Annual Technical Report "Flowfield Computer Graphics" NASA Grant NCC2-430

This final report covers the period:
October 1, 1991 - January 31, 1993

by

Dr. Richard Desautel
Principal Investigator and Professor,
Aerospace Engineering

Annual Technical Report
"Flowfield Computer Graphics"
NASA Grant NCC2-430

This final report covers the period October 1, 1991 - January 31, 1993

by

Dr. Desautel
Principal Investigator and
Professor, Aerospace Engineering

Date: 5/10/93

dick Desautel

Dr. Dick Desautel

Final Report: NCC2-430
"Flowfield Computer Graphics"
(Reporting Period: October 1, 1991 - January 31, 1993)

1.0 Research Objectives

The objectives of this research includes supporting the Aerothermodynamics Branch's research by developing graphical visualization tools for both the branch's adaptive grid code and flow field ray tracing code.

The completed research for the reporting period include development of a graphical user interface (GUI) and implementation into NAS's Flowfield Analysis Software Tool kit (FAST) for both the adaptive grid code (SAGE) and the flow field ray tracing code (CISS).

2.0 Research Progress

The research progress is reported herein in terms of the following subjects:

1) SAGE GUI
2) CISS GUI

2.1 SAGE GUI

SAGE, Self Adapting Grid, by Carol Davies of the Aerothermodynamics Branch, is a grid editor that refines the grid, based on a set of criterias. NASA has several technical reports explaining the concepts and use of SAGE.

2.1.1 SAGE Objective

The objective for SAGE is to develop a graphical user interface that displays all the parameter settings along with a suite of visualization tools. The basic visualization tools are obtained by using NAS's FAST application with SAGE.

2.1 SAGE Use

The use of SAGE through FAST saves research time because changes to the SAGE parameters are displayed immediately on the screen. Otherwise, the changes are ran in a batch fashion and then displayed.

2.1 SAGE GUI Documentation

The SAGE GUI follows the same layout as the other NAS modules. To execute the SAGE module, the user must be familiar with using FAST. Documentation for FAST can be obtained from the NAS Applications Group. To use SAGE, select SAGE from the Modules menu. A window will appear with SAGE at the heading. In the right half of the window, are the SAGE parameters. In the left half of the window, the sliders are used for both the displayed grid and the adaption region. After setting the parameters, click adapt grid button and the result will be displayed. Click undo to return to the original grid. A picture of the GUI along with the code appears in the appendix.

2.1 Results

The implementation of SAGE into FAST was completed with ninety percent of the feature set available from the original SAGE code. The SAGE code itself was preserved in its Fortran state with minimal modifications. The SAGE code and GUI code has been turned over to the NAS Application's group for further development. A picture of the GUI and the source code appear in appendix A.

2.2 CISS GUI

CISS, Constructed Interferograms Schlieren and Shadowgraphs, by Dr. Leslie Yates of the Aerothermodynamics Branch, is a graphical tool that simulates experimental images through the use of psuedo ray tracing techniques. NASA has several technical reports explaining the physics of CISS.

2.2.1 CISS Objective

The objective for CISS is to develop a graphical user interface that displays all the parameter settings along with a suite of visualization tools. The basic visualization tools are obtained by using NAS's FAST application with CISS.

2.2 CISS GUI Documentation

The procedure for accessing CISS is the same as SAGE. Also, the parameter layout in the CISS window is similar to SAGE with the parameters of CISS appearing in the right half of the screen.

To use CISS, first setup the surfaces if there are any. This is done by moving the sliders and selecting the appropriate i or j or k

surface. Then depress the Surface button to record this entry. If there is multiple surfaces, repeat this procedure. Depressing Reset erases all recorded surfaces.

Next, the integration parameter set must be set. These integration parameters are common to all the image options and must be computed separately since this is the time consuming part. The integration parameters are the computational density, the experimental density, the viewing angle psi, theta and phi, scaling, z-depth for 2-dimensions, the number of rays to use, the number of times these rays transverse the flow, and most importantly the exposure that determines the intensity. Depressing integrate button will now execute this.

The interferogram supports finite vertical fringes and infinite vertical fringes. The interferogram parameters are the wavelength of light and the number of fringes. Depressing the interferogram button will display the image immediately.

The shadowgraph has only one parameter, object distance, to compute its image. Depressing the shadowgraph button will display the image immediately.

The schlieren supports both the horizontal and vertical knife edge. Depressing the the schlieren button will display the image immediately.

In displaying any of the images, the integrate section need not be recomputed as long as the parameters in integrate are not changed except for the exposure parameter. The exposure is now separate from the integrate.

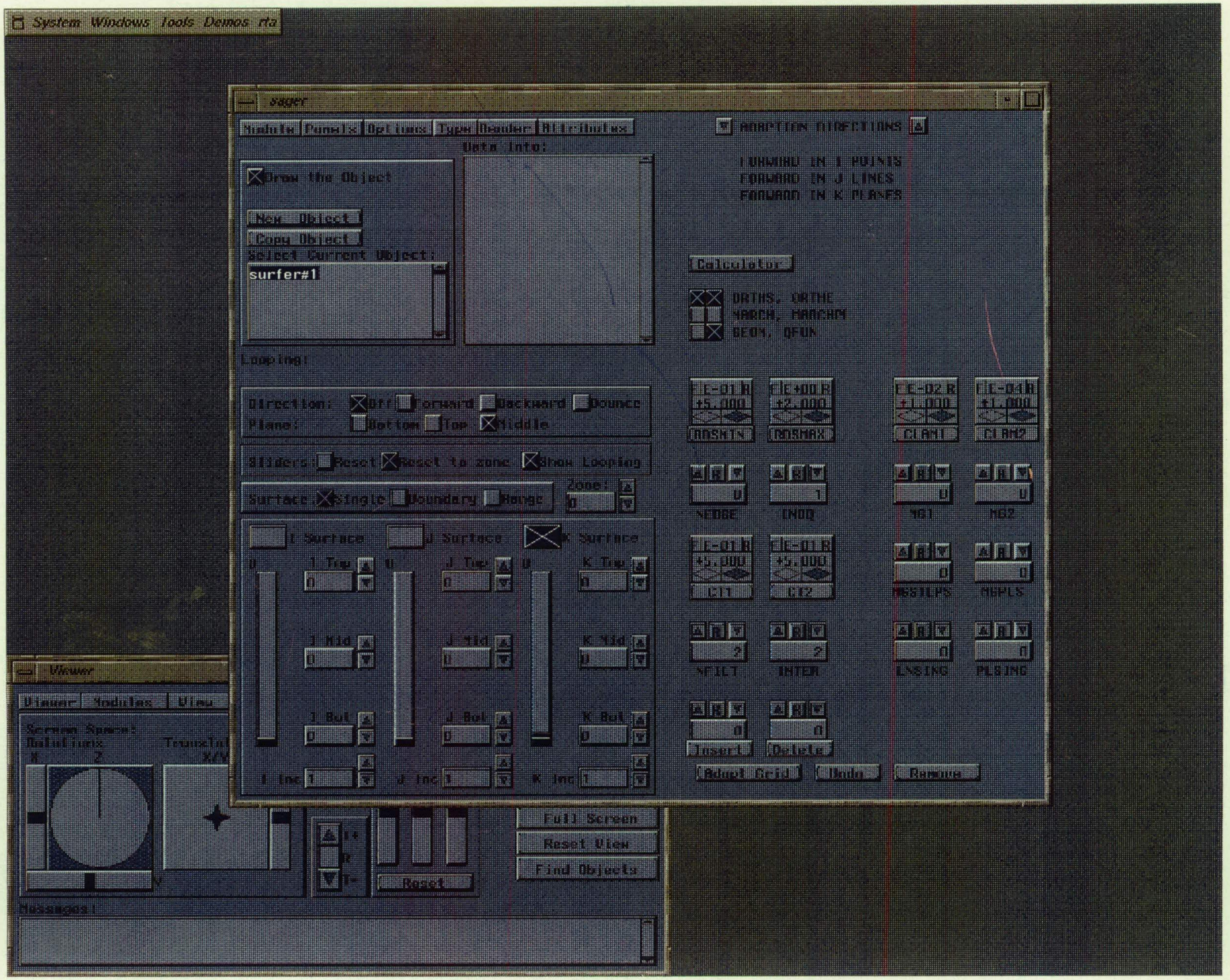
2.3 CISS GUI Result

The implementation of CISS into FAST was completed with ninety-five percent of the feature set available from the original CISS code. The CISS code itself was preserved in its Fortran state with minimal modifications. The CISS code within FAST is know as FISST, Flowfield Interferogram Schlieren Shadowgraph Tookit. The CISS or FISST has been turned over to the NAS Application's group for further development. A picture of the GUI and the source code appear in Appendix B.

3.0 Conclusions

SAGE and CISS have been successfully implemented into the NAS's FAST graphics application. NAS application group will now support and release these two modules from within FAST.

Appendix A: SAGE GUI with Source Code


System Windows Tools Demos rtz
surfer
Modules
Panels
Options
Types
Headers
Attributes
Data Info:
Draw the Object
New Object
Copy Object
Select Current Object:
surfer#1
Looping!
Direction:
Off
Forward
Backward
Bounce
Plane:
Bottom
Top
Middle
Sliders:
Reset
Reset to zone
Show Looping
Surface:
Single
Boundary
Range
Zone:
I Surface
J Surface
K Surface
I Typ
J Typ
K Typ
I Mid
J Mid
K Mid
I Bot
J Bot
K Bot
I Inc
J Inc
K Inc
ANAPTION DIRECTIONS
FORWARD IN I POINTS
FORWARD IN J LINES
FORWARD IN K PLANES
Calculator
ORTHO, ORTHE
MARCH, MARCHM
GEOM, QFUN
Insert
Delete
Adapt Grid
Undo
Remove
Viewer
Viewers
Modules
View
Screen Space:
Rotations
Translate
Full Screen
Reset View
Find Objects
Reset
Messages:

```
#
#       Makefile for first module
#
#       instructions:
#
#               make            compiles code to update executable file
#               make clean      removes object files
#               make clobber    removes object files and the executable file
#               make dirty      touches source files
#               make link       removes executable file
#               make lint       runs lint on each module source file
#


#
#       include global definitions
#

include ../../Makefile.defs

#
#       must define CFLAGS here to be equal to:
#
#               $(IFLAGS)               -I search path for include files
#               $(GLOBALDEFINES)        -D global defines (e.g., -DDEBUG)
#               $(GLOBALCFLAGS)         e.g., -g, -O2 and other optional flags
#
#       (MYFLAGS is for additional flags you may want to add after these)
#

MYCFLAGS =
CFLAGS =        $(IFLAGS) $(GLOBALDEFINES) $(GLOBALCFLAGS) $(MYCFLAGS)
FFLAGS =        -static -mp -pfa -WE,-O=5,-UR=2,-UR2=200,-LM=20000,-LM2=50000

#
#       external library link flags
#

LFLAGS = -lgl_s -lbsd -lm -lF77 -lU77  \
        -lI77_mp -lI77_mp_G0 -lI77 -lI77_G0     \
        -lU77 -lU77_G0 -lisam -lisam_G0 -lm     \
        -lmpc -ll


#
#       module application executable file
#

MOD =           $(INSTALLDIR)sager

#
#       object files
#

MODOBJS =       main.o          \
                panels.o        \
                sager.o         \
                sagerSupport.o

#
#       source files
#

MODSRC =        main.c          \
                panels.c        \
                sagerSupport.c  \
                sager.f

#
#       FAST library archive files
#

FASTLIBS =      $(LIBDIR)libfldpan.a    \
                $(LIBDIR)libfldmtn.a    \
                $(LIBDIR)libviewI.a     \
                $(LIBDIR)libpanu.a      \
                $(LIBDIR)libcmap.a      \
                $(LIBDIR)libfpanel.a    \
                $(LIBDIR)libmem.a       \
                $(LIBDIR)libmodule.a    \
                $(LIBDIR)libobj.a       \
                $(LIBDIR)liblist.a      \
                $(LIBDIR)libcps.a       \
                $(LIBDIR)libfgl.a

#
#       include module definitions
#

include ../Makefile.defs
```



```
#ifndef FUNCS_H
#define FUNCS_H

/*++ funcs.h
 *
 * PURPOSE :
 *
 *              File contains prototyped declarations of public (non-static)
 *              functions of this module.
 *
 * I/O:
 *
 *              None.
 *
 * STANDARDS VIOLATIONS:
 *
 *              None.
 *
 * AUTHORS :
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY :
 *
 *              7/91
 *
 * NOTES :
 *
 --*/

#define MODULE_NAME     "sager"
#define OBJECT_NAME     "surfer$"

/* panels.c */

extern void init_panels( int panel_x, int panel_y );
extern int get_looping( void );
extern void update_looping( void );
extern void update_minmax( void );

extern void wait_until_object_is_redrawn( void );
extern int redraw_object_when_unlocked;
extern int call_update_minmax_after_drawing;

#endif /* FUNCS_H */
```

```
/*++ main.c
 *
 * PURPOSE :
 *
 *              File contains the function main() for the sager program
 *              (a FAST module).
 *
 * I/O:
 *
 *              None.
 *
 * STANDARDS VIOLATIONS:
 *
 *              None.
 *
 * AUTHORS :
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY :
 *
 *              4/89
 *
 * NOTE :
 *
 *      Functions declared in this file :
 *
 *      int     main()
 *
 *      External Funtions called by functions within this file :
 *
 *      extern int      get_no_redraw()                 libpanu
 *      extern int      get_quit_module()               libmodule
 *      extern void     process_hub_command()           libmodule
 *
 *      extern void     init_module()                   file init.c
 *      extern void     exit_module()                   file init.c
 *      extern int      get_looping()                   file panels.c
 *      extern void     update_looping()                file panels.c
 *      extern void     update_minmax()                 file panels.c
 *
 *      External variables used by functions within this file :
 *
 *
 --*/

/*------------------------------ INCLUDES ------------------------------------*/

#include        <stdio.h>       /* for NULL etc.                        */
#include        <fgl.h>         /* for required by panel.h              */
#include        <fast_panel.h>  /* for pnl_naptime & pnl_block          */
#include        <panel_utils.h> /* for get_no_redraw()                  */
#include        <Module.h>      /* for command passing                  */
#include        "funcs.h"       /* for function declarations            */

/*------------------------------ FUNCTIONS -----------------------------------*/
```

```
/*++ int main( int argc, char* argv[] )
 *
 * PURPOSE :
 *
 *              Invokes the sager program and handles the main event loop.
 *
 * AUTHORS :
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY :
 *
 *              4/89
 *
 * INPUT PARAMETERS :
 *
 *              int     argc;           argument count from the hub
 *              char    *argv[];        argument value strings from the hub
 *
 * OUTPUT PARAMETERS :
 *
 *              None
 *
 * FUNCTION RETURN :
 *
 *              int 0
 *
 * GLOBAL VARIABLES USED :
 *
 *              None
 *
 * FILES USED :
 *
 *              None
 *
 * NOTES :
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              extern int      get_no_redraw()         libpanu
 *              extern int      get_quit_module()       libmodule
 *              extern void     process_hub_command()   libmodule
 *
 *              extern void     init_module()           file init.c
 *              extern void     exit_module()           file init.c
 *              extern int      get_looping()           file panels.c
 *              extern void     update_looping()        file panels.c
 *              extern void     update_minmax()         file panels.c
 *
 --*/

/*---------------------------------- main ------------------------------------*/
int* gArgc;
```



```
char** gArgv;

int main( int argc, char* argv[] )
{
        gArgc = &argc;
        gArgv = argv;
        /* set initial panel nap time */

        pnl_naptime = HZ / 5;

        /*
         * Initialize the module:
         * read arguments from the hub and establish a connection then
         * build the panels
         */

        init_module( argc, argv );

        /* Initialize the panels */

        init_panels( get_module_panel_x(), get_module_panel_y() );
/*        init_calc_panels( get_module_panel_x(), get_module_panel_y() ); */

        /* Tell the hub we are done initializing */

        send_hub_command( "DONE_INITIALIZING" );

        /* Enter main event loop */

        while ( get_quit_module() == 0 )
        {
                if ( pnl_dopanel() || get_looping() )
                {
                        if ( ! get_no_redraw() )
                        {
                                update_looping();
                                update_minmax();
                        }
                }
                else if ( call_update_minmax_after_drawing )
                {
                        wait_until_object_is_redrawn();
                        redraw_object_when_unlocked = 0;
                        update_looping();
                        update_minmax();
                        redraw_object_when_unlocked = 1;
                        call_update_minmax_after_drawing = 0;
                }
                else pnl_naptime = HZ / 5; /* reset panel nap time */

                process_hub_command();

                /* process the next command */

                command_buffer_process( MODULE );
        }

        /* exit and clean up */

        exit_module();
        return 0;
}

/*------------------------------ END OF main ---------------------------------*/
```

```
/*------------------------- END OF FILE main.c -------------------------------*/
```

```
/*++ panels.c
 *
 * PURPOSE:
 *
 *              File contains the functions used for creating the panels.
 *
 *
 * I/O:
 *
 *              None.
 *
 * STANDARDS VIOLATIONS:
 *
 *              None.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              6/89
 *              7/91    prototyped functions
 *
 * NOTE :
 *
 *      Functions declared in this file :
 *
 *              void    init_panels()
 *              Panel*  main_panel();
 *              int     get_looping();
 *
 *      Support functions called by the above panel routines:
 *
 *      External functions called by functions within this file :
 *
 *      extern float    Norm()                          Fgl
 *      extern float    Denorm()                        Fgl
 *      extern int      view_start_write_lock()         libviewI
 *      extern int      view_end_write_unlock()         libviewI
 *      extern int      view_single_buffer()            libviewI
 *      extern int      view_set_draw_mode()            libviewI
 *      extern int      shm_get_local_address()         libviewI
 *      extern int      shm_destroy_local_address()     libviewI
 *      extern int      init_fast_cmap()                libcmap
 *      extern int      module_menu()                   libpanu
 *      extern void     draw_palettes()                 libpanu
 *      extern Panel*   color_panel()                   libpanu
 *      extern void     mark_menu()                     libpanu
 *      extern void     close_parent_panel()            libpanu
 *      extern void     clear_typeout()                 libpanu
 *      extern int      set_selection_num()             libpanu
 *      extern void     fix_color_panel()               libpanu
 *      extern void     set_typein_ival()               libpanu
 *      extern void     set_typein_fval()               libpanu
 *      extern float    get_typein_fval()               libpanu
 *      extern int      get_data_minmax()               libflddata
 *      extern int      get_data_list_minmax()          libflddata
 *      extern int      print_field_node_info();        libfldpan
 *      extern int      set_fld_data_selection()        libfldpan
 *      extern void     update_fld_data_panel()         libfldpan
 *      extern Panel*   fld_data_panel()                libfldpan
 *      extern void     Error()                         libmodule
 *      extern void     Warning()                       libmodule
 *
 *      Panel Library functions
 *
--*/

/*------------------------------ INCLUDES ------------------------------------*/

#include        <stdio.h>       /* for NULL etc.                        */
#include        <stdlib.h>      /* for atoi()                           */
#include        <string.h>      /* for string stuff                     */
#include        <ctype.h>       /* for isdigit() etc.                   */
#include        <math.h>        /* for atof()                           */
#include        <fld_pan.h>     /* for GRID_SCALAR_VECTOR_TYPEOUT       */
#include        <panel_utils.h> /* for typedefs and FLOAT_STRING_FORMAT */
#include        <fast_cmap.h>   /* for init_fast_cmap()                 */
#include        <Object.h>      /* for OBJECT_NAME_LENGTH, etc.         */
#include        <grid_surface.h>/* for Grid_Surface typedef             */
#include        <fast_error.h>  /* for ERROR() macro                    */
#include        <fast_memory.h> /* for DDIM3() and DDIM4() macros       */
#include        <fld_list.h>    /* for SCALAR, NUM_VARS etc             */
#include        <get_data.h>    /* for req_*()                          */
#include        <fast_cmap.h>   /* for MAX_SCMAPS & colormap functions  */
#include        <Module.h>      /* for scripting stuff                  */
#include        <ViewI.h>       /* for locking/unlocking                */
#include        "funcs.h"       /* for function declarations            */

/*-------------- FORWARD DECLARATIONS OF PRIVATE FUNCTIONS ----------------*/

static tGraphicObject make_new_object( char* name );
static Grid_Surface* lock_object( tGraphicObject object );
int lock_cur_object( void );
int unlock_cur_object( void );
static int inc_lock_count( void );
static int get_object_id( char* object_name );
static tGraphicObject attach_object( int object_id );
static void detach_cur_object( void );
static void delete_an_object( char* script_command );
static void deallocate_object_data( tGraphicObject object );

static void copy_color( int attribute );
static void copy_colors( void );
static void update_colors( void );
static void set_default_colors( void );

static Panel* main_panel( char* title, int win_x, int win_y );
static Panel* minmax_panel( char* title, int win_x, int win_y );

static Panel* contour_panel( char* title, int win_x, int win_y );
static void set_contour_legend( char* str );

static void file_io_func( char* file_name, int mode );
static void panels_func( int group, int item );
static void attributes_func( int group, int item );
static void set_attributes_func( char* script_command );
static void type_func( int group, int item );
static void set_type_func( char* script_command );
static void render_func( int group, int item );
static void set_render_func( char* script_command );
static void options_func( int group, int item );
static void set_options_func( char* script_command );

static void dump_state( void );
```



```
static void reset_state( int reset_actuators );

static void select_object(char* script_command );
static void new_object( char* script_command );
static void copy_object( char* script_command );
static void update_objects( Actuator* a );

static void surface_buttons_func( int row, int col, int state );
static void set_surface_buttons_func( char* script_command );
static void slider_buttons_func( int row, int col, int state );
static void set_slider_buttons_func( char* script_command );
static void loop_buttons_func( int row, int col, int state );
static void set_loop_buttons_func( char* script_command );
static void zone_func( int new_zone );
static void set_zone_func( char* script_command );
static void toggle_draw_func( char* script_command );
static void ijk_ranges_func( int dir, float ranges[5] );
static void set_ijk_ranges_func( char* script_command );
static void direction_func( int dir );
static void set_direction_func( char* script_command );
static void boundary_surfaces_func( int selections[3][3] );
static void set_boundary_surfaces_func( char* script_command );

static void reset_ijk_ranges( void );

static void data_select( int type, int req_num, int fld_num,
            FLDDataPtr fld_data_ptr );
static void reset_data( int type );

static void vector_scale( int index, float new_value );
static void set_vector_scale( char* script_command );

static void set_minmax_func( char* script_command );
static void reset_minmax( char* script_command );
static void adjust_minmax_func( char* script_command );
static void set_minmax_mode( char* script_command );
static void invert_clip_test( char* script_command );

static void clip_and_norm( int state );
static int copy_normals( void );
static void delete_normals( void );
static void check_delete_normals( int new_dir, int new_ranges[3][5] );

static int copy_contours( void );
static void delete_contours( void );

static void first_object_name( char* name );
void update_object_typeout( void );

void update_actuators( void );
static void update_data_info( void );
static void update_dims( int type, int dims[3] );

static void update_minmax_sliders( float minmax[2][4], ... type );
static void update_legend( float min_val, float max_val, Actuator** act );
static void update_palettes( int clip_or_norm );

/*------------------------------ DEFINES -------------------------------------*/

#ifndef MINIMUM
#define MINIMUM         0
#endif

#ifndef MAXIMUM
#define MAXIMUM         1
#endif

#ifndef MIN
#define MIN(a, b)       ((a) < (b) ? (a) : (b))
#endif

#ifndef MAX
#define MAX(a, b)       ((a) > (b) ? (a) : (b))
#endif

#define ROUND(x)        ((x) >= 0.0 ? ((int) ((x) + 0.5)) : ((int) ((x) - 0.5)))

#define LIMIT_TO_MIN(a, min)    if ((a) < (min)) (a) = (min)
#define LIMIT_TO_MAX(a, max)    if ((a) > (max)) (a) = (max)
#define LIMIT_TO(a, min, max)   {LIMIT_TO_MIN(a, min); LIMIT_TO_MAX(a, max);}

#define MATCH_DIMS(dims1, dims2)                                        \
                                                                        \
    (dims1[I] == dims2[I] && dims1[J] == dims2[J] && dims1[K] == dims2[K])

/*------------------------- Panel Attributes ---------------------------*/

/* main panel */

#define MAIN_TITLE              MODULE_NAME
#define MAIN_WIN_X              660
#define MAIN_WIN_Y              800

/* data panel */

#define DATA_TITLE              MODULE_NAME
#define DATA_WIN_X              750
#define DATA_WIN_Y              10
#define MATCH_GRID_BUTTON       1

static int data_formats[3] = { ALL_FORMATS, STRUCTURED, STRUCTURED };

/* contour panel */

#define SAGER_CONTOUR_LABEL     "Surfer: Contours"
#define SAGER_CONTOUR_WIN_X     800
#define SAGER_CONTOUR_WIN_Y     50

/* scalar minmax panel */

#define MINMAX_TITLE            "Surfer: Scalar Minmax"
#define MINMAX_WIN_X            735
#define MINMAX_WIN_Y            20


/*------------------------- Vector Scale Panel -------------------------*/

#define VECTOR_TITLE            "Surfer: Vector Scale"
#define VECTOR_WIN_X            1125
#define VECTOR_WIN_Y            582

/*---------------------- Vector Panel's Scale Group --------------------*/

#define SCALE_GROUP_X           0.0
#define SCALE_GROUP_Y           (-1.0 * SCALE_GROUP_HEIGHT)
#define SCALE_GROUP_FRAMED      0
#define SCALE_GROUP_NUM_VALUES  2
```

```
static char* scale_group_labels[SCALE_GROUP_NUM_VALUES] =
        { "Vectors", "Frame Lines" };

/* initial vector and frame scale factors */

static float scale_group_values[SCALE_GROUP_NUM_VALUES] = { 1.0, 0.1 };

/* add 100% to vector and 1% to frame while the slider is pegged */

static float scale_group_slider_rates[SCALE_GROUP_NUM_VALUES] = { 1.0, 0.01 };

/* multiply the vector by 2 and the frame by 1.1 on each up button click */

static float scale_group_button_rates[SCALE_GROUP_NUM_VALUES] = { 2.0, 1.1 };

static char* scale_group_script_commands[SCALE_GROUP_NUM_VALUES] =
        {
                "VECTOR_SCALE %f",
                "FRAME_SCALE %f"
        };



/*---------------------------- Color Panel ----------------------------*/

#define COLOR_TITLE             MODULE_NAME
#define COLOR_WIN_X             1030
#define COLOR_WIN_Y             300


static char* color_labels[NUM_GS_COLORS] =
        {
                "Line",
                "Point",
                "Contour",
                "Vector",
                "Polygon",
                "Outline",
                "Glyph"
        };

static int color_indices[NUM_GS_COLORS];
static int default_color_indices[NUM_GS_COLORS] =
        {
                /* LINE_COLOR           */      RED,
                /* POINT_COLOR          */      YELLOW,
                /* CONTOUR_COLOR        */      BLUE,
                /* VECTOR_COLOR         */      CYAN,
                /* POLYGON_COLOR        */      WHITE,
                /* OUTLINE_COLOR        */      WHITE,
                /* GLYPH_COLOR          */      WHITE
        };

static float color_rgbs[NUM_GS_COLORS][3];
static float default_color_rgbs[NUM_GS_COLORS][3] =
        {
                /* LINE_COLOR           */      {1.0, 0.0, 0.0},
                /* POINT_COLOR          */      {1.0, 1.0, 0.0},
                /* CONTOUR_COLOR        */      {0.0, 0.0, 1.0},
                /* VECTOR_COLOR         */      {0.0, 1.0, 1.0},
                /* POLYGON_COLOR        */      {1.0, 1.0, 1.0},
                /* OUTLINE_COLOR        */      {1.0, 1.0, 1.0},
                /* GLYPH_COLOR          */      {1.0, 1.0, 1.0}
        };
        };

/*---------------------------- Panels Menu ----------------------------*/

#define PANELS_MENU_ITEMS       5
#define PANELS_MENU_GROUPS      1
#define PANELS_MENU_MARKABLE    0
#define PANELS_MENU_JUSTIFY     PWL_LEFT_JUSTIFY

static int panels_menu_items_per_group[PANELS_MENU_GROUPS]={PANELS_MENU_ITEMS};
static int panels_menu_selections[PANELS_MENU_ITEMS];
static int panels_menu_markable[PANELS_MENU_GROUPS] = { 0 };
static char* panels_menu_labels[1 + PANELS_MENU_ITEMS] =
        {
                "Panels",
                "Data...",
                "Colors...",
                "Vector Scale...",
                "Scalar Minmax...",
                "Contours..."
        };

static char* panels_menu_script_commands[1 + PANELS_MENU_ITEMS] =
        {
                "OPEN_PANEL %s",
                "DATA",
                "COLORS",
                "VECTOR_SCALE",
                "SCALAR_MINMAX",
                "CONTOURS"
        };



/*------------ panels ----------*/

#define MAIN_PANEL              0
#define DATA_PANEL              1
#define COLOR_PANEL             2
#define VECTOR_PANEL            3
#define MINMAX_PANEL            4
#define CONTOUR_PANEL           5
#define NUM_PANELS              6


/*---------------------------- Options Menu ----------------------------*/

#define OPTIONS_MENU_ITEMS      5
#define OPTIONS_MENU_GROUPS     5
#define OPTIONS_MENU_MARKABLE   1
#define OPTIONS_MENU_JUSTIFY    PWL_LEFT_JUSTIFY

static int options_menu_items_per_group[OPTIONS_MENU_GROUPS] =
        { 1, 1, 1, 1, 1 };
static int options_menu_selections[OPTIONS_MENU_ITEMS] =
        { 0, 0, 0, 0, 0 };
static int options_menu_markable[OPTIONS_MENU_GROUPS] =
        { 1, 1, 0, 0, 0 };
static char* options_menu_labels[1 + OPTIONS_MENU_ITEMS] =
        {
                "Options",
                "Draw Outline",
                "Draw Glyph",
```



```
                "Recompute Normals",
                "Reset",
                "Debug"
        };


/*---------------------------- Type Menu ----------------------------*/

#define TYPE_MENU_ITEMS 3
#define TYPE_MENU_GROUPS        1
#define TYPE_MENU_MARKABLE      1
#define TYPE_MENU_JUSTIFY       PWL_LEFT_JUSTIFY

static int type_menu_items_per_group[TYPE_MENU_GROUPS] =
        { TYPE_MENU_ITEMS };
static int type_menu_selections[TYPE_MENU_ITEMS] = { 1, 0, 0 };
static int type_menu_markable[TYPE_MENU_GROUPS] = { 1 };
static char* type_menu_labels[1 + TYPE_MENU_ITEMS] =
        {
                "Type",
                "Grid",
                "Scalar",
                "Vector"
        };

static char* type_menu_script_commands[1 + TYPE_MENU_ITEMS] =
        {
                "TYPE %s",
                "GRID",
                "SCALAR",
                "VECTOR"
        };


/*---------------------------- Render Menu ----------------------------*/

#define RENDER_MENU_ITEMS       12
#define RENDER_MENU_GROUPS      1
#define RENDER_MENU_MARKABLE    1
#define RENDER_MENU_JUSTIFY     PWL_CENTER_JUSTIFY

static int render_menu_items_per_group[RENDER_MENU_GROUPS] =
        { RENDER_MENU_ITEMS };
static int render_menu_markable[RENDER_MENU_GROUPS] =
        { 1 };
static int render_menu_selections[RENDER_MENU_ITEMS] =
        { 0,0,0,0,0,0,1,0,0,0,0,0 };
static char* render_menu_labels[1 + RENDER_MENU_ITEMS] =
        {
                "Render",
                "Points",
                "Dots",
                "Bubbles",
                "Stars",
                "Grid Lines 1",
                "Grid Lines 2",
                "Grid Lines 1 and 2",
                "Contour Lines",
                "Plain Vectors",
                "Normalized Vectors",
                "Flat Polygons",
                "Smooth Polygons"
        };

static char* render_menu_script_commands[1 + RENDER_MENU_ITEMS] =
        {
                "RENDER %s",
                "POINTS",
                "DOTS",
                "BUBBLES",
                "STARS",
                "GRID_LINES_1",
                "GRID_LINES_2",
                "GRID_LINES_1_AND_2",
                "CONTOUR_LINES",
                "PLAIN_VECTORS",
                "NORMALIZED_VECTORS",
                "FLAT_POLYGONS",
                "SMOOTH_POLYGONS"
        };

/*---------------------------- Attributes Menu ----------------------------*/

#define ATTRIBUTES_MENU_ITEMS           17
#define ATTRIBUTES_MENU_GROUPS          8
#define ATTRIBUTES_MENU_MARKABLE        1
#define ATTRIBUTES_MENU_JUSTIFY         PWL_RIGHT_JUSTIFY

static int attributes_menu_items_per_group[ATTRIBUTES_MENU_GROUPS] =
        { 2, 2, 2, 2, 3, 2, 2, 2 };
static int attributes_menu_selections[ATTRIBUTES_MENU_ITEMS] =
        { 0,1, 1,0, 1,0, 1,0, 1,0,0, 1,0, 1,0, 1,0 };
static int attributes_menu_markable[ATTRIBUTES_MENU_GROUPS] =
        { 1, 1, 1, 1, 1, 1, 1, 1 };
static char* attributes_menu_labels[1 + ATTRIBUTES_MENU_ITEMS] =
        {
                "Attributes",
                "Constant Color Contours",
                "Scalar   Color Contours",
                "Constant Color Vectors",
                "Scalar   Color Vectors",
                "No       Vector Tips",
                "Arrowed Vector Tips",
                "Straight Clip",
                "Faded    Clip",
                "Default Shading",
                "Shading On",
                "Shading Off",
                "Frame Lines Off",
                "Frame Lines On",
                "Standard Normals",
                "Reversed Normals",
                "Surface Normals",
                "Zone Normals"
        };

static char* attributes_menu_script_commands[1 + ATTRIBUTES_MENU_ITEMS] =
        {
                "ATTRIBUTES %s",
                "CONSTANT_COLOR_CONTOURS",
                "SCALAR_COLOR_CONTOURS",
                "CONSTANT_COLOR_VECTORS",
                "SCALAR_COLOR_VECTORS",
                "NO_VECTOR_TIPS",
                "ARROWED_VECTOR_TIPS",
                "STRAIGHT_CLIP",
                "FADED_CLIP",
```

```
                "DEFAULT_SHADING",
                "SHADING_ON",
                "SHADING_OFF",
                "FRAME_LINES_OFF",
                "FRAME_LINES_ON",
                "STANDARD_NORMALS",
                "REVERSED_NORMALS",
                "SURFACE_NORMALS",
                "ZONE_NORMALS"
        };


/*------------------------- Main Panel Buttons -------------------------*/

#define DRAW_BUTTON_LABEL               "Draw the Object"
#define UPDATE_OBJECTS_BUTTON_LABEL     "Update All Objects"
#define NEW_OBJECT_BUTTON_LABEL         "New  Object"
#define COPY_OBJECT_BUTTON_LABEL        "Copy Object"

#define OBJECT_TYPEOUT_LABEL            "Select Current Object:"
#define OBJECT_TYPEOUT_LINES    5
#define OBJECT_TYPEOUT_COLS     OBJECT_NAME_LENGTH

/* data info typeout */

#define DATA_INFO_LABEL         "Data Info:"
#define DATA_INFO_COLS          20
#define DATA_INFO_LINES         13
#define DATA_INFO_BUF_SIZE      (2 * DATA_INFO_COLS * DATA_INFO_LINES)



/*------------------------- Loop Buttons Group -------------------------*/

#define LOOPING_LABEL   "Looping:"

#define NUM_LOOP_BUTTONS                9
#define LOOP_BUTTONS_FRAMED             1
#define LOOP_BUTTONS_RADIO_GROUPING     RADIO_ROW
#define LOOP_BUTTONS_ROWS               3

static int loop_buttons_per_row[LOOP_BUTTONS_ROWS] = { 4, 3, 2 };
static int loop_buttons_types[NUM_LOOP_BUTTONS] =
        { PNL_RADIO_BUTTON, PNL_RADIO_BUTTON, PNL_RADIO_BUTTON, PNL_RADIO_BUTTON,
          PNL_RADIO_BUTTON, PNL_RADIO_BUTTON, PNL_RADIO_BUTTON, PNL_RADIO_BUTTON,
          PNL_RADIO_BUTTON, PNL_RADIO_BUTTON, PNL_RADIO_BUTTON };
static int loop_buttons_selections[NUM_LOOP_BUTTONS] =
        { 1, 0, 0,  0, 0, 1,  1, 0 };
static char* loop_buttons_labels[NUM_LOOP_BUTTONS] =
        {
                "Off", "Forward", "Backward", "Bounce",
                "Bottom", "Top ", "Middle",
                "Single", "Multi"
        };
static char* loop_buttons_row_labels[LOOP_BUTTONS_ROWS] =
        {
                "Direction: ",
                "Plane:     ",
                "Zones:     "
        };
static char* loop_buttons_script_load_commands[LOOP_BUTTONS_ROWS]=
        {
                "LOOP %s",
                "PLANE %s",
                "LOOP_ZONE %s"
        };
static char* loop_buttons_script_param_commands[LOOP_BUTTONS_ROWS][4]=
        {
                { "OFF", "FORWARD", "BACKWARD", "BOUNCE" },
                { "BOTTOM", "TOP", "MIDDLE", " " },
                { "SINGLE", "MULTI", " ", " " }
        };



/*------------------------- Slider Buttons Group -------------------------*/

#define NUM_SLIDER_BUTTONS              3
#define SLIDER_BUTTONS_FRAMED           1
#define SLIDER_BUTTONS_RADIO_GROUPING   RADIO_ROW
#define SLIDER_BUTTONS_ROWS             1

static int slider_buttons_per_row[SLIDER_BUTTONS_ROWS] = {NUM_SLIDER_BUTTONS};
static int slider_buttons_types[NUM_SLIDER_BUTTONS] =
        { PNL_BUTTON, PNL_TOGGLE_BUTTON, PNL_TOGGLE_BUTTON };
static int slider_buttons_selections[NUM_SLIDER_BUTTONS] =
        { 0, 1, 1 };
static char* slider_buttons_labels[NUM_SLIDER_BUTTONS] =
        { "Reset", "Reset to zone", "Show Looping" };
static char* slider_buttons_row_labels[1] = { "Sliders:" };


/*------------------------- Surface Buttons Group -------------------------*/

#define NUM_SURFACE_BUTTONS             3
#define SURFACE_BUTTONS_FRAMED          1
#define SURFACE_BUTTONS_RADIO_GROUPING  RADIO_ROW
#define SURFACE_BUTTONS_ROWS            1

static int surface_buttons_per_row[SURFACE_BUTTONS_ROWS] =
        { NUM_SURFACE_BUTTONS };
static int surface_buttons_types[NUM_SURFACE_BUTTONS] =
        { PNL_RADIO_BUTTON, PNL_RADIO_BUTTON, PNL_RADIO_BUTTON };
static int surface_buttons_selections[NUM_SURFACE_BUTTONS] =
        { 1, 0, 0 };
static char* surface_buttons_labels[NUM_SURFACE_BUTTONS] =
        { "Single", "Boundary", "Range" };
static char* surface_buttons_row_labels[1] = { "Surface:" };
static char* surface_buttons_script_commands[1 + NUM_SURFACE_BUTTONS] =
        {
                "SURFACE %s",
                "SINGLE",
                "BOUNDARY",
                "SURFACE_RANGE"
        };

/*------------------------- Zone Typein Group -------------------------*/

#define ZONE_TYPEIN_TYPE                INT_TYPE
#define ZONE_TYPEIN_FORMAT              "%d"
#define ZONE_TYPEIN_LABEL               "Zone:"
#define ZONE_TYPEIN_LABEL_TYPE          PNL_LABEL_TOP_LEFT
#define ZONE_TYPEIN_WIDTH               5
#define ZONE_TYPEIN_LIMITED             0

static float zone_typein_values[3] = { 0.0, 0.0, 0.0 };
```



```
/*------------------------- Slider Group -------------------------*/

#define SLIDER_GROUP_FRAMED             1
#define SLIDER_GROUP_TYPE               INT_TYPE
#define SLIDER_GROUP_SELECTION_BUTTONS  1

static char slider_group_label_letters[3] = { 'I', 'J', 'K' };
static char* slider_group_direction_label = " Surface";

static float slider_group_values[3][5] = /* [I/J/K][START/END/INC/CUR/DIM] */
        {
                { 0.0, 0.0, 1.0, 0.0, 0.0 },
                { 0.0, 0.0, 1.0, 0.0, 0.0 },
                { 0.0, 0.0, 1.0, 0.0, 0.0 }
        };

static int slider_group_selections[3][3] = /* [I/J/K][START/END/MID] */
        {
                { 1, 1, 1 },
                { 1, 1, 1 },
                { 1, 1, 1 }
        };









/*------------------------- Actuator Attributes -------------------------*/


/* top left corner of the panel */

#define X_ORIGIN                0.0
#define Y_ORIGIN                0.0
#define SPACE                   0.1

/* pull-down menus are at the top */

#define X_MENU                  (X_ORIGIN + 1.25)
#define Y_MENU                  Y_ORIGIN
#define X_MENU_INC              3.0
#define Y_MENU_INC              0.5

/* panel buttons */

#define X_BUTTON_INC            3.0
#define Y_BUTTON_INC            0.6


#define INT_LABEL_FORMAT        "%d"



/*------------------------- Scalar Minmax Stuff -------------------------*/

/* frame coordinates */

#define SMM_FRAME_HEIGHT        15.0

#define X_FRAME_ACT             0.0
#define Y_FRAME_ACT             6.0

/* for string typeins & labels */

#define ZERO_STRING_16          "0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"

/* multislider dimensions */

#define MULTISLIDER_WIDTH       0.5
#define MULTISLIDER_HEIGHT      6.0

/* palette dimensions */

#define PALETTE_WIDTH           1.0
#define PALETTE_HEIGHT          6.0


#define INVERT_CLIP_TEST_LABEL  "Clip Inside"

#define CLIP_LABEL              "  Clip:   "
#define CLIP_TOP_LABEL          "Top:"
#define CLIP_BOT_LABEL          "Bot:"
#define CLIP_TOP_NUMBER_STR     ZERO_STRING_16
#define CLIP_BOT_NUMBER_STR     ZERO_STRING_16
#define RESET_CLIP_LABEL        " Reset "

#define NORM_LABEL              "Normalize:"
#define NORM_TOP_LABEL          "Top:"
#define NORM_BOT_LABEL          "Bot:"
#define NORM_TOP_NUMBER_STR     ZERO_STRING_16
#define NORM_BOT_NUMBER_STR     ZERO_STRING_16
#define RESET_NORM_LABEL        " Reset "

#define LEGEND_LABEL            " Legend:  "
#define LEGEND_MAX_LABEL        "Max:"
#define LEGEND_MIN_LABEL        "Min:"
#define LEGEND_MAX_NUMBER_STR   ZERO_STRING_16
#define LEGEND_MIN_NUMBER_STR   ZERO_STRING_16
#define RESET_LEGEND_LABEL      " Reset "
#define NUM_LEGEND_VALUES       11

#define UPDATE_MMSLIDERS_LABEL  "Update Minmax Sliders"
#define AUTO_MINMAX_UPDATE_LABEL        "Auto Minmax Update"
#define MULTI_ZONE_MINMAX_LABEL         "Multi Zone Minmax"
#define SINGLE_ZONE_MINMAX_LABEL        "Single Zone Minmax"
#define SUBSET_MINMAX_LABEL             "Zone Subset Minmax"
#define SURFACE_MINMAX_LABEL            "Surface Minmax"
#define SURFACE_SUBSET_MINMAX_LABEL     "Surface Subset Minmax"


#define LEGEND                  2
#define LOW                     0
#define MED                     1
#define HI                      2
#define NUM_PALETTE_SETS        3       /* CLIP, NORM, LEGEND */
#define NUM_PALETTE_TYPES       3       /* LOW, MED, HI */

#define MIN_MAP                 ((float) get_function_index(0.0))
#define MAX_MAP                 ((float) get_function_index(1.0))

#define PALETTE_X_INC           5.0
```

```
#define RESET_CLIP_ID           "1"
#define RESET_NORM_ID           "2"
#define RESET_LEGEND_ID         "3"
#define CLIP_MSLIDER_ID         "4"
#define NORM_MSLIDER_ID         "5"
#define CLIP_TOP_ID             "6"
#define CLIP_BOT_ID             "7"
#define NORM_TOP_ID             "8"
#define NORM_BOT_ID             "9"
#define LEGEND_MAX_ID           "10"
#define LEGEND_MIN_ID           "11"
#define AUTO_MINMAX_UPDATE_ID   "12"
#define UPDATE_MMSLIDERS_ID     "13"
#define MULTI_ZONE_MINMAX_ID    "14"
#define SINGLE_ZONE_MINMAX_ID   "15"
#define SUBSET_MINMAX_ID        "16"
#define SURFACE_MINMAX_ID       "17"
#define SURFACE_SUBSET_MINMAX_ID "18"
#define CONTOUR_MIN_ID          "19"
#define CONTOUR_MAX_ID          "20"
#define CONTOUR_NUM_ID          "21"
#define CONTOUR_INC_ID          "22"



/* for routines that lock and unlock the object */

#ifdef DEBUG
#define DEBUG_LOCKING   1
#else
#define DEBUG_LOCKING   0
#endif

#if DEBUG_LOCKING
#define DPRINT(s, v)    printf(s, v)
#else
#define DPRINT(s, v)
#endif


/*------------------------------- TYPEDEFS ---------------------------------*/

/*
 * define structures that will contain pointers to the important actuators
 * and groups of actuators that will be updated by various action funcs
 */

struct main_actuators
{
        MenuGroup*      panels_menu_group;
        MenuGroup*      attributes_menu_group;
        MenuGroup*      type_menu_group;
        MenuGroup*      render_menu_group;
        MenuGroup*      options_menu_group;
        Actuator*       object_frame;
        Actuator*       draw_object_button;
        Actuator*       update_objects_button;
        Actuator*       new_object_button;
        Actuator*       copy_object_button;
        Actuator*       object_typeout;
        Actuator*       looping_label;
        ButtonGroup*    loop_buttons_group;
        ButtonGroup*    slider_buttons_group;
        ButtonGroup*    surface_buttons_group;
        TypeinGroup*    zone_typein_group;
        SliderGroup*    slider_group;
        Actuator*       data_info_typeout;
};

typedef struct  main_actuators  Main_Acts;

struct minmax_actuators
{
        Actuator*       minmax_frame;
        Actuator*       invert_clip_test_button;
        Actuator*       clip_label;
        Actuator*       norm_label;
        Actuator*       legend_label;
        Actuator*       clip_top_typein;
        Actuator*       norm_top_typein;
        Actuator*       legend_max_typein;
        Actuator*       clip_multislider;
        Actuator*       norm_multislider;
        Actuator*       legend_multislider;
        Actuator*       clip_bot_typein;
        Actuator*       norm_bot_typein;
        Actuator*       legend_min_typein;
        Actuator*       reset_clip_button;
        Actuator*       reset_norm_button;
        Actuator*       reset_legend_button;
        Actuator*       update_minmax_sliders_button;
        Actuator*       auto_minmax_update_button;
        Actuator*       mode_buttons[NUM_SCALAR_MINMAX_MODES];
        Actuator*       palettes[NUM_PALETTE_SETS][NUM_PALETTE_TYPES];
        Actuator*       legend_labels[NUM_LEGEND_VALUES];
};

typedef struct  minmax_actuators        Minmax_Acts;

struct contour_actuators
{
        Actuator*       palette;
        Actuator*       contour_min_typein;
        Actuator*       contour_max_typein;
        Actuator*       contour_num_typein;
        Actuator*       contour_inc_typein;
        Actuator*       contour_labels[NUM_LEGEND_VALUES];
};

typedef struct  contour_actuators       Contour_Acts;

/*-------------------------------- GLOBALS ---------------------------------*/

/*
 * Cur_object is a pointer to the current graphical object which contains a
 * grid surface as part of its structure. We allocate and attach to an object
 * during initialization. We remain attached to this object until another one
 * is selected. Cur_object is locked each time before it is accessed in an
 * action func and before exiting the action func it is unlocked (however
 * we still remain attached to it). This enforces mutual exclusion between
 * this process and others (e.g., viewer). Sometimes one action func
 * will call another in which case we do not actually lock the object
 * more then once but rather increment a locked count. This locked count is
 * also decremented after each unlock. However deeply nested the lock are,
 * there must eventually be a matching unlock so that when we are done
 * accessing the object other processes may do so. Sometimes we must read or
```



```
 * write to viewer (indirectly via the hub). In these cases we must be
 * sure that the object is unlocked otherwise we could become deadlocked.
 * (If we are waiting for the hub, the hub is waiting for viewer and
 * viewer is waiting for us to unlock our object so it can draw.)
 */

static tGraphicObject   cur_object;

static char     cur_object_name[OBJECT_NAME_LENGTH]; /* name of object */

/*
 * Grid_surf is a pointer to the current grid surface structure that is part
 * of cur_object. We must call lock_cur_object() each time before accessing
 * this pointer and call unlock_cur_object() each time after accessing it.
 * For example, we must lock at start of every action func that accesses
 * this pointer (nearly all do) and unlock before returning (or exiting).
 */

Grid_Surface*   grid_sager;     /* define globally so sagerSupport can use it */

/* array holds pointers to all Surfer panels */

static Panel*   panels[NUM_PANELS];

/* structures hold pointers to all important actuators */

static Main_Acts        main_acts;
static Minmax_Acts      minmax_acts;
static ScaleGroup*      scale_group;
static Contour_Acts     contour_acts;

/* special flags and modes */

static int      locked;                 /* current object lock count    */
static int      loop_mode;              /* looping status               */
static int      show_looping = 1;       /* redraw sliders when looping? */
static int      update_actuators_mode;  /* set when selecting a new obj */
static int      update_all_objects_mode;/* apply to all objects?        */
static int      interactive;            /* interactive or scripted      */

int     redraw_object_when_unlocked = 1;
int     call_update_minmax_after_drawing = 0;


static User_token_def   tokens[] =
  {
    { "SELECT_OBJECT",          select_object,              REPEAT_YES },
    { "NEW_OBJECT",             new_object,                 REPEAT_YES },
    { "COPY_OBJECT",            copy_object,                REPEAT_YES },
    { "DELETE_OBJECT",          delete_an_object,           REPEAT_YES },
    { "DRAW",                   toggle_draw_func,           REPEAT_YES },

    { "TYPE",                   set_type_func,              REPEAT_YES },
    { "RENDER",                 set_render_func,            REPEAT_YES },
    { "ATTRIBUTES",             set_attributes_func,        REPEAT_YES },
    { "OPTIONS",                set_options_func,           REPEAT_YES },
    { "DIRECTION",              set_direction_func,         REPEAT_YES },
    { "SLIDER",                 set_ijk_ranges_func,        REPEAT_NO  },
    { "SURFACE",                set_surface_buttons_func,   REPEAT_YES },
    { "BOUNDARY",               set_boundary_surfaces_func, REPEAT_NO  },
    { "LOOP",                   set_loop_buttons_func,      REPEAT_YES },
    { "PLANE",                  set_loop_buttons_func,      REPEAT_YES },
    { "LOOP_ZONE",              set_loop_buttons_func,      REPEAT_YES },
    { "SLIDER_ACTION",          set_slider_buttons_func,    REPEAT_YES },
    { "MINMAX_MODE",            set_minmax_modes,           REPEAT_YES },
    { "AUTO_MINMAX",            set_minmax_modes,           REPEAT_YES },
    { "UPDATE_MINMAX",          set_minmax_modes,           REPEAT_YES },
    { "CLIP",                   invert_clip_test,           REPEAT_YES },
    { "MINMAX",                 set_minmax_func,            REPEAT_NO  },
    { "RESET_MINMAX",           reset_minmax,               REPEAT_YES },
    { "ZONE",                   set_zone_func,              REPEAT_YES },
    { "VECTOR_SCALE",           set_vector_scale,           REPEAT_NO  },
    { "FRAME_SCALE",            set_vector_scale,           REPEAT_NO  },
    { "CONTOUR",                set_contour_legend,         REPEAT_YES },
    { NULL, NULL, REPEAT_NO }
  };




/*****************************************************************************/
/************************** PUBLIC FUNCTIONS *********************************/
/*****************************************************************************/




/*++ int get_looping( void )
 *
 * PURPOSE:
 *
 *              Return the looping status of sager.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              12/89
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
```

```
*               int             looping status
*
* GLOBAL VARIABLES USED:
*
*               extern int      loop_mode       defined in this file
*
* FILES USED:
*
*               None
*
* NOTES:
*
* NON-STANDARD CODE :
*
* CALLED BY :
*
* FUNCTIONS CALLED :
*
*               None
*
--*/

/*------------------------------ get_looping ---------------------------------*/

int get_looping( void )
{
        return loop_mode;
}

/*--------------------------- END OF get_looping -----------------------------*/




/*++ void update_looping( void )
 *
 * PURPOSE:
 *
 *              Updates the grid surface structure to specify the next
 *              surface to display based on the current state and the
 *              looping control variables. Note: update sliders button
 *              must be on for this to occur.
 *
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              7/89
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Main_Acts        main_acts;              declared in this file
 *      extern Grid_Surface*    grid_saqer;             declared in this file
 *      extern int              loop_mode               declared in this file
 *      extern int              show_looping            declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *      void            update_data_info()      defined in this file
 *
--*/

/*------------------------------ update_looping ------------------------------*/

void update_looping( void )
{
        int             d;                      /* I, J, or K           */
        int*            ranges;                 /* ranges for d         */
        int             cur_zone;               /* current grid zone    */

#ifdef DEBUG
printf("inside update_looping() with loop_mode = %d, show_looping = %d\n",
loop_mode, show_looping);
#endif

        if ( redraw_object_when_unlocked == 1 )
                if ( loop_mode == LOOP_OFF || show_looping == 0 ) return;

        lock_cur_object();

        /* check if we must advance to the next zone */

        if (    grid_saqer -> loop_zone == MULTI_ZONE &&
                grid_saqer -> loop_new_zone != 0                )
        {
            cur_zone = ACCESS2(main_acts.zone_typein_group, get_ivalue);

            if      (grid_saqer -> loop_new_zone ==  1) ++cur_zone;
            else if (grid_saqer -> loop_new_zone == -1) --cur_zone;

            ACCESS4(main_acts.zone_typein_group, set_ivalue, cur_zone, 1);
            grid_saqer -> loop_new_zone = 0;
```



```
        }

        /* update the data info typeout */

        update_data_info();

        d = grid_saqer -> direction;
        ranges = grid_saqer -> ranges[d];

#ifdef DEBUG
printf("grid_saqer -> loop_mode     = %d\n", grid_saqer -> loop_mode);
printf("grid_saqer -> loop_dir      = %d\n", grid_saqer -> loop_dir);
printf("grid_saqer -> loop_zone     = %d\n", grid_saqer -> loop_zone);
printf("grid_saqer -> loop_new_zone = %d\n", grid_saqer -> loop_new_zone);
#endif

        unlock_cur_object();

        /* redraw the current slider to indicate the new position */

        ACCESS5(main_acts.slider_group, set_ivalues, d, ranges, 0);
}

/*--------------------------- END OF update_looping --------------------------*/




/*++ void update_minmax( void )
 *
 * PURPOSE:
 *
 *              Updates the minmax panel to reflect the possibley altered
 *              grid surface minmax values Note: the minmax mode must be
 *              a surface type and the update slider butto must be on for
 *              this to occur.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              3/90
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Minmax_Acts      minmax_acts     defined in this file
 *      extern Grid_Surface*    grid_saqer;             declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      void            update_minmax_sliders()         defined in this file
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
--*/

/*------------------------------ update_minmax -------------------------------*/

void update_minmax( void )
{
        if (    minmax_acts.update_minmax_sliders_button -> val == 0.0 ||
                minmax_acts.mode_buttons[MULTI_ZONE_MINMAX] -> val == 1.0 ||
                minmax_acts.mode_buttons[SINGLE_ZONE_MINMAX] -> val == 1.0  )
        {
                return;
        }

        /* update and redraw the sliders */

        lock_cur_object();
        update_minmax_sliders(grid_saqer -> minmax, LEGEND);
        update_minmax_sliders(grid_saqer -> minmax, CLIP);
        update_minmax_sliders(grid_saqer -> minmax, NORM);
        unlock_cur_object();

        call_update_minmax_after_drawing = 1;

}

/*--------------------------- END OF update_minmax ---------------------------*/









/*++ void init_panels( int panel_x, int panel_y )
```

```
*
* PURPOSE:
*
*             Creates the panels.
*
* AUTHORS:
*
*             Todd Plessel
*             NASA Ames Research Center
*             Sterling Software
*
* REVISION HISTORY:
*
*             6/89
*
* INPUT PARAMETERS:
*
*             int     panel_x         screen x position of main panel
*             int     panel_y         screen y position of main panel
*
* OUTPUT PARAMETERS:
*
*             None
*
* FUNCTION RETURN:
*
*             None
*
* GLOBAL VARIABLES USED:
*
*       extern Grid_Surface*    grid_sager              defined in this file
*       extern Panel*           panels[]                defined in this file
*       extern Main_Acts        main_acts               defined in this file
*       extern ScaleGroup*      scale_group             defined in this file
*       extern char*            color_labels[NUM_GS_COLORS]     defined in this file
*       extern int              color_indices[NUM_GS_COLORS]    defined in this file
*       extern float            color_rgbs[NUM_GS_COLORS][3]     ""
*       extern tGraphicObject   cur_object              defined in this file
*       extern char     cur_object_name[OBJECT_NAME_LENGTH]     defined in this file
*
* FILES USED:
*
*             None
*
* NOTES:
*
* NON-STANDARD CODE :
*
* CALLED BY :
*
* FUNCTIONS CALLED :
*
*       extern int      init_fast_cmap()                libcmap
*       extern Panel*   make_panel()                    libpanu
*       ScaleGroup*     make_scale_group()              libpanu
*       extern Panel*   color_panel()                   libpanu
*       extern Panel*   fld_data_panel()                libfldpan
*       extern void     update_fld_data_panel()         libfldpan
*       extern void     exit_module()                   file init.c
*       Panel*          main_panel()                    defined in this file
*       Panel*          minmax_panel()                  defined in this file
*       Panel*          contour_panel()                 defined in this file
*       void            data_select()                   defined in this file
*       void            reset_state()                   defined in this file
*       void            set_default_colors()            defined in this file
*       void            update_object_typeout()         defined in this file
*       void            copy_color()                    defined in this file
*       tGraphicObject  make_new_object()               defined in this file
*       tGraphicObject  attach_object()                 defined in this file
*       int             get_object_id()                 defined in this file
*       void            first_object_name()             defined in this file
*       int             lock_cur_object()               defined in this file
*       int             unlock_cur_object()             defined in this file
*
--*/

/*------------------------------ init_panels ---------------------------------*/

void init_panels( int panel_x, int panel_y )
{
        int             must_init_object = 0;   /* init object if new   */

        /* load up the tokens */

        command_init(tokens);

        /* find or create the initial graphic object */

        first_object_name(cur_object_name);

        if (cur_object_name[0] != '\0')
        {
                /* get the id of the graphic object and attach to it */

                cur_object = attach_object(get_object_id(cur_object_name));

                if (cur_object == NULL)
                {
                    Error("Cannot attach to first graphic object! Exiting...");
                    exit_module();
                }
        }
        else /* create the initial graphic object */
        {
                strcpy(cur_object_name, OBJECT_NAME);

                cur_object = make_new_object(cur_object_name);

                if (cur_object == NULL)
                {
                    Error("Cannot make initial graphic object! Exiting...");
                    exit_module();
                }

                must_init_object = 1;
        }

        /* lock it until we are done initializing */

        lock_cur_object();

        /*
         * Redefine the panel library colors
         * Note: EVERY module in the FAST environment that uses panels or
         * the colormap must call this function before invoking its panels!
         */
```



```
        init_fast_cmap(0);

        /* Create the panels: */

        /* create main panel */

        panels[MAIN_PANEL] = main_panel( MAIN_TITLE, panel_x, panel_y );

        if (panels[MAIN_PANEL] == NULL)
        {
                Error("Cannot create a panel! Exiting...\n");
                exit_module();
        }

        /*---------------------- create fld data panel ----------------------*/

        panels[DATA_PANEL] = fld_data_panel(    DATA_TITLE,
                                                DATA_WIN_X,
                                                DATA_WIN_Y,
                                                GRID_SOLUTION_SCALAR_TYPEOUT,
                                                MATCH_GRID_BUTTON,
                                                data_formats,
                                                data_select     );

        if (panels[DATA_PANEL] == NULL)
        {
                Error("Cannot create a panel! Exiting...\n");
                exit_module();
        }

        /*---------------------- create minmax panel ------------------------*/

        panels[MINMAX_PANEL] = minmax_panel(    MINMAX_TITLE,
                                                MINMAX_WIN_X,
                                                MINMAX_WIN_Y    );

        if (panels[MINMAX_PANEL] == NULL)
        {
                Error("Cannot create a panel! Exiting...\n");
                exit_module();
        }

        /*---------------------- create vector scale panel ------------------*/

        panels[VECTOR_PANEL] = make_panel(      VECTOR_TITLE,
                                                VECTOR_WIN_X,
                                                VECTOR_WIN_Y, 0, 1      );

        if (panels[VECTOR_PANEL] == NULL)
        {
                Error("Cannot create a panel! Exiting...\n");
                exit_module();
        }

        /* add a scale group to this panel */

        scale_group = make_scale_group(         panels[VECTOR_PANEL],
                                                SCALE_GROUP_X,
                                                SCALE_GROUP_Y,
                                                SCALE_GROUP_FRAMED,
                                                SCALE_GROUP_NUM_VALUES,
                                                scale_group_labels,
                                                scale_group_values,
                                                scale_group_slider_rates,
                                                scale_group_button_rates,
                                                vector_scale);

        if (scale_group == NULL)
        {
                Error("Cannot create actuators! Exiting...\n");
                exit_module();
        }

        /*---------------------- create color panel -------------------------*/

        /* initilize to the default colors */

        set_default_colors();

        panels[COLOR_PANEL] = color_panel(      COLOR_TITLE,
                                                COLOR_WIN_X,
                                                COLOR_WIN_Y,
                                                NUM_GS_COLORS,
                                                color_labels,
                                                color_indices,
                                                &color_rgbs[0][0],
                                                copy_color );

        if (panels[COLOR_PANEL] == NULL)
        {
                Error("Cannot create a panel! Exiting...\n");
                exit_module();
        }

        /*---------------------- create contour panel -----------------------*/

        panels[CONTOUR_PANEL] =contour_panel(   SAGER_CONTOUR_LABEL,
                                                SAGER_CONTOUR_WIN_X,
                                                SAGER_CONTOUR_WIN_Y     );


        if (panels[CONTOUR_PANEL] == NULL)
        {
                Error("Cannot create a panel! Exiting...\n");
                exit_module();
        }

        if ( must_init_object )
        {
                /* init grid surface data structure */

                reset_state(0);
        }
        else
        {
                /* update the actuators to reflect the grid surface */

                update_actuators();
        }

        /* update the data panel */

        update_fld_data_panel();

        /* unlock the current object so it may be drawn */

        unlock_cur_object();
```

```
        /* this causes communication to viewer so unlock first */

        update_object_typeout();
}

/*------------------------- END OF init_panels ------------------------------*/




/************************************************************************/
/************************** PRIVATE FUNCTIONS ***************************/
/************************************************************************/



/*++ static tGraphicObject make_new_object( char* name )
 *
 * PURPOSE:
 *
 *              Allocates and returns a grid surface object
 *              based on the given name but extended with a unique number.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              11/90
 *
 * INPUT PARAMETERS:
 *
 *              char*           name            name of object
 *
 * OUTPUT PARAMETERS:
 *
 *              char*           name            name of object and number
 *
 * FUNCTION RETURN:
 *
 *              extern  tGraphicObject  grid surface object
 *
 * GLOBAL VARIABLES USED:
 *
 *      None
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 *              void    init_panels()                   defined in this file
 *              void    new_object()                    defined in this file
 *
 * FUNCTIONS CALLED :
 *
 *              extern tGraphicObject   view_single_buffer()    libviewI
 *
 --*/

/*------------------------- make_new_object ---------------------------------*/

static tGraphicObject make_new_object( char* name )
{
        tGraphicObject  object;         /* pointer to a new object      */

        DPRINT("make_new_object(name = %s)...\n", name);
        object = view_single_buffer(GRID_SURFACE, name, sizeof(Grid_Surface));
        DPRINT("    generates object = %d\n", object);
        return object;
}

/*------------------------- END OF make_new_object --------------------------*/




/*++ static Grid_Surface* lock_object( tGraphicObject object )
 *
 * PURPOSE:
 *
 *              Locks the given object and returns a pointer to its
 *              grid surface structure.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              11/90
 *
 * INPUT PARAMETERS:
 *
 *              tGraphicObject  object  object to lock
```



```
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              Grid_Surface*   gs      pointer to a grid surface structure
 *
 * GLOBAL VARIABLES USED:
 *
 *              None
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              extern char*    view_start_write_lock() libviewI
 *
 --*/

/*------------------------------- lock_object -------------------------------*/

static Grid_Surface* lock_object( tGraphicObject object )
{
        Grid_Surface*   gs = NULL;

        DPRINT("lock_object(object = %d)...\n", object);

        if ((gs = (Grid_Surface *) view_start_write_lock(object)) == NULL)
        {
                Error("Cannot lock to object!");
        }

        DPRINT("   generates gs = %d\n", gs);

        return gs;
}

/*------------------------- END OF lock_object ------------------------------*/




/*++ static int lock_cur_object( void )
 *
 * PURPOSE:
 *
 *              Locks the current object (unless it has already been locked).
 *              In either case it increments and returns the locked count.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              7/89
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              int             locked count
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern  tGraphicObject  cur_object;             declared in this file
 *      extern Grid_Surface*    grid_saver;             declared in this file
 *      extern int              locked                  declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              extern char*    view_start_write_lock() libviewI
 *              extern void     exit_module()           file init.c
 *
 --*/

/*------------------------------- lock_cur_object ---------------------------*/

int lock_cur_object( void )
{
        DPRINT("lock_cur_object(): before locked = %d\n", locked);

        if ( locked < 0 )
        {
                printf("%s: locked = %d\n", MODULE_NAME, locked);
                Error( "Too many unlocks - Exiting..." );

#ifdef DEBUG
{
int produce_core_file = 0; produce_core_file /= produce_core_file;
}
#endif

                exit_module();
```

```
        }
        else if ( locked++ == 0 )
        {
                DPRINT("after if ( locked++ == 0 ) locked = %d so LOCKING\n",locked);

                if ((grid_sager = (Grid_Surface *)
                        view_start_write_lock(cur_object)) == NULL)
                {
                        Error( "Cannot lock to object! Exiting..." );
                        exit_module();
                }
        }

        DPRINT("lock_cur_object(): after  locked = %d\n", locked);

        return locked;
}

/*-------------------------- END OF lock_cur_object --------------------------*/




/*++ static int unlock_cur_object( void )
 *
 * PURPOSE:
 *
 *              Unlocks the current object and NULLS grid_sager
 *              (unless it has already been unlocked).
 *              In either case it decrements and returns the locked count.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              7/89
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              int     locked count
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern  tGraphicObject  cur_object;             declared in this file
 *      extern Grid_Surface*    grid_sager;             declared in this file
 *      extern int              locked                  declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              extern ????     view_end_write_unlock()         libviewI
 *
 --*/

/*-------------------------- unlock_cur_object --------------------------*/

int unlock_cur_object( void )
{
        DPRINT("unlock_cur_object(): before locked = %d\n", locked);

        if ( locked <= 0 )
        {
                printf("%s: locked = %d\n", MODULE_NAME, locked);
                Error( "Too many unlocks - Exiting..." );

#ifdef DEBUG
{
int produce_core_file = 0; produce_core_file /= produce_core_file;
}
#endif

                exit_module();
        }
        else if ( locked-- == 1 )
        {
                DPRINT("after if ( locked-- == 1 ) locked = %d so UNLOCKING\n", locked);
                if ( redraw_object_when_unlocked )
                        view_end_write_unlock(cur_object, ERASE_AND_DRAW);
                else    view_end_write_unlock(cur_object, NO_NEED_TO_DRAW);
                grid_sager = NULL;
        }

        DPRINT("unlock_cur_object(): after  locked = %d\n", locked);

        return locked;
}

/*-------------------------- END OF unlock_cur_object ------------------------*/



void wait_until_object_is_redrawn( void )
{
    lock_cur_object();
    unlock_cur_object();
    view_data_was_drawn( cur_object, WAIT );
}
```



```
/*-------------------------- END OF inc_lock_count --------------------------*/




/*++ static int get_object_id( char* object_name )
 *
 * PURPOSE:
 *
 *              Writes to the Hub and gets the shared memory id of the
 *              named object.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              11/90
 *
 * INPUT PARAMETERS:
 *
 *              char*   object_name     name of the object to find
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              int                     shared memory id of the named object
 *
 * GLOBAL VARIABLES USED:
 *
 *              None
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 *              void    select_object()         defined in this file
 *
 * FUNCTIONS CALLED :
 *
 --*/

/*-------------------------- get_object_id --------------------------*/
```

```
static int get_object_id( char* object_name )
{
        int             id;                     /* shared memory id     */
        char            command[32];


        DPRINT("get_object_id(object_name = %s)...\n", object_name);

        sprintf(command, "GET_OBJECT %s", object_name);

        send_hub_command(command);
        module_read_sock ((char*)&id, sizeof(id));

        DPRINT("   got id = %d\n", id);

        return id;
}
/*-------------------------- END OF get_object_id ----------------------------*/
```

```
/*++ static tGraphicObject attach_object( int object_id )
 *
 * PURPOSE:
 *
 *              Attaches to the given object shared memory id and
 *              returns a graphic object.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              11/90
 *
 * INPUT PARAMETERS:
 *
 *              int     object_id               object id to attach to
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              tGraphicObject  object;         graphic object
 *
 * GLOBAL VARIABLES USED:
 *
 *              None
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 *              void    select_object()         defined in this file
 *
 * FUNCTIONS CALLED :
 *
 *              None
 *
 --*/

/*-------------------------- attach_object -----------------------------------*/

static tGraphicObject attach_object( int object_id )
{
        tGraphicObject  object;         /* graphic object       */


        DPRINT("attach_object(object_id = %d)...\n", object_id);

        if (object_id == -1)
        {
                Error("object_id = -1 in attach_object()");
                return NULL;
        }

        object = shm_get_local_address(object_id);

        DPRINT("   generates object %d\n", object);

        return object;
}

/*-------------------------- END OF attach_object ----------------------------*/
```

```
/*++ static void detach_cur_object( void )
 *
 * PURPOSE:
 *
 *              Detaches from the current object and NULLS cur_object and
 *              grid_saqer.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
```



```
 * REVISION HISTORY:
 *
 *              11/90
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern tGraphicObject   cur_object              declared in this file
 *      extern Grid_Surface*    grid_saqer              declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 *              void    select_object()                 defined in this file
 *              void    new_object()                    defined in this file
 *
 * FUNCTIONS CALLED :
 *
 *              None
 *
 --*/

/*-------------------------- detach_cur_object -------------------------------*/

static void detach_cur_object( void )
{
        DPRINT("detach_cur_object(): before cur_object = %d\n", cur_object);

        shm_destroy_local_address(cur_object);
        cur_object = NULL;
        grid_saqer = NULL;
}

/*-------------------------- END OF detach_cur_object ------------------------*/
```

```
/*++ static void delete_an_object( char* script_command )
 *
 * PURPOSE:
 *
 *              Deletes the contents of the object named in the script command.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              11/91
 *
 * INPUT PARAMETERS:
 *
 *              char*   script_command
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *              None
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 *              external Hub/Viewer routines
 *
 * FUNCTIONS CALLED :
 *
 *              None
 *
 --*/

/*-------------------------- delete_an_object --------------------------------*/

static void delete_an_object( char* script_command )
{
        int             object_id;
        char            object_name[OBJECT_NAME_LENGTH];
        tGraphicObject  object = NULL;

        parse_command( script_command, "%s", object_name );

        /* Get the object id from the object name */

        object_id = get_object_id( object_name );

        /* Attach to the object */
```

```
        if ( ( object = attach_object( object_id ) ) == NULL )
        {
                Error( "Cannot attach to object" );
                return;
        }

        /* Deallocate any object-specific data */

        deallocate_object_data( object );

        /* Detach from the object */

        shm_destroy_local_address( object ); object = NULL;

        /* Send the command to remove the object from the object list */

        module_command( "Viewer", script_command, NULL );

        /* Generate an updated listing of available objects and select one */

        update_object_typeout();
}

/*------------------------ END OF delete_an_object --------------------------*/









/*++ static void select_object( char* script_command )
 *
 * PURPOSE:
 *
 *              Sets the current object
 *
 * AUTHORS:
 *
 *              Fergus J. Merritt
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              3/90
 *              5/90    Todd Plessel
 *                      modified to work around locking & handle glyph etc.
 *              11/90   Todd Plessel
 *                      simplified and removed glyph and outline handling etc.
 *              4/91    Paul Kalaita
 *                      added command stuff
 *
 * INPUT PARAMETERS:
 *
 *              char*   script_command          actuator/command
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_sager              defined in this file
 *      extern char             cur_object_name[OBJECT_NAME_LENGTH] * * * *
 *
 * FILES USED:
 *
 *      void            update_actuators()              defined in this file
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      extern  void    exit_module()           init.c
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *      void            detach_cur_object()     defined in this file
 *      tGraphicObject  attach_object()         defined in this file
 *      int             get_object_id()         defined in this file
 *      void            update_object_typeout() define in this file
 *
 --*/

/*------------------------------ select_object ------------------------------*/

static void select_object( char* script_command )
{
        tGraphicObject  new_obj;                        /* new object   */
        char            new_obj_name[OBJECT_NAME_LENGTH];
        int             new_obj_id;                     /* new obj id   */
        Actuator*       a = main_acts.object_typeout;


        if ( is_act( script_command ) )
        {
                /* if there are no object names in the list then return */

                if ( ! get_selection_name( a, new_obj_name,
                        OBJECT_NAME_LENGTH ) )
                {
                        return;
                }

                /* if the same object was selected then just return */

                if ( strcasecmp( new_obj_name, cur_object_name ) == 0 ) return;

                load_command( "SELECT_OBJECT %s", new_obj_name );
                return;
        }

        parse_command( script_command, "%s", new_obj_name );
        set_selection_name( a, new_obj_name );
```



```
        /* if the same object was selected then just return */

        if ( strcasecmp( new_obj_name, cur_object_name ) == 0 ) return;

        hourglass_cursor( ON );

        /* write to the hub for the shared memory id of the new selection */

        new_obj_id = get_object_id( new_obj_name );

        /* check the shared memory id for validity */

        if (new_obj_id == -1)
        {
                Warning("Selected object is no longer available!");
                update_object_typeout();
                hourglass_cursor( OFF );
                return;
        }

        /* attach to the new object's shared memory id */

        if ((new_obj = attach_object(new_obj_id)) == NULL)
        {
                Error("Cannot attach to new grid surface object!");
                hourglass_cursor( OFF );
                return;
        }

        /* detach from the current object's shared memory id */

        detach_cur_object();

        /* update the current object to the new one (and copy name) */

        cur_object = new_obj;
        strcpy(cur_object_name, new_obj_name);

        /* update all actuators */

        update_actuators();

        hourglass_cursor( OFF );

}

/*--------------------------- END OF select_object ---------------------------*/









/*++ static void new_object( char* script_command )
 *
 * PURPOSE:
 *
 *              Creates a new object (a default one)
 *              and makes it the current object to modify.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              5/91
 *
 * INPUT PARAMETERS:
 *
 *              char*   script_command          actuator/command
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_sager              defined in this file
 *      extern tGraphicObject   cur_object              defined in this file
 *      extern char             cur_object_name[OBJECT_NAME_LENGTH] * * * *
 *      extern char             cur_object_name[OBJECT_NAME_LENGTH] * * * *
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      void            update_object_typeout()         defined in this file
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *      Grid_Surface*   lock_object()           defined in this file
 *      tGraphicObject  make_new_object()       defined in this file
 *      int             inc_lock_count()        defined in this file
 *      void            detach_cur_object()     defined in this file
 *
 --*/

/*------------------------------ new_object ---------------------------------*/

static void new_object( char* script_command )
{
        Grid_Surface*   new_grid_sager;         /* -> new surface obj   */
        tGraphicObject  new_obj;                /* temp new object      */
        char            new_obj_name[OBJECT_NAME_LENGTH];       /* name */
        int             dir;                    /* I,J,K direction      */
        int             type;                   /* START,END,MID,ZONE   */
```

```
        if ( is_act( script_command ) )
        {
                load_command( "NEW_OBJECT" );
                return;
        }

        /* allocate a new object */

        strcpy(new_obj_name, OBJECT_NAME);

        new_obj = make_new_object(new_obj_name);

        if (new_obj == NULL)
        {
                Error("Cannot allocate new object!");
                return;
        }

        /* lock the new object and point to a new grid surface structure */

        if ((new_grid_sager = lock_object(new_obj)) == (Grid_Surface *)NULL)
        {
                Error("Cannot lock new object!");
                return;
        }

        /* lock the current object while copying (grid_sager is current one) */

        lock_cur_object();

        /* copy the old grid surface object to the new one */

        memcpy( new_grid_sager, grid_sager, sizeof(Grid_Surface) );

        /* Clear the shared memory ids of the normals in the new surface */

        for ( dir = 0; dir < 3; ++dir )
          for ( type = 0; type < 4; ++type )
            new_grid_sager -> field_ids[NORM_ID_INDEX( dir, type )] = -1;

        /* Clear the shared memory id of the contours data */

        new_grid_sager -> field_ids[CONTOURS_ID] = -1;

        /* unlock and detach from the current grid surface object */

        unlock_cur_object();
        detach_cur_object();

        /* update the current object to the new one (and copy name) */

        cur_object = new_obj;
        grid_sager  = new_grid_sager;
        strcpy(cur_object_name, new_obj_name);

        /*
         * must increment locked count here (to account for locked new object)
         */

        inc_lock_count();

        /* Reset everything to the default state */

        reset_state( 1 );

        unlock_cur_object();

        /*
         * update typeout (after lock since this does communication)
         * and select the new object
         */

        update_object_typeout();

}

/*--------------------------- END OF new_object ----------------------------*/




/*++ static void copy_object( char* script_command )
 *
 * PURPOSE:
 *
 *              Copies the current object and makes the copy the current
 *              object to modify.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              6/89
 *
 * INPUT PARAMETERS:
 *
 *              char*   script_command          actuator/command
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_sager              defined in this file
 *      extern tGraphicObject   cur_object              defined in this file
 *      extern char             cur_object_name[OBJECT_NAME_LENGTH] " " " "
 *
 * FILES USED:
 *
```



```
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      int             copy_normals()                  defined in this file
 *      void            delete_normals()                defined in this file
 *      void            update_object_typeout()         defined in this file
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *      Grid_Surface*   lock_object()           defined in this file
 *      tGraphicObject  make_new_object()       defined in this file
 *      int             inc_lock_count()        defined in this file
 *      void            detach_cur_object()     defined in this file
 *              macro USES_NORMALS() is made available by grid_surface.h
 *
 --*/

/*---------------------------- copy_object ---------------------------------*/

static void copy_object( char* script_command )
{
        Grid_Surface*   new_grid_sager;         /* -> new surface obj   */
        tGraphicObject  new_obj;                /* temp new object      */
        char    new_obj_name[OBJECT_NAME_LENGTH];/* temp object name    */


        if ( is_act( script_command ) )
        {
                load_command( "COPY_OBJECT" );
                return;
        }

        /* allocate a new object */

        strcpy(new_obj_name, OBJECT_NAME);

        new_obj = make_new_object(new_obj_name);

        if (new_obj == NULL)
        {
                Error("Cannot allocate new object!");
                return;
        }

        /* lock the new object and point to a new grid surface structure */

        if ((new_grid_sager = lock_object(new_obj)) == (Grid_Surface *)NULL)
        {
                Error("Cannot lock new object!");
                return;
        }

        /* lock the current object while copying (grid_sager is current one) */

        lock_cur_object();

        /* copy the old grid surface object to the new one */

        memcpy( new_grid_sager, grid_sager, sizeof(Grid_Surface) );

        /* unlock and detach from the current grid surface object */

        unlock_cur_object();

        detach_cur_object();

        /* update the current object to the new one (and copy name) */

        cur_object = new_obj;
        grid_sager  = new_grid_sager;
        strcpy(cur_object_name, new_obj_name);

        /*
         * must increment locked count here (to account for locked new object)
         */

        inc_lock_count();

        /* copy any existing normals data if it is needed */

        if (USES_NORMALS(grid_sager)) if (!copy_normals()) delete_normals();

        /* also copy any existing contours data if it is needed */

        if ( grid_sager -> render_mode == CONTOUR_LINES )
          if (!copy_contours()) delete_contours();

        unlock_cur_object();

        /*
         * update typeout (after lock since this does communication)
         * and select the new object
         */

        update_object_typeout();

}

/*--------------------------- END OF copy_object ---------------------------*/




/*++ static void update_objects( Actuator* a )
 *
 * PURPOSE:
 *
 *              Toggles the mode that makes all objects updated whenever
 *              the current object changes.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
```

```
*               Sterling Software
*
* REVISION HISTORY:
*
*               12/90
*
* INPUT PARAMETERS:
*
*               Actuator*       a       update objects button
*
* OUTPUT PARAMETERS:
*
*               None
*
* FUNCTION RETURN:
*
*               None
*
* GLOBAL VARIABLES USED:
*
*       extern int      update_all_objects_mode         defined in this file
*
* FILES USED:
*
*               None
*
* NOTES:
*
* NON-STANDARD CODE :
*
* CALLED BY :
*
* FUNCTIONS CALLED :
*
*               None
*
--*/

/*---------------------------- update_objects -------------------------------*/

static void update_objects( Actuator* a )
{
        update_all_objects_mode = !update_all_objects_mode;
}

/*------------------------ END OF update_objects ----------------------------*/









/*++ static void toggle_draw_func( char* script_command )
 *
 * PURPOSE:
 *
 *               Toggling the draw flag.
 *
 * AUTHORS:
 *
 *               Todd Plessel
 *               NASA Ames Research Center
 *               Sterling Software
 *
 * REVISION HISTORY:
 *
 *               9/90
 *
 *
 * INPUT PARAMETERS:
 *
 *               char*   script_command  script command or actuator
 *
 * OUTPUT PARAMETERS:
 *
 *               None
 *
 * FUNCTION RETURN:
 *
 *               None
 *
 * GLOBAL VARIABLES USED:
 *
 *               extern Grid_Surface*    grid_saqer      defined in this file
 *
 * FILES USED:
 *
 *               None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *       int             lock_cur_object()       defined in this file
 *       int             unlock_cur_object()     defined in this file
 *
--*/

/*---------------------------- toggle_draw_func -----------------------------*/

static void toggle_draw_func( char* script_command )
{
        Actuator*       a = main_acts.draw_object_button;
        char            on_off_str[16];


        if ( is_act( script_command ) )
        {
                interactive = 1;
                load_command( "DRAW %s", ON_OR_OFF( a -> val ) );
                return;
        }

        parse_command( script_command, "%s", on_off_str );

        if ( ! interactive )
        {
                a -> val = (float) ON_OR_OFF_VAL( on_off_str );
                pnl_fixact( a );
```



```
        }

        interactive = 0;

        lock_cur_object();
        grid_saqer -> draw = (int) a -> val;
        unlock_cur_object();
}

/*------------------------ END OF toggle_draw_func ---------------------------*/









/*++ static void deallocate_object_data( tGraphicObject object )
 *
 * PURPOSE:
 *
 *               Deallocates the data associated with the given object.
 *
 * AUTHORS:
 *
 *               Todd Plessel
 *               NASA Ames Research Center
 *               Sterling Software
 *
 * REVISION HISTORY:
 *
 *               11/91
 *
 * INPUT PARAMETERS:
 *
 *               tGraphicObject  object
 *
 * OUTPUT PARAMETERS:
 *
 *               None
 *
 * FUNCTION RETURN:
 *
 *               None
 *
 * GLOBAL VARIABLES USED:
 *
 *               None
 *
 * FILES USED:
 *
 *               None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 *               void    delete_an_object()              defined in this file
 *
 * FUNCTIONS CALLED :
 *
 *               None
 *
--*/

/*------------------------ deallocate_object_data -------------------------*/

static void deallocate_object_data( tGraphicObject object )
{
        Grid_Surface*   gs = NULL;
        int             i;

        /* Lock this object */

        if ( ( gs = lock_object( object ) ) == NULL )
        {
                Error( "Cannot lock to object" );
                return;
        }

        /* Delete field ids for object-specific data */

        for ( i = START_I_NORM_ID; i < NUM_FIELDS; ++i )
        {
                if ( gs -> field_ids[i] != -1 )
                {
                        SDEALLOCATE( gs -> field_ids[i] );
                        gs -> field_ids[i] = -1;
                }
        }

        /* Unlock this object and mark it so it WON'T be drawn */

        view_end_write_unlock( object, NO_DRAW );
}

/*------------------- END OF deallocate_object_data ------------------------*/









/*++ static void copy_colors( int attribute )
 *
 * PURPOSE:
 *
 *               Copy the current color into grid surface structure
 *
 * AUTHORS:
 *
 *               Todd Plessel
 *               NASA Ames Research Center
 *               Sterling Software
 *
 * REVISION HISTORY:
 *
```

```
*               4/90
*
* INPUT PARAMETERS:
*
*               int     attribute       index of changed color
*
* OUTPUT PARAMETERS:
*
*               None
*
* FUNCTION RETURN:
*
*               None
*
* GLOBAL VARIABLES USED:
*
*       extern int      color_indices[NUM_GS_COLORS]; this file
*       extern float    color_rgbs[NUM_GS_COLORS][3]; this file
*       extern Grid_Surface*    grid_sager;             this file
*
* FILES USED:
*
*               None
*
* NOTES:
*
* NON-STANDARD CODE :
*
* CALLED BY :
*
* FUNCTIONS CALLED :
*
*       int             lock_cur_object()       defined in this file
*       int             unlock_cur_object()     defined in this file
*
--*/

/*----------------------------- copy_color ---------------------------------*/

static void copy_color( int attribute )
{
  if ( attribute >= 0 && attribute < NUM_GS_COLORS )
  {
    lock_cur_object();
    grid_sager -> color_indices[attribute] = color_indices[attribute];
    grid_sager -> color_rgbs[attribute][R] = color_rgbs[attribute][R];
    grid_sager -> color_rgbs[attribute][G] = color_rgbs[attribute][G];
    grid_sager -> color_rgbs[attribute][B] = color_rgbs[attribute][B];
    unlock_cur_object();
  }
}

/*----------------------------- END OF copy_color ----------------------------*/




/*++ static void copy_colors( void )
```

```
*
* PURPOSE:
*
*               Copy the current colors into grid surface structure
*
* AUTHORS:
*
*               Todd Plessel
*               NASA Ames Research Center
*               Sterling Software
*
* REVISION HISTORY:
*
*               4/90
*
* INPUT PARAMETERS:
*
*               None
*
* OUTPUT PARAMETERS:
*
*               None
*
* FUNCTION RETURN:
*
*               None
*
* GLOBAL VARIABLES USED:
*
*       extern int      color_indices[NUM_GS_COLORS]; this file
*       extern float    color_rgbs[NUM_GS_COLORS][3]; this file
*       extern Grid_Surface*    grid_sager;             this file
*
* FILES USED:
*
*               None
*
* NOTES:
*
* NON-STANDARD CODE :
*
* CALLED BY :
*
* FUNCTIONS CALLED :
*
*       int             lock_cur_object()       defined in this file
*       int             unlock_cur_object()     defined in this file
*
--*/

/*----------------------------- copy_colors ---------------------------------*/

static void copy_colors( void )
{
  int i;
  for ( i = 0; i < NUM_GS_COLORS; ++i ) copy_color( i );
}

/*----------------------------- END OF copy_colors ---------------------------*/
```



```
/*++ static void update_colors( void )
*
* PURPOSE:
*
*               Copy the colors from the grid surface structure into
*               the global colors array and then fixes the color edit
*               panel to display these colors.
*
* AUTHORS:
*
*               Todd Plessel
*               NASA Ames Research Center
*               Sterling Software
*
* REVISION HISTORY:
*
*               4/90
*
* INPUT PARAMETERS:
*
*               None
*
* OUTPUT PARAMETERS:
*
*               None
*
* FUNCTION RETURN:
*
*               None
*
* GLOBAL VARIABLES USED:
*
*       extern int      color_indices[NUM_GS_COLORS]; this file
*       extern float    color_rgbs[NUM_GS_COLORS][3]; this file
*       extern Grid_Surface*    grid_sager;             this file
*
* FILES USED:
*
*               None
*
* NOTES:
*
* NON-STANDARD CODE :
*
* CALLED BY :
*
* FUNCTIONS CALLED :
*
*       extern void     fix_color_panel()               1.2.anu
*       int             lock_cur_object()       defined in this file
*       int             unlock_cur_object()     defined in this file
*
--*/

/*----------------------------- update_colors ------------------------------*/

static void update_colors( void )
{
        int             i;                      /* loop on colors       */
        lock_cur_object();

        for (i = 0; i < NUM_GS_COLORS; i++)
        {
                color_indices[i] = grid_sager -> color_indices[i];

                color_rgbs[i][R] = grid_sager -> color_rgbs[i][R];
                color_rgbs[i][G] = grid_sager -> color_rgbs[i][G];
                color_rgbs[i][B] = grid_sager -> color_rgbs[i][B];
        }

        fix_color_panel();

        unlock_cur_object();

}

/*----------------------------- END OF update_colors -----------------------*/




/*++ static void set_default_colors( void )
*
* PURPOSE:
*
*               Copy the default colors into globals
*
* AUTHORS:
*
*               Todd Plessel
*               NASA Ames Research Center
*               Sterling Software
*
* REVISION HISTORY:
*
*               4/90
*
* INPUT PARAMETERS:
*
*               None
*
* OUTPUT PARAMETERS:
*
*               None
*
* FUNCTION RETURN:
*
*               None
*
* GLOBAL VARIABLES USED:
*
*       extern int      default_color_indices[NUM_GS_COLORS]; this file
*       extern float    default_color_rgbs[NUM_GS_COLORS][3]; this file
*       extern int      color_indices[NUM_GS_COLORS]; this file
*       extern float    color_rgbs[NUM_GS_COLORS][3]; this file
```

```
*
* FILES USED:
*
*               None
*
* NOTES:
*
* NON-STANDARD CODE :
*
* CALLED BY :
*
* FUNCTIONS CALLED :
*
*               None
*
--*/

/*------------------------- set_default_colors -------------------------*/

static void set_default_colors( void )
{
        memcpy( color_indices, default_color_indices, sizeof color_indices );
        memcpy( color_rgbs, default_color_rgbs, sizeof color_rgbs );
}

/*---------------------- END OF set_default_colors -----------------------*/









/*++ static Panel* main_panel( char* title, int win_x, int win_y )
*
* PURPOSE:
*
*               Creates and displays the main panel.
*               The panel is titled if title is not NULL, and is positioned at the
*               screen coordinates (win_x, win_y).
*
* AUTHORS:
*
*               Todd Plessel
*               NASA Ames Research Center
*               Sterling Software
*
* REVISION HISTORY:
*
*               6/89
*               12/90           rewrote to use groups
*
* INPUT PARAMETERS:
*
*               char*   title                   title for panel window
*               int     win_x                   initial x position of window
*               int     win_y                   initial y position of window
*
* OUTPUT PARAMETERS:
*
*               None
*
* FUNCTION RETURN:
*
*               Panel*  p                       a pointer to the panel
*
* GLOBAL VARIABLES USED:
*
*       extern Main_Acts        main_acts;      declared in this file
*
* FILES USED:
*
*               None
*
* NOTES:
*
* NON-STANDARD CODE :
*
* CALLED BY :
*
* FUNCTIONS CALLED :
*
*               extern void     exit_module()           file init.c
*               void            file_io_func()          defined in this file
*               void            panels_func()           defined in this file
*               void            attributes_func()       defined in this file
*               void            type_func()             defined in this file
*               void            render_func()           defined in this file
*               void            options_func()          defined in this file
*               void            update_objects()        defined in this file
*               void            new_object()            defined in this file
*               void            copy_object()           defined in this file
*               void            dump_state()            defined in this file
*               void            reset_state()           defined in this file
*               void            loop_buttons_func()     defined in this file
*               void            slider_buttons_func()   defined in this file
*               void            surface_buttons_func()  defined in this file
*               void            ijk_ranges_func()       defined in this file
*               void            direction_func()        defined in this file
*               void            boundary_surfaces_func()defined in this file
*               void            select_object()         defined in this file
*               void            toggle_draw_func()      defined in this file
*               void            zone_func()             defined in this file
*
--*/

/*------------------------------ main_panel ------------------------------*/

static Panel* main_panel( char* title, int win_x, int win_y )
{
        extern void SeqerPanel(Panel*,float,float);     /* seger panel */
        Panel*          p;                      /* tmp panel           */
        Actuator*       e;                      /* tmp actuator        */
        Actuator*       f;                      /* tmp frame actuator  */
        float           x = X_ORIGIN;           /* act x position      */
        float           y = Y_ORIGIN;           /* act y position      */


        /*--------------------------- Main Panel ---------------------------*/

        if ( ( p = make_panel(title, win_x, win_y, 1, 0 ) ) == NULL )
        {
                Error("Cannot create a panel! Exiting...\n");
```



```
                exit_module();
        }

        /*--------------------------- Module menu ---------------------------*/

        panels[MAIN_PANEL] = p;

        if ( !module_menu( MODULE_NAME, panels, NUM_PANELS, file_io_func ) )
        {
                Error("Cannot create a module menu! Exiting...\n");
                exit_module();
        }

        x += MODULE_MENU_WIDTH;

        /*--------------------------- Panels Menu ---------------------------*/

        main_acts.panels_menu_group =
                make_menu_group(p,
                                x,
                                y,
                                PANELS_MENU_GROUPS,
                                panels_menu_items_per_group,
                                panels_menu_labels,
                                panels_menu_selections,
                                panels_menu_markable,
                                PANELS_MENU_JUSTIFY,
                                panels_func);

        if ( main_acts.panels_menu_group == NULL )
        {
                Error("Could not make a menu! Exiting...\n");
                exit_module();
        }

        x += menu_group_width( main_acts.panels_menu_group );

        /*--------------------------- Options Menu ---------------------------*/

        main_acts.options_menu_group =
                make_menu_group(p,
                                x,
                                y,
                                OPTIONS_MENU_GROUPS,
                                options_menu_items_per_group,
                                options_menu_labels,
                                options_menu_selections,
                                options_menu_markable,
                                OPTIONS_MENU_JUSTIFY,
                                options_func);

        if ( main_acts.options_menu_group == NULL )
        {
                Error("Could not make a menu! Exiting...\n");
                exit_module();
        }

        x += menu_group_width( main_acts.options_menu_group );

        /*--------------------------- Type Menu ---------------------------*/

        main_acts.type_menu_group =
                make_menu_group(p,
                                x,
                                y,
                                TYPE_MENU_GROUPS,
                                type_menu_items_per_group,
                                type_menu_labels,
                                type_menu_selections,
                                type_menu_markable,
                                TYPE_MENU_JUSTIFY,
                                type_func);

        if ( main_acts.type_menu_group == NULL )
        {
                Error("Could not make a menu! Exiting...\n");
                exit_module();
        }

        x += menu_group_width( main_acts.type_menu_group );

        /*--------------------------- Render Menu ---------------------------*/

        main_acts.render_menu_group =
                make_menu_group(p,
                                x,
                                y,
                                RENDER_MENU_GROUPS,
                                render_menu_items_per_group,
                                render_menu_labels,
                                render_menu_selections,
                                render_menu_markable,
                                RENDER_MENU_JUSTIFY,
                                render_func);

        if ( main_acts.render_menu_group == NULL )
        {
                Error("Could not make a menu! Exiting...\n");
                exit_module();
        }

        x += menu_group_width( main_acts.render_menu_group );

        /*--------------------------- Attributes Menu ---------------------------*/

        main_acts.attributes_menu_group =
                make_menu_group(p,
                                x,
                                y,
                                ATTRIBUTES_MENU_GROUPS,
                                attributes_menu_items_per_group,
                                attributes_menu_labels,
                                attributes_menu_selections,
                                attributes_menu_markable,
                                ATTRIBUTES_MENU_JUSTIFY,
                                attributes_func);

        if ( main_acts.attributes_menu_group == NULL )
        {
                Error("Could not make a menu! Exiting...\n");
                exit_module();
        }

        /*--------------------------- Object Frame ---------------------------*/

        x = X_ORIGIN;
        y = Y_ORIGIN - 6.15;
```

```
        main_acts.object_frame =
        f = add_frame(p, x, y);
        if ( f == NULL )
        {
                Error("Could not make an actuator! Exiting...\n");
                exit_module();
        }

        /*------------------------- Don't Draw Button ----------------------*/

        x = X_ORIGIN;
        y = Y_ORIGIN - SPACE;

        main_acts.draw_object_button =
        a = make_actuator( pnl_toggle_button, x, y, DRAW_BUTTON_LABEL );
        if ( a == NULL )
        {
                Error("Could not make actuators! Exiting...\n");
                exit_module();
        }
        a -> val = 1.0;
        a -> upfunc = (PNL_AFUNC) toggle_draw_func;
        pnl_addsubact(a, f);

        y -= Y_BUTTON_INC;

        /*------------------ Update All Objects Button -------------------*/

        main_acts.update_objects_button =
        a = make_actuator(pnl_toggle_button,x,y,UPDATE_OBJECTS_BUTTON_LABEL);
        if ( a == NULL )
        {
                Error("Could not make an actuator! Exiting...\n");
                exit_module();
        }
        a -> upfunc = (PNL_AFUNC) update_objects;
/* HACK: unimplemented in this release so hide it */ a -> visible = 0;
        pnl_addsubact(a, f);

        y -= Y_BUTTON_INC;

        /*--------------------- New Object Button ----------------------*/

        main_acts.new_object_button =
        a = make_actuator( pnl_wide_button, x, y, NEW_OBJECT_BUTTON_LABEL );
        if ( a == NULL )
        {
                Error("Could not make an actuator! Exiting...\n");
                exit_module();
        }
        a -> upfunc = (PNL_AFUNC) new_object;
        pnl_addsubact(a, f);

        y -= Y_BUTTON_INC;

        /*--------------------- Copy Object Button ---------------------*/

        main_acts.copy_object_button =
        a = make_actuator( pnl_wide_button, x, y, COPY_OBJECT_BUTTON_LABEL );
        if ( a == NULL )
        {
                Error("Could not make an actuator! Exiting...\n");
                exit_module();
        }
        a -> upfunc = (PNL_AFUNC) copy_object;
        pnl_addsubact(a, f);

        /*------------------------ Object Typeout ------------------------*/

        y -= 0.5 * OBJECT_TYPEOUT_LINES + 0.25;

        main_acts.object_typeout =
        a = make_actuator( pnl_typeout, x, y, OBJECT_TYPEOUT_LABEL );
        if ( a == NULL )
        {
                Error("Could not make an actuator! Exiting...\n");
                exit_module();
        }
        a -> labeltype = PNL_LABEL_TOP_LEFT;
        a -> upfunc = (PNL_AFUNC) select_object;
        PNL_ACCESS(Typeout, a, size) = OBJECT_BUF_SIZE;
        PNL_ACCESS(Typeout, a, lin) = OBJECT_TYPEOUT_LINES;
        PNL_ACCESS(Typeout, a, col) = OBJECT_TYPEOUT_COLS;
        PNL_ACCESS(Typeout, a, delimstr) = "\n";
        PNL_ACCESS(Typeout, a, mode) |= PNL_TOM_NOCURSOR;
        pnl_addsubact(a, f);
        if ( PNL_ACCESS(Typeout, a, buf) == NULL ) return NULL;

        /*---------------------- Data Info Typeout ---------------------*/

        x += 6.5;
        y = Y_ORIGIN - 6.15;

        main_acts.data_info_typeout =
        a = make_actuator( pnl_typeout, x, y, DATA_INFO_LABEL );
        if ( a == NULL )
        {
                Error("Could not make an actuator! Exiting...\n");
                exit_module();
        }
        a -> labeltype = PNL_LABEL_TOP_LEFT;
        PNL_ACCESS(Typeout, a, size) = DATA_INFO_BUF_SIZE;
        PNL_ACCESS(Typeout, a, lin) = DATA_INFO_LINES;
        PNL_ACCESS(Typeout, a, col) = DATA_INFO_COLS;
        PNL_ACCESS(Typeout, a, delimstr) = "\n";
        PNL_ACCESS(Typeout, a, mode) |= PNL_TOM_NOCURSOR;
        PNL_ACCESS(Typeout, a, mode) |= PNL_TOM_NOREGION;
        pnl_addact(a, p);
        if ( PNL_ACCESS(Typeout, a, buf) == NULL ) return NULL;

        /*------------------------- Looping Label ------------------------*/

        x = X_ORIGIN;
        y -= Y_BUTTON_INC;

        main_acts.looping_label =
        a = make_actuator( pnl_label, x, y, LOOPING_LABEL );
        if ( a == NULL )
        {
                Error("Could not make an actuator! Exiting...\n");
                exit_module();
        }
        pnl_addact(a, p);

        /*------------------------- Loop Buttons -------------------------*/

        y -= 0.5 * (1 + LOOP_BUTTONS_ROWS) + SPACE;
```



```
        main_acts.loop_buttons_group =
                make_button_group(      p,
                                        x,
                                        y,
                                        LOOP_BUTTONS_FRAMED,
                                        LOOP_BUTTONS_RADIO_GROUPING,
                                        LOOP_BUTTONS_ROWS - 1 /* HACK */,
                                        loop_buttons_per_row,
                                        loop_buttons_types,
                                        loop_buttons_labels,
                                        loop_buttons_row_labels,
                                        NULL,
                                        loop_buttons_selections,
                                        loop_buttons_func       );

        if (main_acts.loop_buttons_group == NULL)
        {
                Error("Could not make actuators! Exiting...\n");
                exit_module();
        }


        /*------------------------- Slider Buttons -------------------------*/

        y -= 0.5 * (1 + SLIDER_BUTTONS_ROWS) + SPACE;

        main_acts.slider_buttons_group =
                make_button_group(      p,
                                        x,
                                        y,
                                        SLIDER_BUTTONS_FRAMED,
                                        SLIDER_BUTTONS_RADIO_GROUPING,
                                        SLIDER_BUTTONS_ROWS,
                                        slider_buttons_per_row,
                                        slider_buttons_types,
                                        slider_buttons_labels,
                                        slider_buttons_row_labels,
                                        NULL,
                                        slider_buttons_selections,
                                        slider_buttons_func     );

        if (main_acts.slider_buttons_group == NULL)
        {
                Error("Could not make actuators! Exiting...\n");
                exit_module();
        }



        /*------------------------- Surface Buttons ------------------------*/

        y -= 0.5 * (1 + SURFACE_BUTTONS_ROWS) + SPACE;

        main_acts.surface_buttons_group =
                make_button_group(      p,
                                        x,
                                        y,
                                        SURFACE_BUTTONS_FRAMED,
                                        SURFACE_BUTTONS_RADIO_GROUPING,
                                        SURFACE_BUTTONS_ROWS,
                                        surface_buttons_per_row,
                                        surface_buttons_types,
                                        surface_buttons_labels,
                                        surface_buttons_row_labels,
                                        NULL,
                                        surface_buttons_selections,
                                        surface_buttons_func    );

        if (main_acts.surface_buttons_group == NULL)
        {
                Error("Could not make actuators! Exiting...\n");
                exit_module();
        }


        /*------------------------ Zone Typein Group ---------------------*/

        x += 9.5;

        main_acts.zone_typein_group =
                make_typein_group(      p,
                                        NULL,
                                        x,
                                        y,
                                        INT_TYPE,
                                        ZONE_TYPEIN_FORMAT,
                                        ZONE_TYPEIN_WIDTH,
                                        zone_typein_values,
                                        ZONE_TYPEIN_LIMITED,
                                        ZONE_TYPEIN_LABEL,
                                        ZONE_TYPEIN_LABEL_TYPE,
                                        zone_func               );

        if (main_acts.zone_typein_group == NULL)
        {
                Error("Could not make actuators! Exiting...\n");
                exit_module();
        }


        /*------------------------ Slider Group --------------------------*/

        x = X_ORIGIN;
        y -= 5.25;

        main_acts.slider_group =
                make_slider_group(      p,
                                        x,
                                        y,
                                        SLIDER_GROUP_FRAMED,
                                        SLIDER_GROUP_SELECTION_BUTTONS,
                                        SLIDER_GROUP_TYPE,
                                        slider_group_values,
                                        slider_group_label_letters,
                                        slider_group_direction_label,
                                        ijk_ranges_func,
                                        direction_func,
                                        boundary_surfaces_func  );

        if (main_acts.slider_group == NULL)
        {
                Error("Could not make actuators! Exiting...\n");
                exit_module();
        }

        /* hide the selection buttons until we enter multi surface mode */
```

```
        ACCESS2(main_acts.slider_group, hide_select_buttons);

        /* highlight the middle slider */

        ACCESS3(main_acts.slider_group, highlight_slider, MID);

        /************ create sager panel *********************************/

        SagerPanel(p,X_ORIGIN,Y_ORIGIN);

        return p;
}


/*------------------------- END OF main_panel ------------------------------*/




/*++ static Panel* minmax_panel( char* title, int win_x, int win_y )
 *
 * PURPOSE:
 *
 *              Creates and displays the scalar minmax panel for sager.
 *              The panel is titled if title is not NULL, and is positioned at the
 *              screen coordinates (win_x, win_y).
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              6/89
 *
 * INPUT PARAMETERS:
 *
 *              char*   title                   title for panel window
 *              int     win_x                   initial x position of window
 *              int     win_y                   initial y position of window
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              Panel*  p                       a pointer to the panel
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Minmax_Acts      minmax_acts;    declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      extern void     close_parent_panel()            libpanu
 *      extern Panel*   make_panel()                    libpanu
 *      extern Actuator*        make_actuator()         libpanu
 *
 *      void            set_minmax_func()               defined in this file
 *      void            reset_minmax()                  defined in this file
 *      void            adjust_minmax_func()            defined in this file
 *      void            set_minmax_modes()              defined in this file
 *
 --*/

/*------------------------------ minmax_panel ------------------------------*/

static Panel* minmax_panel( char* title, int win_x, int win_y )
{
        Panel*          p;                      /* tmp panel            */
        Actuator*       a;                      /* tmp actuator         */
        Actuator*       ta;                     /* tmp actuator         */
        Actuator*       f;                      /* frame actuator       */
        float           x;                      /* act x position       */
        float           y;                      /* act y position       */
        float           h;                      /* act height           */
        float           w;                      /* act height           */
        int             i;                      /* looping index        */
        float           y_inc;                  /* y position increment */
        float           y_save;                 /* save y position      */
        static char     clip_ms_label[2][16];   /* clip multislider     */
        static char     norm_ms_label[2][16];   /* norm multislider     */
        static char     legend_labels[NUM_LEGEND_VALUES][16];
                                                /* legend values        */



        /*------------------------ Minmax Panel ------------------------*/

        p = pnl_mkpanel();
        p -> label = title;
        p -> x = (long) win_x;
        p -> y = (long) win_y;
        p -> ppu = 36.0;
        p -> visible = 0;

        x = X_ORIGIN;
        y = Y_ORIGIN;

        /*------------------ window close button ----------------*/

        a = pnl_mkact(pnl_wide_button);
        a -> label = "Close";
        a -> x = x;
        a -> y = y;
        a -> upfunc = (PNL_AFUNC) close_parent_panel;
        pnl_addact(a, p);
```



```
        /*--------- Minmax Frame -------------*/

        y -= SMM_FRAME_HEIGHT;

        f = pnl_mkact(pnl_frame);
        f -> x = x;
        f -> y = y;
        PNL_ACCESS3(frame, f, mode) |= PNL_FM_FREE;
        pnl_addact(f, p);

        minmax_acts.minmax_frame = f;


        /*-------- Minmax Modes --------*/

        x = X_FRAME_ACT;
        y = Y_FRAME_ACT;

        a = pnl_mkact(pnl_toggle_button);
        a -> label = AUTO_MINMAX_UPDATE_LABEL;
        a -> x = x;
        a -> y = y;
        a -> val = 1.0;
        a -> u = AUTO_MINMAX_UPDATE_ID;
        a -> upfunc = (PNL_AFUNC) set_minmax_modes;
        pnl_addsubact(a, f);

        minmax_acts.auto_minmax_update_button = a;

        y -= 0.75;

        a = pnl_mkact(pnl_toggle_button);
        a -> label = UPDATE_MMSLIDERS_LABEL;
        a -> x = x;
        a -> y = y;
        a -> u = UPDATE_MMSLIDERS_ID;
        a -> val = 1.0;
        a -> upfunc = (PNL_AFUNC) set_minmax_modes;
        pnl_addsubact(a, f);

        minmax_acts.update_minmax_sliders_button = a;

        x += 6.0;
        y += 0.75;

        a = pnl_mkact(pnl_radio_button);
        a -> label = MULTI_ZONE_MINMAX_LABEL;
        a -> x = x;
        a -> y = y;
        a -> u = MULTI_ZONE_MINMAX_ID;
        a -> val = 1.0;
        a -> upfunc = (PNL_AFUNC) set_minmax_modes;
        pnl_addsubact(a, f);

        minmax_acts.mode_buttons[MULTI_ZONE_MINMAX] = a;

        y -= 0.75;

        a = pnl_mkact(pnl_radio_button);
        a -> label = SINGLE_ZONE_MINMAX_LABEL;
        a -> x = x;
        a -> y = y;
        a -> u = SINGLE_ZONE_MINMAX_ID;
        a -> upfunc = (PNL_AFUNC) set_minmax_modes;
        pnl_addsubact(a, f);

        minmax_acts.mode_buttons[SINGLE_ZONE_MINMAX] = a;

        y -= 0.75;

        a = pnl_mkact(pnl_radio_button);
        a -> label = SUBSET_MINMAX_LABEL;
        a -> x = x;
        a -> y = y;
        a -> u = SUBSET_MINMAX_ID;
        a -> upfunc = (PNL_AFUNC) set_minmax_modes;
        pnl_addsubact(a, f);

        minmax_acts.mode_buttons[SUBSET_MINMAX] = a;

        y -= 0.75;

        a = pnl_mkact(pnl_radio_button);
        a -> label = SURFACE_MINMAX_LABEL;
        a -> x = x;
        a -> y = y;
        a -> u = SURFACE_MINMAX_ID;
        a -> upfunc = (PNL_AFUNC) set_minmax_modes;
        pnl_addsubact(a, f);

        minmax_acts.mode_buttons[SURFACE_MINMAX] = a;

        y -= 0.75;

        a = pnl_mkact(pnl_radio_button);
        a -> label = SURFACE_SUBSET_MINMAX_LABEL;
        a -> x = x;
        a -> y = y;
        a -> u = SURFACE_SUBSET_MINMAX_ID;
        a -> upfunc = (PNL_AFUNC) set_minmax_modes;
        pnl_addsubact(a, f);

        minmax_acts.mode_buttons[SURFACE_SUBSET_MINMAX] = a;

        pnl_endgroup(p);


        x = X_FRAME_ACT;

        /*-------- Invert Clip Test Button ----------*/

        a = pnl_mkact(pnl_toggle_button);
        a -> label = INVERT_CLIP_TEST_LABEL;
        a -> x = x;
        a -> y = y;
        a -> upfunc = (PNL_AFUNC) invert_clip_test;
        pnl_addsubact(a, f);

        minmax_acts.invert_clip_test_button  = a;

        y -= 1.0;

        /*-------- Clip, Norm & Legend Labels --------*/

        a = pnl_mkact(pnl_label);
        a -> label = CLIP_LABEL;
        a -> x = x;
```

```
a -> y = y;
pnl_addsubact(a, f);

minmax_acts.clip_label = a;

x += PALETTE_X_INC;

a = pnl_mkact(pnl_label);
a -> label = NORM_LABEL;
a -> x = x;
a -> y = y;
pnl_addsubact(a, f);

minmax_acts.norm_label = a;

x += PALETTE_X_INC;

a = pnl_mkact(pnl_label);
a -> label = LEGEND_LABEL;
a -> x = x;
a -> y = y;
pnl_addsubact(a, f);

minmax_acts.legend_label = a;

/*----- Clip, Norm & Legend Top/Max Typeins -----*/

x = X_FRAME_ACT;
y -= 1.0;

a = pnl_mkact(pnl_typein);
a -> labeltype = PNL_LABEL_LEFT;
a -> label = CLIP_TOP_LABEL;
a -> x = x;
a -> y = y;
a -> u = CLIP_TOP_ID;
a -> upfunc = (PNL_AFUNC) set_minmax_func;
PNL_ACCESS(Typein, a, len) = FLOAT_STRING_WIDTH;
PNL_ACCESS(Typein, a, str) = CLIP_TOP_NUMBER_STR;
pnl_addsubact(a, f);

minmax_acts.clip_top_typein = a;

x += PALETTE_X_INC;

a = pnl_mkact(pnl_typein);
a -> labeltype = PNL_LABEL_LEFT;
a -> label = NORM_TOP_LABEL;
a -> x = x;
a -> y = y;
a -> u = NORM_TOP_ID;
a -> upfunc = (PNL_AFUNC) set_minmax_func;
PNL_ACCESS(Typein, a, len) = FLOAT_STRING_WIDTH;
PNL_ACCESS(Typein, a, str) = NORM_TOP_NUMBER_STR;
pnl_addsubact(a, f);

minmax_acts.norm_top_typein = a;

x += PALETTE_X_INC;

a = pnl_mkact(pnl_typein);
a -> labeltype = PNL_LABEL_LEFT;
a -> label = LEGEND_MAX_LABEL;
a -> x = x;
a -> y = y;
a -> u = LEGEND_MAX_ID;
a -> upfunc = (PNL_AFUNC) set_minmax_func;
PNL_ACCESS(Typein, a, len) = FLOAT_STRING_WIDTH;
PNL_ACCESS(Typein, a, str) = LEGEND_MAX_NUMBER_STR;
pnl_addsubact(a, f);

minmax_acts.legend_max_typein = a;

/*------ Clip & Norm Multisliders ------*/

x = X_FRAME_ACT;
y -= MULTISLIDER_HEIGHT + 0.5;
w = MULTISLIDER_WIDTH;
h = MULTISLIDER_HEIGHT;

/* store the labels for the Clip multislider */

for (i = 0; i < 2; ++i)
{
        sprintf(clip_ms_label[i], INT_STRING_FORMAT, 0);
}

a = pnl_mkact(pnl_multislider);
a -> x = x;
a -> y = y;
a -> w = w;
a -> h = h;
a -> u = CLIP_MSLIDER_ID;
a -> activefunc = (PNL_AFUNC) adjust_minmax_func;
a -> minval = 0.0;
a -> maxval = 1.0;
PNL_ACCESS(Multislider, a, n) = 2;
PNL_ACCESS(Multislider, a, mode) = PNL_MSM_CONSTRAINED;
pnl_addsubact(a, f);

ta = a -> al;
ta -> extval = 1.0;
ta -> label = clip_ms_label[0];
ta = ta -> next;
ta -> extval = 0.0;
ta -> label = clip_ms_label[1];
pnl_fixact(ta);

minmax_acts.clip_multislider = a;

x += PALETTE_X_INC;

/* store the labels for the Norm multislider */

for (i = 0; i < 2; ++i)
{
        sprintf(norm_ms_label[i], INT_STRING_FORMAT, 0);
}

a = pnl_mkact(pnl_multislider);
a -> x = x;
a -> y = y;
a -> w = w;
a -> h = h;
a -> minval = 0.0;
a -> maxval = 1.0;
a -> u = NORM_MSLIDER_ID;
a -> activefunc = (PNL_AFUNC) adjust_minmax_func;
```



```
PNL_ACCESS(Multislider, a, n) = 2;
PNL_ACCESS(Multislider, a, mode) = PNL_MSM_CONSTRAINED;
pnl_addsubact(a, f);

ta = a -> al;
ta -> extval = 1.0;
ta -> label = norm_ms_label[0];
ta = ta -> next;
ta -> extval = 0.0;
ta -> label = norm_ms_label[1];
pnl_fixact(ta);

minmax_acts.norm_multislider = a;


/*------ Clip, Norm & Legend Palettes ------*/

x = X_FRAME_ACT - PALETTE_WIDTH;

/* Clip LOW */

a = pnl_mkact(pnl_palette);
a -> x = x;
a -> y = y;
a -> w = PALETTE_WIDTH;
a -> h = 0.0;
a -> minval = MIN_MAP;
a -> maxval = MIN_MAP;
pnl_addsubact(a, f);

minmax_acts.palettes[CLIP][LOW] = a;

/* Clip MED */

a = pnl_mkact(pnl_palette);
a -> x = x;
a -> y = y;
a -> w = PALETTE_WIDTH;
a -> h = PALETTE_HEIGHT;
a -> minval = MIN_MAP;
a -> maxval = MAX_MAP;
pnl_addsubact(a, f);

minmax_acts.palettes[CLIP][MED] = a;

/* Clip HI */

a = pnl_mkact(pnl_palette);
a -> x = x;
a -> y = y + PALETTE_HEIGHT;
a -> w = PALETTE_WIDTH;
a -> h = 0.0;
a -> minval = MAX_MAP;
a -> maxval = MAX_MAP;
pnl_addsubact(a, f);

minmax_acts.palettes[CLIP][HI] = a;

x += PALETTE_X_INC;

/* Norm LOW does not exist */

minmax_acts.palettes[NORM][LOW] = NULL;

/* Norm MED */

a = pnl_mkact(pnl_palette);
a -> x = x;
a -> y = y;
a -> w = PALETTE_WIDTH;
a -> h = PALETTE_HEIGHT;
a -> minval = MIN_MAP;
a -> maxval = MAX_MAP;
pnl_addsubact(a, f);

minmax_acts.palettes[NORM][MED] = a;

/* Norm HI does not exist */

minmax_acts.palettes[NORM][HI] = NULL;

x += PALETTE_X_INC;

/* Legend LOW */

a = pnl_mkact(pnl_palette);
a -> x = x;
a -> y = y;
a -> w = PALETTE_WIDTH;
a -> h = 0.0;
a -> minval = MIN_MAP;
a -> maxval = MIN_MAP;
pnl_addsubact(a, f);

minmax_acts.palettes[LEGEND][LOW] = a;

/* Legend MED */

a = pnl_mkact(pnl_palette);
a -> x = x;
a -> y = y;
a -> w = PALETTE_WIDTH;
a -> h = PALETTE_HEIGHT;
a -> minval = MIN_MAP;
a -> maxval = MAX_MAP;
pnl_addsubact(a, f);

minmax_acts.palettes[LEGEND][MED] = a;

/* Legend HI */

a = pnl_mkact(pnl_palette);
a -> x = x;
a -> y = y + PALETTE_HEIGHT;
a -> w = PALETTE_WIDTH;
a -> h = 0.0;
a -> minval = MAX_MAP;
a -> maxval = MAX_MAP;
pnl_addsubact(a, f);

minmax_acts.palettes[LEGEND][HI] = a;


/*----- Legend Number Labels -----*/

x += PALETTE_WIDTH + MULTISLIDER_WIDTH + 0.1;
y_inc = h / (float) (NUM_LEGEND_VALUES - 1);
y_save = y;
```

```
        y -= 0.15;

        for (i = 0; i < NUM_LEGEND_VALUES; ++i)
        {
                a = pnl_mkact(pnl_label);
                sprintf(legend_labels[i], FLOAT_STRING_FORMAT, 0.0);
                a -> label = legend_labels[i];
                a -> x = x;
                a -> y = y;
                pnl_addsubact(a, f);

                minmax_acts.legend_labels[i] = a;

                y += y_inc;
        }


        /*----- Clip, Norm & Legend Bot/Min Typeins -----*/

        x = X_FRAME_ACT;
        y = y_save - 1.0;

        a = pnl_mkact(pnl_typein);
        a -> labeltype = PNL_LABEL_LEFT;
        a -> label = CLIP_BOT_LABEL;
        a -> x = x;
        a -> y = y;
        a -> u = CLIP_BOT_ID;
        a -> upfunc = (PNL_AFUNC) set_minmax_func;
        PNL_ACCESS(Typein, a, len) = FLOAT_STRING_WIDTH;
        PNL_ACCESS(Typein, a, str) = CLIP_BOT_NUMBER_STR;
        pnl_addsubact(a, f);

        minmax_acts.clip_bot_typein = a;

        x += PALETTE_X_INC;

        a = pnl_mkact(pnl_typein);
        a -> labeltype = PNL_LABEL_LEFT;
        a -> label = NORM_BOT_LABEL;
        a -> x = x;
        a -> y = y;
        a -> u = NORM_BOT_ID;
        a -> upfunc = (PNL_AFUNC) set_minmax_func;
        PNL_ACCESS(Typein, a, len) = FLOAT_STRING_WIDTH;
        PNL_ACCESS(Typein, a, str) = NORM_BOT_NUMBER_STR;
        pnl_addsubact(a, f);

        minmax_acts.norm_bot_typein = a;

        x += PALETTE_X_INC;

        a = pnl_mkact(pnl_typein);
        a -> labeltype = PNL_LABEL_LEFT;
        a -> label = LEGEND_MIN_LABEL;
        a -> x = x;
        a -> y = y;
        a -> u = LEGEND_MIN_ID;
        a -> upfunc = (PNL_AFUNC) set_minmax_func;
        PNL_ACCESS(Typein, a, len) = FLOAT_STRING_WIDTH;
        PNL_ACCESS(Typein, a, str) = LEGEND_MIN_NUMBER_STR;
        pnl_addsubact(a, f);

        minmax_acts.legend_min_typein = a;

        /*-------- Clip, Norm & Legend Reset Buttons -------*/

        x = X_FRAME_ACT;
        y -= 1.0;

        a = pnl_mkact(pnl_wide_button);
        a -> label = RESET_CLIP_LABEL;
        a -> x = x;
        a -> y = y;
        a -> u = RESET_CLIP_ID;
        a -> upfunc = (PNL_AFUNC) reset_minmax;
        pnl_addsubact(a, f);

        minmax_acts.reset_clip_button = a;

        x += PALETTE_X_INC;

        a = pnl_mkact(pnl_wide_button);
        a -> label = RESET_NORM_LABEL;
        a -> x = x;
        a -> y = y;
        a -> u = RESET_NORM_ID;
        a -> upfunc = (PNL_AFUNC) reset_minmax;
        pnl_addsubact(a, f);

        minmax_acts.reset_norm_button = a;

        x += PALETTE_X_INC;

        a = pnl_mkact(pnl_wide_button);
        a -> label = RESET_LEGEND_LABEL;
        a -> x = x;
        a -> y = y;
        a -> u = RESET_LEGEND_ID;
        a -> upfunc = (PNL_AFUNC) reset_minmax;
        pnl_addsubact(a, f);

        minmax_acts.reset_legend_button = a;


        return p;
}

/*--------------------------- END OF minmax_panel -------------------------*/




/*++ static Panel* contour_panel( char* title, int win_x, int win_y )
 *
 * PURPOSE :
 *
 *              The panel is titled if title is not NULL, and is positioned at
 *              the screen coordinates (win_x, win_y).
 *
```



```
 * AUTHORS :
 *
 *              Paul Kelaita
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY :
 *
 *              12/90
 *
 * INPUT PARAMETERS :
 *
 *              char*   title                   title for panel window
 *              int     win_x                   initial x position of window
 *              int     win_y                   initial y position of window
 *
 * OUTPUT PARAMETERS :
 *
 *              None
 *
 * FUNCTION RETURN :
 *
 *              Panel*  p                       a pointer to the panel
 *
 * GLOBAL VARIABLES USED :
 *
 *
 * FILES USED :
 *
 *              None
 *
 * NOTES :
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      extern void     close_parent_panel()            libpanu
 *
 *
 --*/

/*------------------------------- contour_panel -------------------------------*/

static Panel* contour_panel( char* title, int win_x, int win_y )
{
        Panel*          p;                      /* tmp panel            */
        Actuator*       a;                      /* tmp actuator         */
        float           x;                      /* act x position       */
        float           y;                      /* act y position       */
        int             i;                      /* looping index        */
        float           y_inc;                  /* y position increment */
        float           y_save;                 /* save y position      */
        static char     contour_labels[NUM_LEGEND_VALUES][16];
                                                /* legend values        */



        /*---------------------- SAGER Contour Panel ----------------------*/

        p = pnl_mkpanel();

        p -> label = title;
        p -> x = (long) win_x;
        p -> y = (long) win_y;
        p -> ppu = 36.0;
        p -> visible = 0;

        x = X_ORIGIN + 1.5;
        y = Y_ORIGIN + 5.0;

        /*------------------ window close button ----------------*/

        a = pnl_mkact(pnl_wide_button);
        a -> label = "Close";
        a -> x = x;
        a -> y = y;
        a -> upfunc = (PNL_AFUNC) close_parent_panel;
        pnl_addact(a, p);

        /*------------------ contour legend ----------------*/

        x = X_ORIGIN + 3.0;
        y = Y_ORIGIN - 6.0;

        a = pnl_mkact(pnl_palette);
        a -> x = x;
        a -> y = y;
        a -> w = PALETTE_WIDTH;
        a -> h = PALETTE_HEIGHT;
        a -> minval = MIN_MAP;
        a -> maxval = MAX_MAP;
        pnl_addact(a, p);

        contour_acts.palette = a;

        /*---------------- Legend Number labels ----------------*/

        x += PALETTE_WIDTH + MULTISLIDER_WIDTH + 0.1;
        y_inc = MULTISLIDER_HEIGHT / (float) (NUM_LEGEND_VALUES - 1);
        y_save = y;
        y -= 0.15;

        for (i = 0; i < NUM_LEGEND_VALUES; ++i)
        {
                a = pnl_mkact(pnl_label);
                sprintf(contour_labels[i], FLOAT_STRING_FORMAT, 0.0);
                a -> label = contour_labels[i];
                a -> x = x;
                a -> y = y;
                pnl_addact(a, p);

                contour_acts.contour_labels[i] = a;

                y += y_inc;
        }

        /*---------------- Contour Minimum/Maximum Typeins ----------------*/

        x -= 3.0;
        y = y_save - 1.0;

        a = pnl_mkact(pnl_typein);
        a -> labeltype = PNL_LABEL_BOTTOM;
        a -> label = "Contour Min";
        a -> x = x;
```

```
        a -> y = y;
        a -> u = CONTOUR_MIN_ID;
        a -> downfunc = (PNL_AFUNC) clear_typein;
        a -> upfunc = (PNL_AFUNC) set_contour_legend;
        PNL_ACCESS(Typein, a, len) = FLOAT_STRING_WIDTH;
        PNL_ACCESS(Typein, a, str) = LEGEND_MIN_NUMBER_STR;
        pnl_addact(a, p);

        contour_acts.contour_min_typein = a;

        y += MULTISLIDER_HEIGHT + 1.5;

        a = pnl_mkact(pnl_typein);
        a -> labeltype = PNL_LABEL_TOP;
        a -> label = "Contour Max";
        a -> x = x;
        a -> y = y;
        a -> u = CONTOUR_MAX_ID;
        a -> downfunc = (PNL_AFUNC) clear_typein;
        a -> upfunc = (PNL_AFUNC) set_contour_legend;
        PNL_ACCESS(Typein, a, len) = FLOAT_STRING_WIDTH;
        PNL_ACCESS(Typein, a, str) = LEGEND_MAX_NUMBER_STR;
        pnl_addact(a, p);

        contour_acts.contour_max_typein = a;

        /*---------- Contours/Increment Typeins ----------*/

        x = X_ORIGIN + 3.0;
        y = Y_ORIGIN + 3.5;

        a = pnl_mkact(pnl_typein);
        a -> labeltype = PNL_LABEL_TOP;
        a -> label = "Number of Contours";
        a -> x = x + 1.0;
        a -> y = y;
        a -> u = CONTOUR_NUM_ID;
        a -> downfunc = (PNL_AFUNC) clear_typein;
        a -> upfunc = (PNL_AFUNC) set_contour_legend;
        PNL_ACCESS(Typein, a, len) = INT_STRING_WIDTH;
        PNL_ACCESS(Typein, a, str) = "40";
        pnl_addact(a, p);

        contour_acts.contour_num_typein = a;

        y -= 1.5;

        a = pnl_mkact(pnl_typein);
        a -> labeltype = PNL_LABEL_TOP;
        a -> label = "Increment of Contours";
        a -> x = x;
        a -> y = y;
        a -> u = CONTOUR_INC_ID;
        a -> upfunc = (PNL_AFUNC) set_contour_legend;
        PNL_ACCESS(Typein, a, len) = FLOAT_STRING_WIDTH;
        PNL_ACCESS(Typein, a, str) = ZERO_STRING_16;
        pnl_addact(a, p);

        contour_acts.contour_inc_typein = a;

        return p;

}
```

```
/*---------------------- END OF contour_panel ----------------------*/




/*++ static void set_contour_legend( char* str )
 *
 * PURPOSE :
 *
 *              Sets the contours legend according to min, max, numcontours,
 *              and increment
 *
 *
 * AUTHORS :
 *
 *              Paul Kelaita
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY :
 *
 *              12/90
 *              5/91            added scripting
 *
 * INPUT PARAMETERS :
 *
 *              char*   str             command/actuator
 *
 * OUTPUT PARAMETERS :
 *
 *              None
 *
 * FUNCTION RETURN :
 *
 *              None
 *
 * GLOBAL VARIABLES USED :
 *
 *              extern Minmax_Acts      minmax_acts     defined in this file
 *              extern Grid_Surface*    grid_eager      defined in this file
 *
 * FILES USED :
 *
 *              None
 *
 * NOTES :
 *
 *              Requires that math.h be included for atof()
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
```



```
 *
 *      void            update_legend()                 defined in this file
 *      int             lock_cur_object()               defined in this file
 *      int             unlock_cur_object()             defined in this file
 *
 --*/

/*---------------------------- set_contour_legend ----------------------------*/

static void set_contour_legend( char* str )
{

#define IS_REMAINDER(max, min, inc)     \
        (((max - min)/inc) == (int)((max - min)/inc) ? 0 : 1)

        Actuator*       a;

        char*   min_str;
        char*   max_str;
        char*   numc_str;
        char*   inc_str;

        float   inc;
        float   t1, t2;
        int     numc;
        char    mode[16];
        int     inc_flag, minmax_flag;



        if (is_act(str))
        {
           a = (Actuator*) str;
           if (a == contour_acts.contour_min_typein ||
               a == contour_acts.contour_max_typein   )
           {
              min_str = PNL_ACCESS(Typein,contour_acts.contour_min_typein, str);
              max_str = PNL_ACCESS(Typein,contour_acts.contour_max_typein, str);
              load_command("CONTOUR %s %f %f", "MINMAX",
                                        atof(min_str), atof(max_str));
           }
           else if (a == contour_acts.contour_num_typein)
           {
              load_command("CONTOUR %s %d", "NUMBER",
                                atoi(PNL_ACCESS(Typein, a, str)));
           }
           else if (a == contour_acts.contour_inc_typein)
           {
              load_command("CONTOUR %s %f", "INC",
                                atof(PNL_ACCESS(Typein, a, str)));
           }
           return;
        }

        parse_command(str, "%s", mode);

        /* grab all the strings first */

        min_str = PNL_ACCESS(Typein, contour_acts.contour_min_typein, str);

        max_str = PNL_ACCESS(Typein, contour_acts.contour_max_typein, str);

        numc_str = PNL_ACCESS(Typein, contour_acts.contour_num_typein, str);

        inc_str = PNL_ACCESS(Typein, contour_acts.contour_inc_typein, str);

        inc_flag = minmax_flag = 0;

        if (strcasecmp(mode, "MINMAX") == 0)
        {
           minmax_flag  = 1;
           parse_command(str, "%s %f %f", mode, &t1, &t2);
           sprintf(min_str, FLOAT_STRING_FORMAT, t1);
           sprintf(max_str, FLOAT_STRING_FORMAT, t2);
           pnl_fixact(contour_acts.contour_min_typein);
           pnl_fixact(contour_acts.contour_max_typein);
        }
        else if (strcasecmp(mode, "NUMBER") == 0)
        {
           parse_command(str, "%s %d", mode, &numc);
           sprintf(numc_str, INT_STRING_FORMAT, numc);
           pnl_fixact(contour_acts.contour_num_typein);
        }
        else if (strcasecmp(mode, "INC") == 0)
        {
           inc_flag     = 1;
           parse_command(str, "%s %f", mode, &t1);
           sprintf(inc_str, FLOAT_STRING_FORMAT, t1);
           pnl_fixact(contour_acts.contour_inc_typein);
        }

        /*
           get validity of values
        */
        if ((int)atof(numc_str) < 2)
           sprintf(numc_str, INT_STRING_FORMAT, 2);

        if (atof(inc_str) <= 0.0)
           sprintf(inc_str, FLOAT_STRING_FORMAT, 0.1);

        /*
           begin checks
        */
        if (minmax_flag)
        {
           update_legend(atof(min_str), atof(max_str),
                         contour_acts.contour_labels);
        }

        else if (inc_flag)
        {
           numc = (int)((atof(max_str) - atof(min_str)) / atof(inc_str)) + 1 +
                  IS_REMAINDER(atof(max_str), atof(min_str), atof(inc_str));
           sprintf(numc_str, INT_STRING_FORMAT, numc);
           pnl_fixact(contour_acts.contour_num_typein);
        }

        /* update the increment no matter what */

        if (!inc_flag)
        {
           inc  = (atof(max_str) - atof(min_str)) / (atof(numc_str) - 1.0);
           sprintf(inc_str, FLOAT_STRING_FORMAT, inc);
           pnl_fixact(contour_acts.contour_inc_typein);
        }

        lock_cur_object();
```

```
        grid_seqer -> num_contours      = atoi(numc_str);
        grid_seqer -> contours_inc      = atof(inc_str);
        grid_seqer -> contours_min      = atof(min_str);
        grid_seqer -> contours_max      = atof(max_str);

        delete_contours();

        unlock_cur_object();

}

/*------------------------- END OF set_contour_legend ----------------------*/









/*++ static void file_io_func( char* file_name, int mode )
 *
 * PURPOSE:
 *
 *              Call back for save/restore file io.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              6/91
 *
 * INPUT PARAMETERS:
 *
 *              char*           file_name       pathed file name to read/write
 *              int             mode            0 = read, 1 = write
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *              None
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              None
 *
 --*/

/*------------------------------ file_io_func ------------------------------*/

static void file_io_func( char* file_name, int mode )
{
        Message("file_io_func: Unimplemented feature...");
}

/*------------------------- END OF file_io_func ----------------------------*/









/*++ static void panels_func( int group, int item )
 *
 * PURPOSE:
 *
 *              Handles the panels menu functions.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              12/90
 *
 * INPUT PARAMETERS:
 *
 *              int             group           menu group number
 *              int             item            item number within group
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
```



```
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *
 --*/

/*------------------------------ panels_func -------------------------------*/

static void panels_func( int group, int item )
{
        load_command(   panels_menu_script_commands[0],
                        panels_menu_script_commands[item + 1]   );
}

/*------------------------- END OF panels_func -----------------------------*/









/*++ static void attributes_func( int group, int item )
 *
 * PURPOSE:
 *
 *              Handles the attributes menu functions.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              6/89
 *              12/90   converted to a menu group call-back
 *              4/91    added for command stuff
 *              6/91    rewrote command stuff
 *
 * INPUT PARAMETERS:
 *
 *              int             group           menu group number
 *              int             item            item number within group
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *
 --*/

/*---------------------------- attributes_func -----------------------------*/

static void attributes_func( int group, int item )
{
        int     index = group_item_to_index( group, item,
                        attributes_menu_items_per_group,
                        ATTRIBUTES_MENU_GROUPS  );

        interactive = 1;
        load_command(   attributes_menu_script_commands[0],
                        attributes_menu_script_commands[index + 1]      );
}

/*------------------------- END OF attributes_func -------------------------*/









/*++ static void set_attributes_func( char* script_command )
 *
 * PURPOSE:
 *
 *              Handles the attributes menu functions from a command
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              6/89
 *              12/90   converted to a menu group call-back
 *              4/91    added command stuff
 *              6/91    rewrote command stuff
 *
 * INPUT PARAMETERS:
 *
 *              char*   script_command
 *
 * OUTPUT PARAMETERS:
 *
 *              None
```

```
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_seger;             declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              extern int      command_index()         libpanu
 *              void            delete_normals()        defined in this file
 *              int             lock_cur_object()       defined in this file
 *              int             unlock_cur_object()     defined in this file
 *
 --*/

/*---------------------------- set_attributes_func ----------------------------*/

static void set_attributes_func( char* script_command )
{
        char    param[32];
        int     index, item, group;


        parse_command( script_command, "%s", param );

        if ( ( index = command_index( param, attributes_menu_script_commands,
                        ATTRIBUTES_MENU_ITEMS + 1 ) ) == -1 )
        {
                return;
        }

        index_to_group_item( --index, &group, &item,
                attributes_menu_items_per_group, ATTRIBUTES_MENU_GROUPS );


        lock_cur_object();

        switch ( group )
        {
            case 0:
               grid_seger -> contour_color_type = item;
               if ( ! interactive )
                   ACCESS6(main_acts.attributes_menu_group, set_selection,
                           0, item, 1, 0);
               break;
            case 1:
               grid_seger -> vector_color_type  = item;
               if ( ! interactive )
                   ACCESS6(main_acts.attributes_menu_group, set_selection,
                           1, item, 1, 0);
               break;
            case 2:
               grid_seger -> vector_tip_type    = item;
               if ( ! interactive )
                   ACCESS6(main_acts.attributes_menu_group, set_selection,
                           2, item, 1, 0);
               break;
            case 3:
               grid_seger -> clip_mode          = item;
               if ( ! interactive )
                   ACCESS6(main_acts.attributes_menu_group, set_selection,
                           3, item, 1, 0);
               break;
            case 4:
               grid_seger -> shaded             = item;
               if ( ! interactive )
                   ACCESS6(main_acts.attributes_menu_group, set_selection,
                           4, item, 1, 0);
               if ( IS_SCALAR_COLORED( grid_seger ) && IS_SHADED( grid_seger ) )
                  module_command( "Viewer", "SET_COLOR_MODE TRUE_COLOR", NULL );
               break;
            case 5:
               grid_seger -> framed             = item;
               if ( ! interactive )
                   ACCESS6(main_acts.attributes_menu_group, set_selection,
                           5, item, 1, 0);
               break;
            case 6:
               if (grid_seger -> raw_normals != item)
                  delete_normals();
               grid_seger -> raw_normals                = item;
               if ( ! interactive )
                   ACCESS6(main_acts.attributes_menu_group, set_selection,
                           6, item, 1, 0);
               break;
            case 7:
               if (grid_seger -> zone_normals != item)
                  delete_normals();
               grid_seger -> zone_normals       = item;
               if ( ! interactive )
                   ACCESS6(main_acts.attributes_menu_group, set_selection,
                           7, item, 1, 0);
               break;
            default:
               break;
        }

        interactive = 0;
        unlock_cur_object();

}

/*------------------------ END OF set_attributes_func -------------------------*/




/*++ static void type_func( int group, int item )
 *
```



```
 * PURPOSE:
 *
 *              Handles the type menu functions for commands
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/89
 *              12/90   converted to a menu group call-back
 *              4/91    Paul Kelaita -- added command stuff
 *              6/91    rewrote command stuff
 *
 * INPUT PARAMETERS:
 *
 *              int             group           menu group number
 *              int             item            item number within group
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *
 --*/

/*------------------------------- type_func -------------------------------*/

static void type_func( int group, int item )
{
        interactive = 1;
        load_command(   type_menu_script_commands[0],
                        type_menu_script_commands[item + 1]     );
}

/*---------------------------- END OF type_func ----------------------------*/




/*++ static void set_type_func( char* script_command )
 *
 * PURPOSE:
 *
 *              Handles the type menu functions from a command
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/89
 *              12/90   converted to a menu group call-back
 *              4/91    Paul Kelaita -- added command stuff
 *              6/91    rewrote command stuff
 *
 * INPUT PARAMETERS:
 *
 *              char*   script_command          command
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_seger;             declared in this file
 *      extern Main_Acts*       main_acts               declared in this file
 *      extern Minmax_Acts      minmax_acts             declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              extern int      command_index()         libpanu
 *              void            delete_normals()        defined in this file
 *              int             lock_cur_object()       defined in this file
 *              int             unlock_cur_object()     defined in this file
 *
 --*/

/*------------------------------ set_type_func ------------------------------*/

static void set_type_func( char* script_command )
{
```

```
    int             old_type;               /* previous type        */
    char            param[32];
    int             index;


    parse_command( script_command, "%s", param );

    if ( ( index = command_index( param, type_menu_script_commands,
                TYPE_MENU_ITEMS + 1 ) ) == -1 )
    {
        return;
    }

    --index;


    lock_cur_object();

    /* save the old type */

    old_type = grid_seger -> type;

    if ( old_type == index )
    {
        unlock_cur_object();
        return;
    }

    /* store new grid surface type and reset the panel if neccessary */

    grid_seger -> type = index;

    if ( ! interactive )
        ACCESS6( main_acts.type_menu_group, set_selection, 0,
                 grid_seger -> type, 1, 0 );
    interactive = 0;

    if (grid_seger -> type != GRID_TYPE)
    {
        /* if in auto update mode then reset the scalar minmax */

        if (minmax_acts.auto_minmax_update_button -> val == 1.0)
        {
            reset_minmax( "RESET_MINMAX LEGEND" );
        }
    }

    /* if neccessary deallocate any normals data */

    if (USES_NORMALS(grid_seger))
    {
        if (old_type == VECTOR_TYPE ||
            grid_seger->type == VECTOR_TYPE)
        {
            delete_normals();
        }
    }

    /* if neccessary deallocate any contours data */

    if ( grid_seger -> render_mode == CONTOUR_LINES )
    {
        delete_contours();
    }

    /*
     * if the render mode is inappropriate for the type then
     * then change the render mode to lines 1 and 2
     */

    if (((grid_seger -> render_mode == PLAIN_VECTORS ||
          grid_seger -> render_mode == NORM_VECTORS) &&
         grid_seger -> type != VECTOR_TYPE) ||
        (grid_seger -> render_mode == CONTOUR_LINES &&
         grid_seger -> type != SCALAR_TYPE))
    {
      ACCESS6(main_acts.render_menu_group, set_selection, 0, LINES_1_2, 1, 0);
      set_render_func( "RENDER GRID_LINES_1_AND_2" );
    }

    unlock_cur_object();

}
/*------------------------- END OF set_type_func ----------------------------*/




/*++ static void render_func( int group, int item )
 *
 * PURPOSE:
 *
 *              Handles the render menu functions.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              6/89
 *              12/90   converted to a menu group call-back
 *              4/91    added command stuff
 *              6/91    rewrote command stuff
 *
 * INPUT PARAMETERS:
 *
 *              int             group           menu group number
 *              int             item            item number within group
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
```



```
 * GLOBAL VARIABLES USED:
 *
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *
 --*/

/*------------------------------- render_func -------------------------------*/

static void render_func( int group, int item )
{
    interactive = 1;
    load_command(   render_menu_script_commands[0],
                    render_menu_script_commands[item + 1]   );
}

/*------------------------- END OF render_func ------------------------------*/




/*++ static void set_render_func( char* script_command )
 *
 * PURPOSE:
 *
 *              Handles the render menu functions from a command
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              6/89
 *              12/90   converted to a menu group call-back
 *              4/91    Paul Kolaits added command stuff
 *              6/91    rewrote command stuff
 *
 * INPUT PARAMETERS:
 *
 *              char*   script_command          command
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_seger;             declared in this file
 *      extern Main_Acts*       main_acts               declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              extern int      command_index()         libpanu
 *              int             lock_cur_object()       defined in this file
 *              int             unlock_cur_object()     defined in this file
 *              void            delete_contours()       defined in this file
 *
 --*/

/*------------------------- set_render_func -------------------------------*/

static void set_render_func( char* script_command )
{
    char            param[32];
    int             index;


    parse_command( script_command, "%s", param );

    if ( ( index = command_index( param, render_menu_script_commands,
                    RENDER_MENU_ITEMS + 1 ) ) == -1 )
    {
        return;
    }

    --index;


    lock_cur_object();

    if ( grid_seger -> render_mode == index )
    {
        unlock_cur_object();
        return;
    }

    grid_seger -> render_mode = index;

    if ( ! interactive )
        ACCESS6(main_acts.render_menu_group, set_selection, 0,
                grid_seger -> render_mode, 1, 0);
    interactive = 0;

    if (grid_seger -> render_mode == CONTOUR_LINES)
    {
      delete_contours();
```

```
        ACCESS6(main_acts.type_menu_group,set_selection,0,SCALAR_TYPE,1, 0);
        set_type_func( "TYPE SCALAR" );
      }

      /* if a vector type was selected then select vector type */

      if (    grid_sager -> render_mode == PLAIN_VECTORS ||
              grid_sager -> render_mode == NORM_VECTORS       )
      {
        ACCESS6(main_acts.type_menu_group,set_selection,0,VECTOR_TYPE,1, 0);
        set_type_func( "TYPE VECTOR" );
      }

      unlock_cur_object();
}

/*------------------------- END OF set_render_func -------------------------*/




/*++ static void options_func( int group, int item )
 *
 * PURPOSE:
 *
 *              Handles the options menu functions.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              6/89
 *              12/90   converted to a menu group call-back
 *
 * INPUT PARAMETERS:
 *
 *              int             group           menu group number
 *              int             item            item number within group
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *
 --*/

/*------------------------------ options_func ------------------------------*/

static void options_func( int group, int item )
{
        int     val;

        interactive = 1;
        lock_cur_object();

        switch (group)
        {
          case 0:
                val = !grid_sager->draw_outline;
                load_command("OPTIONS %s %s", "OUTLINE", ON_OR_OFF(val));
                break;
          case 1:
                val = !grid_sager->draw_glyph;
                load_command("OPTIONS %s %s", "GLYPH", ON_OR_OFF(val));
                break;
          case 2:
                load_command("OPTIONS %s", "DELETE_NORMALS");
                break;
          case 3:
                load_command("OPTIONS %s", "RESET");
                break;
          case 4:
                load_command("OPTIONS %s", "DEBUG");
                break;
          default:
                interactive = 0;
                break;
        }

        unlock_cur_object();
}

/*------------------------- END OF options_func -----------------------------*/




/*++ static void set_options_func( char* script_command )
 *
 * PURPOSE:
 *
 *              Handles the options menu functions.
 *
 * AUTHORS:
 *
```



```
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              6/89
 *              12/90   converted to a menu group call-back
 *              5/91    converted to scripting
 *
 * INPUT PARAMETERS:
 *
 *              char*   script_command  command/act
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_sager;             declared in this file
 *      extern Main_Acts*       main_acts               declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
 --*/

/*------------------------------ set_options_func ---------------------------*/

static void set_options_func( char* script_command )
{
        char            param[32];
        char            val[8];


        parse_command(script_command, "%s", param);

        lock_cur_object();

        if (strcasecmp(param, "OUTLINE") == 0)
        {
          parse_command(script_command, "%s %s", param, val);
          grid_sager->draw_outline = ON_OR_OFF_VAL(val);
          ACCESS6(main_acts.options_menu_group, set_selection, 0, 0, grid_sager->draw_o
utline, 0);
        }
        else if (strcasecmp(param, "GLYPH") == 0)
        {
          parse_command(script_command, "%s %s", param, val);
          grid_sager->draw_glyph = ON_OR_OFF_VAL(val);
          ACCESS6(main_acts.options_menu_group, set_selection, 1, 0, grid_sager->draw_gl
yph, 0);
        }
        else if (strcasecmp(param, "DELETE_NORMALS") == 0)
        {
          delete_normals();
        }
        else if (strcasecmp(param, "RESET") == 0)
        {
          reset_state(1);
        }
        else if (strcasecmp(param, "DEBUG") == 0)
        {
          dump_state();
        }

        unlock_cur_object();

}

/*------------------------- END OF set_options_func -------------------------*/




/*++ static void dump_state( void )
 *
 * PURPOSE:
 *
 *              Prints the contents of the grid_sager structure and globals.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              6/89
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
```

```
*
*       extern Grid_Surface*    grid_sager      declared in this file
*
* FILES USED:
*
*               None
*
* NOTES:
*
* NON-STANDARD CODE :
*
* CALLED BY :
*
* FUNCTIONS CALLED :
*
*       int             lock_cur_object()       defined in this file
*       int             unlock_cur_object()     defined in this file
*
--*/

/*------------------------------- dump_state ---------------------------------*/

static void dump_state( void )
{
        lock_cur_object();

        printf("\n");
        printf("object_type                     = %d\n",
                grid_sager -> object_type);
        printf("draw                            = %d\n",
                grid_sager -> draw);
        printf("draw_outline                    = %d\n",
                grid_sager -> draw_outline);
        printf("draw_glyph                      = %d\n",
                grid_sager -> draw_glyph);
        printf("rev_normals                     = %d\n",
                grid_sager -> rev_normals);
        printf("zone_normals                    = %d\n",
                grid_sager -> zone_normals);
        printf("framed                          = %d\n",
                grid_sager -> framed);
        printf("shaded                          = %d\n",
                grid_sager -> shaded);
        printf("type                            = %d\n",
                grid_sager -> type);
        printf("contour_color_type              = %d\n",
                grid_sager -> contour_color_type);
        printf("vector_color_type               = %d\n",
                grid_sager -> vector_color_type);
        printf("vector_tip_type                 = %d\n",
                grid_sager -> vector_tip_type);
        printf("surface_mode                    = %d\n",
                grid_sager -> surface_mode);
        printf("render_mode                     = %d\n",
                grid_sager -> render_mode);
        printf("clip_mode                       = %d\n",
                grid_sager -> clip_mode);
        printf("minmax_mode                     = %d\n",
                grid_sager -> minmax_mode);
        printf("invert_clip_test                = %d\n",
                grid_sager -> invert_clip_test);
        printf("auto_minmax_update              = %d\n",
                grid_sager -> auto_minmax_update);
        printf("update_minmax_sliders           = %d\n",
                grid_sager -> update_minmax_sliders);
        printf("direction                       = %d\n",
                grid_sager -> direction);
        printf("loop_mode                       = %d\n",
                grid_sager -> loop_mode);
        printf("loop_dir                        = %d\n",
                grid_sager -> loop_dir);
        printf("loop_surface                    = %d\n",
                grid_sager -> loop_surface);
        printf("loop_zone                       = %d\n",
                grid_sager -> loop_zone);
        printf("loop_new_zone                   = %d\n",
                grid_sager -> loop_new_zone);
        printf("prev                            = %d\n",
                grid_sager -> prev);
        printf("reset_to_zone                   = %d\n",
                grid_sager -> reset_to_zone);
        printf("show_looping                    = %d\n",
                grid_sager -> show_looping);
        printf("boundary_flags[I J K][START END MID]   = \");
        printf("[%2d %2d %2d] [%2d %2d %2d] [%2d %2d %2d]\n",
                grid_sager -> boundary_flags[I][START],
                grid_sager -> boundary_flags[I][END],
                grid_sager -> boundary_flags[I][MID],
                grid_sager -> boundary_flags[J][START],
                grid_sager -> boundary_flags[J][END],
                grid_sager -> boundary_flags[J][MID],
                grid_sager -> boundary_flags[K][START],
                grid_sager -> boundary_flags[K][END],
                grid_sager -> boundary_flags[K][MID]);
        printf("scale_factors[VECTOR, FRAME]    = [%f %f]\n",
                grid_sager -> scale_factors[VECTOR_SCALE],
                grid_sager -> scale_factors[FRAME_SCALE]);
        printf("color_indices[LINE_COLOR]       = %d\n",
                grid_sager -> color_indices[LINE_COLOR]);
        printf("color_rgbs[LINE_COLOR]          = (%f %f %f)\n",
                grid_sager->color_rgbs[LINE_COLOR][R],
                grid_sager->color_rgbs[LINE_COLOR][G],
                grid_sager->color_rgbs[LINE_COLOR][B] );
        printf("color_indices[POINT_COLOR]      = %d\n",
                grid_sager -> color_indices[POINT_COLOR]);
        printf("color_rgbs[POINT_COLOR]         = (%f %f %f)\n",
                grid_sager->color_rgbs[POINT_COLOR][R],
                grid_sager->color_rgbs[POINT_COLOR][G],
                grid_sager->color_rgbs[POINT_COLOR][B] );
        printf("color_indices[CONTOUR_COLOR]    = %d\n",
                grid_sager -> color_indices[CONTOUR_COLOR]);
        printf("color_rgbs[CONTOUR_COLOR]       = (%f %f %f)\n",
                grid_sager->color_rgbs[CONTOUR_COLOR][R],
                grid_sager->color_rgbs[CONTOUR_COLOR][G],
                grid_sager->color_rgbs[CONTOUR_COLOR][B] );
        printf("color_indices[VECTOR_COLOR]     = %d\n",
                grid_sager -> color_indices[VECTOR_COLOR]);
        printf("color_rgbs[VECTOR_COLOR]        = (%f %f %f)\n",
                grid_sager->color_rgbs[VECTOR_COLOR][R],
                grid_sager->color_rgbs[VECTOR_COLOR][G],
                grid_sager->color_rgbs[VECTOR_COLOR][B] );
        printf("color_indices[POLYGON_COLOR]    = %d\n",
                grid_sager -> color_indices[POLYGON_COLOR]);
        printf("color_rgbs[POLYGON_COLOR]       = (%f %f %f)\n",
                grid_sager->color_rgbs[POLYGON_COLOR][R],
                grid_sager->color_rgbs[POLYGON_COLOR][G],
                grid_sager->color_rgbs[POLYGON_COLOR][B] );
        printf("color_indices[OUTLINE_COLOR]    = %d\n",
```



```
                grid_sager -> color_indices[OUTLINE_COLOR]);
        printf("color_rgbs[OUTLINE_COLOR]       = (%f %f %f)\n",
                grid_sager->color_rgbs[OUTLINE_COLOR][R],
                grid_sager->color_rgbs[OUTLINE_COLOR][G],
                grid_sager->color_rgbs[OUTLINE_COLOR][B] );
        printf("color_indices[GLYPH_COLOR]      = %d\n",
                grid_sager -> color_indices[GLYPH_COLOR]);
        printf("color_rgbs[GLYPH_COLOR]         = (%f %f %f)\n",
                grid_sager->color_rgbs[GLYPH_COLOR][R],
                grid_sager->color_rgbs[GLYPH_COLOR][G],
                grid_sager->color_rgbs[GLYPH_COLOR][B] );
        printf("zones[GRID_TYPE][REG FLD]       = %d %d\n",
                grid_sager -> zones[GRID_TYPE][REG],
                grid_sager -> zones[GRID_TYPE][FLD]);
        printf("zones[SCALAR_TYPE][REG FLD]     = %d %d\n",
                grid_sager -> zones[SCALAR_TYPE][REG],
                grid_sager -> zones[SCALAR_TYPE][FLD]);
        printf("zones[VECTOR_TYPE][REG FLD]     = %d %d\n",
                grid_sager -> zones[VECTOR_TYPE][REG],
                grid_sager -> zones[VECTOR_TYPE][FLD]);
        printf("dims[GRID_TYPE][I J K]          = [%d %d %d]\n",
                grid_sager -> dims[GRID_TYPE][I],
                grid_sager -> dims[GRID_TYPE][J],
                grid_sager -> dims[GRID_TYPE][K]);
        printf("dims[SCALAR_TYPE][I J K]        = [%d %d %d]\n",
                grid_sager -> dims[SCALAR_TYPE][I],
                grid_sager -> dims[SCALAR_TYPE][J],
                grid_sager -> dims[SCALAR_TYPE][K]);
        printf("dims[VECTOR_TYPE][I J K]        = [%d %d %d]\n",
                grid_sager -> dims[VECTOR_TYPE][I],
                grid_sager -> dims[VECTOR_TYPE][J],
                grid_sager -> dims[VECTOR_TYPE][K]);
        printf("ranges[I][START END INC CUR DIM]   = [%d %d %d %d %d]\n",
                grid_sager -> ranges[I][START],
                grid_sager -> ranges[I][END],
                grid_sager -> ranges[I][INC],
                grid_sager -> ranges[I][CUR],
                grid_sager -> ranges[I][DIM]);
        printf("ranges[J][START END INC CUR DIM]   = [%d %d %d %d %d]\n",
                grid_sager -> ranges[J][START],
                grid_sager -> ranges[J][END],
                grid_sager -> ranges[J][INC],
                grid_sager -> ranges[J][CUR],
                grid_sager -> ranges[J][DIM]);
        printf("ranges[K][START END INC CUR DIM]   = [%d %d %d %d %d]\n",
                grid_sager -> ranges[K][START],
                grid_sager -> ranges[K][END],
                grid_sager -> ranges[K][INC],
                grid_sager -> ranges[K][CUR],
                grid_sager -> ranges[K][DIM]);
        printf("minmax[CLIP][MINI MAXI BOTTOM TOP] = [%f %f %f %f]\n",
                grid_sager -> minmax[CLIP][MINI],
                grid_sager -> minmax[CLIP][MAXI],
                grid_sager -> minmax[CLIP][BOTTOM],
                grid_sager -> minmax[CLIP][TOP]);
        printf("minmax[NORM][MINI MAXI BOTTOM TOP] = [%f %f %f %f]\n",
                grid_sager -> minmax[NORM][MINI],
                grid_sager -> minmax[NORM][MAXI],
                grid_sager -> minmax[NORM][BOTTOM],
                grid_sager -> minmax[NORM][TOP]);
        printf("field_ids[GRID ISLANK SCALAR VECTOR] = [%d %d %d %d]\n",
                grid_sager -> field_ids[GRID_ID],
                grid_sager -> field_ids[ISLANK_ID],
                grid_sager -> field_ids[SCALAR_ID],
                grid_sager -> field_ids[VECTOR_ID]);
        printf("field_ids[I: START END MID ZONE]   = [%d %d %d %d]\n",
                grid_sager -> field_ids[START_I_NORM_ID],
                grid_sager -> field_ids[END_I_NORM_ID],
                grid_sager -> field_ids[MID_I_NORM_ID],
                grid_sager -> field_ids[ZONE_I_NORM_ID]);
        printf("field_ids[J: START END MID ZONE]   = [%d %d %d %d]\n",
                grid_sager -> field_ids[START_J_NORM_ID],
                grid_sager -> field_ids[END_J_NORM_ID],
                grid_sager -> field_ids[MID_J_NORM_ID],
                grid_sager -> field_ids[ZONE_J_NORM_ID]);
        printf("field_ids[K: START END MID ZONE]   = [%d %d %d %d]\n",
                grid_sager -> field_ids[START_K_NORM_ID],
                grid_sager -> field_ids[END_K_NORM_ID],
                grid_sager -> field_ids[MID_K_NORM_ID],
                grid_sager -> field_ids[ZONE_K_NORM_ID]);
        printf("field_ids[CONTOURS]             = [%d]\n",
                grid_sager -> field_ids[CONTOURS_ID]);
        printf("num_contours                    = %d\n",
                grid_sager -> num_contours);
        printf("contours_inc                    = %f\n",
                grid_sager -> contours_inc);
        printf("contours_min                    = %f\n",
                grid_sager -> contours_min);
        printf("contours_max                    = %f\n",
                grid_sager -> contours_max);
        printf("num_con_points                  = %d\n",
                grid_sager -> num_con_points);

        printf("\n");

        unlock_cur_object();

}

/*---------------------------- END OF dump_state -----------------------------*/




/*++ static void reset_state( int reset_actuators )
*
* PURPOSE:
*
*               Resets the contents of the grid_sager structure to the default
*               state (empty) and halts looping. If reset_actuators is 1 then
*               the actuators will be reset to their default states.
*
* AUTHORS:
*
*               Todd Plessel
*               NASA Ames Research Center
*               Sterling Software
*
* REVISION HISTORY:
*
*               6/89
```

```
*
* INPUT PARAMETERS:
*
*               int             reset_actuators         reset actuators too?
*
* OUTPUT PARAMETERS:
*
*               None
*
* FUNCTION RETURN:
*
*               None
*
* GLOBAL VARIABLES USED:
*
*       extern Grid_Surface*    grid_sager              declared in this file
*       extern Main_Acts        main_acts               defined in this file
*       extern Minmax_Acts      minmax_acts             defined in this file
*       extern ScaleGroup*      scale_group             defined in this file
*
* FILES USED:
*
*               None
*
* NOTES:
*
* NON-STANDARD CODE :
*
* CALLED BY :
*
* FUNCTIONS CALLED :
*
*       extern void     clear_typeout()                 libpanu
*       extern void     fix_color_panel()               libpanu
*       extern void     set_typein_fval()               libpanu
*
*       int             lock_cur_object()               defined in this file
*       int             unlock_cur_object()             defined in this file
*       void            update_legend()                 defined in this file
*       void            delete_normals()                defined in this file
*       void            set_default_colors()            defined in this file
*       void            copy_colors()                   defined in this file
*
--*/

/*------------------------------ reset_state ------------------------------*/

static void reset_state( int reset_actuators )
{
        Actuator*       ta;                     /* temp act pointer     */
        int             i;                      /* 1st loop index       */
        int             j;                      /* 2nd loop index       */


        /* if called from the button, fix actuators to reflect state */

        if (reset_actuators == 1)
        {
                /* reset type menu */

                ACCESS3(main_acts.type_menu_group, reset_selections, 0);

                /* reset render menu */

                ACCESS3(main_acts.render_menu_group, reset_selections, 0);

                /* reset attributes menu */

                ACCESS3(main_acts.attributes_menu_group, reset_selections, 0);

                /* reset options menu */

                ACCESS3(main_acts.options_menu_group, reset_selections, 0);

                /* delete normals */

                delete_normals();

                /* delete contours */

                delete_contours();

                /* reset loop buttons group */

                ACCESS3(main_acts.loop_buttons_group, reset_selections, 0);

                /* reset slider buttons group */

                ACCESS3(main_acts.slider_buttons_group, reset_selections, 0);

                /* reset surface buttons group */

                ACCESS3(main_acts.surface_buttons_group, reset_selections, 0);

                /* reset zone typein group */

                ACCESS4(main_acts.zone_typein_group, set_ivalue, 0, 0);

                /* reset slider group */

                ACCESS5(main_acts.slider_group, set_fvalues, I, slider_group_values[I], 0
);

                ACCESS5(main_acts.slider_group, set_fvalues, J, slider_group_values[J], 0
);

                ACCESS5(main_acts.slider_group, set_fvalues, K, slider_group_values[K], 0
);

                ACCESS4(main_acts.slider_group, set_selections, slider_group_selections,
0);

                ACCESS2(main_acts.slider_group, hide_select_buttons);
                ACCESS4(main_acts.slider_group, set_direction, K, 0);
                ACCESS3(main_acts.slider_group, highlight_slider, MID);

                /* data info typeout */

                clear_typeout(main_acts.data_info_typeout);


                /* vector panel's scale group */

                ACCESS3(scale_group, reset_values, 0);


                /* draw button */

                main_acts.draw_object_button -> val = 1.0;
                pnl_fixact(main_acts.draw_object_button);
```



```
                /* reset scalar minmax: */

                /* clip, norm, and legend minmax typeins */

                set_typein_fval(minmax_acts.clip_top_typein, 0.0,
                        FLOAT_STRING_FORMAT);
                set_typein_fval(minmax_acts.clip_bot_typein, 0.0,
                        FLOAT_STRING_FORMAT);
                set_typein_fval(minmax_acts.norm_top_typein, 0.0,
                        FLOAT_STRING_FORMAT);
                set_typein_fval(minmax_acts.norm_bot_typein, 0.0,
                        FLOAT_STRING_FORMAT);
                set_typein_fval(minmax_acts.legend_max_typein, 0.0,
                        FLOAT_STRING_FORMAT);
                set_typein_fval(minmax_acts.legend_min_typein, 0.0,
                        FLOAT_STRING_FORMAT);

                /* clip and norm multisliders */

                ta = minmax_acts.clip_multislider;
                ta = ta -> al;
                ta -> extval = 1.0;
                sprintf(ta -> label, FLOAT_STRING_FORMAT, 0.0);
                ta = ta -> next;
                ta -> extval = 0.0;
                sprintf(ta -> label, FLOAT_STRING_FORMAT, 0.0);
                pnl_fixact(minmax_acts.clip_multislider);

                ta = minmax_acts.norm_multislider;
                ta = ta -> al;
                ta -> extval = 1.0;
                sprintf(ta -> label, FLOAT_STRING_FORMAT, 0.0);
                ta = ta -> next;
                ta -> extval = 0.0;
                sprintf(ta -> label, FLOAT_STRING_FORMAT, 0.0);
                pnl_fixact(minmax_acts.norm_multislider);

                /* restore default minmax modes */

                minmax_acts.auto_minmax_update_button -> val = 1.0;
                pnl_fixact(minmax_acts.auto_minmax_update_button);

                minmax_acts.update_minmax_sliders_button -> val = 1.0;
                pnl_fixact(minmax_acts.update_minmax_sliders_button);

                minmax_acts.invert_clip_test_button -> val = 0.0;
                pnl_fixact(minmax_acts.invert_clip_test_button);

                fix_radio_buttons( minmax_acts.mode_buttons,
                                   NUM_SCALAR_MINMAX_MODES,
                                   MULTI_ZONE_MINMAX );
        }


        lock_cur_object();

        /* reinitialize the grid surface data structure */

        memset( grid_sager, 0, sizeof (Grid_Surface) );

        grid_sager -> object_type               = GRID_SURFACE;
        grid_sager -> surface_mode              = SINGLE_SURFACE;
        grid_sager -> type                      = GRID_TYPE;
        grid_sager -> render_mode               = LINES_1_2;
        grid_sager -> scale_factors[VECTOR_SCALE] = scale_group_values[0];
        grid_sager -> scale_factors[FRAME_SCALE]        = scale_group_values[1];
        grid_sager -> contour_color_type        = SCALAR_COLOR;
        grid_sager -> vector_color_type         = CONST_COLOR;
        grid_sager -> vector_tip_type           = NO_TIP;
        grid_sager -> direction                 = K;
        grid_sager -> ranges[I][INC]            = 1;
        grid_sager -> ranges[J][INC]            = 1;
        grid_sager -> ranges[K][INC]            = 1;
        grid_sager -> minmax_mode               = MULTI_ZONE_MINMAX;

        for (i = 0; i < NUM_GS_FIELD_IDS; ++i)
                grid_sager -> field_ids[i]      = -1;

        for (i = 0; i < 3; ++i)
          for (j = 0; j < 3; ++j)
            grid_sager -> boundary_flags[i][j]  = 1;

        grid_sager -> draw                      = 1;
        grid_sager -> shaded                    = DEFAULT_SHADING;
        grid_sager -> clip_mode                 = STRAIGHT_CLIP;
        grid_sager -> loop_mode = loop_mode     = LOOP_OFF;
        grid_sager -> loop_dir                  = LOOP_FORWARD;
        grid_sager -> loop_surface              = MID;
        grid_sager -> loop_zone                 = SINGLE_ZONE;
        grid_sager -> num_contours              = 40;

        grid_sager -> reset_to_zone             = 1;
        grid_sager -> show_looping              = 1;

        grid_sager -> auto_minmax_update        = 1;
        grid_sager -> update_minmax_sliders     = 1;


        unlock_cur_object();


        /* reset colors */

        set_default_colors();
        fix_color_panel();
        copy_colors();

        /* clear legend values */

        update_legend(0.0, 0.0, minmax_acts.legend_labels);

        /* Update data */

        update_fld_data_panel();
}

/*------------------------- END OF reset_state -------------------------*/
```

```
/*++ static void surface_buttons_func( int row, int col, int state )
 *
 * PURPOSE:
 *
 *              Handles setting the current surface button selections for
 *              commands
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/90
 *              12/90           converted to button group call-back
 *              4/91            added command stuff
 *
 * INPUT PARAMETERS:
 *
 *              int     row     selected button's row
 *              int     col     selected button's column
 *              int     state   selected button's state (0 or 1)
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *
 --*/

/*-------------------------- surface_buttons_func --------------------------*/

static void surface_buttons_func( int row, int col, int state )
{
        interactive = 1;
        load_command(   surface_buttons_script_commands[0],
                        surface_buttons_script_commands[1 + col] );
}

/*---------------------- END OF surface_buttons_func ----------------------*/
```

```
/*++ static void set_surface_buttons_func( char* script_command )
 *
 * PURPOSE:
 *
 *              Handles setting the current surface button selections from
 *              a command
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/90
 *              12/90           converted to button group call-back
 *              4/91            added command stuff
 *
 * INPUT PARAMETERS:
 *
 *               char* script_command   script command or actuator
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_sager;             declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              extern int      command_index()         libpanu
 *              int             lock_cur_object()       defined in this file
 *              int             unlock_cur_object()     defined in this file
 --*/

/*-------------------------- set_surface_buttons_func ----------------------*/

static void set_surface_buttons_func( char* script_command )
```



```
{
        char            param[32];
        int             selections[NUM_SURFACE_BUTTONS];
        int             index;

        parse_command( script_command, "%s", param );

        if ( ( index = command_index( param, surface_buttons_script_commands,
                        NUM_SURFACE_BUTTONS + 1 ) ) == -1 )
        {
                return;
        }

        --index;

        lock_cur_object();

        grid_sager -> surface_mode = index;

        if ( !interactive )
        {
                memset( selections, 0, sizeof selections );
                selections[grid_sager -> surface_mode] = 1;
                ACCESS4( main_acts.surface_buttons_group,
                        set_selections, selections, 0 );
        }

        delete_contours();

        unlock_cur_object();

        /* show the select buttons if boundary surfaces mode */

        if ( index == BOUNDARY_SURFACES )
        {
                ACCESS2( main_acts.slider_group, show_select_buttons );
        }
        else /* hide them */
        {
                ACCESS2( main_acts.slider_group, hide_select_buttons );
        }
}

/*---------------------- END OF set_surface_buttons_func ----------------------*/
```

```
/*++ static void slider_buttons_func( int row, int col, int state )
 *
 * PURPOSE:
 *
 *              Handles setting the current slider button selections.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/90
 *              12/90           converted to button group call-back
 *
 * INPUT PARAMETERS:
 *
 *              int     row     selected button's row
 *              int     col     selected button's column
 *              int     state   selected button's state (0 or 1)
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *
 --*/

/*-------------------------- slider_buttons_func --------------------------*/

static void slider_buttons_func( int row, int col, int state )
{
        interactive = 1;

        switch ( col )
        {
            case 0: load_command( "SLIDER_ACTION %s", "RESET" ); break;
            case 1: load_command( "SLIDER_ACTION %s %s", "RESET_ZONE",
                                                        ON_OR_OFF(state) );
                    break;
            case 2: load_command( "SLIDER_ACTION %s %s", "SHOW_LOOPING",
                                                        ON_OR_OFF(state) );
                    break;
            default: interactive = 0; break;
        }
}

/*---------------------- END OF slider_buttons_func ----------------------*/
```

```
/*++ static void set_slider_buttons_func( char* script_command )
 *
 * PURPOSE:
 *
 *              Handles setting the current slider button selections.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/90
 *              12/90           converted to button group call-back
 *              5/91            converted to scripting
 *
 * INPUT PARAMETERS:
 *
 *              char*   script_command  script command or actuator
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_sager;             declared in this file
 *      extern int              show_looping            declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              int             lock_cur_object()       defined in this file
 *              int             unlock_cur_object()     defined in this file
 *              void            reset_ijk_ranges()      defined in this file
 *
 --*/

/*------------------------- set_slider_buttons_func ----------------------*/

static void set_slider_buttons_func( char* script_command )
{
        int             col;
        char            param[16];
        char            cval[8];
        int             selections[3];


        parse_command( script_command, "%s", param );

        if      ( strcasecmp( param, "RESET"        ) == 0 )  col = 0;
        else if ( strcasecmp( param, "RESET_ZONE"   ) == 0 )  col = 1;
        else if ( strcasecmp( param, "SHOW_LOOPING" ) == 0 )  col = 2;
        else { interactive = 0; return; }

        lock_cur_object();

        switch ( col )
        {
                case 0:
                        reset_ijk_ranges();
                        break;
                case 1:
                        parse_command( script_command, "%s %s", param, cval );
                        grid_sager -> reset_to_zone = ON_OR_OFF_VAL(cval);
                        break;
                case 2:
                        parse_command( script_command, "%s %s", param, cval );
                        show_looping = ON_OR_OFF_VAL( cval );
                        grid_sager -> show_looping = show_looping;
                        break;
                default:
                        break;
        }

        if ( ! interactive )
        {
                selections[0] = 0;
                selections[1] = grid_sager -> reset_to_zone;
                selections[2] = show_looping;
                ACCESS4( main_acts.slider_buttons_group,
                        set_selections, selections, 0 );
        }

        interactive = 0;

        unlock_cur_object();
}

/*----------------------- END OF set_slider_buttons_func --------------------*/



/*++ static void loop_buttons_func( int row, int col, int state )
 *
 * PURPOSE:
 *
 *              Handles setting the current loop button selections.
 *
```



```
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/90
 *              12/90           converted to button group call-back
 *
 * INPUT PARAMETERS:
 *
 *              int     row     selected button's row
 *              int     col     selected button's column
 *              int     state   selected button's state (0 or 1)
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_sager;             declared in this file
 *      extern Main_Acts*       main_acts               declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              None
 *
 --*/

/*---------------------------- loop_buttons_func ---------------------------*/

static void loop_buttons_func( int row, int col, int state )
{
        interactive = 1;
        load_command(   loop_buttons_script_load_commands[row],
                        loop_buttons_script_param_commands[row][col]    );
}

/*----------------------- END OF loop_buttons_func ------------------------*/
/*++ static void set_loop_buttons_func( char* script_command )
 *
 * PURPOSE:
 *
 *              Handles setting the current loop button selections.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/90
 *              12/90           converted to button group call-back
 *
 * INPUT PARAMETERS:
 *
 *              char*           script command
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_sager;             declared in this file
 *      extern Main_Acts*       main_acts               declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              extern int      view_set_draw_mode()    libviewI
 *              void            delete_contours()       defined in this file
 *              int             lock_cur_object()       defined in this file
 *              int             unlock_cur_object()     defined in this file
 *
 --*/

/*---------------------------- set_loop_buttons_func ---------------------------*/

static void set_loop_buttons_func( char* script_command )
{
        int             d;                      /* I,J,K direction      */
        int             new_loop_surface;       /* START or END plane   */
        int             old_loop_surface;       /* START or END plane   */
        int*            ranges;                 /* grid ranges[d]       */
        int             i, row, col;
        int             selections[NUM_LOOP_BUTTONS];
        char            command[32];
```

```
        char            param[32];


        sscanf( script_command, "%s", command );
        parse_command( script_command, "%s", param );

        if ( ( row = command_index( command,
                loop_buttons_script_load_commands, LOOP_BUTTONS_ROWS)) == -1 )
        {
                interactive = 0;
                return;
        }

        if ( ( col = command_index( param,
                loop_buttons_script_param_commands[row], 4 ) ) == -1 )
        {
                interactive = 0;
                return;
        }


        lock_cur_object();

        delete_contours();

        new_loop_surface = old_loop_surface = grid_seqer -> loop_surface;

        switch ( row )
        {
            case 0: loop_mode = grid_seqer -> loop_mode = col; break;
            case 1: old_loop_surface = grid_seqer -> loop_surface;
                    new_loop_surface = grid_seqer -> loop_surface = col;
                    ACCESS3( main_acts.slider_group, highlight_slider, col );
                    break;
            case 2: grid_seqer -> loop_zone = col; break;
            default: interactive = 0; return;
        }

        /* If necessary, delete the old surface normals and contours */

        if (    grid_seqer -> surface_mode == SINGLE_SURFACE &&
                grid_seqer -> zone_normals == 0 &&
                new_loop_surface != old_loop_surface    )
        {
            i = NORM_ID_INDEX( grid_seqer->direction, grid_seqer->loop_surface );

            if ( grid_seqer -> field_ids[i] != -1 )
            {

#ifdef DEBUG
printf("delete surface normals for loop_surface = %d\n", grid_seqer -> loop_surface);
#endif

                SDEALLOCATE( grid_seqer -> field_ids[i] );
                grid_seqer -> field_ids[i] = -1;
            }
        }

        /* if not interactive then fix the loop buttons */

        if ( ! interactive )
        {
                memset( selections, 0, sizeof selections );

                selections[grid_seqer -> loop_mode] = 1;
                i = loop_buttons_per_row[0];
                selections[i + grid_seqer -> loop_surface] = 1;
                i += loop_buttons_per_row[1];
                selections[i + grid_seqer -> loop_zone] = 1;

                ACCESS4( main_acts.loop_buttons_group,
                        set_selections, selections, 0 );
        }

        interactive = 0;


        /* set draw mode based on current looping status */

        if ( grid_seqer -> loop_mode == LOOP_OFF )
        {
            grid_seqer -> prev = 0;
            view_set_draw_mode( cur_object, NO_NEED_TO_DRAW );
        }
        else
        {
            d           = grid_seqer -> direction;
            ranges      = grid_seqer -> ranges[d];

            if ( old_loop_surface != MID && grid_seqer -> prev != 0 )
                ranges[old_loop_surface] = grid_seqer -> prev;

            if ( new_loop_surface != MID )
                grid_seqer -> prev = ranges[new_loop_surface];

            /* redraw the sliders to indicate the new position */

            ACCESS5( main_acts.slider_group, set_ivalues, d, ranges, 0 );

            view_set_draw_mode(cur_object, ALWAYS_DRAW);
        }

        update_data_info();

        update_minmax();

        unlock_cur_object();
}

/*---------------------- END OF set_loop_buttons_func -----------------------*/



/*++ static void zone_func( int new_zone )
 *
 * PURPOSE:
 *
 *              Handles setting the current zone.
 *
```



```
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/90
 *              12/90   converted to typein group call-back
 *
 * INPUT PARAMETERS:
 *
 *              int     new_zone;       new zone value to set
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *              None
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *
 --*/

/*------------------------------ zone_func ----------------------------------*/

static void zone_func( int new_zone )
{
        interactive = 1;
        load_command( "ZONE %d", new_zone );
}

/*-------------------------- END OF zone_func -------------------------------*/



/*++ static void set_zone_func( char* script_command )
 *
 * PURPOSE:
 *
 *              Handles setting the current zone.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/90
 *              12/90   converted to typein group call-back
 *              5/91    converted to scripting
 *
 * INPUT PARAMETERS:
 *
 *              char*   script_command  script command or actuator
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *              None
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              extern int      set_fld_data_selection()        libfldpan
 *              extern void     update_fld_data_panel()         libfldpan
 *
 --*/

/*---------------------------- set_zone_func ----------------------------------*/

static void set_zone_func( char* script_command )
{
        int             new_zone;


        parse_command( script_command, "%d", &new_zone );

        /* select the new grid and update data */

        set_fld_data_selection( GRID, 0, new_zone );

        /* this will invoke the call-back function: data_select() */
```

```
        update_fld_data_panel();

        if ( ! interactive )
        {
                lock_cur_object();
                ACCESS4( main_acts.zone_typein_group, set_ivalue,
                                grid_seger -> zones[GRID_TYPE][FLD], 0 );
                unlock_cur_object();
        }

        interactive = 0;
}

/*---------------------- END OF set_zone_func ------------------------------*/









/*++ static void ijk_ranges_func( int dir, float ranges[5] )
 *
 * PURPOSE:
 *
 *              Adjusts the IJK ranges based on the slider
 *              specifications.
 *
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              4/89
 *              12/90   converted to slider group call-back
 *
 * INPUT PARAMETERS:
 *
 *              int             dir;            I/J/K
 *              float           ranges[5]       START/END/INC/CUR/DIM
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *
 --*/

/*-------------------------- ijk_ranges_func ------------------------------*/

static void ijk_ranges_func( int dir, float ranges[5] )
{
        static char*    dir_str[3] = { "I", "J", "K" };
        static char*    slider_str[5] = { "START", "END", "INC", "CUR", "DIM" };
        int             s, plane;

        lock_cur_object();
        interactive = 1;
        for ( s = 0; s < 5; ++s )
        {
            plane = ROUND( ranges[s] );
            if ( plane != grid_seger -> ranges[dir][s] )
                load_command( "SLIDER %s %s %d",
                              dir_str[dir], slider_str[s], plane );
        }
        unlock_cur_object();
}

/*------------------------- END OF ijk_ranges_func -------------------------*/









/*++ static void set_ijk_ranges_func( char* script_command )
 *
 * PURPOSE:
 *
 *              Adjusts the IJK ranges based on the slider
 *              specifications from a script command
 *
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              4/89
 *              12/90   converted to slider group call-back
 *              4/91    P. Kelaita -- added command handling
 *
 * INPUT PARAMETERS:
 *
 *              char*           str
```



```
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *              extern Grid_Surface*    grid_seger      declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              void            delete_contours()       defined in this file
 *              void            update_data_info()      defined in this file
 *              int             lock_cur_object()       defined in this file
 *              int             unlock_cur_object()     defined in this file
 *              void            check_delete_normals()  defined in this file
 *
 --*/

/*-------------------------- set_ijk_ranges_func ---------------------------*/

static void set_ijk_ranges_func( char* script_command )
{
        static char*    dir_str[3] = { "I", "J", "K" };
        static char*    slider_str[5] = { "START", "END", "INC", "CUR", "DIM" };
        int             d;                      /* IJK direction        */
        int             s;                      /* START/END/INC/CUR/DIM*/
        int             dir;
        int             plane;
        int             ranges[3][5];
        char            dir_str_param[8];
        char            slider_str_param[8];



        parse_command( script_command, "%s %s %d",
                       dir_str_param, slider_str_param, &plane );

        for ( dir = 0; dir < 3; ++dir )
          if ( strcasecmp( dir_str_param, dir_str[dir] ) == 0 ) break;
        if ( dir == 3 ) { interactive = 0; return; }

        for ( s = 0; s < 5; ++s )
          if ( strcasecmp( slider_str_param, slider_str[s] ) == 0 ) break;
        if ( s == 5 ) { interactive = 0; return; }

        if ( ! interactive ) /* Update slider group */
        {
          ACCESS6( main_acts.slider_group, set_ivalue, dir, s, plane, 0 );
        }

        interactive = 0;

        /* Get the new (possibly adjusted) ranges for this direction */

        ACCESS4( main_acts.slider_group, get_ivalues, dir, ranges[dir] );

        lock_cur_object();

        /* get all of the ranges */

        for ( d = 0; d < 3; ++d )
          for ( s = 0; s < 5; ++s )
            if ( d != dir ) ranges[d][s] = grid_seger -> ranges[d][s];

        /* If necessary, delete the normals data */

        check_delete_normals( grid_seger -> direction, ranges );

        /* Delete the contours data */

        delete_contours();

        /* update to these new ranges */

        for (s = 0; s < 5; ++s) grid_seger->ranges[dir][s] = ranges[dir][s];

        if ( grid_seger -> render_mode == CONTOUR_LINES ) delete_contours();

        unlock_cur_object();

        /* update the data info typeout */

        update_data_info();

        /* Update the scalar minmax panel */

        update_minmax();
}

/*---------------------- END OF set_ijk_ranges_func -------------------------*/









/*++ static void direction_func( int dir )
 *
 * PURPOSE:
 *
 *              Handles setting the current IJK direction, set up command
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
```

```
 *
 *              6/89
 *              12/90   converted to a slider group call-back
 *              4/91    changed to handle commands
 *
 * INPUT PARAMETERS:
 *
 *              int     dir     current direction
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *
 --*/

/*---------------------------- direction_func ---------------------------------*/

static void direction_func( int dir )
{
        static char*    dir_str[3] = { "I", "J", "K" };

        interactive = 1;
        load_command( "DIRECTION %s", dir_str[dir] );
}

/*------------------- END OF direction_func ---------------------------------*/




/*++ static void set_direction_func( char* script_command )
 *
 * PURPOSE:
 *
 *              Handles setting the current IJK direction from a command
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              4/91 added command capability
 *
 * INPUT PARAMETERS:
 *
 *              char*   script_command  script command or actuator
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_sager;             declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              void            delete_contours()       defined in this file
 *              int             lock_cur_object()       defined in this file
 *              int             unlock_cur_object()     defined in this file
 *
 --*/

/*---------------------------- set_direction_func ----------------------------*/

static void set_direction_func( char* script_command )
{
        int             old_dir;                /* I,J,K direction      */
        int             loop_surface;           /* START or END plane   */
        int*            ranges;                 /* grid ranges[d]       */
        int             dir;
        char            dir_str[8];


        parse_command( script_command, "%s", dir_str );

        if      ( strcasecmp( dir_str, "I" ) == 0 ) dir = I;
        else if ( strcasecmp( dir_str, "J" ) == 0 ) dir = J;
        else if ( strcasecmp( dir_str, "K" ) == 0 ) dir = K;
        else { interactive = 0; return; }

        lock_cur_object();

        old_dir = grid_sager -> direction;
        ranges = grid_sager -> ranges[old_dir];
```



```
        /* If necessary, delete the normals data */

        check_delete_normals( dir, grid_sager -> ranges );

        /* Delete the contours data */

        delete_contours();

        grid_sager -> direction = dir;

        if ( ! interactive )
            ACCESS4( mmin_acts.slider_group, set_direction, dir, 0 );
        interactive = 0;

        if ( old_dir != dir && grid_sager -> prev != 0 )
        {
            loop_surface = grid_sager -> loop_surface;

            if ( loop_surface != MID && grid_sager -> prev != 0 )
            {
                ranges[loop_surface] = grid_sager -> prev;
                grid_sager -> prev = grid_sager -> ranges[dir][loop_surface];

                /* redraw the sliders to indicate the new position */

                ACCESS5( mmin_acts.slider_group,
                        set_ivalues,old_dir, grid_sager->ranges[old_dir], 0 );
            }
        }

        if ( grid_sager -> render_mode == CONTOUR_LINES ) delete_contours();

        update_minmax();

        unlock_cur_object();
}

/*---------------------- END OF set_direction_func ---------------------------*/




/*++ static void boundary_surfaces_func( int selections[3][3] )
 *
 * PURPOSE:
 *
 *              Handles setting the current boundary surface selections for
 *              commands.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/90
 *              12/90           converted to button group call-back
 *              4/91            P. Kelaita -- added command stuff
 *
 * INPUT PARAMETERS:
 *
 *              int     selections[3][3]        new surface button selections
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *
 --*/

/*------------------------ boundary_surfaces_func -------------------------*/

static void boundary_surfaces_func( int selections[3][3] )
{
        static char*    off_on_str[2] = { "OFF", "ON" };
        static char*    dir_str[3] = { "I", "J", "K" };
        static char*    slider_str[3] = { "START", "END", "MID" };
        int             dir, s, off_on;

        lock_cur_object();
        interactive = 1;
        for ( dir = 0; dir < 3; ++dir )
        {
          for ( s = 0; s < 3; ++s )
          {
            off_on = selections[dir][s];
            if ( off_on != grid_sager -> boundary_flags[dir][s] )
                load_command( "BOUNDARY %s %s %s",
                              dir_str[dir], slider_str[s], off_on_str[off_on] );
          }
        }
        unlock_cur_object();
}

/*---------------------- END OF boundary_surfaces_func ----------------------*/
```

```
/*++ static void set_boundary_surfaces_func( char* script_command )
 *
 * PURPOSE:
 *
 *              Handles setting the current boundary surface selections for
 *              commands
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/90
 *              12/90           converted to button group call-back
 *              4/91            added command stuff
 *
 * INPUT PARAMETERS:
 *
 *              char*   script_command  script command or actuator
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_sager;             declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              int             lock_cur_object()       defined in this file
 *              int             unlock_cur_object()     defined in this file
 *              void            check_delete_normals()  defined in this file
 *
 --*/

/*------------------------- set_boundary_surfaces_func ---------------------*/

static void set_boundary_surfaces_func( char* script_command )
{
        static char*    dir_str[3] = { "I", "J", "K" };
        static char*    slider_str[3] = { "START", "END", "MID" };
        int             dir, s, off_on, searching;
        char            param_dir_str[8];
        char            param_slider_str[8];
        char            param_off_on_str[8];

        parse_command( script_command, "%s %s %s",
                       param_dir_str, param_slider_str, param_off_on_str );

        if      ( strcasecmp( param_off_on_str, "OFF" ) == 0 ) off_on = OFF;
        else if ( strcasecmp( param_off_on_str, "ON"  ) == 0 ) off_on = ON;
        else { interactive = 0; return; }

        for ( dir = 0, searching = TRUE; searching && dir < 3; ++dir )
        {
          if ( strcasecmp( param_dir_str, dir_str[dir] ) == 0 )
          {
            for ( s = 0; searching && s < 3; ++s )
            {
              if ( strcasecmp( param_slider_str, slider_str[s] ) == 0 )
              {
                searching = FALSE;
              }
            }
          }
        }

        --dir; --s;

        if ( searching ) return;

        interactive = 0;

        lock_cur_object();

        grid_sager -> boundary_flags[dir][s] = off_on;

        if ( ! interactive )
                ACCESS4( main_acts.slider_group, set_selections,
                         grid_sager -> boundary_flags, 0 );

        check_delete_normals( grid_sager -> direction, grid_sager -> ranges );
        delete_contours();

        unlock_cur_object();
}

/*------------------------ END OF set_boundary_surfaces_func ------------------*/
```

```
/*++ static void reset_ijk_ranges( void )
 *
 * PURPOSE:
 *
 *              Resets the IJK ranges either to full dims or partially.
```



```
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              1/90
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *              extern Grid_Surface*    grid_sager      defined in this file
 *              extern Main_Acts        main_acts;      declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *      void            update_data_info()      defined in this file
 *
 --*/

/*---------------------------- reset_ijk_ranges ----------------------------*/

static void reset_ijk_ranges( void )
{
        int             full_reset;             /* reset to full dims?  */
        int             dir;                    /* direction: I, J, K   */
        int             start;                  /* starting I, J, K     */
        int             end;                    /* ending I, J, K       */
        int             dim;                    /* dimension: I, J, K   */
        int             min;                    /* 1 or 0 if dim is 0   */
        int             temp[20];               /* to check buttons     */


        /* check if this is a full or partial reset */

        ACCESS3(main_acts.slider_buttons_group, get_selections, temp);
        full_reset = temp[1]; /* reset when zone changes? */

        lock_cur_object();

        /* loop on direction and reset each component */

        for ( dir = 0; dir < 3; ++dir )
        {
                /* set the dimension of the range */

                dim = grid_sager -> dims[GRID_TYPE][dir];
                grid_sager -> ranges[dir][DIM] = dim;

                /* if the dimension is zero then zero out other components */

                if ( dim == 0 )
                {
                        grid_sager -> ranges[dir][START]        = start = 0;
                        grid_sager -> ranges[dir][END]  = end   = 0;
                        grid_sager -> ranges[dir][INC]  = 1;
                        grid_sager -> ranges[dir][CUR]  = 0;
                }

                /* else if this is a full reset then do it */

                else if ( full_reset == 1 )
                {
                        grid_sager -> ranges[dir][START]        = start = 1;
                        grid_sager -> ranges[dir][END]  = end   = dim;
                        grid_sager -> ranges[dir][INC]  = 1;
                        grid_sager -> ranges[dir][CUR]  = ROUND( dim / 2.0 );

                        if ( grid_sager -> ranges[dir][CUR] == 0 )
                                grid_sager -> ranges[dir][CUR] = 1;
                }

                /* else only change if out-of-bounds */

                else
                {
                        /*
                         * only make changes required to have
                         * 1 <= start <= cur <= end <= dim and
                         * 1 <= inc <= dim
                         * note: LIMIT_TO(a, min, max) insures that
                         * min <= a <= max
                         */

                        min = MIN(1, dim);

                        LIMIT_TO( grid_sager -> ranges[dir][START], min, dim );
                        start =   grid_sager -> ranges[dir][START];
                        LIMIT_TO( grid_sager -> ranges[dir][END], start, dim );
                        end =     grid_sager -> ranges[dir][END];
                        LIMIT_TO( grid_sager -> ranges[dir][CUR], start, end );
                        LIMIT_TO_MAX( grid_sager -> ranges[dir][INC], dim );
                        LIMIT_TO_MIN( grid_sager -> ranges[dir][INC], 1 );
                }
        }

        /* update the sliders */

        ACCESS5( main_acts.slider_group,
                 set_ivalues, I, grid_sager -> ranges[I], 0 );
```

```
        ACCESS5( main_acts.slider_group,
                 set_ivalues, J, grid_sager -> ranges[J], 0 );
        ACCESS5( main_acts.slider_group,
                 set_ivalues, K, grid_sager -> ranges[K], 0 );

        delete_contours();

        unlock_cur_object();

        /* update data info typeout */

        update_data_info();
}

/*------------------------ END OF reset_ijk_ranges ------------------------*/
```

```
/*++ static void data_select( int type, int reg_num, int fld_num,
 *              FLDDataPtr fld_data_ptr )
 *
 * PURPOSE:
 *
 *              This routine is called from Surfer's fld data panel.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/89
 *
 * INPUT PARAMETERS:
 *
 *              int             type            GRID,SOLUTION,SCALAR,VECTOR
 *              int             reg_num         number of register selected
 *              int             fld_num         number of field selected
 *              FLDDataPtr      fld_data_ptr    -> fld data structure
 *                                              containing info about the
 *                                              selected data. Or this could
 *                                              be NULL indicating a reset
 *                                              affecting the given type
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Minmax_Acts      minmax_acts     this file
 *      extern Grid_Surface*    grid_sager              defined in this file
 *      extern int              update_actuators_mode;  declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      extern void     set_typein_ival()       libpanu
 *      void            update_dims()           defined in this file
 *      void            reset_data()            defined in this file
 *      void            reset_minmax()          defined in this file
 *      void            delete_normals()        defined in this file
 *      void            update_data_info()      defined in this file
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
 --*/

/*------------------------------ data_select ------------------------------*/

static void data_select( int type, int reg_num, int fld_num,
                FLDDataPtr fld_data_ptr )
{
        int     must_delete_normals = 0; /* only if data changed */
        int     new_scalar_field    = 0; /* only if data changed */


        /*
         * check if the data is NULL, if so this indicates that a
         * selection was on a field that is no longer available (and
         * the fld data panel automatically updated its typeouts) but we must
         * now call Surfer's own reset data function and pass type
         * to indicate that the reset pertains only to this type
         */

        if ( fld_data_ptr == NULL )
        {
                reset_data( type );
                return;
        }

        lock_cur_object();

        /* otherwise a field was selected so switch on the type */

        switch ( type )
        {
            case GRID:

                /* update grid zone info */

                grid_sager -> zones[GRID_TYPE][FLD] = fld_num;
                ACCESS4( main_acts.zone_typein_group, set_ivalue, fld_num, 0 );

                /* copy grid data id into grid surface structure */
```



```
                if (    grid_sager -> field_ids[GRID_ID] !=
                        fld_data_ptr -> field_ids[VF]   )
                {
                        must_delete_normals = 1;
                        grid_sager -> field_ids[GRID_ID] =
                                fld_data_ptr -> field_ids[VF];
                }

                /* copy grid iblanking data id */

                if ( fld_data_ptr -> attributes[IS_IBLANKED] )
                {
                        grid_sager -> field_ids[IBLANK_ID] =
                                fld_data_ptr -> field_ids[ISF];
                }
                else
                {
                        grid_sager -> field_ids[IBLANK_ID] = -1;
                }

                if (!MATCH_DIMS(grid_sager->dims[GRID_TYPE],fld_data_ptr->dims))
                {
                        must_delete_normals = 1;
                        update_dims(GRID_TYPE, fld_data_ptr -> dims);
                }

                break;

            case SCALAR:

                /* update zone info in grid surface struct */

                grid_sager -> zones[SCALAR_TYPE][REG] = reg_num;
                grid_sager -> zones[SCALAR_TYPE][FLD] = fld_num;

                /* copy scalar data id into grid surface structure */

                if (    grid_sager -> field_ids[SCALAR_ID] !=
                        fld_data_ptr -> field_ids[SF]   )
                {
                        new_scalar_field = 1;
                        grid_sager -> field_ids[SCALAR_ID] =
                                fld_data_ptr->field_ids[SF];
                }

                /* update the dims */

                if (!MATCH_DIMS(grid_sager->dims[SCALAR_TYPE],fld_data_ptr->dims))
                {
                        update_dims(SCALAR_TYPE, fld_data_ptr -> dims);
                }

                /* if in auto update mode then reset the scalar minmax */

                if ( minmax_acts.auto_minmax_update_button -> val == 1.0 &&
                     ( ! update_actuators_mode || new_scalar_field ) )
                {
                        reset_minmax( "RESET_MINMAX LEGEND" );
                }

                break;

            case SOLUTION:

                /* update zone info in grid surface struct */

                grid_sager -> zones[VECTOR_TYPE][REG] = reg_num;
                grid_sager -> zones[VECTOR_TYPE][FLD] = fld_num;

                /* copy vector data id into grid surface structure */

                if (    grid_sager -> field_ids[VECTOR_ID] !=
                        fld_data_ptr -> field_ids[VF]   )
                {
                        if (    grid_sager -> type == VECTOR_TYPE &&
                                USES_NORMALS(grid_sager)        )
                        {
                                must_delete_normals = 1;
                        }

                        grid_sager -> field_ids[VECTOR_ID] =
                                fld_data_ptr->field_ids[VF];
                }

                /* update the dims */

                if (!MATCH_DIMS(grid_sager->dims[VECTOR_TYPE],fld_data_ptr->dims))
                {
                        if (    grid_sager -> type == VECTOR_TYPE &&
                                USES_NORMALS(grid_sager)        )
                        {
                                must_delete_normals = 1;
                        }

                        update_dims(VECTOR_TYPE, fld_data_ptr -> dims);
                }

                /* if in auto update mode then reset the scalar minmax */

                if ( grid_sager -> type == VECTOR_TYPE )
                {
                    if ( minmax_acts.auto_minmax_update_button -> val == 1.0 )
                    {
                        reset_minmax( "RESET_MINMAX LEGEND" );
                    }
                }

                break;

            default:

                return;
        }


        /* if needed, delete any old normals and contours data if it exists */

        if ( must_delete_normals )
        {
                /* reset this mode to allow normals to be deleted */

                update_actuators_mode = 0;
                delete_normals();
                delete_contours();
        }
```

```
        /* update the contour data if need be */

        if ( grid_seqer -> render_mode == CONTOUR_LINES ) delete_contours();

        /* update the data info typeout */

        update_data_info();

        unlock_cur_object();

}

/*--------------------------- END OF data_select ----------------------------*/









/*++ static void reset_data( int type )
 *
 * PURPOSE:
 *
 *              Resets the zone data of the indicated type
 *              in the grid_seqer structure to the default state (empty).
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              6/89
 *
 * INPUT PARAMETERS:
 *
 *              int             type    GRID, SCALAR, VECTOR
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURNS:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_seqer              declared in this file
 *      extern Main_Acts        main_acts               this file
 *      extern Minmax_Acts      minmax_acts     this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      extern void     clear_typeout()                 libpanu
 *      extern void     set_typeout_fval()              libpanu
 *      extern void     update_fld_data_panel()         libfldpan
 *      void            delete_normals()                defined in this file
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
 --*/

/*------------------------------ reset_data -------------------------------*/

static void reset_data( int type )
{
        Actuator*       ta;                     /* temp act pointer     */


        /* reset zone data: clear appropriate registers and fields: */

        if ( type == GRID )
        {
                /* delete normals */

                delete_normals();

                /* delete contours */

                delete_contours();

                /* reset zone typein group */

                ACCESS4( main_acts.zone_typein_group, set_ivalue, 0, 0 );

                /* reset slider group */

                ACCESS5(main_acts.slider_group, set_fvalues, I, slider_group_values[I], 0
);
                ACCESS5(main_acts.slider_group, set_fvalues, J, slider_group_values[J], 0
);
                ACCESS5(main_acts.slider_group, set_fvalues, K, slider_group_values[K], 0
);
                ACCESS4(main_acts.slider_group, set_selections, slider_group_selections,
0);

                /* data info typeout */

                clear_typeout(main_acts.data_info_typeout);

                /* reinitialize part of the grid surface data structure */

                lock_cur_object();

                grid_seqer -> dims[GRID_TYPE][I]                = 0;
                grid_seqer -> dims[GRID_TYPE][J]                = 0;
                grid_seqer -> dims[GRID_TYPE][K]                = 0;
                grid_seqer -> ranges[I][START]          = 0;
                grid_seqer -> ranges[I][END]            = 0;
```



```
                grid_seqer -> ranges[I][INC]            = 1;
                grid_seqer -> ranges[I][CUR]            = 0;
                grid_seqer -> ranges[I][DIM]            = 0;
                grid_seqer -> ranges[J][START]          = 0;
                grid_seqer -> ranges[J][END]            = 0;
                grid_seqer -> ranges[J][INC]            = 1;
                grid_seqer -> ranges[J][CUR]            = 0;
                grid_seqer -> ranges[J][DIM]            = 0;
                grid_seqer -> ranges[K][START]          = 0;
                grid_seqer -> ranges[K][END]            = 0;
                grid_seqer -> ranges[K][INC]            = 1;
                grid_seqer -> ranges[K][CUR]            = 0;
                grid_seqer -> ranges[K][DIM]            = 0;
                grid_seqer -> field_ids[GRID_ID]                = -1;
                grid_seqer -> field_ids[IBLANK_ID]      = -1;

                unlock_cur_object();
        }
        else if ( type == SCALAR )
        {
                /* delete contours */

                delete_contours();

                /* reinitialize part of the grid surface data structure */

                lock_cur_object();

                grid_seqer -> zones[SCALAR_TYPE][REG]   = 0;
                grid_seqer -> zones[SCALAR_TYPE][FLD]   = 0;
                grid_seqer -> dims[SCALAR_TYPE][I]      = 0;
                grid_seqer -> dims[SCALAR_TYPE][J]      = 0;
                grid_seqer -> dims[SCALAR_TYPE][K]      = 0;
                grid_seqer -> minmax[CLIP][MINI]                = 0.0;
                grid_seqer -> minmax[CLIP][MAXI]                = 0.0;
                grid_seqer -> minmax[CLIP][BOTTOM]      = 0.0;
                grid_seqer -> minmax[CLIP][TOP]         = 0.0;
                grid_seqer -> minmax[NORM][MINI]                = 0.0;
                grid_seqer -> minmax[NORM][MAXI]                = 0.0;
                grid_seqer -> minmax[NORM][BOTTOM]      = 0.0;
                grid_seqer -> minmax[NORM][TOP]         = 0.0;
                grid_seqer -> field_ids[SCALAR_ID]      = -1;

                /* update and redraw the sliders */

                update_minmax_sliders(grid_seqer -> minmax, LEGEND);
                update_minmax_sliders(grid_seqer -> minmax, CLIP);
                update_minmax_sliders(grid_seqer -> minmax, NORM);

                unlock_cur_object();
        }
        else if ( type == VECTOR )
        {
                lock_cur_object();

                /* if vector shaded surface deallocate any normals data */

                if ( grid_seqer->type == VECTOR && USES_NORMALS( grid_seqer ) )
                {
                        delete_normals();
                }

                /* reinitialize part of the grid surface data structure */

                grid_seqer -> zones[VECTOR_TYPE][REG]   = 0;
                grid_seqer -> zones[VECTOR_TYPE][FLD]   = 0;
                grid_seqer -> dims[VECTOR_TYPE][I]      = 0;
                grid_seqer -> dims[VECTOR_TYPE][J]      = 0;
                grid_seqer -> dims[VECTOR_TYPE][K]      = 0;
                grid_seqer -> field_ids[VECTOR_ID]      = -1;

                unlock_cur_object();
        }
}

/*--------------------------- END OF reset_data ----------------------------*/









/*++ static void vector_scale( int index, float new_value )
 *
 * PURPOSE:
 *
 *              Sets the vector or frame scale value to this new value.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              11/90
 *              12/90   converted to scale group call-back
 *
 * INPUT PARAMETERS:
 *
 *              int             index           index of value changed
 *              float           new_value       new value
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURNS:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
```

```
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *
 --*/

/*---------------------------- vector_scale ---------------------------------*/

static void vector_scale( int index, float new_value )
{
        interactive = 1;
        load_command( scale_group_script_commands[index], new_value );
}

/*------------------------- END OF vector_scale -----------------------------*/









/*++ static void set_vector_scale( char* script_command )
 *
 * PURPOSE:
 *
 *              Sets the vector or frame scale value to this new value.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              11/90
 *              12/90   converted to scale group call-back
 *              5/91    converted to scripting
 *
 * INPUT PARAMETERS:
 *
 *              char*   script_command  script command or actuator
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *              extern Grid_Surface*    grid_seger      defined in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
 --*/

/*---------------------------- set_vector_scale -----------------------------*/

static void set_vector_scale( char* script_command )
{
        float           new_value;
        int             index;
        char            command[32];

        sscanf( script_command, "%s", command );
        parse_command( script_command, "%f", &new_value );

        if ( ( index = command_index( command,
                scale_group_script_commands, SCALE_GROUP_NUM_VALUES ) ) == -1 )
        {
                interactive = 0;
                return;
        }

        lock_cur_object();


        grid_seger -> scale_factors[index] = new_value;

        if ( ! interactive )
                ACCESS4( scale_group, set_values, grid_seger -> scale_factors, 0);
        interactive = 0;

        unlock_cur_object();
}

/*----------------------- END OF set_vector_scale ---------------------------*/









/*++ static void set_minmax_func( char* script_command )
 *
 * PURPOSE:
 *
 *              Sets the clip and norm top and bottom values to the
 *              values entered in the typeins.
 *
 *
 * AUTHORS:
```



```
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              4/89
 *
 * INPUT PARAMETERS:
 *
 *              char*   script_command          actuator / command
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *              extern Minmax_Acts      minmax_acts this file
 *              extern Grid_Surface*    grid_seger      defined in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 *              Requires that math.h be included for atof()
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      void            update_minmax_sliders()         defined in this file
 *      int             lock_cur_object()               defined in this file
 *      int             unlock_cur_object()             defined in this file
 *
 --*/

/*---------------------------- set_minmax_func ------------------------------*/

static void set_minmax_func( char* script_command )
{
        char*           num_str;                /* -> typein string     */
        float           minmax[2];              /* entered minmax value */
        int             clip_or_norm;           /* CLIP or NORM typein  */
        int             legend = 0;             /* 1 if act is legend   */
        Actuator*       a;
        Actuator*       a1;
        Actuator*       a2;
        float           min, max;
        char*           minval;
        char*           maxval;
        char            mode[16];

        if (is_act(script_command))
        {
           a = (Actuator*) script_command;

           if (strcmp(a -> u, CLIP_BOT_ID) == 0 ||
               strcmp(a -> u, CLIP_TOP_ID) == 0     )
           {
                minval = PNL_ACCESS(Typein, minmax_acts.clip_bot_typein, str);
                maxval = PNL_ACCESS(Typein, minmax_acts.clip_top_typein, str);
                load_command("MINMAX %s %f %f", "CLIP", atof(minval),
                                                        atof(maxval));
           }
           else if (strcmp(a->u, NORM_BOT_ID) == 0 ||
                    strcmp(a->u, NORM_TOP_ID) == 0     )
           {
                minval = PNL_ACCESS(Typein, minmax_acts.norm_bot_typein, str);
                maxval = PNL_ACCESS(Typein, minmax_acts.norm_top_typein, str);
                load_command("MINMAX %s %f %f", "NORM", atof(minval),
                                                        atof(maxval));
           }
           else if (strcmp(a -> u, LEGEND_MIN_ID) == 0 ||
                    strcmp(a -> u, LEGEND_MAX_ID) == 0     )
           {
                minval = PNL_ACCESS(Typein, minmax_acts.legend_min_typein, str);
                maxval = PNL_ACCESS(Typein, minmax_acts.legend_max_typein, str);
                load_command("MINMAX %s %f %f", "LEGEND", atof(minval),
                                                        atof(maxval));
           }
           else
           {
                return;
           }
           return;
        }

        parse_command(script_command, "%s %f %f", mode, &min, &max);

        if (strcasecmp(mode, "CLIP") == 0)
        {
                clip_or_norm    = CLIP;
                a1 = minmax_acts.clip_bot_typein;
                a2 = minmax_acts.clip_top_typein;
        }
        else if (strcasecmp(mode, "NORM") == 0)
        {
                clip_or_norm    = NORM;
                a1 = minmax_acts.norm_bot_typein;
                a2 = minmax_acts.norm_top_typein;
        }
        else if (strcasecmp(mode, "LEGEND") == 0)
        {
                legend          = 1;
                a1 = minmax_acts.legend_min_typein;
                a2 = minmax_acts.legend_max_typein;
        }
        else
        {
                return;
        }

        /* store the float equivalent in minmax */

        minmax[0] = min;
        minmax[1] = max;
```

```
        /* make sure the typein always displays a valid float number */

        if (minmax[0] == 0.0)
        {
                num_str = PNL_ACCESS(Typein, a1, str);
                sprintf(num_str, FLOAT_STRING_FORMAT, 0.0);
                pnl_fimact(a1);
        }
        if (minmax[1] == 0.0)
        {
                num_str = PNL_ACCESS(Typein, a2, str);
                sprintf(num_str, FLOAT_STRING_FORMAT, 0.0);
                pnl_fimact(a2);
        }

        /* make sure that max >= min */

        if (minmax[1] < minmax[0])
        {
                minmax[0] = minmax[1];

                num_str = PNL_ACCESS(Typein, a1, str);
                sprintf(num_str, FLOAT_STRING_FORMAT, minmax[0]);
                num_str = PNL_ACCESS(Typein, a2, str);
                sprintf(num_str, FLOAT_STRING_FORMAT, minmax[1]);
        }

        /* if the legend minmax was changed, limit the clip & norm minmax */

        lock_cur_object();

        if (legend)
        {
                /* set the clip & norm minmax to these values */

                grid_sager -> minmax[CLIP][MAXI]                = minmax[1];
                grid_sager -> minmax[CLIP][TOP]         = minmax[1];
                grid_sager -> minmax[CLIP][BOTTOM]      = minmax[0];
                grid_sager -> minmax[CLIP][MINI]                = minmax[0];
                grid_sager -> minmax[NORM][MAXI]                = minmax[1];
                grid_sager -> minmax[NORM][TOP]         = minmax[1];
                grid_sager -> minmax[NORM][BOTTOM]      = minmax[0];
                grid_sager -> minmax[NORM][MINI]                = minmax[0];

                /* update all of these actuators */

                update_minmax_sliders(grid_sager -> minmax, LEGEND);
                update_minmax_sliders(grid_sager -> minmax, CLIP);
                update_minmax_sliders(grid_sager -> minmax, NORM);
        }

        /* else just adjusted a top or bot value so fix only that group */

        else
        {
                /* make sure values entered are within the legend minmax */

                if (minmax[0] < grid_sager -> minmax[clip_or_norm][MINI])
                {
                        minmax[0] = grid_sager -> minmax[clip_or_norm][MINI];
                }

                if (minmax[1] > grid_sager -> minmax[clip_or_norm][MAXI])
                {
                        minmax[1] = grid_sager -> minmax[clip_or_norm][MAXI];
                }

                /* reset the active minmax to this value */

                grid_sager -> minmax[clip_or_norm][BOTTOM] = minmax[0];
                grid_sager -> minmax[clip_or_norm][TOP] = minmax[1];

                /* update and redraw the sliders */

                update_minmax_sliders(grid_sager -> minmax, clip_or_norm);
        }

        unlock_cur_object();

}

/*------------------------- END OF set_minmax_func --------------------------*/




/*++ static void reset_minmax( char* script_command )
 *
 * PURPOSE:
 *
 *              Resets the clip and norm minmax values to the actual
 *              data minmax.
 *
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              4/89
 *              5/91            Added scripting
 *
 * INPUT PARAMETERS:
 *
 *              char*           script_command  command/actuator
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *              extern Minmax_Acts      minmax_acts;    defined in this file
 *              extern Grid_Surface*    grid_sager      defined in this file
```



```
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      extern int      get_data_minmax()               libfldata
 *      extern int      get_data_list_minmax()          libfldata
 *      void            update_minmax_sliders()         defined in this file
 *      int             lock_cur_object()               defined in this file
 *      int             unlock_cur_object()             defined in this file
 *
 --*/

/*------------------------------ reset_minmax -------------------------------*/

static void reset_minmax( char* script_command )
{
        float           minmax[NUM_VARS][2];    /* zone minmax values   */
        int             type;                   /* CLIP, NORM or LEGEND */
        Actuator*       a;                      /* Actuator             */
        char            mode[16];



        if (is_act(script_command))
        {
            a = (Actuator*) script_command;

            if (strcmp(a -> u, RESET_CLIP_ID) == 0)
            {
                    load_command("RESET_MINMAX %s", "CLIP");
            }
            else if (strcmp(a -> u, RESET_NORM_ID) == 0)
            {
                    load_command("RESET_MINMAX %s", "NORM");
            }
            else if (strcmp(a -> u, RESET_LEGEND_ID) == 0)
            {
                    load_command("RESET_MINMAX %s", "LEGEND");
            }
            else
            {
                    return;
            }
            return;
        }

        parse_command(script_command, "%s", mode);

        if (strcasecmp(mode, "CLIP") == 0)
        {
                type = CLIP;
        }
        else if (strcasecmp(mode, "NORM") == 0)
        {
                type = NORM;
        }
        else if (strcasecmp(mode, "LEGEND") == 0)
        {
                type = LEGEND;
        }
        else
        {
                return;
        }

        lock_cur_object();

        if (type == LEGEND)
        {
                /* if we are not attached to any data then just return */

                if (grid_sager -> zones[SCALAR_TYPE][FLD] == 0)
                {
                        unlock_cur_object();
                        return;
                }

                /* if multi zone then get the minmax of all fields in reg */

                if (minmax_acts.mode_buttons[MULTI_ZONE_MINMAX] -> val == 1.0)
                {
                    if (!get_data_list_minmax(  SCALAR +
                                grid_sager -> zones[SCALAR_TYPE][REG],
                                minmax))
                    {
                        Error("No minmax data available for register!");
                        unlock_cur_object();
                        return;
                    }
                }

                /* else if we are in single zone mode, get this zone minmax */

                else if (minmax_acts.mode_buttons[SINGLE_ZONE_MINMAX] -> val == 1.0)
                {
                    if (!get_data_minmax(       SCALAR +
                                grid_sager -> zones[SCALAR_TYPE][REG],
                                grid_sager -> zones[SCALAR_TYPE][FLD],
                                minmax))
                    {
                        Error("No minmax for scalar field!\n");
                        unlock_cur_object();
                        return;
                    }
                }

                /* reset the active minmax to these data values */

                grid_sager -> minmax[CLIP][MINI]                = minmax[0][0];
                grid_sager -> minmax[CLIP][MAXI]                = minmax[0][1];
                grid_sager -> minmax[CLIP][BOTTOM]      = minmax[0][0];
                grid_sager -> minmax[CLIP][TOP]         = minmax[0][1];

                grid_sager -> minmax[NORM][MINI]                = minmax[0][0];
                grid_sager -> minmax[NORM][MAXI]                = minmax[0][1];
                grid_sager -> minmax[NORM][BOTTOM]      = minmax[0][0];
                grid_sager -> minmax[NORM][TOP]         = minmax[0][1];
```

```
            /* update and redraw the sliders */

            update_minmax_sliders(grid_sager -> minmax, LEGEND);
            update_minmax_sliders(grid_sager -> minmax, CLIP);
            update_minmax_sliders(grid_sager -> minmax, NORM);
        }

        else
        {
            /* reset the top & bottom sliders to their minmax */

            grid_sager -> minmax[type][BOTTOM] =
                        grid_sager -> minmax[type][MINI];
            grid_sager -> minmax[type][TOP] =
                        grid_sager -> minmax[type][MAXI];

            /* update and redraw the sliders */

            update_minmax_sliders(grid_sager -> minmax, type);

        }

        unlock_cur_object();

}

/*--------------------------- END OF reset_minmax ----------------------------*/




/*++ static void adjust_minmax_func( char* script_command )
 *
 * PURPOSE:
 *
 *              Adjusts the clip and norm minmax values based on the slider
 *              specifications.
 *
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              4/89
 *              5/91    Added scripting
 *
 * INPUT PARAMETERS:
 *
 *               char* script_command           Actuator/command
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *              extern Minmax_Acts      minmax_acts this file
 *              extern Grid_Surface*    grid_sager      defined in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      extern float    Denorm()                Fgl
 *      void            update_palettes()       defined in this file
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
 --*/

/*--------------------------- adjust_minmax_func -----------------------------*/

static void adjust_minmax_func( char* script_command )
{
        Actuator*       a;                      /* points to slider act */
        float           minval, maxval;         /* typein act floats    */


        if (is_act(script_command))
        {
                a = (Actuator*) script_command;

                lock_cur_object();

                if (strcmp(a -> u, CLIP_MSLIDER_ID) == 0)
                {
                    a = minmax_acts.clip_multislider;
                    minval = Denorm((a -> al) -> next) -> extval,
                                grid_sager -> minmax[CLIP][MINI],
                                grid_sager -> minmax[CLIP][MAXI]);
                    maxval = Denorm(a -> al) -> extval,
                                grid_sager -> minmax[CLIP][MINI],
                                grid_sager -> minmax[CLIP][MAXI]);
                    load_command("MINMAX %s %f %f", "CLIP", minval, maxval);
                }
                else if (strcmp(a -> u, NORM_MSLIDER_ID) == 0)
                {
                    a = minmax_acts.norm_multislider;
                    minval = Denorm(a -> al -> next -> extval,
                                grid_sager -> minmax[NORM][MINI],
                                grid_sager -> minmax[NORM][MAXI]);
                    maxval = Denorm(a -> al -> extval,
                                grid_sager -> minmax[NORM][MINI],
                                grid_sager -> minmax[NORM][MAXI]);
                    load_command("MINMAX %s %f %f", "NORM", minval, maxval);
                }
```



```
                unlock_cur_object();
                return;
        }
}

/*----------------------- END OF adjust_minmax_func ----------------------------*/




/*++ static void invert_clip_test( char* script_command )
 *
 * PURPOSE:
 *
 *              Sets the clip test mode based on the button selections.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/91
 *
 * INPUT PARAMETERS:
 *
 *              char* script_command            command/actuator
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Minmax_Acts      minmax_acts     defined in this file
 *      extern Grid_Surface*    grid_sager              declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              int     lock_cur_object()       defined in this file
 *              int     unlock_cur_object()     defined in this file
 *
 --*/

/*--------------------------- invert_clip_test -------------------------------*/

static void invert_clip_test( char* script_command )
{
        Actuator*       a = minmax_acts.invert_clip_test_button;
        char            mode[16];


        if ( is_act( script_command ) )
        {
            interactive = 1;
            load_command( "CLIP %s", a -> val == 1.0 ? "INSIDE" : "OUTSIDE" );
            return;
        }

        parse_command( script_command, "%s", mode );

        if ( ! interactive )
        {
            a -> val = (float) strcasecmp( mode, "OUTSIDE" );
            pnl_fixact( a );
        }

        interactive = 0;

        lock_cur_object();
        grid_sager -> invert_clip_test = (int) a -> val;
        unlock_cur_object();
}

/*------------------------- END OF invert_clip_test ---------------------------*/




/*++ static void set_minmax_modes( char* script_command )
 *
 * PURPOSE:
 *
 *              Sets the minmax mode based on the button selections.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              7/89    original version
 *              9/90    added multi zone minmax mode
 *              5/91    added scripting
```

```
 *
 * INPUT PARAMETERS:
 *
 *              char* script_command            command/actuator
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Minmax_Acts      minmax_acts     defined in this file
 *      extern Grid_Surface*    grid_seqer              declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              void    reset_minmax()          defined in this file
 *              int     lock_cur_object()       defined in this file
 *              int     unlock_cur_object()     defined in this file
 *
 --*/

/*------------------------------ set_minmax_modes ----------------------------*/

static void set_minmax_modes( char* script_command )
{
        Actuator*       a;
        char            mode[32], val[8];
        int             fix_minmax_buttons = 0;

        if (is_act(script_command))
        {
            interactive = 1;

            a = (Actuator*) script_command;
            if (strcmp(a -> u, AUTO_MINMAX_UPDATE_ID) == 0)
            {
                load_command("AUTO_MINMAX %s", ON_OR_OFF(a -> val));
            }
            else if (strcmp(a -> u, UPDATE_MMSLIDERS_ID) == 0)
            {
                load_command("UPDATE_MINMAX %s", ON_OR_OFF(a -> val));
            }
            else if (strcmp(a -> u, MULTI_ZONE_MINMAX_ID) == 0)
            {
                load_command("MINMAX_MODE %s", "MULTI_ZONE");
            }
            else if (strcmp(a -> u, SINGLE_ZONE_MINMAX_ID) == 0)
            {
                load_command("MINMAX_MODE %s", "SINGLE_ZONE");
            }
            else if (strcmp(a -> u, SUBSET_MINMAX_ID) == 0)
            {
                load_command("MINMAX_MODE %s", "ZONE_SUBSET");
            }
            else if (strcmp(a -> u, SURFACE_MINMAX_ID) == 0)
            {
                load_command("MINMAX_MODE %s", "SURFACE");
            }
            else if (strcmp(a -> u, SURFACE_SUBSET_MINMAX_ID) == 0)
            {
                load_command("MINMAX_MODE %s", "SURFACE_SUBSET");
            }
            else
            {
                interactive = 0;
            }
            return;
        }

        parse_command(script_command, "%s", mode);

        lock_cur_object();

        if (strncasecmp( script_command, "AUTO_MINMAX", 11 ) == 0)
        {
          grid_seqer -> auto_minmax_update = (int) ON_OR_OFF_VAL( mode );
          if ( grid_seqer -> auto_minmax_update == 1 )
          {
            if ( grid_seqer -> field_ids[SCALAR_ID] != -1 )
              reset_minmax( "RESET_MINMAX LEGEND" );
          }
          if ( ! interactive )
          {
            minmax_acts.auto_minmax_update_button -> val = ON_OR_OFF_VAL(val);
            pnl_fixact(minmax_acts.auto_minmax_update_button);
          }
        }
        else if (strncasecmp( script_command, "UPDATE_MINMAX", 13 ) == 0)
        {
          grid_seqer -> update_minmax_sliders = (int) ON_OR_OFF_VAL( mode );
          if ( ! interactive )
          {
            minmax_acts.update_minmax_sliders_button -> val = ON_OR_OFF_VAL(val);
            pnl_fixact(minmax_acts.update_minmax_sliders_button);
          }
        }
        else if (strcasecmp(mode, "MULTI_ZONE") == 0)
        {
          grid_seqer -> minmax_mode = MULTI_ZONE_MINMAX;
          reset_minmax( "RESET_MINMAX LEGEND" );
          clip_and_norm( TRUE );
          if ( ! interactive ) fix_minmax_buttons = 1;
        }
        else if (strcasecmp(mode, "SINGLE_ZONE") == 0)
        {
          grid_seqer -> minmax_mode = SINGLE_ZONE_MINMAX;
          reset_minmax( "RESET_MINMAX LEGEND" );
          clip_and_norm( TRUE );
          if ( ! interactive ) fix_minmax_buttons = 1;
        }
        else if (strcasecmp(mode, "ZONE_SUBSET") == 0)
        {
          grid_seqer -> minmax_mode = SUBSET_MINMAX;
```



```
          clip_and_norm( FALSE );
          if ( ! interactive ) fix_minmax_buttons = 1;
        }
        else if (strcasecmp(mode, "SURFACE") == 0)
        {
          grid_seqer -> minmax_mode = SURFACE_MINMAX;
          clip_and_norm( FALSE );
          if ( ! interactive ) fix_minmax_buttons = 1;
        }
        else if (strcasecmp(mode, "SURFACE_SUBSET") == 0)
        {
          grid_seqer -> minmax_mode = SURFACE_SUBSET_MINMAX;
          clip_and_norm( FALSE );
          if ( ! interactive ) fix_minmax_buttons = 1;
        }

        if ( fix_minmax_buttons )
        {
          fix_radio_buttons( minmax_acts.mode_buttons,
                             NUM_SCALAR_MINMAX_MODES,
                             grid_seqer -> minmax_mode );
        }

        interactive = 0;

        unlock_cur_object();

        /*
         * Must wait here until the object has been redrawn
         * since redrawing can change the minmax values.
         */

        wait_until_object_is_redrawn();

        /*
         * After drawing (which updates the minmax values),
         * update the scalar minmax panel (legend etc.).
         */

        update_minmax();
}

/*------------------------ END OF set_minmax_modes --------------------------*/









/*++ static void clip_and_norm( int state )
 *
 * PURPOSE:
 *
 *              Enables or disables use of clipping and normalization sliders.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/91
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_seqer;             declared in this file
 *      extern Minmax_Acts      minmax_acts;            declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              int     lock_cur_object()       defined in this file
 *              int     unlock_cur_object()     defined in this file
 *
 --*/

/*------------------------------ clip_and_norm -------------------------------*/

static void clip_and_norm( int state )
{
        int     i;

        lock_cur_object();

        if ( state == FALSE ) /* FALSE so disable and hide the actuators */
        {
                /* Reset & hide clip test button (to prevent 100% clipping) */

                grid_seqer -> invert_clip_test = FALSE;
                minmax_acts.invert_clip_test_button ->  val = 0.0;
                minmax_acts.invert_clip_test_button ->  selectable = 0;

                /* Hide the clipping and normalization actuators */

                minmax_acts.clip_label ->               selectable = 0;
                minmax_acts.norm_label ->               selectable = 0;
                minmax_acts.clip_top_typein ->          selectable = 0;
                minmax_acts.norm_top_typein ->          selectable = 0;
                minmax_acts.legend_max_typein ->        selectable = 0;
```

```
        minmax_acts.clip_multislider ->         selectable = 0;
        minmax_acts.norm_multislider ->         selectable = 0;
        minmax_acts.clip_bot_typein ->          selectable = 0;
        minmax_acts.norm_bot_typein ->          selectable = 0;
        minmax_acts.legend_min_typein ->        selectable = 0;
        minmax_acts.reset_clip_button ->        selectable = 0;
        minmax_acts.reset_norm_button ->        selectable = 0;
        minmax_acts.reset_legend_button ->      selectable = 0;

        for ( i = 0; i < NUM_PALETTE_TYPES; ++i )
        {
          if ( minmax_acts.palettes[NORM][i] )
              minmax_acts.palettes[NORM][i]->   selectable = 0;
          minmax_acts.palettes[CLIP][i] ->      selectable = 0;
        }
    }
    else /* TRUE so enable and show the actuators */
    {
        /* Show clip test button */

        minmax_acts.invert_clip_test_button ->  selectable = 1;

        /* Show the clipping and normalization actuators */

        minmax_acts.clip_label ->               selectable = 1;
        minmax_acts.norm_label ->               selectable = 1;
        minmax_acts.clip_top_typein ->          selectable = 1;
        minmax_acts.norm_top_typein ->          selectable = 1;
        minmax_acts.legend_max_typein ->        selectable = 1;
        minmax_acts.clip_multislider ->         selectable = 1;
        minmax_acts.norm_multislider ->         selectable = 1;
        minmax_acts.clip_bot_typein ->          selectable = 1;
        minmax_acts.norm_bot_typein ->          selectable = 1;
        minmax_acts.legend_min_typein ->        selectable = 1;
        minmax_acts.reset_clip_button ->        selectable = 1;
        minmax_acts.reset_norm_button ->        selectable = 1;
        minmax_acts.reset_legend_button ->      selectable = 1;

        for ( i = 0; i < NUM_PALETTE_TYPES; ++i )
        {
          if ( minmax_acts.palettes[NORM][i] )
              minmax_acts.palettes[NORM][i]->   selectable = 1;
          minmax_acts.palettes[CLIP][i] ->      selectable = 1;
        }
    }

    pnl_fixsct( minmax_acts.invert_clip_test_button );
    pnl_fixsct( minmax_acts.clip_label );
    pnl_fixsct( minmax_acts.norm_label );
    pnl_fixsct( minmax_acts.clip_top_typein );
    pnl_fixsct( minmax_acts.norm_top_typein );
    pnl_fixsct( minmax_acts.legend_max_typein );
    pnl_fixsct( minmax_acts.clip_multislider );
    pnl_fixsct( minmax_acts.norm_multislider );
    pnl_fixsct( minmax_acts.clip_bot_typein );
    pnl_fixsct( minmax_acts.norm_bot_typein );
    pnl_fixsct( minmax_acts.legend_min_typein );
    pnl_fixsct( minmax_acts.reset_clip_button );
    pnl_fixsct( minmax_acts.reset_norm_button );
    pnl_fixsct( minmax_acts.reset_legend_button );

    for ( i = 0; i < NUM_PALETTE_TYPES; ++i )
    {
      if ( minmax_acts.palettes[NORM][i] )
          pnl_fixsct( minmax_acts.palettes[NORM][i] );
        pnl_fixsct( minmax_acts.palettes[CLIP][i] );
    }

    unlock_cur_object();
}

/*---------------------------- END OF clip_and_norm ----------------------------*/




/*++ static int copy_normals( void )
 *
 * PURPOSE:
 *
 *              Allocates a new set of polygon normals data
 *              if needed, and copies the existing data into it.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              11/89
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              int             1 if successful else 0
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_saver;             declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
```



```
 *
 *              SALLOCATE() macro
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
 --*/

/*------------------------------ copy_normals ------------------------------*/

static int copy_normals( void )
{
    int             old_id;                 /* old normals id       */
    float*          old_data;               /* -> old normals data  */
    int             new_id;                 /* new normals id       */
    float*          new_data;               /* -> new normals data  */
    int             dir;                    /* I/J/K                */
    int             type;                   /* START/END/MID/ZONE   */
    int             i;                      /* index of id          */
    int*            dims;                   /* -> IJK dimensions    */
    int             size;                   /* norm size (in bytes) */


#ifdef DEBUG
printf("debug: copying normals\n");
#endif

    lock_cur_object();

    dims = &(grid_saver -> dims[GRID_TYPE][0]);

    for ( dir = 0; dir < 3; ++dir )
    {
        for ( type = 0; type < 4; ++type )
        {
            i = NORM_ID_INDEX( dir, type );
            size = 3 * sizeof (float);

            if ( type == ZONE ) size *= dims[I] * dims[J] * dims[K];
            else
            {
                switch ( dir )
                {
                    case I: size *= dims[K] * dims[J]; break;
                    case J: size *= dims[K] * dims[I]; break;
                    case K: size *= dims[J] * dims[I]; break;
                    default: break;
                }
            }

            if ( ( old_id = grid_saver -> field_ids[i] ) != -1 )
            {
                if ( ( old_da'     (float*) ATTACH( old_id ) ) == NULL )
                {
                    Error("Could not attach to existing normals data!");
                    unlock_cur_object();
                    return 0;
                }

                if ( ( new_id = SALLOCATE( size ) ) == -1 )
                {
                    Error("Could not allocate new normals data!");
                    DETACH( old_data );
                    unlock_cur_object();
                    return 0;
                }

                if ( ( new_data = (float*) ATTACH( new_id ) ) == NULL )
                {
                    Error("Could not attach to new normals data!");
                    DETACH( old_data );
                    SDEALLOCATE( new_id );
                    unlock_cur_object();
                    return 0;
                }

                memcpy( new_data, old_data, size );
                DETACH( old_data );
                DETACH( new_data );

                grid_saver -> field_ids[i] = new_id;
            }
        }
    }

    unlock_cur_object();

    return 1;
}

/*---------------------------- END OF copy_normals ----------------------------*/




/*++ static void delete_normals( void )
 *
 * PURPOSE:
 *
 *              Deallocates the polygon normals data.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              11/89
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
```

```
*
*               None
*
* GLOBAL VARIABLES USED:
*
*       extern Grid_Surface*    grid_sager;             declared in this file
*       extern int              update_actuators_mode;  declared in this file
*
* FILES USED:
*
*               None
*
* NOTES:
*
* NON-STANDARD CODE :
*
* CALLED BY :
*
* FUNCTIONS CALLED :
*
*               SDEALLOCATE() macro
*       int             lock_cur_object()       defined in this file
*       int             unlock_cur_object()     defined in this file
*
--*/

/*---------------------------- delete_normals ----------------------------*/

static void delete_normals( void )
{
        int             dir;                    /* I/J/K                */
        int             type;                   /* START/END/MID/ZONE   */
        int             i;                      /* index of id          */


        if ( update_actuators_mode == 1 ) return;

#ifdef DEBUG
printf("debug: deleting normals\n");
#endif
        lock_cur_object();

        /* deallocate existing normals data and set ids to -1 */

        for ( dir = 0; dir < 3; ++dir )
        {
            for ( type = 0; type < 4; ++type )
            {
                i = NORM_ID_INDEX( dir, type );

                if ( grid_sager -> field_ids[i] != -1 )
                {
                    SDEALLOCATE(grid_sager->field_ids[i]);
                    grid_sager -> field_ids[i] = -1;
                }
            }
        }

        unlock_cur_object();

        /* (new normals data will be generated by the drawing routines) */

}

/*------------------------- END OF delete_normals -------------------------*/



/*++ static void check_delete_normals( int new_dir, int new_ranges[3][5] )
*
* PURPOSE:
*
*               Checks and then deallocates the normals data if necessary.
*
* AUTHORS:
*
*               Todd Plessel
*               NASA Ames Research Center
*               Sterling Software
*
* REVISION HISTORY:
*
*               6/91
*
* INPUT PARAMETERS:
*
*               int             new_dir         I, J, or K
*               int             new_ranges[3][5] [I/J/K][START/END/INC/CUR/DIM]
*
* OUTPUT PARAMETERS:
*
*               None
*
* FUNCTION RETURN:
*
*               None
*
* GLOBAL VARIABLES USED:
*
*       extern Grid_Surface*    grid_sager;             declared in this file
*       extern int              update_actuators_mode;  declared in this file
*
* FILES USED:
*
*               None
*
* NOTES:
*
* NON-STANDARD CODE :
*
* CALLED BY :
*
* FUNCTIONS CALLED :
*
*               SDEALLOCATE() macro
*       int             lock_cur_object()       defined in this file
*       int             unlock_cur_object()     defined in this file
*       void            delete_normals()        defined in this file
*
--*/
```



```
/*---------------------------- check_delete_normals ----------------------------*/

static void check_delete_normals( int new_dir, int new_ranges[3][5] )
{
        int             old_dir;                /* IJK direction        */
        int*            old_range;              /* START/END/MID/ZONE   */
        int             loop_surface;           /* START/END/MID        */
        int             zone_normals;           /* use zone normals?    */
        int             changed = 0;            /* delete them?         */
        int             cur;                    /* loop_surface index   */
        int             count;                  /* count surfaces drawn */
        int             i;                      /* index on normals id  */
        int             drawn;                  /* boundary surf drawn? */
        int             dir;                    /* IJK direction        */
        int             type;                   /* START/END/MID/ZONE   */


#ifdef DEBUG
printf("debug: check deleting normals\n");
#endif

        lock_cur_object();

        zone_normals    = grid_sager -> zone_normals;
        loop_surface    = grid_sager -> loop_surface;
        old_dir         = grid_sager -> direction;
        old_range       = &(grid_sager -> ranges[old_dir][0]);


        switch ( grid_sager -> surface_mode )
        {
            case SINGLE_SURFACE:

                if ( new_dir != old_dir ) changed = 1;
                else if ( ! zone_normals )
                {
                    cur = loop_surface == MID ? CUR : loop_surface;
                    changed = old_range[cur] != new_ranges[old_dir][cur];

                    for ( dir = 0; dir < 3; ++dir )
                    {
                        i = NORM_ID_INDEX( dir, ZONE );

                        if ( grid_sager -> field_ids[i] != -1 )
                        {
#ifdef DEBUG
printf("debug: deleting normals for surface dir = %d, ZONE\n", dir );
#endif

                            SDEALLOCATE( grid_sager -> field_ids[i] );
                            grid_sager -> field_ids[i] = -1;
                        }
                    }
                }

                break;

            case BOUNDARY_SURFACES:

                for ( dir = 0; dir < 3; ++dir )
                {
                    count = 0;
                    old_range = &(grid_sager -> ranges[dir][0]);

                    for ( type = 0; type < 4; ++type )
                    {
                        i = NORM_ID_INDEX( dir, type );
                        drawn = 1;

                        if ( type != ZONE )
                          count += drawn = grid_sager->boundary_flags[dir][type];

                        cur = type == MID ? CUR : type;
                        changed = !zone_normals &&
                                old_range[cur] != new_ranges[dir][cur];

                        if ( drawn == 0 || changed ||
                             ( zone_normals && type == ZONE && count == 0 ) )
                        {
                            if ( grid_sager -> field_ids[i] != -1 )
                            {
#ifdef DEBUG
printf("debug: deleting normals for surface dir = %d, type = %d\n", dir, type);
#endif

                                SDEALLOCATE( grid_sager -> field_ids[i] );
                                grid_sager -> field_ids[i] = -1;
                            }
                        }
                    }
                }

                changed = 0;
                break;

            case SURFACE_RANGE: changed = old_dir != new_dir; break;

            default: break;
        }

        if ( changed ) delete_normals();

        unlock_cur_object();
}

/*------------------------ END OF check_delete_normals ------------------------*/



/*++ static int copy_contours( void )
*
* PURPOSE:
*
*               Allocates a new set of contours data
*               if needed, and copies the existing data into it.
*
```

```
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              11/89
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              int             1 if successful else 0
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_sager;             declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              SALLOCATE() macro
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
 --*/

/*---------------------------- copy_contours -----------------------------*/

static int copy_contours( void )
{
        int             old_id;                 /* old contours id      */
        float*          old_data;               /* -> old contours data */
        int             new_id;                 /* new contours id      */
        float*          new_data;               /* -> new contours data */
        int             size;                   /* size (in bytes)      */


#ifdef DEBUG
printf("debug: copying contours\n");
#endif

        lock_cur_object();

        size = grid_sager -> num_con_points * 4 * sizeof (float);

        if ( ( old_id = grid_sager -> field_ids[CONTOURS_ID] ) != -1 )
        {
          if ( ( old_data = (float*) ATTACH( old_id ) ) == NULL )
          {
            Error("Could not attach to existing contours data!");
            unlock_cur_object();
            return 0;
          }

          if ( ( new_id = SALLOCATE( size ) ) == -1 )
          {
            Error("Could not allocate new contours data!");
            DETACH( old_data );
            unlock_cur_object();
            return 0;
          }

          if ( ( new_data = (float*) ATTACH( new_id ) ) == NULL )
          {
            Error("Could not attach to new contours data!");
            DETACH( old_data );
            SDEALLOCATE( new_id );
            unlock_cur_object();
            return 0;
          }

          memcpy( new_data, old_data, size );
          DETACH( old_data );
          DETACH( new_data );

          grid_sager -> field_ids[CONTOURS_ID] = new_id;
        }

        unlock_cur_object();

        return 1;
}

/*------------------------- END OF copy_contours --------------------------*/



/*++ static void delete_contours( void )
 *
 * PURPOSE:
 *
 *              Deallocates the contours data.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
```



```
 *
 *              11/89
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_sager;             declared in this file
 *      extern int              update_actuators_mode;  declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              SDEALLOCATE() macro
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
 --*/

/*---------------------------- delete_contours ----------------------------*/

static void delete_contours( void )
{
        if ( update_actuators_mode == 1 ) return;

#ifdef DEBUG
printf("debug: deleting contours\n");
#endif
        lock_cur_object();

        /* deallocate existing contours data and set ids to -1 */

        if ( grid_sager -> field_ids[CONTOURS_ID] != -1 )
        {
          SDEALLOCATE(grid_sager->field_ids[CONTOURS_ID]);
          grid_sager -> field_ids[CONTOURS_ID] = -1;
        }

        unlock_cur_object();

        /* (new contours data will be generated by the drawing routines) */
}

/*------------------------- END OF delete_contours ------------------------*/



/*++ static void first_object_name( char* name )
 *
 * PURPOSE:
 *
 *              Passes back the name of the first available object or an empty
 *              string if there are none available.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              1/91
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *              None
 *
 * FILES USED:
 *
 *              None
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 *      void            init_panels()                   defined in this file
 *
 * FUNCTIONS CALLED :
 *
 --*/

/*---------------------------- first_object_name --------------------------*/

static void first_object_name( char* name )
{
        static char     buf[OBJECT_BUF_SIZE];   /* object name listing  */
```

```
        char            command[32];
        int             i;                      /* index into buf/name  */


DPRINT("  inside first_object_name(%s):\n", name);

        memset( buf, 0, sizeof buf );

DPRINT("  buf = '%s'\n", buf);

        /* get an updated listing into the typeout buffer */

        sprintf (command, "LIST_OBJECTS %d", GRID_SURFACE);
        send_hub_command(command);
        module_read_sock(buf, sizeof(buf));

DPRINT("  back from module_read_sock( with buf = '%s')\n", buf);

        /* extract the name of the first object */

        i = 0;

        while (buf[i] != '\0' && buf[i] != '\n' && i < OBJECT_NAME_LENGTH - 1)
        {
                name[i] = buf[i];
                ++i;
        }

        name[i] = '\0';
}

/*------------------------ END OF first_object_name --------------------------*/




/*++ void update_object_typeout( void )
 *
 * PURPOSE:
 *
 *              Updates the object typeout listing.
 *
 * AUTHORS:
 *
 *              Fergus J. Merritt
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              3/90
 *              11/90   Todd Plessel
 *                      added mutually recursive call and exit_module().
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Main_Acts        main_acts;      declared in this file
 *
 * FILES USED:
 *
 *              None
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 *      void            init_panels()                   defined in this file
 *      void            new_object()            defined in this file
 *
 * FUNCTIONS CALLED :
 *
 *      extern void     clear_typeout()                 libpanu
 *      extern int      set_selection_num()             libpanu
 *      extern void     select_object()                 defined in this file
 *
--*/

/*------------------------ update_object_typeout -----------------------*/

void update_object_typeout( void )
{
        int             buf_size;
        char*           buf;            /* points into typeout buf      */
        char            command[32];


        clear_typeout(main_acts.object_typeout);

        buf = PNL_ACCESS(Typeout, main_acts.object_typeout, buf);
        buf_size = PNL_ACCESS(Typeout, main_acts.object_typeout, size);

        /* get an updated listing into the typeout buffer */

        sprintf (command, "LIST_OBJECTS %d", GRID_SURFACE);
        send_hub_command(command);
        module_read_sock(buf, buf_size);

        /* try to reselect the current object */

        if (set_selection_name(main_acts.object_typeout, cur_object_name) == 0)
        {
            /* if that failed try to select the first line */

            if (set_selection_num(main_acts.object_typeout, 1) == 0)
            {
                /* if that failed then there are no objects left so exit */
```



```
                Warning("There are no objects left so I'm exiting!\n");
                exit_module();
            }

            /* otherwise update grid surface object with this new selection */

            select_object( (char*) main_acts.object_typeout );
        }
}

/*------------------- END OF update_object_typeout -------------------------*/




/*++ void update_actuators( void )
 *
 * PURPOSE:
 *
 *              Resets the panel based on the contents of the grid_saver
 *              structure.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              3/90
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_saver              declared in this file
 *      extern Main_Acts        main_acts               this file
 *      extern Minmax_Acts      minmax_acts     this file
 *      extern int              update_actuators_mode;  declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      extern void     set_typein_ival()               libpanu
 *      extern void     set_typein_fval()               libpanu
 *      extern int      set_fld_data_selection()        libfldpan
 *      extern void     update_fld_data_panel()         libfldpan
 *      void            update_colors()                 defined in this file
 *      void            update_minmax_sliders()         defined in this file
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
--*/

/*---------------------------- update_actuators ------------------------------*/

void update_actuators( void )
{
        int             i;
        int             temp[20];               /* for button updates   */


        /* set global flag to prevent unneeded deleting of normals data */

        update_actuators_mode = 1;

        lock_cur_object();


        /* fix the Draw the Object button */

        if ( ((int) main_acts.draw_object_button->val) != grid_saver->draw )
        {
                main_acts.draw_object_button->val = (float) grid_saver->draw;
                pnl_fixact( main_acts.draw_object_button );
        }

        /* fix type menu */

        ACCESS6(main_acts.type_menu_group, set_selection, 0, grid_saver -> type, 1, 0);


        /* fix render menu */

        ACCESS6(main_acts.render_menu_group, set_selection, 0, grid_saver -> render_mode,
1, 0);


        /* fix attributes menu */

        ACCESS6(main_acts.attributes_menu_group, set_selection, 0, grid_saver -> contour_
color_type, 1, 0);
        ACCESS6(main_acts.attributes_menu_group, set_selection, 1, grid_saver -> vector_c
olor_type, 1, 0);
        ACCESS6(main_acts.attributes_menu_group, set_selection, 2, grid_saver -> vector_t
ip_type, 1, 0);
        ACCESS6(main_acts.attributes_menu_group, set_selection, 3, grid_saver -> clip_mod
e, 1, 0);
        ACCESS6(main_acts.attributes_menu_group, set_selection, 4, grid_saver -> shaded,
1, 0);
        ACCESS6(main_acts.attributes_menu_group, set_selection, 5, grid_saver -> framed,
```

```
1, 0);
        ACCESS6(main_acts.attributes_menu_group, set_selection, 6, grid_sager -> rev_nor
mals, 1, 0);
        ACCESS6(main_acts.attributes_menu_group, set_selection, 7, grid_sager -> zone_no
rmals, 1, 0);


        /* fix the options menu */

        ACCESS6(main_acts.options_menu_group, set_selection, 0, 0, grid_sager->draw_outl
ine, 0);
        ACCESS6(main_acts.options_menu_group, set_selection, 1, 0, grid_sager->draw_glyp
h, 0);


        /* fix the loop buttons */

        memset( temp, 0, sizeof temp );

        temp[grid_sager -> loop_mode] = 1;
        i = loop_buttons_per_row[0];
        temp[i + grid_sager -> loop_surface] = 1;
        i += loop_buttons_per_row[1];
        temp[i + grid_sager -> loop_zone] = 1;

        ACCESS4(main_acts.loop_buttons_group, set_selections, temp, 0);

        /* also set global */

        loop_mode = grid_sager -> loop_mode;


        /* fix the slider buttons */

        memset( temp, 0, sizeof temp );
        temp[1] = grid_sager -> reset_to_zone;
        temp[2] = grid_sager -> show_looping;

        ACCESS4(main_acts.slider_buttons_group, set_selections, temp, 0);


        /* fix the surface mode buttons */

        memset( temp, 0, sizeof temp );
        temp[grid_sager -> surface_mode] = 1;

        ACCESS4(main_acts.surface_buttons_group, set_selections, temp, 0);

        /* show the select buttons if boundary surfaces mode */

        if (grid_sager -> surface_mode == BOUNDARY_SURFACES)
        {
                ACCESS2(main_acts.slider_group, show_select_buttons);
        }
        else /* hide them */
        {
                ACCESS2(main_acts.slider_group, hide_select_buttons);
        }

        /* fix the zone typein */

        ACCESS4(main_acts.zone_typein_group, set_ivalue, grid_sager -> zones[GRID_TYPE][
FLD], 0);
```

```
        /* fix the sliders */

        ACCESS5(main_acts.slider_group, set_ivalues, I, grid_sager -> ranges[I], 0);
        ACCESS5(main_acts.slider_group, set_ivalues, J, grid_sager -> ranges[J], 0);
        ACCESS5(main_acts.slider_group, set_ivalues, K, grid_sager -> ranges[K], 0);

        ACCESS4(main_acts.slider_group, set_selections, grid_sager -> boundary_flags, 0);

        ACCESS4(main_acts.slider_group, set_direction, grid_sager -> direction, 0);

        if (grid_sager -> surface_mode != BOUNDARY_SURFACES)
                ACCESS2(main_acts.slider_group, hide_select_buttons);

        i = grid_sager -> loop_surface;

        ACCESS3(main_acts.slider_group, highlight_slider, i);


        /* fix vector and frame scale factors */

        ACCESS4(scale_group, set_values, grid_sager -> scale_factors, 0);


        /* update the scalar minmax panel */

        /* fix the invert clip test button */

        if ( ((int) minmax_acts.invert_clip_test_button->val) !=
                grid_sager->invert_clip_test )
        {
                minmax_acts.invert_clip_test_button->val =
                        (float) grid_sager->invert_clip_test;
                pnl_fixact( minmax_acts.invert_clip_test_button );
        }

        /* and the auto minmax update button */

        if ( ((int) minmax_acts.auto_minmax_update_button->val) !=
                grid_sager->auto_minmax_update )
        {
                minmax_acts.auto_minmax_update_button->val =
                        (float) grid_sager->auto_minmax_update;
                pnl_fixact( minmax_acts.auto_minmax_update_button );
        }

        /* and the update minmax sliders button */

        if ( ((int) minmax_acts.update_minmax_sliders_button->val) !=
                grid_sager->update_minmax_sliders )
        {
                minmax_acts.update_minmax_sliders_button->val =
                        (float) grid_sager->update_minmax_sliders;
                pnl_fixact( minmax_acts.update_minmax_sliders_button );
        }

        /* and the minmax mode buttons */

        fix_radio_buttons( minmax_acts.mode_buttons, NUM_SCALAR_MINMAX_MODES,
                                grid_sager -> minmax_mode );

        update_minmax_sliders(grid_sager -> minmax, CLIP);
        update_minmax_sliders(grid_sager -> minmax, NORM);
        update_minmax_sliders(grid_sager -> minmax, LEGEND);
```



```
        /* vector panel's scale group */

        ACCESS4( scale_group, set_values, grid_sager -> scale_factors, 0 );

        /* update the color edit panel to display the new colors */

        update_colors();


        /* update the data panel by trying to select this object's zones */

        set_fld_data_selection( GRID,
                                grid_sager -> zones[GRID_TYPE][REC],
                                grid_sager -> zones[GRID_TYPE][FLD]);

        set_fld_data_selection( SCALAR,
                                grid_sager -> zones[SCALAR_TYPE][REC],
                                grid_sager -> zones[SCALAR_TYPE][FLD]);

        set_fld_data_selection( VECTOR,
                                grid_sager -> zones[VECTOR_TYPE][REC],
                                grid_sager -> zones[VECTOR_TYPE][FLD]);


        unlock_cur_object();

        /* this will invoke the call-back function: data_select() */

        update_fld_data_panel();

        /* reset global flag to normal */

        update_actuators_mode = 0;

}

/*------------------------- END OF update_actuators ------------------------*/
```

```
/*++ static void update_data_info( void )
 *
 * PURPOSE:
 *
 *              Updates the data info typeout.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              3/90
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Main_Acts        main_acts               defined in this file
 *      extern Grid_Surface*    grid_sager;             declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      extern int      print_field_node_info()         libfldpan
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
 --*/

#define DIMS_MATCH(dims1, dims2)        (dims1[I] == dims2[I] &&        \
                                         (dims1[J] == dims2[J] &&        \
                                          dims1[K] == dims2[K]))

/*------------------------- update_data_info ------------------------*/

static void update_data_info( void )
{
        int             ind[NUM_DIMS];          /* holds current IJK    */
        int             dims[NUM_DIMS];         /* holds dimensions IJK */
        char*           buf;                    /* points into typeout  */
        int             temp_scalar_id;         /* pass to update data  */
        int             temp_vector_id;         /* pass to update data  */
        int             dir;                    /* IJK direction        */


        lock_cur_object();

        buf = PHL_ACCESS(Typeout, main_acts.data_info_typeout, buf);
        ind[I] = grid_sager -> ranges[I][CUR];
        ind[J] = grid_sager -> ranges[J][CUR];
        ind[K] = grid_sager -> ranges[K][CUR];
        dims[I] = grid_sager -> ranges[I][DIM];
        dims[J] = grid_sager -> ranges[J][DIM];
        dims[K] = grid_sager -> ranges[K][DIM];

        dir = grid_sager -> direction;
```

```
        if ( grid_seqer -> loop_surface != MID )
            ind[dir] = grid_seqer -> ranges[dir][grid_seqer -> loop_surface];

        /* check that the dimensions are valid for the other data */

        temp_scalar_id = -1;
        temp_vector_id = -1;

        if (DIMS_MATCH(grid_seqer -> dims[GRID_TYPE], grid_seqer -> dims[SCALAR_TYPE]))
        {
                temp_scalar_id = grid_seqer -> field_ids[SCALAR_ID];
        }

        if (DIMS_MATCH(grid_seqer -> dims[GRID_TYPE], grid_seqer -> dims[VECTOR_TYPE]))
        {
                temp_vector_id = grid_seqer -> field_ids[VECTOR_ID];
        }

        print_field_node_info(ind, dims,
                grid_seqer -> field_ids[GRID_ID],
                grid_seqer -> field_ids[IBLANK_ID],
                temp_scalar_id,
                temp_vector_id,
                buf, DATA_INFO_BUF_SIZE);

        pnl_fixact(mmin_acts.data_info_typeout);

        unlock_cur_object();
}

/*------------------------ END OF update_data_info --------------------------*/
```

```
/*++ static void update_dims( int type, int dims[3] )
 *
 * PURPOSE:
 *
 *              Updates the grid surface structure's dims and possible range.
 *
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              4/89
 *
 * INPUT PARAMETERS:
 *
 *              int     type            GRID_TYPE, SCALAR_TYPE, VECTOR_TYPE
 *              int     dims[NUM_DIMS]  IJK names
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *              extern Grid_Surface*    grid_seqer      defined in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      void            delete_contours()               defined in this file
 *      void            reset_ijk_ranges()              defined in this file
 *      int             lock_cur_object()               defined in this file
 *      int             unlock_cur_object()             defined in this file
 *
 --*/

/*------------------------------ update_dims --------------------------------*/

static void update_dims( int type, int dims[3] )
{
        if (type != GRID_TYPE && type != SCALAR_TYPE && type != VECTOR_TYPE)
        {
                Error("Invalid type!\n");
                return;
        }

        /* update dims */

        lock_cur_object();

        grid_seqer -> dims[type][I] = dims[I];
        grid_seqer -> dims[type][J] = dims[J];
        grid_seqer -> dims[type][K] = dims[K];

        if (grid_seqer->render_mode == CONTOUR_LINES) delete_contours();

        unlock_cur_object();

        /* if grid type then reset the IJK ranges to within these dims */

        if (type == GRID_TYPE) reset_ijk_ranges();
}

/*---------------------------- END OF update_dims ----------------------------*/
```



```
/*++ static void update_minmax_sliders( float minmax[2][4], int type )
 *
 * PURPOSE:
 *
 *              Updates the scalar minmax sliders/typeins to
 *              the specified minmax values.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              6/89
 *
 * INPUT PARAMETERS:
 *
 *              float   minmax[2][4]    CLIP/NORM, MINI..TOP minmax values
 *              int     type;           CLIP or NORM
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Minmax_Acts      minmax_acts;    this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      extern float    Norm()                  libfgl
 *      extern void     set_typein_fval()       libpanu
 *      void            update_legend()         defined in this file
 *      void            update_palettes()       defined in this file
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
 --*/

/*------------------------ update_minmax_sliders ----------------------------*/

static void update_minmax_sliders( float minmax[2][4], int type )
{
        Actuator*       ms;                     /* -> a multislider     */
        Actuator*       sa;                     /* -> a slider in ms    */
        Actuator*       mina;                   /* min typein           */
        Actuator*       maxa;                   /* max typein           */


        /* select appropriate multislider */

        if (type == CLIP)
        {
                ms      = minmax_acts.clip_multislider;
                mina    = minmax_acts.clip_bot_typein;
                maxa    = minmax_acts.clip_top_typein;
        }
        else if (type == NORM)
        {
                ms      = minmax_acts.norm_multislider;
                mina    = minmax_acts.norm_bot_typein;
                maxa    = minmax_acts.norm_top_typein;
        }
        else if (type == LEGEND)
        {
                mina    = minmax_acts.legend_min_typein;
                maxa    = minmax_acts.legend_max_typein;
        }
        else
        {
                return;
        }

        if (type != LEGEND)
        {
                /* update the top slider */

                sa = ms -> al;
                sprintf(sa -> label, FLOAT_STRING_FORMAT, minmax[type][TOP]);
                sa -> extval = Norm(    minmax[type][TOP],
                                        minmax[type][MINI], minmax[type][MAXI]  );

                /* update the bottom slider */

                sa = sa -> next;
                sprintf(sa -> label, FLOAT_STRING_FORMAT, minmax[type][BOTTOM]);
                sa -> extval = Norm(    minmax[type][BOTTOM],
                                        minmax[type][MINI], minmax[type][MAXI]  );

                /* redraw the appropriate multislider */

                pnl_fixact(ms);

                /* update the top and bottom typeins */

                set_typein_fval(maxa, minmax[type][TOP], FLOAT_STRING_FORMAT);
                set_typein_fval(mina, minmax[type][BOTTOM], FLOAT_STRING_FORMAT);

                /* update the palettes */

                update_palettes(type);
        }
        else /* legend */
        {
                /* update the min and max typeins */
```

```
                set_typein_fval(maxa, minmax[0][MAXI], FLOAT_STRING_FORMAT);
                set_typein_fval(mina, minmax[0][MINI], FLOAT_STRING_FORMAT);

                /* redraw legend numbers */

                update_legend(minmax[0][MINI], minmax[0][MAXI],
                              minmax_acts.legend_labels);


                /* now take care of the contour stuff */

                lock_cur_object();

                grid_seger -> contours_min = minmax[0][MINI];
                grid_seger -> contours_max = minmax[0][MAXI];
                grid_seger -> contours_inc =
                        (minmax[0][MAXI] - minmax[0][MINI]) /
                                (float)grid_seger -> num_contours;

                /* update the min/max typein */

                mina = contour_acts.contour_min_typein;
                maxa = contour_acts.contour_max_typein;

                set_typein_fval(maxa, minmax[0][MAXI], FLOAT_STRING_FORMAT);
                set_typein_fval(mina, minmax[0][MINI], FLOAT_STRING_FORMAT);

                /* redraw contour legend numbers */

                update_legend(minmax[0][MINI], minmax[0][MAXI],
                              contour_acts.contour_labels);

                set_typein_ival(contour_acts.contour_num_typein,
                        grid_seger -> num_contours, INT_STRING_FORMAT);

                set_typein_fval(contour_acts.contour_inc_typein,
                        grid_seger -> contours_inc, FLOAT_STRING_FORMAT);

                unlock_cur_object();
        }
}

/*-------------------- END OF update_minmax_sliders ------------------------*/









/*++ static void update_legend( float min_val, float max_val, Actuator** act )
 *
 * PURPOSE:
 *
 *              Updates the legend to the specified minmax values.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/89
 *
 * INPUT PARAMETERS:
 *
 *              float   min_val                 minumum legend value
 *              float   max_val                 maximum legend value
 *              Actuator*       *act
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Minmax_Acts      minmax_acts     this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 *              void update_minmax_sliders()    defined in this file
 *
 * FUNCTIONS CALLED :
 *
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
 --*/

/*------------------------------ update_legend --------------------------------*/

static void update_legend( float min_val, float max_val, Actuator** act )
{
        int             i;                      /* loop on labels       */
        float           val;                    /* value of label       */
        float           val_inc;                /* increment value      */
        Actuator*       ta;                     /* points at label acts */

        /* calc incremental value */

        val_inc = (max_val - min_val) / (float) (NUM_LEGEND_VALUES - 1);

        /* set initial value */

        val = min_val;

        /* fix each label (from the botton up) with incremental value */
```



```
        for (i = 0; i < NUM_LEGEND_VALUES; ++i)
        {
                ta = act[i];
                sprintf(ta -> label, FLOAT_STRING_FORMAT, val);
                pnl_fixact(ta);
                val += val_inc;
        }
}

/*------------------------ END OF update_legend ------------------------------*/









/*++ static void update_palettes( int clip_or_norm )
 *
 * PURPOSE:
 *
 *              Updates the palettes to reflect the new settings of
 *              the specified slider.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/89
 *
 * INPUT PARAMETERS:
 *
 *              int     clip_or_norm            CLIP or NORM multislider
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Minmax_Acts      minmax_acts;    this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 *              void update_minmax_sliders()            defined in this file
 *              void adjust_minmax_func()               defined in this file
 *
 * FUNCTIONS CALLED :
 *
 *              extern void     draw_palettes()         libpanu
 *              int             lock_cur_object()       defined in this file
 *              int             unlock_cur_object()     defined in this file
 *
 --*/

/*------------------------------ update_palettes ------------------------------*/

static void update_palettes( int clip_or_norm )
{
        Actuator*       ms;                     /* clip or norm multislider */
        float           norm_bot_val;           /* norm bottom slider val */
        float           norm_top_val;           /* norm top slider val  */
        float           clip_bot_val;           /* clip bottom slider val */
        float           clip_top_val;           /* clip top slider val  */
        float           val;                    /* tmp normalized value */
        float           vals[NUM_PALETTE_TYPES][2]; /* bot/top pal vals */
        float           origin[2];              /* x/y coord of bot left */
        float           colors[NUM_PALETTE_TYPES][2]; /* bot/top color vals */




        /* get state of norm multislider */

        ms = minmax_acts.norm_multislider;
        norm_top_val = ms -> sl -> extval;
        norm_bot_val = ms -> sl -> next -> extval;

        origin[Y] = ms -> y;

        if (clip_or_norm == NORM)
        {
                /* fix norm palette */

                origin[X] = ms -> x - PALETTE_WIDTH;

                vals[MED][MAXIMUM] = norm_top_val;
                vals[MED][MINIMUM] = norm_bot_val;

                colors[MED][MAXIMUM] = MAX_MAP;
                colors[MED][MINIMUM] = MIN_MAP;


                draw_palettes(  NUM_PALETTE_TYPES,
                                minmax_acts.palettes[NORM],
                                vals,
                                colors,
                                PALETTE_HEIGHT,
                                origin);


                /* fix legend palettes */
```

```
                origin[X] += PALETTE_X_INC;

                vals[HI][MAXIMUM] = 1.0;
                vals[HI][MINIMUM] = norm_top_val;

                vals[LOW][MAXIMUM] = norm_bot_val;
                vals[LOW][MINIMUM] = 0.0;

                colors[HI][MAXIMUM] = MAX_MAP;
                colors[HI][MINIMUM] = MAX_MAP;

                colors[LOW][MAXIMUM] = MIN_MAP;
                colors[LOW][MINIMUM] = MIN_MAP;


                draw_palettes(  NUM_PALETTE_TYPES,
                                minmax_acts.palettes[LEGEND],
                                vals,
                                colors,
                                PALETTE_HEIGHT,
                                origin);

        }

        else
        {
                colors[HI][MAXIMUM] = MAX_MAP;
                colors[HI][MINIMUM] = MAX_MAP;
                colors[LOW][MAXIMUM] = MIN_MAP;
                colors[LOW][MINIMUM] = MIN_MAP;
        }


        /* fix clip palettes */

        /* get state of clip multislider */

        ms = minmax_acts.clip_multislider;
        clip_top_val = ms -> sl -> extval;
        clip_bot_val = ms -> sl -> next -> extval;

        origin[X] = ms -> x - PALETTE_WIDTH;

        /* calc positions and MED colors */

        if (clip_top_val >= norm_top_val)
        {
                vals[HI][MAXIMUM] = clip_top_val;
                vals[HI][MINIMUM] = MAX(clip_bot_val, norm_top_val);

                colors[MED][MAXIMUM] = MAX_MAP;
        }

        else
        {
                vals[HI][MAXIMUM] = clip_top_val;
                vals[HI][MINIMUM] = clip_top_val;

                val = Norm(clip_top_val, norm_bot_val, norm_top_val);
                colors[MED][MAXIMUM] = get_function_index(val);
        }

        if (clip_bot_val <= norm_bot_val)
        {
                vals[LOW][MAXIMUM] = MIN(clip_top_val, norm_bot_val);
                vals[LOW][MINIMUM] = clip_bot_val;

                colors[MED][MINIMUM] = MIN_MAP;
        }

        else
        {
                vals[LOW][MAXIMUM] = clip_bot_val;
                vals[LOW][MINIMUM] = clip_bot_val;

                val = Norm(clip_bot_val, norm_bot_val, norm_top_val);
                colors[MED][MINIMUM] = get_function_index(val);
        }

        /* MED palette is always between LOW and HI palettes */

        vals[MED][MAXIMUM] = vals[HI][MINIMUM];
        vals[MED][MINIMUM] = vals[LOW][MAXIMUM];


        draw_palettes(  NUM_PALETTE_TYPES,
                        minmax_acts.palettes[CLIP],
                        vals,
                        colors,
                        PALETTE_HEIGHT,
                        origin);

        pal_fixact(minmax_acts.minmax_frame);
}

/*------------------------- END OF update_palettes -------------------------*/




/*--------------------------- END OF FILE panels.c ---------------------------*/
```



```
c**********************************************************************
c**********************************************************************
c
c
      subroutine sager()
c
c
c**********************************************************************
c**********************************************************************
c
c
c  All modifications to sage will be done in lower case and the modified lines
c  will be commented out.
c
c
c**********************************************************************
c**********************************************************************
c
c
c
c      PROGRAM SAGE
c
C Latest amendment: 27th Apr, 1992
C THIS VERSION 3-D SAGE CAN ALSO HANDLE 2-D DATASETS.
C IT USES THE 3.1 VERSION OF 2-D SAGE AS A BASE,
C THUS IT DOES NOT HANDLE PERIODIC OR CYCLICAL BOUNDARIES
C
C        THE MATHEMATICAL METHOD FOLLOWS THE APPROACH DEVISED
C        BY NAKAHASHI AND DEIWERT
C
C      WRITTEN BY CAROL DAVIES, STERLING SOFTWARE.
C      UNPUBLISHED DOCUMENTATION IS AVAILABLE, JAN 91
C
C   ***** THIS IS NOT A GUARANTEED BUG-FREE CODE *****
C            PLEASE CONTACT ME IF YOU FIND ANY!
C
C      TELEPHONE 415-604-6204
C
C*************************************************************
C
C                     MAIN ROUTINE
C
C THIS IS THE DRIVING ROUTINE.
C SUBROUTINES ARE CALLED TO PERFORM ALL  FUNCTIONS
C
C CALLS: FBAR,INITIAL,LINE1,OUTPUT,NOADAPT,BLOCK,READAT,TORCOF,TORSION,
C        SETUPJ,SETUPK,SOLUT,UPDATE,WTEDGE,MARCHJ
C
C*************************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      common/comxyz/x(id,jd,kd),y(id,jd,kd),z(id,jd,kd)
      common/comq/q(id,jd,kd,ndim)
      COMMON/COM2/F(IMX),FS(IMX),WEIGHT(IMX),A,B,WDS,WDE
      COMMON/COM3/SS(IMX),SMS(IMX),DS(IMX),SN(IMX),SNM(IMX),SNMX(IMX)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM5/RDSMAX,RDSMIN,INDQ,IQ(NDIM+3)
      COMMON/COM6/DSMAX,DSMIN,WT(IMX),DLENGS,DLENGE,WDSS,WDES
      COMMON/COM7/COSE(IMX,3),COSU(IMX,3),COSB(IMX,3)
      COMMON/COM8/XJ(IMX,3,3),YJ(IMX,3,3),ZJ(IMX,3,3)
      COMMON/COM9/SP(IMX),SPPL(IMX),DAP(IMX),DAPPL(IMX),
     C            REDGE,MC1,MC2
      COMMON/COM10/CLAM(2),CT(2),TN(2),CKN(2),CLAMN(2),CTN(2),TNN(2)
      COMMON/COM11/NFILT,INTER,NGSTEPS,MGPLS,GEOM,QFUN,NFLAG
      COMMON/COM12/FSMACH,ALPHA,RE,TIME,NITG,NITQ
      COMMON/COM13/ISTEP,JSTEP,KSTEP,IINVERSE,JINVERSE,KINVERSE,TWOD
      COMMON/COM14/IJPLANE,IKPLANE,JKPLANE,MARCH,MARCHPL,SAVE,OK
      COMMON/COM15/CONV,MAXITS,NOUP,LNSING,PLSING,BR1
      COMMON/COM16/ORTHS,ORTHE
      COMMON/COM17/FQ(IMX),FG(IMX),FGS(IMX),SMSS(IMX),NOFG,FGN,FQN
      COMMON/COM18/SUB,LSTSUB,LENDSUB
      COMMON/COM19/ADD,LSTADD,LENDADD
      COMMON/COM20/COSEK(IMX,3),COSUK(IMX,3),COSBK(IMX,3)
      LOGICAL GEOM,QFUN,NOUP,ORTHS,ORTHE,NOMORE
      LOGICAL ISTEP,JSTEP,KSTEP,IINVERSE,JINVERSE,KINVERSE,TWOD
      LOGICAL IJPLANE,IKPLANE,JKPLANE,MARCH,MARCHPL,SAVE,OK
      INTEGER ADD,SUB,PLSING
      logical debug
      integer oldmgsteps, oldmgpls, oldmgl, oldmg2

      debug = .true.
      nads=1
c
c save original values of mgsteps, mgpls and reset them before leaving
c
      oldmgsteps = mgsteps
      oldmgpls   = mgpls
      oldmgl     = mgl
      oldmg2     = mg2
C
C START ADAPTIONS
C
c     DO 200 NADS=1,10
C
C READ CONTROL CARDS AND INITIALISE,
C 'NOMORE' INDICATES REQUESTED ADAPTIONS COMPLETE
C
      call priVar()
      CALL INITIAL(NOMORE,NADS)
      IF(NOMORE) GO TO 900
      IF(NOUP) GO TO 150
C
C START MAIN SOLUTION LOOP. ADAPT EACH LINE IN A PLANE
C AND THEN STEP TO NEXT PLANE
C
      DO 125 K=KST,KEND
       IF(K.NE.KST) CALL TORCOF(2,K,KST,KEND,MGPLS,MARCHPL)
       DO 100 J=JST,JEND
        IF ((J.EQ.JST.AND.MGSTEPS.NE.0) .OR.
     C      (J.EQ.JST  .AND.K.NE.KST.AND.LNSING.EQ.JST) .OR.
     C      (K.EQ.KST  .AND. MGPLS.NE.0)) THEN
         CALL NOADAPT(J,K)
         GO TO 100
        END IF
        CALL BLOCK(J,K)
        CALL FBAR(J,K)
        IF(J.EQ.JST.AND.K.EQ.KST) THEN
         CALL LINE1
         IF(PLSING.EQ.KST) THEN
          CALL SINGPLN
          GO TO 125
         END IF
        ELSE
         IF(J.NE.JST) CALL SETUPJ(J,K)
         IF(K.NE.KST) CALL SETUPK(J,K)
         CALL TORCOF(1,J,JST,JEND,MGSTEPS,MARCH)
         CALL TORSION(J,K)
         CALL SOLUT(J,K)
```

```
        END IF
        CALL UPDATE(J,K)
90      CONTINUE
        IF(.NOT. OK)  THEN
         CALL OUTPUT(NADS)
         GO TO 900
        END IF
        IF(NEDGE.EQ.0) CALL WIEDGE(J,K)
100     CONTINUE
        IF(MARCH .AND. J.LE.JMAX) CALL MARCHJ(K)
125     CONTINUE
C
C END MAIN SOLUTION LOOP
C
150     CONTINUE
        CALL OUTPUT(NADS)
c200    CONTINUE
C
C END OF ADAPTIONS
C
c sage modified nqsteps and nqpls.  so let's reset them
c
        nqsteps = oldnqsteps
        nqpls   = oldnqpls
        nq1     = oldnq1
        nq2     = oldnq2

900     CONTINUE
c       STOP
        return
        END
C
        SUBROUTINE ADDPTS
C
C*************************************************
C
C THIS ROUTINE INCREASES THE NO. OF POINTS ALONG
C THE ADAPTION LINE IF  REQUESTED ON INPUT
C
C CALLED BY: INITIAL
C
C CALLS : NONE
C
C*************************************************
C
        PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
        common/comxyz/x(id,jd,kd),y(id,jd,kd),z(id,jd,kd)
        common/comq/q(id,jd,kd,ndim)
        COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
        COMMON/COM13/ISTEP,JSTEP,KSTEP,IINVERSE,JINVERSE,KINVERSE,TWOD
        COMMON/COM14/IJPLANE,IKPLANE,JKPLANE,MARCH,MARCHPL,SAVE,OK
        COMMON/COM19/ADD,LSTADD,LENDADD
        DIMENSION XT(IMX),YT(IMX),ZT(IMX),QT(IMX,NDIM)
        LOGICAL ISTEP,JSTEP,KSTEP,IINVERSE,JINVERSE,KINVERSE,TWOD
        LOGICAL IJPLANE,IKPLANE,JKPLANE,MARCH,MARCHPL,SAVE,OK
        INTEGER ADD
        character*50 string
C
C INITIALIZE START AND END LIMITS TO BE CONSISTENT
C WITH ADAPTION DOMAIN
C
        IF(LSTADD.EQ.0) LSTADD=IST
        IF(LENDADD.EQ.0) LENDADD=IEND
        IF(IINVERSE) THEN
         L1=LENDADD
         LENDADD=IMAX-LSTADD+1
         LSTADD=IMAX-L1+1
        END IF
        IF(LSTADD.GT.LENDADD) THEN
c         WRITE (6,1020)
c1020     FORMAT(/' NO POINTS ADDED')
         call warning("NO POINTS ADDED")
         GO TO 999
        END IF
        IMAXB=IMAX
        IMAX=IMAX+(LENDADD-LSTADD)*(ADD)
        IF(IMAX.GT.IMX) THEN
c         WRITE(6,1000)
c1000     FORMAT(/,' ADD OPTION EXCEEDS DIMENSION')
         call warning("ADD OPTION EXCEEDS DIMENSION")
        END IF
C
C INCREASE  POINTS IN REQUESTED REGION
C
        MEND=(LENDADD-LSTADD-1)*(ADD+1)+LSTADD
        DO 500 K=1,KMAX
        DO 500 J=1,JMAX
         M=LSTADD-(ADD+1)
         DO 75 I=LSTADD,LENDADD-1
           M=M+ADD+1
           XT(M)=X(I,J,K)
           YT(M)=Y(I,J,K)
           ZT(M)=Z(I,J,K)
           DO 10 N=1,NDIM
            QT(M,N)=Q(I,J,K,N)
10         CONTINUE
           XS=(X(I+1,J,K)-X(I,J,K))/(ADD+1)
           YS=(Y(I+1,J,K)-Y(I,J,K))/(ADD+1)
           ZS=(Z(I+1,J,K)-Z(I,J,K))/(ADD+1)
           DO 30 L=1,ADD
            XT(M+L)=X(I,J,K)+XS*L
            YT(M+L)=Y(I,J,K)+YS*L
            ZT(M+L)=Z(I,J,K)+ZS*L
            DO 20 N=1,NDIM
             QT(M+L,N)=Q(I,J,K,N)+L*(Q(I+1,J,K,N)-Q(I,J,K,N))/(ADD+1)
20          CONTINUE
30         CONTINUE
75      CONTINUE
C
C PUT REMAINING DATA INTO TEMPORARY  ARRAYS
C
        DO 150 I=LENDADD,IMAXB
         INTO=M+ADD+1+(I-LENDADD)
         XT(INTO)=X(I,J,K)
         YT(INTO)=Y(I,J,K)
         ZT(INTO)=Z(I,J,K)
         DO 125 N=1,NDIM
          QT(INTO,N)=Q(I,J,K,N)
125      CONTINUE
150     CONTINUE
C
C RETURN DATA IN ORIGINAL ARRAYS
C
        DO 200 I=LSTADD,IMAX
         X(I,J,K)=XT(I)
         Y(I,J,K)=YT(I)
         Z(I,J,K)=ZT(I)
         DO 175 N=1,NDIM
```



```
          Q(I,J,K,N)=QT(I,N)
175       CONTINUE
200     CONTINUE
500     CONTINUE
        IEND=IEND+(LENDADD-LSTADD)*ADD
900     CONTINUE
c        WRITE(6,1010) IMAXB,IMAX
        encode(50,1010,string) imaxb,imax
1010    FORMAT(' NO. OF PTS INCREASED FROM ',I3,' TO ',I3)
        call warning(string)
999     CONTINUE
        RETURN
        END
C
        SUBROUTINE ADDV(COSX1,COSY1,COSZ1,A1,COSX2,COSY2,COSZ2,A2,
     C                  COSX,COSY,COSZ)
C
C*************************************************************
C
C THIS ROUTINE FINDS THE DIRECTION COSINES OF THE UNIT VECTOR
C OF THE SUM OF 2 VECTORS
C A1,A2 ARE MULTIPLIERS (TO ENABLE PROPORTIONS TO BE ADDED)
C
C CALLED BY: SETUP,TORSION,VMERGE
C
C CALLS: NONE
C
C*************************************************************
C
        COSX=A1*COSX1+A2*COSX2
        COSY=A1*COSY1+A2*COSY2
        COSZ=A1*COSZ1+A2*COSZ2
        VMOD=SQRT(COSX**2+COSY**2+COSZ**2)
        IF(VMOD.NE..0) THEN
         COSX=COSX/VMOD
         COSY=COSY/VMOD
         COSZ=COSZ/VMOD
        END IF
        RETURN
        END
C
        SUBROUTINE BLOCK(J,K)
C
C*****************************************************************
C
C THIS ROUTINE STORES THE BLOCK OF THE GRID AROUND THE CURRENT J LINE
C IE PLANES K-1,K,K+1 AND LINES J-1,J,J+1 FOR ALL I
C THIS BLOCK IS USED FOR THE MAJOR CALCULATIONS, MOST SPECIFICALLY
C FOR THE NORMALS: IT PREVENTS THE ORIGINAL GRID FROM BEING CONTAMINATED
C WHEN PROJECTIONS ARE REQUIRED.
C
C CALLED BY: MAIN
C
C CALLS: NONE
C
C*****************************************************************
C
        PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
        common/comxyz/x(id,jd,kd),y(id,jd,kd),z(id,jd,kd)
        common/comq/q(id,jd,kd,ndim)
        COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
        COMMON/COM8/XJ(IMX,3,3),YJ(IMX,3,3),ZJ(IMX,3,3)
        K1ST=1
        K1END=3
        J1ST=1
        J1END=3
        IF(J.EQ.1) THEN
         J1ST=2
         DO 20 K1=1,3
         DO 20 I=1,NIPTS
          XJ(I,1,K1)=999.0
20       CONTINUE
        END IF
        IF(J.EQ.JMAX) THEN
         J1END=2
         DO 30 K1=1,3
         DO 30 I=1,NIPTS
          XJ(I,3,K1)=999.0
30       CONTINUE
        END IF
        IF(K.EQ.1) THEN
         K1ST=2
         DO 50 J1=1,3
         DO 50 I=1,NIPTS
          XJ(I,J1,1)=999.0
50       CONTINUE
        END IF
        IF(K.EQ.KMAX) THEN
         K1END=2
         DO 70 J1=1,3
         DO 70 I=1,NIPTS
          XJ(I,J1,3)=999.0
70       CONTINUE
        END IF
        DO 100 K1=K1ST,K1END
        DO 100 J1=J1ST,J1END
        DO 100 I=1,NIPTS
         LI=IST+I-1
         LJ=J-2+J1
         LK=K-2+K1
         XJ(I,J1,K1)=X(LI,LJ,LK)
         YJ(I,J1,K1)=Y(LI,LJ,LK)
         ZJ(I,J1,K1)=Z(LI,LJ,LK)
100     CONTINUE
        RETURN
        END
C
        SUBROUTINE CROSSV(XT,YT,ZT,XT1,YT1,ZT1,DST,COSV,AAP,DAP,ICROSS,J)
C
C*****************************************************************
C
C THIS ROUTINE FINDS INTERSECTION OF A VECTOR, T (FROM LINE J-1)
C AND J LINE SEGMENT. COSV IS VECTOR COSINE OF T.
C COMPUTES DAP,AAP AND ICROSS
C
C CALLED BY: TORSION
C
C CALLS: UNITV
C
C*****************************************************************
C
        PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
        common/comxyz/x(id,jd,kd),y(id,jd,kd),z(id,jd,kd)
        common/comq/q(id,jd,kd,ndim)
        COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
        COMMON/COM11/NFILT,INTER,NGSTEPS,NGPLS,GEOM,QFUN,NFLAG
        LOGICAL GEOM,QFUN
        DIMENSION SSX(IMX),SSY(IMX),SSZ(IMX),AAP(IMX),DAP(IMX),ICROSS(IMX)
```

```
      DIMENSION XT(IMX),YT(IMX),ZT(IMX),XT1(IMX),YT1(IMX),ZT1(IMX)
      DIMENSION DST(IMX),COSV(IMX,3)
C
C FIND S SEGMENT ALONG J LINE
C
      DO 50 I=1,NIPTS-1
       CALL UNITV(XT(I),YT(I),ZT(I),XT(I+1),YT(I+1),ZT(I+1),
     C            SSX(I),SSY(I),SSZ(I))
50    CONTINUE
C
C FIND DAP,AAP LENGTHS THAT DEFINE S'
C
      ICROSS(1)=1
      DO 200 I=2,NIPTS-1
       DO 120 L=ICROSS(I-1),NIPTS-1
C
C FIND AD VECTOR AND N, NORMAL TO PLANE
C
        CALL UNITV(XT1(I),YT1(I),ZT1(I),XT(L),YT(L),ZT(L),DAX,DAY,DAZ)
        CALL PURPLE(SSX(L),SSY(L),SSZ(L),DAX,DAY,DAZ,
     C              DNX,DNY,DNZ,NFLAG)
        V1=-COSV(I,1)
        V2=-COSV(I,2)
        V3=-COSV(I,3)
        ADX=XT1(I)-XT(L)
        ADY=YT1(I)-YT(L)
        ADZ=ZT1(I)-ZT(L)
        CALL DETERM(SSX(L),V1,DNX,SSY(L),V2,DNY,SSZ(L),V3,DNZ,BIGD)
        IF(BIGD.EQ..0) GO TO 120
        CALL DETERM(ADX,V1,DNX,ADY,V2,DNY,ADZ,V3,DNZ,AAD)
        AAP(I)=AAD/BIGD
        IF(AAP(I).LT..0) AAP(I)=.0
        IF(AAP(I).LT.DST(L)) THEN
C
C AAP FOUND: NOW FIND DAP
C
         DAPX=AAP(I)*SSX(L)-ADX
         DAPY=AAP(I)*SSY(L)-ADY
         DAPZ=AAP(I)*SSZ(L)-ADZ
         DAP(I)=SQRT(DAPX**2+DAPY**2+DAPZ**2)
         ICROSS(I)=L
         GO TO 199
        END IF
120    CONTINUE
       ICROSS(I)=ICROSS(I-1)
       DAP(I)=DAP(I-1)
199    CONTINUE
       IF(DAP(I).LE. .0) DAP(I)=DAP(I-1)
200   CONTINUE
      RETURN
      END
C
      SUBROUTINE CSPLIN(NT,S,V,SPF)
C
C*************************************************************
C
C THIS ROUTINE  EVALUATES SPLINE COEFFICIENTS (SPF) USED BY SPEVAL
C THESE ARE REQUIRED ONLY WHEN THE GEOMTRY OPTION IS REQUESTED (GEOM=T)
C OR IF SPLINE INTERPOLATION REQUESTED (INTER=4)
C
C CALLED BY: INTF,INTXYEQ,WALLS
C
C CALLS: NONE
C
C
C CALLS: NONE
C
C*************************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      DIMENSION S(IMX),V(IMX),A(IMX),B(IMX),C(IMX),SPF(IMX)
      DO 10 I = 2,NT-1
       A(I)   = S(I)-S(I-1)
       B(I)   = 2.*(S(I+1)-S(I-1))
       C(I)   = S(I+1)-S(I)
       SPF(I) = 6.*((V(I+1)-V(I))/C(I)-(V(I)-V(I-1))/A(I))
   10 CONTINUE
C
C..... BOUNDARY CONDITIONS
C
      A(1)    = 0.
      B(1)    = 1.
      C(1)    = -.5
      SPF(1)  = 0.
      A(NT)   = -.5
      B(NT)   = 1.
      C(NT)   = 0.
      SPF(NT) = 0.
C
C..... SOLVE THE SCALAR TRIDIAGONAL SYSTEM
C
      C(1)   = C(1)/B(1)
      SPF(1) = SPF(1)/B(1)
      DO 40 I = 2,NT
       B(I)   = 1./(B(I)-A(I)*C(I-1))
       SPF(I) = (SPF(I)-A(I)*SPF(I-1))*B(I)
       C(I)   = C(I)*B(I)
   40 CONTINUE
      DO 50 I = NT-1,1,-1
       SPF(I) = SPF(I)-C(I)*SPF(I+1)
   50 CONTINUE
      RETURN
      END
C
      SUBROUTINE DETERM(A1,B1,C1,A2,B2,C2,A3,B3,C3,DET)
C
C*************************************************************
C
C THIS ROUTINE COMPUTES THE 3-ORDER DETERMINANT
C
CALLED BY: CROSSV
C
CALLS: UNITV
C
C*************************************************************
C
      CROSSA=B2*C3-C2*B3
      CROSSB=A2*C3-C2*A3
      CROSSC=A2*B3-B2*A3
      DET=A1*CROSSA-B1*CROSSB+C1*CROSSC
      RETURN
      END
C
      SUBROUTINE DLENG(JL,K)
C
C*************************************************************
C
C THIS ROUTINE FINDS THE EDGE MULTIPLIERS FOR EDGE TREATMENT
C WHEN NEDGE NOT ZERO
C
C CALLED BY: WTEDGE
```



```
C
C CALLS: NONE
C
C*************************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      common/commxyz/x(id,jd,kd),y(id,jd,kd),z(id,jd,kd)
      common/commq/q(id,jd,kd,ndim)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM6/DSMAX,DSMIN,WT(IMX),DLENGS,DLENGE,WDSS,WDES
C
C CALC OF DLENG DEPENDS ON WHETHER DATA EXISTS EXTERNAL TO
C ADAPTION DOMAIN
C
C  (1) AT FIRST EDGE POINT
C
      IF(IST.EQ.1 .OR. (X(IST-1,JL,K).EQ .0  .AND.
     C  ( X(IST-1,JL-1,K).EQ .0 .OR. X(IST-1,JL+1,K).EQ..0))) THEN
        DLENGS=SQRT((X(IST+1,JL,K)-X(IST,JL,K))**2+
     C              (Y(IST+1,JL,K)-Y(IST,JL,K))**2+
     C              (Z(IST+1,JL,K)-Z(IST,JL,K))**2)
       ELSE
        DLENGS=SQRT((X(IST,JL,K)-X(IST-1,JL,K))**2+
     C              (Y(IST,JL,K)-Y(IST-1,JL,K))**2+
     C              (Z(IST,JL,K)-Z(IST-1,JL,K))**2)
       END IF
C
C  (2)  AT LAST EDGE POINT
C
      IF(IEND.EQ.IMAX .OR. (X(IEND+1,JL,K).EQ .0  .AND.
     C  (X(IEND+1,JL-1,K).EQ .0 .OR. X(IEND+1,JL+1,K).EQ..0))) THEN
        DLENGE=SQRT((X(IEND,JL,K)-X(IEND-1,JL,K))**2+
     C              (Y(IEND,JL,K)-Y(IEND-1,JL,K))**2+
     C              (Z(IEND,JL,K)-Z(IEND-1,JL,K))**2)
       ELSE
        DLENGE=SQRT((X(IEND+1,JL,K)-X(IEND,JL,K))**2+
     C              (Y(IEND+1,JL,K)-Y(IEND,JL,K))**2+
     C              (Z(IEND+1,JL,K)-Z(IEND,JL,K))**2)
       END IF
       RETURN
       END
C
      SUBROUTINE EDGENG(VAR)
C
C*************************************************************
C
C THIS ROUTINE MERGES THE END VALUES OF VAR INTO NEXT 'MG' MESH
C POINTS. L1 IS START EDGE (OFTEN 1), L2 IS END EDGE (OFTEN NIPTS)
C IF EITHER IS ZERO, THEN NO MERGING AT THAT END
C
C CALLED BY: LINE1,SOLUT
C
C CALLS: NONE
C
C*************************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM9/SP(IMX),SPPL(IMX),DAP(IMX),DAPPL(IMX),
     C            NEDGE,MG1,MG2
      DIMENSION VAR(IMX)
      IF(MG1.NE.0) THEN
       DO 150 I=2,MG1
        VAR(I)=(VAR(I)*(I-1)+VAR(1)*(MG1+1-I))/MG1
150    CONTINUE
       END IF
       IF(MG2.NE.0) THEN
        DO 200 I=2,MG2
         L=NIPTS-I
         VAR(L)=(VAR(L)*(I-1)+VAR(NIPTS-1)*(MG2+1-I))/MG2
200    CONTINUE
       END IF
       RETURN
       END
C
      SUBROUTINE FBAR(J,K)
C
C*************************************************************
C
C THIS ROUTINE COMPUTES THE FLOWFIELD GRADIENTS, (FQ);
C THE GEOMETRY GRADIENTS, (FG);
C NORMALIZES (TO FBAR); FINDS B BY CALLING GETB;
C COMPUTES S FROM X,Y; FINDS MIDPTS OF S;
C AND REPROPORTIONS S FOR INITIAL GUESS WITHIN DATA BLOCK
C AND FINALLY INTERPOLATES FOR X,Y,FB ETC AT NEW S
C
C CALLED BY: MAIN
C
C CALLS: INTF,WALLS,MGWALLS,NORM,GETB,FILTER,PROPS
C
C*************************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      common/commxyz/x(id,jd,kd),y(id,jd,kd),z(id,jd,kd)
      common/commq/q(id,jd,kd,ndim)
      COMMON/COM2/F(IMX),FB(IMX),WEIGHT(IMX),A,B,WDS,WDE
      COMMON/COM3/SS(IMX),SMS(IMX),DS(IMX),SN(IMX),SNM(IMX),SNME(IMX)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM5/RDSMAX,RDSMIN,INDQ,IQ(NDIM+3)
      COMMON/COM6/DSMAX,DSMIN,WT(IMX),DLENGS,DLENGE,WDSS,WDES
      COMMON/COM8/XJ(IMX,3,3),YJ(IMX,3,3),ZJ(IMX,3,3)
      COMMON/COM11/NFILT,INTER,NGSTEPS,NGPLS,GEOM,QFUN,NFLAG
      COMMON/COM17/FQ(IMX),FG(IMX),FGS(IMX),SMSS(IMX),MXFG,FGN,FQN
      DIMENSION AMACH(IMX),PRES(IMX),TRAT(IMX),FDQ(IMX)
      LOGICAL GEOM,QFUN
      GAM=1.4
C
C FIND S ARRAY (SS AND SMS ARE EVALUATED AT INPUT POINTS
C AND STAY FIXED FOR EACH LINE-USED FOR INTERPOLATION)
C SN IS UPDATED S ARRAY, SNMK IS S ARRAY AT K-1
C
      SS(1)=.0
      SNMK(1)=.0
      DO 100 I=1,NIPTS-1
       L=IST+I-1
       DS(I)=SQRT((X(L+1,J,K)-X(L,J,K))**2+
     C            (Y(L+1,J,K)-Y(L,J,K))**2+
     C            (Z(L+1,J,K)-Z(L,J,K))**2)
       SS(I+1)=SS(I)+DS(I)
       IF(K.GT.KST) THEN
        DSNMK=SQRT((X(L+1,J,K-1)-X(L,J,K-1))**2+
     C             (Y(L+1,J,K-1)-Y(L,J,K-1))**2+
     C             (Z(L+1,J,K-1)-Z(L,J,K-1))**2)
        SNMK(I+1)=SNMK(I)+DSNMK
       ELSE
        SNMK(I+1)=.0
       END IF
100    CONTINUE
```

```
      DO 200 I=1,NIPTS-1
       SMS(I)=(SS(I+1)+SS(I))*.5
200   CONTINUE
      AVEDS=SS(NIPTS)/(NIPTS-1)
      DSMAX=RDSMAX*AVEDS
      DSMIN=RDSMIN*AVEDS
C
C FIND DQ/DS
C
      DO 300 N=1,NDIM
       DO 240 I=1,NIPTS-1
        L=I+IST-1
        FDQ(I)=(ABS(Q(L+1,J,K,N)-Q(L,J,K,N)))/DS(I)
240    CONTINUE
       CALL NORM(FDQ,NIPTS-1)
       DO 260 I=1,NIPTS-1
        IF(N.EQ.1) FQ(I)=.0
        FQ(I)=FQ(I)+IQ(N)*FDQ(I)
260    CONTINUE
300   CONTINUE
C
C    EVALUATE PRESSURE AND MACH NUMBER IF USED
C (THESE ARE NORMALLY STORED IN INDEX 6 & 7, BUT CODE
C  IS SET THIS WAY FOR USERS WHO INCREASE NDIM FOR
C  MORE Q VARIABLES)
C
      IF(IQ(NDIM+1).NE.0 .OR. IQ(NDIM+2).NE.0 .OR.
     C   IQ(NDIM+3).NE.0) THEN
       DO 350 I=1,NIPTS
        L=I+IST-1
        P1=(Q(L,J,K,2)**2+Q(L,J,K,3)**2+Q(L,J,K,4)**2)/Q(L,J,K,1)
        PRES(L)=(GAM-1.0)*(Q(L,J,K,5)-.5*P1)
        AMACH(L)=SQRT(P1/(GAM*PRES(L)))
        TRAT(L)=GAM*PRES(L)/Q(L,J,K,1)
350    CONTINUE
       DO 400 I=1,NIPTS-1
        L=I+IST-1
        FDQP=ABS(PRES(L+1)-PRES(L))
        FDQAM=ABS(AMACH(L+1)-AMACH(L))
        FDQT=ABS(TRAT(L+1)-TRAT(L))
        FQ(I)=FQ(I)+(IQ(NDIM+1)*FDQP+IQ(NDIM+2)*FDQAM
     C         +IQ(NDIM+3)*FDQT)/DS(I)
400    CONTINUE
      END IF
C
C FIND DG/DS, GEOMETRY GRADIENTS
C
      IF(GEOM) THEN
       IF(J.EQ.JST .OR. .NOT. QFUN) CALL WALLS(J,K)
       IF(QFUN.AND.J.EQ.((JEND-JST)/2+1)) CALL WALLS(JEND,K)
C
C FIND FGW, COEFFICIENT OF GEOMETRY FUNCTION
C
       IF (.NOT. QFUN) THEN
        FGW=.0
        FGW=1.0
       ELSE
        CALL NGWALLS(J)
        IF(FGW.EQ..0) GO TO 460
       END IF
C
C INTERP FG (USING FGS COMPUTED IN WALLS)
C AND NORMALISE FQ AND FG
C
       CALL INTF(FGS,FG,SS,SMSS,NIPTS-1)
       CALL NORM(FG,NIPTS-1)
       CALL NORM(FQ,NIPTS-1)
      END IF
C
C COMPUTE F
C
460   CONTINUE
      DO 475 I=1,NIPTS-1
       F(I)=FQW*FQ(I)+FGW*FG(I)
475   CONTINUE
C
C IF REQUIRED, FILTER F
C
      IF(NFILT.GT.0) CALL FILTER(F,NIPTS,NFILT)
C
C F IS NOW FINALIZED AND EVALUATED AT SS
C STORE F IN FB BEFORE NORMALIZING (USE INTF)
C
      CALL INTF(F,FB,SS,SMS,NIPTS-1)
      CALL NORM(FB,NIPTS-1)
C
C FIND B USING DATA EVALUATED AT INPUT NODES
C
      CALL GETB(J,K)
C
C INITIALISE SN ARRAY - IF 1ST TIME,=SS, ELSE PROPORTION FROM
C CONVERGED SOLUTION AT J-1 LINE AND RE-EVALUATE F,DS
C
      IF(J.EQ.JST.AND.K.EQ.KST) THEN
       DO 520 I=1,NIPTS
        SN(I)=SS(I)
520    CONTINUE
      ELSE
       CALL PROPS(J,K)
      END IF
      DO 600 I=1,NIPTS-1
       DS(I)=SN(I+1)-SN(I)
600   CONTINUE
C
C INTERPOLATE FOR F AT NEW S, AND COMPUTE FB
C
      CALL INTF(F,FB,SN,SMS,NIPTS-1)
      CALL NORM(FB,NIPTS-1)
      RETURN
      END
C
      SUBROUTINE FILTER(VAR,NIPTS,NFILT)
C
C*************************************************************
C
C     THIS SUBROUTINE FILTERS (SMOOTHS) GIVEN VARIABLE
C
C CALLED BY: FBAR,SOLUT,LINE1
C
C CALLS: NONE
C
C*************************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      DIMENSION VAR(IMX),VT(IMX)
C
      DO 200  L=1,NFILT
       DO 100 I=2,NIPTS-2
```



```
        VT(I)=.75*VAR(I)+.125*(VAR(I+1)+VAR(I-1))
100    CONTINUE
       DO 150 I=2,NIPTS-2
        VAR(I)=VT(I)
150    CONTINUE
200   CONTINUE
      RETURN
      END
C
      SUBROUTINE GETB(J,K)
C
C*************************************************************
C
C THIS ROUTINE FINDS B BY ITERATION, USING INITIAL GRID SPACING.
C B CONVERGES WHEN INPUT DS MIN=COMPUTED DS MIN.
C B CONTROLS THE MINIMUM ALLOWED S MESH SPACING
C
C CALLED BY: FBAR
C
C CALLS: INTF
C
C*************************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      COMMON/COM2/F(IMX),FB(IMX),WEIGHT(IMX),A,B,NDS,NDE
      COMMON/COM3/SS(IMX),SMS(IMX),DS(IMX),SN(IMX),SNM(IMX),SNMX(IMX)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM6/DSMAX,DSMIN,WT(IMX),DLENGS,DLENGE,NDSS,NDES
      COMMON/COM15/CONV,MAXITS,NOOP,LWSING,PLSING,BK1
      DIMENSION ALWF(IMX),FINT(IMX),WIT(IMX)
      DIMENSION SNB(IMX),DSB(IMX)
      LOGICAL NOOP
      INTEGER PLSING
C
C IF FIRST LINE SET INITIAL GUESS OF B=1.0
C AND PERMIT MORE ITERATIONS FOR CONVERGENCE
C OTHERWISE FIRST GUESS ON B IS THE CURRENT VALUE FROM
C THE J-1 LINE (OR IF JST, THEN K-1 LINE)
C
      MAXBITS=MAXITS
      IF(J.EQ.JST) THEN
       IF (K.EQ.KST) THEN
        B=1.0
        MAXBITS=MAXITS+20
       ELSE
        B=BK1
       END IF
      END IF
      IF(J.EQ.JST+1) BK1=B
      BJ1=B
      IF(A.EQ..0) GO TO 999
      DB=.0
      SNB(1)=.0
C
C CONVERGENCE OF DSMIN MUST BE BETTER THAN
C REQUESTED DSMIN, HENCE BCONV IS FUNCTION OF DSMIN
C
      BCONV=DSMIN*.02
C
C ITERATE TO FIND CORRECT B
C
      DO 500 ITER=1,MAXBITS
C
C COMPUTE WEIGHT BASED ON CURRENT B
C USING INITIAL GRID SPACING
C
      DO 100 L=1,NIPTS-1
       WEIGHT(L)=1.0+A*FB(L)**B
100   CONTINUE
C
C FIND NEW DS, USING THIS B
C
      WTSUM=.0
      DO 120 I=1,NIPTS-1
       WTSUM=WTSUM+1.0/WEIGHT(I)
120   CONTINUE
      DO 130 I=1,NIPTS-1
       DSB(I)=SS(NIPTS)/(WEIGHT(I)*WTSUM)
       SNB(I+1)=SNB(I)+DSB(I)
130   CONTINUE
C
C FIND MIN VALUE OF DS
C
      DSIMIN=DSB(1)
      DO 150 L=2,NIPTS-1
       IF(DSB(L).LT.DSIMIN) DSIMIN=DSB(L)
150   CONTINUE
C
C TEST FOR CONVERGENCE
C
      TSTCONV=ABS(DSMIN-DSIMIN)
      IF(TSTCONV .LE. BCONV) GO TO 999
C
C NO CONVERGENCE, COMPUTE DB FOR NEXT ITERATION
C  (1) FIND NEW FB AND W AT THIS NEW SPACING
C
       CALL INTF(WEIGHT,WIT,SNB,SMS,NIPTS-1)
       CALL INTF(FB,FINT,SNB,SMS,NIPTS-1)
180   CONTINUE
C
C (2) FIND DERIV OF DXIMIN WRT B
C
      SUM=.0
      DO 200 L=1,NIPTS-1
       ALWF(L)=LOG(FINT(L))
       SUM=SUM+FINT(L)**B*ALWF(L)/WIT(L)**2
200   CONTINUE
      DMINSDB=A*(1.0+A)*DSIMIN**2*SUM/SS(NIPTS)
      IF(DMINSDB.EQ..0) GO TO 998
      DB=(DSMIN-DSIMIN)/DMINSDB
      B=B+DB
C
C MAINTAIN REASONABLE LIMIT FOR DB, FIRST ITERATION
C MAY BE EXTREME
C
      IF(B.GT.5.0) B=3.0
      IF(B.LT..0) B=.1
300   CONTINUE
C
C NO CONVERGENCE ON B, USE OLD VALUE
C
998   CONTINUE
      IF(J.EQ.JST) THEN
       B=1.0
      ELSE
       B=BJ1
      END IF
999   CONTINUE
```

```
      RETURN
      END
c
      SUBROUTINE GETWT
c
C*******************************************************
c
C THIS ROUTINE DETERMINES THE MULTIPLIER
C OF THE WEIGHT CONSTANTS (WT). THESE ARE AN ADDITIONAL
C CONTROL ON THE MAX AND MIN ALLOWABLE DS'S. AM IS A
C VARIABLE THAT PREVENTS A 'FLIP-FLOP' CONDITION.
c
C CALLED BY: LINE1,SOLUT
c
C CALLS: NONE
c
C*******************************************************
c
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      COMMON/COM3/SS(IMX),SMS(IMX),DS(IMX),SN(IMX),SNM(IMX),SNMX(IMX)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM6/DSMAX,DSMIN,WT(IMX),DLENGS,DLENGE,WDSS,WDES
      COMMON/COM9/SP(IMX),SPPL(IMX),DAP(IMX),DAPPL(IMX),
     C            NEDGE,MG1,MG2
c
      AM=.5
      DO 50 I=1,NIPTS-1
       WT(I)=1.0
50    CONTINUE
c
C IF NEDGE SET, PERMIT DS TO BE TOO SMALL
c
      I1=1
      IF(MG1.NE.0) I1=MG1
      I2=NIPTS-1
      IF(MG2.NE.0) I2=NIPTS-MG2
      DO 100 I=1,NIPTS-1
       IF(DS(I).GT.DSMAX) THEN
        DT=DS(I)/DSMAX
        WT(I)=(DT-1.0)*AM+1.0
       END IF
100   CONTINUE
      DO 200 I=I1,I2
       IF(DS(I).LT.DSMIN) THEN
        DT=DS(I)/DSMIN
        WT(I)=(DT-1.0)*AM+1.0
       END IF
200   CONTINUE
      RETURN
      END
c
      SUBROUTINE INITIAL(NOMORE,NADS)
c
C**********************************************
c
C THIS ROUTINE READS THE INPUT CONTROL VARIABLES,
C INITIALISES DEFAULTS AND SWAPS DATA IN
C X,Y ARRAYS IF NECESSARY
c
C CALLED BY: MAIN
c
C CALLS: ADDPTS,SUBPTS,SWAPXYZ,SWAPINV
c
C**********************************************
c
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      common/comxyz/x(id,jd,kd),y(id,jd,kd),z(id,jd,kd)
      common/comq/q(id,jd,kd,ndim)
      COMMON/COM2/F(IMX),FB(IMX),WEIGHT(IMX),A,B,WDS,WDE
      COMMON/COM3/SS(IMX),SMS(IMX),DS(IMX),SN(IMX),SNM(IMX),SNMX(IMX)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM5/RDSMAX,RDSMIN,INDQ,IQ(NDIM+3)
      COMMON/COM6/DSMAX,DSMIN,WT(IMX),DLENGS,DLENGE,WDSS,WDES
      COMMON/COM9/SP(IMX),SPPL(IMX),DAP(IMX),DAPPL(IMX),
     C            NEDGE,MG1,MG2
      COMMON/COM10/CLAM(2),CT(2),TN(2),CNM(2),CLAMW(2),CTM(2),TNM(2)
      COMMON/COM11/NFILT,INTER,MGSTEPS,MGPLS,GEOM,QFUN,NFLAG
      COMMON/COM12/FSMACH,ALPHA,RE,TIME,NITG,NITO
      COMMON/COM13/ISTEP,JSTEP,KSTEP,IINVERSE,JINVERSE,KINVERSE,TWOD
      COMMON/COM14/IJPLANE,IKPLANE,JKPLANE,MARCH,MARCHPL,SAVE,OK
      COMMON/COM15/CONV,MAXITS,NOUP,LNSING,PLSING,BK1
      COMMON/COM16/ORTHS,ORTHE
      COMMON/COM17/FQ(IMX),FC(IMX),FGS(IMX),SMSS(IMX),MXFC,FGN,FQN
      COMMON/COM18/SUB,LSTSUB,LENDSUB
      COMMON/COM19/ADD,LSTADD,LENDADD
      LOGICAL ISTEP,JSTEP,KSTEP,IINVERSE,JINVERSE,KINVERSE,TWOD
      LOGICAL IJPLANE,IKPLANE,JKPLANE,MARCH,MARCHPL,SAVE,OK
      LOGICAL GEOM,QFUN,NOMORE,NOUP,ORTHS,ORTHE,RSWAP
      INTEGER ADD,SUB,PLSING,REMOVE
c
CIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
c                                                                  I
C IST,IEND=FIRST AND LAST ADAPTION POINTS IN I DIRECTION           I
C JST,JEND= ------------------------------ J ---------             I
C KST,KEND=------------------------------- K ---------             I
C ISTEP,JSTEP,KSTEP=TRUE FOR  STEPPING DIRECTION                   I
C IJPLANE,IKPLANE,JKPLANE=TRUE FOR MARCHING PLANE                  I
C RDSMIN,RDSMAX=MIN AND MAX ALLOWABLE GRID SPACINGS                I
C CLAM(2)=MAGNITUDE OF TORSION(1=IJPLANE,2=IKPLANE)                I
C NEDGE=EDGE SPACING CONTROL,1=BOTH,2=ST,3=END                     I
C CT(2)=PROP. OF STRAIGHTNESS TO ORTHOGONALITY,1=STRAIGHT,0=NORMAL I
C MGSTEPS=NO OF STEPS BEFORE FULL ADAPTION REQUIRED                I
C MGPLS=NO OF PLANES BEFORE FULL ADAPTION OCCURS                   I
C GEOM=TRUE, INCLUDE GEOMETRY GRADIENTS CLOSE TO WALL              I
C QFUN=FALSE-USED ONLY WHEN GEOM=TRUE AND ONLY ADAPT TO GEOMETRY   I
C INDQ=INDEX ON FLOWFIELD VARIABLE                                 I
C IQ(NDIM+3)=IF INDQ=0, PROPORTION OF FLOWFIELD VARIABLES          I
C NOUP=TRUE IF NO ADAPTION REQUIRED (EG IF ADDING PTS ONLY)        I
C MARCH=TRUE FOR EXTRAPOLATING LAST ADAPTED LINE (IF JST  LT JMAX) I
C MARCHPL=TRUE FOR " LAST ADAPTED PLANE TO REMAINING PLANES        I
C ADD=INTEGER FOR ADDING POINTS IN ADAPTION DIRECTION              I
C LSTADD,LENDADD=RANGE WITHIN TO ADD PTS                           I
C SUB=INTEGER FOR DELETING POINTS                                  I
C LSTSUB,LENDSUB=RANGE FOR DELETING POINTS                         I
C REMOVE=NO OF PTS TO REMOVE FROM OUTER BOUNDARY OF ADAPTION LINE  I
C SAVE=FALSE TO SUPPRESS O/P OF ADAPTED FILES                      I
C ORTHS,ORTHE=FALSE TO SUPPRESS ORTHOGONALITY AT WALL BOUNDARIES   I
C NFILT=INDEX ON AMOUNT OF FILTERING OF RAW DATA                   I
C INTER=ORDER OF INTERPOLATION                                     I
C LNSING=N, SAME LINE IN EACH PLANE: ADAPT ONCE ONLY               I
C PLSING=N, PLANE COLLAPSED TO A LINE                              I
C MG1,MG2=CHANGES NO. OF POINTS IN NEDGE CONTROL                   I
C TWOD=.true. FOR 2-D DATASETS                                     I
c                                                                  I
CIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
c
      NAMELIST/NAMEL/IST,IEND,JST,JEND,KST,KEND,RDSMAX,RDSMIN,
     C ISTEP,JSTEP,KSTEP,IJPLANE,IKPLANE,JKPLANE,
```



```
     C CLAM,NEDGE,CT,MGSTEPS,MGPLS,INDQ,IQ,NOUP,TWOD,REMOVE,
     C MARCH,MARCHPL,ADD,LSTADD,LENDADD,SUB,LSTSUB,LENDSUB,SAVE,
     C ORTHS,ORTHE,NFILT,INTER,GEOM,QFUN,LNSING,PLSING,MG1,MG2
c
C SET INPUT DEFAULT VALUES
c
c       IST=1
c       IEND=0
c       JST=1
c       JEND=0
c       KST=1
c       KEND=0
c       ISTEP=.FALSE.
c       JSTEP=.TRUE.
c       KSTEP=.FALSE.
c       IJPLANE=.TRUE.
c       JKPLANE=.FALSE.
c       IKPLANE=.FALSE.
c       RDSMAX=2.0
c       RDSMIN=.5
c       CLAM(1)=.01
c       CLAM(2)=.0001
c       NEDGE=0
c       CT(1)=.5
c       CT(2)=.5
c       MGSTEPS=0
c       MGPLS=0
c       LNSING=0
c       PLSING=0
c       INDQ=1
c       NOUP=.FALSE.
c       MARCH=.FALSE.
c       MARCHPL=.FALSE.
c       ADD=0
c       LSTADD=0
c       LENDADD=0
c       SUB=0
c       LSTSUB=0
c       LENDSUB=0
c       SAVE=.TRUE.
c       ORTHS=.TRUE.
c       ORTHE=.TRUE.
c       NFILT=2
c       GEOM=.FALSE.
c       QFUN=.TRUE.
c       INTER=2
c       TWOD=.FALSE.
c
C SET OTHER START VALUES
c
c       OK=.TRUE.
c       FQN=1.0
c       FGN=.0
      NOMORE=.FALSE.
c       IINVERSE=.FALSE.
c       JINVERSE=.FALSE.
c       KINVERSE=.FALSE.
c       TN(1)=.5
c       TN(2)=.5
c       CONV=.001
c       MG1=0
c       MG2=0
c       MAXITS=20
      RSWAP=.FALSE.
c       NFLAG=1
c       WDS=.0
c       WDE=.0
      REMOVE=0
c
C READ USER INPUT PARAMETER FILE
c
c       READ(5,NML=NAMEL,END=230)
c       IF(IKPLANE.OR.JKPLANE) IJPLANE=.FALSE.
c       IF(ISTEP.OR.KSTEP) JSTEP=.FALSE.
c       IF(IJPLANE.AND.KSTEP .OR. IKPLANE.AND.JSTEP
c      C   .OR. JKPLANE.AND.ISTEP) THEN
c        WRITE(6,1000)
c1000    FORMAT('  INCONSISTENT PLANE AND STEP')
c        GO TO 230
c       END IF
c
C READ GRID AND FUNCTION FILES
c
      IF (NADS.EQ.1) THEN
c       CALL READAT
       CALL SIZING(IER)
       IF (IER.EQ.1) GO TO 230
      END IF
      IF (IEND.EQ.0) IEND=IMAX
      IF (JEND.EQ.0) JEND=JMAX
      IF (KEND.EQ.0) KEND=KMAX
c
C SET UP INDICES FOR EDGE TREATMENT, BASED ON NEDGE
c
      IF((NEDGE.EQ.1 .OR. NEDGE.EQ.2) .AND.MG1.EQ.0) MG1=4
      IF((NEDGE.EQ.1 .OR. NEDGE.EQ.3) .AND.MG2.EQ.0) MG2=4
c
C SWAP COORDINATES IF NEC
C (CODE ALWAYS ADAPTS 'I' STEPPING IN 'J' LINES AND 'K' PLANES)
c
      IF (TWOD) CALL SWAP2D(1)
      IF(.NOT.(IJPLANE .AND. JSTEP)) CALL SWAPXYZ(RSWAP)
c
C IF IST>IEND OR JST>JEND, SWAP DATA
C (ANALYSIS ASSUMES I AND J INDICES ALWAYS INCREASE)
c
      IF(IST.GT.IEND) IINVERSE=.TRUE.
      IF(JST.GT.JEND) JINVERSE=.TRUE.
      IF(KST.GT.KEND) KINVERSE=.TRUE.
      IF(JST.EQ.JEND .AND. JST.EQ.JMAX) JINVERSE=.TRUE.
      IF(IINVERSE.OR.JINVERSE.OR. KINVERSE) CALL SWAPINV
c
C IF ADD IS SET, DENSE UP POINTS IN THE ADAPTED DIRECTION
c
      IF(ADD.NE.0) CALL ADDPTS
c
C IF SUB IS SET, DELETE POINTS IN ADAPTION DIRECTION
c
      IF(SUB.NE.0) CALL SUBPTS
c
C REMOVE ANY POINTS FROM OUTER BOUNADRY
c
      IMAX=IMAX-REMOVE
      IF(IEND.GT.IMAX) IEND=IMAX
c
C COMPUTE NUMBER OF PTS IN THE ADAPTIVE SECTION
c
      NIPTS=IEND-IST+1
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
C
C COMPUTE MESH SIZE CONSTANT A
C
      A=RDSMAX/RDSMIN-1.0
C
C IF IQ NOT INPUT, INITIALIZE TO
C ADAPTION VARIABLE REQUESTED (INDQ80)
C
      IF(INDQ.NE.0) THEN
       DO 50 I=1,NDIM+3
        IF(I.EQ.INDQ) THEN
         IQ(I)=1
        ELSE
         IQ(I)=0
        END IF
50     CONTINUE
      END IF
C
C CORRECT NSRGE NOS. TO BE ACTUAL VALUE
C
      IF(NGSTEPS.NE.0) NGSTEPS=NGSTEPS+JST
      IF(NGPLS.NE.0) NGPLS=NGPLS+KST
      GO TO 250
230   NOMORE=.TRUE.
250   CONTINUE
      RETURN
      END
C
      SUBROUTINE INTF(F1,F2,S1,SMID,NPTS)
C
C*****************************************************
C
C F1 IS A VARIABLE ASSOCIATED WITH THE MID-PT (SMID) OF THE
C GRID ELEMENT (EG, DERIVATIVES). THIS SUBROUTINE COMPUTES THE
C MID-PT OF THE NEW S ARRAY (S1) AND INTERPOLATES
C FOR F1, RETURNING IT IN F2
C
C CALLED BY: FBAR,SOLUT,LINE1,GETB
C
C CALLS: LAGCOF,CSPLIN,SPEVAL
C
C*****************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      COMMON/COM11/NFILT,INTER,NGSTEPS,NGPLS,GEOM,QFUN,NFLAG
      DIMENSION F1(IMX),F2(IMX),S1(IMX),SMID(IMX)
      DIMENSION SM(IMX),SPF(IMX)
      LOGICAL GEOM,QFUN
      IF(INTER.EQ.4) CALL CSPLIN(NPTS,SMID,F1,SPF)
C
C FIND NEW MIDPOINT OF S AND USE LAGRANGE
C POLYNOMIALS TO INTERPOLATE FOR F2 USING ORIGINAL F1
C
      DO 100 I=1,NPTS
       SM(I)=(S1(I+1)+S1(I))*.5
       IF(INTER.EQ.4) THEN
        CALL SPEVAL(NPTS,SMID,F1,SPF,SM(I),F2(I),DUM1,DUM2)
       ELSE
        CALL LAGCOF(SM(I),SMID,NPTS,M,P1,P2,P3)
        F2(I)=P1*F1(M)+P2*F1(M+1)+P3*F1(M+2)
       END IF
       IF(F2(I).LE..0) F2(I)=1.0E-05
100   CONTINUE
      RETURN
      END
C
      SUBROUTINE INTXYZQ(J,K,J1,K1,SS,SN,QJ)
C
C*****************************************************
C
C GIVEN X,Y,Z,Q AT SS, FIND SAME AT SN
C
C CALLED BY: PROPS,MARCHJ,MARCHK,UPDATE
C
C CALLS: LAGCOF,CSPLIN,SPEVAL
C
C*****************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      common/comxyz/x(id,jd,kd),y(id,jd,kd),z(id,jd,kd)
      common/comq/q(id,jd,kd,ndim)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM8/XJ(IMX,3,3),YJ(IMX,3,3),ZJ(IMX,3,3)
      COMMON/COM11/NFILT,INTER,NGSTEPS,NGPLS,GEOM,QFUN,NFLAG
      DIMENSION X1(IMX),Y1(IMX),Z1(IMX),X2(IMX),Y2(IMX),Z2(IMX)
      DIMENSION SPFX(IMX),SPFY(IMX),SPFZ(IMX),SPFQ(IMX,NDIM)
      DIMENSION Q1(IMX,NDIM),QJ(IMX,NDIM),SS(IMX),SN(IMX)
      LOGICAL GEOM,QFUN
C
C FIND SPLINE COEFFS IF SPLINE INTERPOLATION
C
      IF(INTER.EQ.4) THEN
       DO 20 I=1,NIPTS
        X1(I)=X(IST-1+I,J,K)
        Y1(I)=Y(IST-1+I,J,K)
        Z1(I)=Z(IST-1+I,J,K)
        DO 20 N=1,NDIM
         Q1(I,N)=Q(IST-1+I,J,K,N)
20     CONTINUE
       CALL CSPLIN(NIPTS,SS,X1,SPFX)
       CALL CSPLIN(NIPTS,SS,Y1,SPFY)
       CALL CSPLIN(NIPTS,SS,Z1,SPFZ)
       DO 30 N=1,NDIM
        CALL CSPLIN(NIPTS,SS,Q1(1,N),SPFQ(1,N))
30     CONTINUE
      END IF
C
C INTERPOLATE FOR NEW X,Y,Q AT COMPUTED SN
C
      DO 100 I=1,NIPTS
       IF(INTER.EQ.4) THEN
        CALL SPEVAL(NIPTS,SS,X1,SPFX,SN(I),X2(I),D1,D2)
        CALL SPEVAL(NIPTS,SS,Y1,SPFY,SN(I),Y2(I),D1,D2)
        CALL SPEVAL(NIPTS,SS,Z1,SPFZ,SN(I),Z2(I),D1,D2)
        DO 40 N=1,NDIM
         M=IST+I-1
         CALL SPEVAL(NIPTS,SS,Q1(1,N),SPFQ(1,N),SN(I),
     C                QJ(I,N),D1,D2)
40      CONTINUE
       ELSE
        CALL LAGCOF(SN(I),SS,NIPTS,MS,P1,P2,P3)
        M=MS+IST-1
        X2(I)=P1*X(M,J,K)+P2*X(M+1,J,K)+P3*X(M+2,J,K)
        Y2(I)=P1*Y(M,J,K)+P2*Y(M+1,J,K)+P3*Y(M+2,J,K)
        Z2(I)=P1*Z(M,J,K)+P2*Z(M+1,J,K)+P3*Z(M+2,J,K)
        DO 50 N=1,NDIM
         QJ(I,N)=P1*Q(M,J,K,N)+P2*Q(M+1,J,K,N)+P3*Q(M+2,J,K,N)
50      CONTINUE
```



```
       END IF
100   CONTINUE
C
C STORE INTO BLOCK
C
      DO 150 I=2,NIPTS-1
       XJ(I,J1,K1)=X2(I)
       YJ(I,J1,K1)=Y2(I)
       ZJ(I,J1,K1)=Z2(I)
150   CONTINUE
      RETURN
      END
C
      SUBROUTINE LAGCOF(SNEW,SARR,NPTS,M,P1,P2,P3)
C
C*****************************************************
C
C CALLED IF INTER=2 OR 3
C FINDS LAGRANGE COEFFS (IE. POLYNOMIALS, P1,P2,P3) NEEDED
C FOR INTERPOLATION (SNEW W.R.T. SARR ARRAY). M IS
C THE FIRST INDEX TO USE ON THE INTERPOLATED VARIABLE
C
C CALLED BY: INTF,INTXYZQ
C
C CALLS: NONE
C
C*****************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      COMMON/COM11/NFILT,INTER,NGSTEPS,NGPLS,GEOM,QFUN,NFLAG
      DIMENSION SARR(IMX)
      LOGICAL GEOM,QFUN
C
C IF DATA OUTSIDE RANGE, EXTRAPOLATE
C
      IF(SNEW.LE.SARR(1)) THEN
       M=1
       P1=(SARR(2)-SNEW)/(SARR(2)-SARR(1))
       P2=1.0-P1
       P3=0.0
       GO TO 999
      END IF
      IF(SNEW.GE.SARR(NPTS)) THEN
       M=NPTS-1
       P2=(SNEW-SARR(NPTS-1))/(SARR(NPTS)-SARR(NPTS-1))
       P1=1.0-P2
       P3=.0
       GO TO 999
      END IF
C
C LINEAR INTERPOLATION
C
      IF(INTER.EQ.2) THEN
       DO 20 I=1,NPTS
        IF(SNEW.GT.SARR(I) .AND. SNEW .LE.SARR(I+1)) THEN
         M=I
         P1=(SARR(M+1)-SNEW)/(SARR(M+1)-SARR(M))
         P2=1.0-P1
         P3=.0
         GO TO 999
        END IF
20     CONTINUE
      END IF
C
C 3 PT INTERPOLATION
C
      IF(SNEW.LE.SARR(2)) THEN
       M=1
       DSARR=SARR(2)-SARR(1)
       P1=(SARR(2)-SNEW)/DSARR
       P2=(SNEW-SARR(1))/DSARR
       P3=.0
       GO TO 999
      END IF
      IF(SNEW.GE.SARR(NPTS-1)) THEN
       M=NPTS-1
       DSARR=SARR(NPTS)-SARR(NPTS-1)
       P1=(SARR(NPTS)-SNEW)/DSARR
       P2=(SNEW-SARR(NPTS-1))/DSARR
       P3=.0
       GO TO 999
      END IF
      DO 100 I=2,NPTS-2
       IF(SNEW.GE. SARR(I) .AND.SNEW.LT.SARR(I+1)) THEN
        M=I-1
C
C IF INTER=3, TEST TO SEE IF BACKWARD OR
C FORWARD DIFFERENCING IS BEST
C
        IF(SNEW.GT. .5*(SARR(I)+SARR(I+1))) M=I
        S1=SARR(M)
        S2=SARR(M+1)
        S3=SARR(M+2)
        P1=(SNEW-S2)*(SNEW-S3)/((S1-S2)*(S1-S3))
        P2=(SNEW-S1)*(SNEW-S3)/((S2-S1)*(S2-S3))
        P3=(SNEW-S2)*(SNEW-S1)/((S3-S1)*(S3-S2))
        GO TO 999
       END IF
100   CONTINUE
999   CONTINUE
      RETURN
      END
C
      SUBROUTINE LINE1
C
C*****************************************************
C
C ADAPTS THE FIRST LINE ON THE FIRST PLANE
C  USING A 1-D TECHNIQUE
C
C CALLED BY: MAIN
C
C CALLS: EDGEMG,GETWT,INTF,WTEDGE,FILTER,NORM
C
C*****************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      COMMON/COM2/F(IMX),FB(IMX),WEIGHT(IMX),A,B,WDS,WDE
      COMMON/COM3/SS(IMX),SMS(IMX),DS(IMX),SN(IMX),SNM(IMX),SNMM(IMX)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM6/DSMAX,DSMIN,WT(IMX),DLENCS,DLENGE,WDSS,WDES
      COMMON/COM9/SP(IMX),SPPL(IMX),DAP(IMX),DAPPL(IMX),
     C            NEDGE,MG1,MG2
      COMMON/COM11/NFILT,INTER,NGSTEPS,NGPLS,GEOM,QFUN,NFLAG
      COMMON/COM15/CONV,MAXITS,NOUP,LNSING,PLSING,BK1
      DIMENSION SWT(IMX),SNBEST(IMX),WTBEST(IMX),DSBEST(IMX)
      LOGICAL GEOM,QFUN,NOUP
      INTEGER PLSING
```

```
      character*60 string
C
      MAXLITS=MAXITS+30
      DO 500 ITER=1,MAXLITS
       DO 100 I=1,NIPTS-1
        WEIGHT(I)=(1.0+A*FB(I)**B)
 100   CONTINUE
C
C IF EDGE PROBLEMS, OVERRIDE INFLUENCE OF FB
C
       IF (NEDGE.NE.0) THEN
        IF(ITER.EQ.1) CALL WTEDGE(JST-1,KST)
        IF(MG1.NE.0) WEIGHT(1)=WDS
        IF(MG2.NE.0) WEIGHT(NIPTS-1)=WDE
        CALL EDGEMG(WEIGHT)
       END IF
       IF(ITER.NE.1) THEN
        CALL GETWT
        DO 150 I=1,NIPTS-1
         WEIGHT(I)=WEIGHT(I)*WT(I)
 150    CONTINUE
       END IF
       IF(NFILT.NE.0) CALL FILTER(WEIGHT,NIPTS,NFILT)
       WTSUM=.0
       DO 200 L=1,NIPTS-1
        WTSUM=WTSUM+1.0/WEIGHT(L)
 200   CONTINUE
C
C NEW VALUES OF S,DS
C
       SNT(1)=.0
       DO 300 I=1,NIPTS-1
        DS(I)=SS(NIPTS)/(WEIGHT(I)*WTSUM)
        SNT(I+1)=SNT(I)+DS(I)
 300   CONTINUE
C
C TEST ON SOLUTION ERROR
C
       ERR=.0
       DO 400 L=1,NIPTS-1
        ERR=ERR+ABS((SNT(L)-SN(L))/SS(NIPTS))
        SN(L)=SNT(L)
 400   CONTINUE
       IF(ERR.LT.CONV) GO TO 530
       IF(ITER.EQ.1 .OR. (ITER.GT.1 .AND. ERR.LT.ERRMIN)) THEN
        DO 425 L=1,NIPTS
         SNBEST(L)=SNT(L)
         WTBEST(L)=WEIGHT(L)
         DSBEST(L)=DS(L)
 425    CONTINUE
        ERRMIN=ERR
       END IF
C
C IF NO CONVERGENCE, INTERPOLATE FOR F AT
C NEW VALUES OF S. AND RE-EVALUATE FB
C
       CALL INTF(F,FB,SN,SNS,NIPTS-1)
       CALL NORM(FB,NIPTS-1)
 500   CONTINUE
C
C END OF ITERATION LOOP
C
      DO 510 I=1,NIPTS
       SN(I)=SNBEST(I)
       WEIGHT(I)=WTBEST(I)
       DS(I)=DSBEST(I)
 510   CONTINUE
C      WRITE(6,1000) ERRMIN
       encode(60,1000,string) errmin
 1000  FORMAT(' NO CONVERGENCE ALONG INITIAL LINE, ERRMIN= ',E9.3)
       call warning(string)
 530   CONTINUE
 999   CONTINUE
       RETURN
       END
C
      SUBROUTINE MARCHJ(K)
C
C*************************************************************
C
C THIS ROUTINE EXTENDS THE PROPORTIONS OF THE LAST ADAPTED J LINE
C THROUGHOUT REST OF CURRENT PLANE, UP TO JMAX
C
C CALLED BY: OUTPUT
C
C CALLS: INTXYZQ
C
C*************************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      common/comxyz/x(id,jd,kd),y(id,jd,kd),z(id,jd,kd)
      common/comq/q(id,jd,kd,ndim)
      COMMON/COM3/SS(IMX),SNS(IMX),DS(IMX),SN(IMX),SNM(IMX),SNMK(IMX)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM8/XJ(IMX,3,3),YJ(IMX,3,3),ZJ(IMX,3,3)
      DIMENSION QJ(IMX,NDIM)
C
C PROPORTION REMAINING LINES
C
      SS(1)=.0
      DO 200 J=JEND+1,JMAX
       DO 110 I=1,NIPTS-1
        L=IST+I-1
        DSN=SQRT((X(L+1,J,K)-X(L,J,K))**2+(Y(L+1,J,K)-Y(L,J,K))**2+
     C           (Z(L+1,J,K)-Z(L,J,K))**2)
        SS(I+1)=SS(I)+DSN
 110   CONTINUE
       DO 120 I=1,NIPTS
        SN(I)=SNM(I)*SS(NIPTS)/SNM(NIPTS)
 120   CONTINUE
C
C FIND Q,X AND Y AT NEW S
C
       CALL INTXYZQ(J,K,2,2,SS,SN,QJ)
       DO 150 I=2,NIPTS-1
        L=IST+I-1
        X(L,J,K)=XJ(I,2,2)
        Y(L,J,K)=YJ(I,2,2)
        Z(L,J,K)=ZJ(I,2,2)
        DO 150 N=1,NDIM
         Q(L,J,K,N)=QJ(I,N)
 150   CONTINUE
 200   CONTINUE
       RETURN
       END
C
      SUBROUTINE MARCHK
C
```



```
C*************************************************************
C
C THIS ROUTINE EXTENDS THE PROPORTIONS OF THE LAST ADAPTED K PLANE
C THROUGHOUT THE REMAINING GRID, UP TO KMAX
C
C CALLED BY: OUTPUT
C
C CALLS: INTXYZQ
C
C*************************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      common/comxyz/x(id,jd,kd),y(id,jd,kd),z(id,jd,kd)
      common/comq/q(id,jd,kd,ndim)
      COMMON/COM3/SS(IMX),SNS(IMX),DS(IMX),SN(IMX),SNM(IMX),SNMK(IMX)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM8/XJ(IMX,3,3),YJ(IMX,3,3),ZJ(IMX,3,3)
      COMMON/COM14/IJPLANE,IKPLANE,JKPLANE,MARCH,MARCHPL,SAVE,OK
      DIMENSION SNK(IMX),QJ(IMX,NDIM)
      LOGICAL IJPLANE,IKPLANE,JKPLANE,MARCH,MARCHPL,SAVE,OK
      IF(MARCH) THEN
       JM=JMAX
      ELSE
       JM=JEND
      END IF
C
C FIND S ON LAST ADAPTED PLANE, FOR EACH LINE
C
      SNK(1)=.0
      DO 200 J=1,JM
       DO 110 I=1,NIPTS-1
        L=IST+I-1
        DSN=SQRT((X(L+1,J,KEND)-X(L,J,KEND))**2+
     C           (Y(L+1,J,KEND)-Y(L,J,KEND))**2+
     C           (Z(L+1,J,KEND)-Z(L,J,KEND))**2)
        SNK(I+1)=SNK(I)+DSN
 110   CONTINUE
C
C FOR EACH K PLANE WITH THIS LINE, PROPORTION NEW S
C
       DO 180 K=KEND+1,KMAX
       SS(1)=.0
       DO 115 I=1,NIPTS-1
        L=IST+I-1
        DSN=SQRT((X(L+1,J,K)-X(L,J,K))**2+
     C           (Y(L+1,J,K)-Y(L,J,K))**2+
     C           (Z(L+1,J,K)-Z(L,J,K))**2)
        SS(I+1)=SS(I)+DSN
 115   CONTINUE
       DO 120 I=1,NIPTS
        SN(I)=SNK(I)*SS(NIPTS)/SNK(NIPTS)
 120   CONTINUE
C
C FIND Q,X AND Y AT NEW S
C
       CALL INTXYZQ(J,K,2,2,SS,SN,QJ)
       DO 150 I=2,NIPTS-1
        L=IST+I-1
        X(L,J,K)=XJ(I,2,2)
        Y(L,J,K)=YJ(I,2,2)
        Z(L,J,K)=ZJ(I,2,2)
        DO 150 N=1,NDIM
         Q(L,J,K,N)=QJ(I,N)
 150   CONTINUE
 180   CONTINUE
 200   CONTINUE
       RETURN
       END
C
      SUBROUTINE MGWALLS(J)
C
C*************************************************************
C
C WHEN WALL GEOMETRY HAS LARGE GRADIENTS, AN INPUT PARAMETER
C (GEOM) REQUESTS THE INCLUSION OF GEOMETRY GRADIENTS (F(G))
C INTO THE ADAPTION VARIABLE. THIS FUNCTION SHOULD BE GRADUALLY
C DECREASED AWAY FROM BOTH WALL BOUNDARIES BY COMPUTING THE
C COEFFICIENT FGW. FGW IS A FUNCTION OF ASPECT RATIO.
C
C CALLED BY: FBAR
C
C CALLS: NONE
C
C*************************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      COMMON/COM3/SS(IMX),SNS(IMX),DS(IMX),SN(IMX),SNM(IMX),SNMK(IMX)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM9/SP(IMX),SPPL(IMX),DAP(IMX),DAPPL(IMX),
     C            NEDGE,MG1,MG2
      COMMON/COM17/FQ(IMX),FG(IMX),FGS(IMX),SMSS(IMX),MXFG,FGW,FQW
C
C COMPUTE ASPECT RATIO (USE AVERAGE OF NEW AND OLD DELTA S)
C NO GEOMETRY EFFECT IF ASPECT RATIO > 1/4
C
      IF(J.LE.JST+1 .OR. J.GE.JEND-1) THEN
       FGW=1.0
      ELSE
       DS1=SS(MXFG+1)-SS(MXFG)
       DS2=SN(MXFG+1)-SN(MXFG)
       FGASP=8.0*ABS(DAP(MXFG))/(DS1+DS2)
C
C IF ASPECT RATIO INDICATES NO GEOM, ENSURE SMOOTH TURN OFF
C
       IF(FGASP.GT.1.0) THEN
        IF(J.GT.JST+4.AND. J.LT.JEND-4) THEN
         FGW=.0
        ELSE
         IF(J.LE.JST+4) FGW=FGW*(JST-FLOAT(J-2)/4.0)
         IF(J.GE.JEND-4) FGW=FLOAT((J-JEND+5))/4.0
        END IF
       ELSE
        FGW=1.0-FGASP
       END IF
      END IF
      RETURN
      END
C
      SUBROUTINE NOADAPT(J,K)
C
C*************************************************************
C
C THIS ROUTINE UPDATES VARIABLES NEEDED WHEN STEPPING
C TO NEXT LINE WHEN THE CURRENT LINE IS NOT ADAPTED
C
CALLED BY: MAIN
C
CALLS: NONE
```

```
C
C*************************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      common/comxyz/x(id,jd,kd),y(id,jd,kd),z(id,jd,kd)
      common/comq/q(id,jd,kd,ndim)
      COMMON/COM3/SS(IMX),SMS(IMX),DS(IMX),SB(IMX),SBM(IMX),SMBK(IMX)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM15/CONV,MAXITS,NOOP,LNSING,PLSING,SK1
      LOGICAL NOOP
      INTEGER PLSING
C
C IF SINGULAR LINE, STORE DATA FROM LINE 1, PLANE 1
C
      IF(LNSING.EQ.JST .AND. K.NE.KST) THEN
       DO 100 I=1,NIPTS
        L=I+IST-1
        X(L,J,K)=X(L,JST,KST)
        Y(L,J,K)=Y(L,JST,KST)
        Z(L,J,K)=Z(L,JST,KST)
        DO 100 N=1,NDIM
          Q(L,J,K,N)=Q(L,JST,KST,N)
 100    CONTINUE
       END IF
C
       SBM(1)=.0
       DO 150 I=1,NIPTS-1
        L=I+IST-1
        DSS=SQRT((X(L+1,J,K)-X(L,J,K))**2+
     C           (Y(L+1,J,K)-Y(L,J,K))**2+
     C           (Z(L+1,J,K)-Z(L,J,K))**2)
        SBM(I+1)=SBM(I)+DSS
 150   CONTINUE
       RETURN
       END
C
       SUBROUTINE NORM(F1,NPTS)
C
C*************************************************
C
C THIS SUBROUTINE RECEIVES VARIABLE F1, NORMALISES
C IT AND RETURNS IT IN F1. THERE ARE NPTS IN F1 ARRAY.
C IF F1 'INPUT' IS  CONSTANT, DON'T NORMALISE.
C
C CALLED BY: FBAR,LINE1,SOLUT
C
C CALLS: NONE
C
C*************************************************
C
       PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
       DIMENSION F1(IMX),F2(IMX)
C
C FIND MAX AND MIN F, AND NORMALISE
C
      FMIN=F1(1)
      FMAX=F1(1)
      DO 100 I=2,NPTS
       IF(F1(I).GT.FMAX) FMAX=F1(I)
       IF(F1(I).LT.FMIN) FMIN=F1(I)
 100  CONTINUE
C
C CHECK FOR CONSTANT FLOWFIELD
C
      EPS=1.0E-04
      IF(ABS(FMAX-FMIN).LT.EPS) FMAX=FMIN
      DO 200 I=1,NPTS
       IF(FMAX.EQ.FMIN .OR. F1(I).EQ.FMIN) THEN
         F2(I)=1.0E-05
       ELSE
         F2(I)=(F1(I)-FMIN)/(FMAX-FMIN)
         F2(I)=MAX(F2(I),1.0E-05)
       END IF
 200  CONTINUE
      DO 300 I=1,NPTS
       F1(I)=F2(I)
 300  CONTINUE
      RETURN
      END
C
C
      SUBROUTINE NORMPT(IP,JP,KP,INDPL,PA,PB,PC,SING)
C
C*************************************************************
C
C THIS ROUTINE TAKES A POINT, (IP,JP,KP) AND A PLANE, INDPL
C AND FINDS THE NORMAL VECTOR TO THE PLANE AT THAT POINT.
C THIS REQUIRES FINDING THE NORMAL TO THE 4 PLANES SURROUNDING
C THE POINT, PTS SURROUNDING A ARE B,C,D,E
C
C INDPL FOR IKPLANE=1, IJPLANE=2
C
CALLED BY: SETUJP,SETUKP
C
CALLS: UNITV,PURPLE,ADDV
C
C*************************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM8/XJ(IMX,3,3),YJ(IMX,3,3),ZJ(IMX,3,3)
      COMMON/COM11/NFILT,INTER,MGSTEPS,MGPLS,GEOM,QFUN,NFLAG
      LOGICAL GEOM,QFUN
      SING=.0
      XA=XJ(IP,JP,KP)
      YA=YJ(IP,JP,KP)
      ZA=ZJ(IP,JP,KP)
      IF(INDPL.EQ.1) THEN
       XB=XJ(IP+1,JP,KP)
       YB=YJ(IP+1,JP,KP)
       ZB=ZJ(IP+1,JP,KP)
       XC=XJ(IP,JP,KP+1)
       YC=YJ(IP,JP,KP+1)
       ZC=ZJ(IP,JP,KP+1)
       XD=XJ(IP-1,JP,KP)
       YD=YJ(IP-1,JP,KP)
       ZD=ZJ(IP-1,JP,KP)
       XE=XJ(IP,JP,KP-1)
       YE=YJ(IP,JP,KP-1)
       ZE=ZJ(IP,JP,KP-1)
      END IF
      IF(INDPL.EQ.2) THEN
       XB=XJ(IP,JP+1,KP)
       YB=YJ(IP,JP+1,KP)
       ZB=ZJ(IP,JP+1,KP)
       XC=XJ(IP+1,JP,KP)
       YC=YJ(IP+1,JP,KP)
       ZC=ZJ(IP+1,JP,KP)
```



```
       XD=XJ(IP,JP-1,KP)
       YD=YJ(IP,JP-1,KP)
       ZD=ZJ(IP,JP-1,KP)
       XE=XJ(IP-1,JP,KP)
       YE=YJ(IP-1,JP,KP)
       ZE=ZJ(IP-1,JP,KP)
      END IF
C
C FIND UNIT VECTORS BETWEEN A AND OTHER POINTS
C
      IF(XB.EQ.999.0) THEN
       XAB=.0
       YAB=.0
       ZAB=.0
      ELSE
       CALL UNITV(XA,YA,ZA,XB,YB,ZB,XAB,YAB,ZAB)
      END IF
      IF(XC.EQ.999.0) THEN
       XAC=.0
       YAC=.0
       ZAC=.0
      ELSE
       CALL UNITV(XA,YA,ZA,XC,YC,ZC,XAC,YAC,ZAC)
      END IF
      IF(XD.EQ.999.0) THEN
       XDA=.0
       YDA=.0
       ZDA=.0
      ELSE
       CALL UNITV(XD,YD,ZD,XA,YA,ZA,XDA,YDA,ZDA)
      END IF
      IF(XE.EQ.999.0) THEN
       XEA=.0
       YEA=.0
       ZEA=.0
      ELSE
       CALL UNITV(XE,YE,ZE,XA,YA,ZA,XEA,YEA,ZEA)
      END IF
C
C FIND NORMAL TO EACH PLANE
C
      CALL PURPLE(XAC,YAC,ZAC,XAB,YAB,ZAB,S1A,S1B,S1C,NFLAG)
      CALL PURPLE(XAC,YAC,ZAC,XDA,YDA,ZDA,S2A,S2B,S2C,NFLAG)
      CALL PURPLE(XEA,YEA,ZEA,XDA,YDA,ZDA,S3A,S3B,S3C,NFLAG)
      CALL PURPLE(XEA,YEA,ZEA,XAB,YAB,ZAB,S4A,S4B,S4C,NFLAG)
C
C AVERAGE THE NORMALS
C
      CALL ADDV(S1A,S1B,S1C,1.0,S2A,S2B,S2C,1.0,S12A,S12B,S12C)
      CALL ADDV(S3A,S3B,S3C,1.0,S4A,S4B,S4C,1.0,S34A,S34B,S34C)
      CALL ADDV(S12A,S12B,S12C,1.0,S34A,S34B,S34C,1.0,PA,PB,PC)
      TST=PA**2+PB**2+PC**2
      IF(TST.LT..9) SING=1.0
      RETURN
      END
C
      SUBROUTINE OUTPUT(NADS)
C
C*************************************************
C
C THIS ROUTINE RE-SWAPS THE X,Y AND Q DATA
C (MARCHES IF REQUESTED ON INPUT)
C AND THEN OUTPUTS IN A FORMAT FOR PLOT3D -
C UNIT NUMBERS DEPEND ON NUMBER OF PASSES
C
C CALLED BY: MAIN
C
C CALLS: MARCHK,SWAPINV,SWAPXYZ
C
C*************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      common/comxyz/x(id,jd,kd),y(id,jd,kd),z(id,jd,kd)
      common/comq/q(id,jd,kd,ndim)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM12/FSMACH,ALPHA,RE,TIME,NITC,NITQ
      COMMON/COM13/ISTEP,JSTEP,KSTEP,IINVERSE,JINVERSE,KINVERSE,TWOD
      COMMON/COM14/IJPLANE,IKPLANE,JKPLANE,MARCH,MARCHPL,SAVE,OK
      LOGICAL ISTEP,JSTEP,KSTEP,IINVERSE,JINVERSE,KINVERSE,TWOD
      LOGICAL IJPLANE,IKPLANE,JKPLANE,MARCH,MARCHPL,SAVE,OK
      LOGICAL RSWAP
      character*50 string
C
C IF CONTINUITY REQUIRED AT NON ADAPTED POINTS (MARCH=7)
C INTERPOLATE FOR NEW X,Y,Q AT REMAINING K PLANES
C
      IF(MARCHPL.AND.KEND.LT.KMAX) CALL MARCHK
C
C IF NECESSARY, RE-INVERT ARRAYS (EG. 1 THRU N BECOMES N THRU 1)
C IF STEPPING IN I, X AND Y DATA NEEDS TO BE SWAPPED
C
      IF(IINVERSE .OR. JINVERSE.OR. KINVERSE) CALL SWAPINV
      RSWAP=.TRUE.
      IF(.NOT. (IJPLANE .AND. JSTEP)) CALL SWAPXYZ(RSWAP)
      IF (TWOD) CALL SWAP2D(2)
C
C COMPUTE UNIT NUMBERS
C
      IF(NADS.EQ.1) THEN
       NITC=8
       NITQ=9
      END IF
      IF(.NOT. OK .OR. SAVE) THEN
       NITC=NITC+2
       NITQ=NITQ+2
C
C OUTPUT ADAPTED GRID AND FLOWFIELD
C
c       CALL WRITOUT
      END IF
      IF(OK) then
c       WRITE(6,1000) NADS
c       encode(50,1000,string) nads
c1000   FORMAT(' ADAPTION ',I2,' COMPLETE')
c       call warning(string)
      else
       encode(50,1010,string)
1010   FORMAT(' ADAPTION DID NOT COMPLETE')
       call warning(string)
      endif
999   CONTINUE
      RETURN
      END
C
      SUBROUTINE PROPS(J,K)
C
C*************************************************
C
```

```
C THIS ROUTINE TAKES THE CONVERGED VALUE OF S ON THE J-1,K
C LINE AND PROPORTIONS IT TO THE REMAINING BLOCK OF XJ,YJ,ZJ
C
CALLED BY :FBAR
C
CALLS: INTXYZQ
C
C*************************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      common/comxyz/x(id,jd,kd),y(id,jd,kd),z(id,jd,kd)
      common/comq/q(id,jd,kd,ndim)
      COMMON/COM3/SS(IMX),SMS(IMX),DS(IMX),SN(IMX),SNM(IMX),SNMK(IMX)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      DIMENSION DOM(IMX,NDIM),SSNK(IMX)
      DIMENSION SSK(IMX),SNK(IMX),SSJK(IMX),SNJK(IMX)
C
C PROPORTION J LINE INTO SN
C
      SN(1)=.0
      IF(J.EQ.JST) THEN
       DO 100 I=2,NIPTS
        SN(I)=SNMK(I)*SS(NIPTS)/SNMK(NIPTS)
100    CONTINUE
      ELSE
       DO 150 I=2,NIPTS
        SN(I)=SNM(I)*SS(NIPTS)/SNM(NIPTS)
150    CONTINUE
      END IF
      CALL INTXYZQ(J,K,2,2,SS,SN,DUM)
C
C AT J+1,K (FOR COSSK)
C
      IF(J.NE.JEND) THEN
       SSK(1)=.0
       DO 175 I=1,NIPTS-1
        L=IST+I-1
        DSSK=SQRT((X(L+1,J+1,K)-X(L,J+1,K))**2+
     C            (Y(L+1,J+1,K)-Y(L,J+1,K))**2+
     C            (Z(L+1,J+1,K)-Z(L,J+1,K))**2)
        SSK(I+1)=SSK(I)+DSSK
175    CONTINUE
       IF(J.EQ.JST) THEN
        DO 180 I=1,NIPTS
         SSNK(I)=SNMK(I)*SSK(NIPTS)/SNMK(NIPTS)
180     CONTINUE
       ELSE
        DO 185 I=1,NIPTS
         SSNK(I)=SNM(I)*SSK(NIPTS)/SNM(NIPTS)
185     CONTINUE
       END IF
       CALL INTXYZQ(J+1,K,3,2,SSK,SSNK,DUM)
      END IF
C
C K+1 PLANE, BOTH J-1 (FOR NORMAL U) AND J (FOR NORMAL B)
C
      IF(K.NE.KMAX .AND.J.NE. JST) THEN
C
C AT J-1,K+1
C
       SSK(1)=.0
       DO 200 I=1,NIPTS-1
        L=IST+I-1
        DSSK=SQRT((X(L+1,J-1,K+1)-X(L,J-1,K+1))**2+
     C            (Y(L+1,J-1,K+1)-Y(L,J-1,K+1))**2+
     C            (Z(L+1,J-1,K+1)-Z(L,J-1,K+1))**2)
        SSK(I+1)=SSK(I)+DSSK
200    CONTINUE
       DO 250 I=1,NIPTS
        SNK(I)=SNM(I)*SSK(NIPTS)/SNM(NIPTS)
250    CONTINUE
       CALL INTXYZQ(J-1,K+1,1,3,SSK,SNK,DUM)
C
C AT J,K+1
C
       SSJK(1)=.0
       DO 400 I=1,NIPTS-1
        L=IST+I-1
        DSSK=SQRT((X(L+1,J,K+1)-X(L,J,K+1))**2+
     C            (Y(L+1,J,K+1)-Y(L,J,K+1))**2+
     C            (Z(L+1,J,K+1)-Z(L,J,K+1))**2)
        SSJK(I+1)=SSJK(I)+DSSK
400    CONTINUE
       DO 450 I=1,NIPTS
        SNJK(I)=SNM(I)*SSJK(NIPTS)/SNM(NIPTS)
450    CONTINUE
       CALL INTXYZQ(J,K+1,2,3,SSJK,SNJK,DUM)
      END IF
      RETURN
      END
C
      SUBROUTINE PURPLE(A1,A2,A3,B1,B2,B3,V1,V2,V3,NFL)
C
C*************************************************************
C
C THIS ROUTINE TAKES THE CROSS-PRODUCT OF UNIT VECTORS A & B
C AND NORMALIZES TO GIVE THE UNIT VECTOR, V
C PERPENDICULAR TO PLANE DESCRIBED BY A & B
C
CALLED BY: CROSSV,NORMPT
C
CALLS: NONE
C
C*************************************************************
C
      V1=(A2*B3-A3*B2)*NFL
      V2=(A3*B1-A1*B3)*NFL
      V3=(A1*B2-A2*B1)*NFL
      VMOD=SQRT(V1**2+V2**2+V3**2)
      IF(VMOD.NE.0.0) THEN
       V1=V1/VMOD
       V2=V2/VMOD
       V3=V3/VMOD
      END IF
      RETURN
      END
C
      SUBROUTINE READAT
C
C*************************************************************
C
C THIS ROUTINE READS THE GRID AND FUNCTION FILES.
C IF DATASETS ARE IN 2-D, THE DATA IS REARRANGED
C TO GIVE 3-D FORM
C
CALLED BY INITIAL
C
CALLS: NONE
```



```
C
C*************************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      common/comxyz/x(id,jd,kd),y(id,jd,kd),z(id,jd,kd)
      common/comq/q(id,jd,kd,ndim)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM12/FSMACH,ALPHA,RE,TIME,NITG,NITQ
      COMMON/COM13/ISTEP,JSTEP,KSTEP,IINVERSE,JINVERSE,KINVERSE,TWOD
      LOGICAL ISTEP,JSTEP,KSTEP,IINVERSE,JINVERSE,KINVERSE,TWOD
C
C READ X,Y,Z GRID POINTS AND CORRESPONDING FLOW VARIABLES (Q)
C
      IF (.NOT.TWOD) THEN
       READ(7) IMAX,JMAX,KMAX
       READ(7) (((X(I,J,K),I=1,IMAX),J=1,JMAX),K=1,KMAX),
     C         (((Y(I,J,K),I=1,IMAX),J=1,JMAX),K=1,KMAX),
     C         (((Z(I,J,K),I=1,IMAX),J=1,JMAX),K=1,KMAX)
       READ(8) IMAX,JMAX,KMAX
       READ(8) FSMACH,ALPHA,RE,TIME
       READ(8) ((((Q(I,J,K,N),I=1,IMAX),J=1,JMAX),K=1,KMAX),
     C          N=1,NDIM)
      ELSE
       READ(7) IMAX,JMAX
       READ(7) ((X(I,J,1),I=1,IMAX),J=1,JMAX),
     C         ((Y(I,J,1),I=1,IMAX),J=1,JMAX)
       READ(8) IMAX,JMAX
       READ(8) FSMACH,ALPHA,RE,TIME
       READ(8) (((Q(I,J,1,N),I=1,IMAX),J=1,JMAX),N=1,NDIM-1)
      END IF
      RETURN
      END
      SUBROUTINE SETUPJ(J,K)
C
C*************************************************************
C
C THIS ROUTINE FINDS THE DIRECTION COSINES OF THE VECTORS
C USED IN THE SOLUT ROUTINE THAT ARE NOT A FUNCTION
C OF THE ITERATIONS: NORMAL VECTOR (U) FROM J-1 LINE,
C NORMAL VECTOR (B) FROM J LINE, AND STRAIGHTNESS VECTOR(E).
C
C CALLED BY: MAIN
C
C CALLS: ADDV,UNITV,VMERGE,NORMPT
C
C*************************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      common/comxyz/x(id,jd,kd),y(id,jd,kd),z(id,jd,kd)
      common/comq/q(id,jd,kd,ndim)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM7/COSE(IMX,3),COSU(IMX,3),COSB(IMX,3)
      COMMON/COM8/XJ(IMX,3,3),YJ(IMX,3,3),ZJ(IMX,3,3)
      COMMON/COM11/NFILT,INTER,MGSTEPS,MGPLS,GEOM,QFUN,NFLAG
      DIMENSION COSD(IMX,3)
      LOGICAL GEOM,QFUN
C
C FIND U AS AVERAGE  NORMAL TO 4 PLANES AT I,J-1,K
C
      DO 100 I=2,NIPTS-1
       CALL NORMPT(I,1,2,1,COSU(I,1),COSU(I,2),COSU(I,3),SING)
       IF(SING.EQ.1.0) CALL UNITV(XJ(I,1,2),YJ(I,1,2),ZJ(I,1,2),
     C   XJ(I,2,2),YJ(I,2,2),ZJ(I,2,2),COSU(I,1),COSU(I,2),COSU(I,3))
100   CONTINUE
C
C FIND B VECTOR, NORMAL TO J LINE
C
      DO 120 I=2,NIPTS-1
       CALL NORMPT(I,2,2,1,COSB(I,1),COSB(I,2),COSB(I,3),SING)
       IF(SING.EQ.1.0) CALL UNITV(XJ(I,1,2),YJ(I,1,2),ZJ(I,1,2),
     C   XJ(I,2,2),YJ(I,2,2),ZJ(I,2,2),COSB(I,1),COSB(I,2),COSB(I,3))
120   CONTINUE
C
C FIND D VECTOR (USED FOR E VECTOR, STRAIGHTNESS),
C (IF 2ND J LINE AND JST=1, THEN NO J-2 LINE EXISTS)
C
      IF(J.EQ.2) THEN
       DO 150 I=2,NIPTS-1
        CALL UNITV(XJ(I,1,2),YJ(I,1,2),ZJ(I,1,2),
     C             XJ(I,2,2),YJ(I,2,2),ZJ(I,2,2),
     C             COSD(I,1),COSD(I,2),COSD(I,3))
150    CONTINUE
      ELSE
       DO 200 I=2,NIPTS-1
        L=IST+I-1
        CALL UNITV(X(L,J-2,K),Y(L,J-2,K),Z(L,J-2,K),
     C             X(L,J-1,K),Y(L,J-1,K),Z(L,J-1,K),
     C             COSD(I,1),COSD(I,2),COSD(I,3))
       IF(COSD(I,1).EQ..0.AND.COSD(I,2).EQ..0.AND.COSD(I,3).EQ..0)
     C  CALL UNITV(XJ(I,1,2),YJ(I,1,2),ZJ(I,1,2),
     C             XJ(I,2,2),YJ(I,2,2),ZJ(I,2,2),
     C             COSD(I,1),COSD(I,2),COSD(I,3))
200    CONTINUE
      END IF
C
C FIND ACTUAL BOUNDARY LINE FOR EDGE CONDITIONS
C
      CALL UNITV(X(IST,J-1,K),Y(IST,J-1,K),Z(IST,J-1,K),
     C           X(IST,J,K),Y(IST,J,K),Z(IST,J,K),
     C           COSD(1,1),COSD(1,2),COSD(1,3))
      CALL UNITV(X(IEND,J-1,K),Y(IEND,J-1,K),Z(IEND,J-1,K),
     C           X(IEND,J,K),Y(IEND,J,K),Z(IEND,J,K),
     C           COSD(NIPTS,1),COSD(NIPTS,2),COSD(NIPTS,3))
      CALL VMERGE(COSD,1,NIPTS)
C
C MERGE ACTUAL ENDS INTO NORMAL VECTORS TOO
C
      DO 230 M=1,3
       COSU(1,M)=COSD(1,M)
       COSU(NIPTS,M)=COSD(NIPTS,M)
       COSB(1,M)=COSD(1,M)
       COSB(NIPTS,M)=COSD(NIPTS,M)
       COSE(1,M)=COSD(1,M)
       COSE(NIPTS,M)=COSD(NIPTS,M)
230   CONTINUE
      CALL VMERGE(COSU,1,NIPTS)
      CALL VMERGE(COSB,1,NIPTS)
C
C FIND E AS AVERAGE  D (I+1,I,I-1)  (AT ENDS, E=0)
C
      DO 350 I=2,NIPTS-1
       CALL ADDV(COSD(I-1,1),COSD(I-1,2),COSD(I-1,3),1.0,
     C           COSD(I,1),COSD(I,2),COSD(I,3),1.0,CFX,CFY,CFZ)
       CALL ADDV(CFX,CFY,CFZ,1.0,COSD(I+1,1),COSD(I+1,2),COSD(I+1,3),
     C           1.0,COSE(I,1),COSE(I,2),COSE(I,3))
350   CONTINUE
      RETURN
      END
```

```
C
      SUBROUTINE SETUPK(J,K)
C
C*************************************************************
C
C THIS ROUTINE FINDS THE DIRECTION COSINES OF THE VECTORS
C USED TO COMPUTE THE TORSION VECTOR FROM THE K-1 PLANE
C NORMAL VECTOR, U AT (J,K-1)
C AND NORMAL VECTOR B FROM (J,K) AND STRAIGHTNESS VECTOR(E).
C
C CALLED BY: MAIN
C
C CALLS: ADDV,UNITV,VMERGE,NORMPT
C
C*************************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      common/comxyz/x(id,jd,kd),y(id,jd,kd),z(id,jd,kd)
      common/comq/q(id,jd,kd,ndim)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM8/XJ(IMX,3,3),YJ(IMX,3,3),ZJ(IMX,3,3)
      COMMON/COM11/NFILT,INTER,MGSTEPS,MGPLS,GEOM,QFUN,NFLAG
      COMMON/COM20/COSEK(IMX,3),COSUK(IMX,3),COSBK(IMX,3)
      DIMENSION COSD(IMX,3)
      LOGICAL GEOM,QFUN
C
C FIND U AS AVERAGE  NORMAL TO 4 PLANES AT I,J,K-1
C
      DO 100 I=2,NIPTS-1
       CALL NORMPT(I,2,1,2,COSUK(I,1),COSUK(I,2),COSUK(I,3),SING)
100   CONTINUE
C
C FIND B VECTOR, NORMAL TO J LINE
C
      DO 120 I=2,NIPTS-1
       CALL NORMPT(I,2,2,2,COSBK(I,1),COSBK(I,2),COSBK(I,3),SING)
120   CONTINUE
C
C FIND D VECTOR (USED FOR E VECTOR, STRAIGHTNESS),
C (IF 2ND K PLANE AND KST=1, THEN NO K-2 LINE EXISTS)
C
      IF(K.EQ.2) THEN
       DO 150 I=2,NIPTS-1
        CALL UNITV(XJ(I,2,1),YJ(I,2,1),ZJ(I,2,1),
     C             XJ(I,2,2),YJ(I,2,2),ZJ(I,2,2),
     C             COSD(I,1),COSD(I,2),COSD(I,3))
150    CONTINUE
      ELSE
       DO 200 I=2,NIPTS-1
        L=IST+I-1
        CALL UNITV(X(L,J,K-2),Y(L,J,K-2),Z(L,J,K-2),
     C             X(L,J,K-1),Y(L,J,K-1),Z(L,J,K-1),
     C             COSD(I,1),COSD(I,2),COSD(I,3))
        IF(COSD(I,1).EQ..0.AND.COSD(I,2).EQ..0.AND.COSD(I,3).EQ.0)
     C   CALL UNITV(XJ(I,2,1),YJ(I,2,1),ZJ(I,2,1),
     C              XJ(I,2,2),YJ(I,2,2),ZJ(I,2,2),
     C              COSD(I,1),COSD(I,2),COSD(I,3))
200    CONTINUE
       END IF
C
C AT END, D IS ACTUAL BOUNDARY LINE
C
       CALL UNITV(X(IST,J,K-1),Y(IST,J,K-1),Z(IST,J,K-1),
     C  X(IST,J,K),Y(IST,J,K),Z(IST,J,K),
     C  COSD(1,1),COSD(1,2),COSD(1,3))
       CALL UNITV(X(IEND,J,K-1),Y(IEND,J,K-1),Z(IEND,J,K-1),
     C            X(IEND,J,K),Y(IEND,J,K),Z(IEND,J,K),
     C            COSD(NIPTS,1),COSD(NIPTS,2),COSD(NIPTS,3))
       CALL VMERGE(COSD,1,NIPTS)
C
C MERGE INTO NORMAL VECTORS TOO
C
      DO 230 N=1,3
       COSUK(1,N)=COSD(1,N)
       COSUK(NIPTS,N)=COSD(NIPTS,N)
       COSBK(1,N)=COSD(1,N)
       COSBK(NIPTS,N)=COSD(NIPTS,N)
       COSEK(1,N)=COSD(1,N)
       COSEK(NIPTS,N)=COSD(NIPTS,N)
230   CONTINUE
      CALL VMERGE(COSUK,1,NIPTS)
      CALL VMERGE(COSBK,1,NIPTS)
C
C FIND E AS AVERAGE  D (I+1,I,I-1) (AT ENDS, E=D)
C
      DO 350 I=2,NIPTS-1
       CALL ADDV(COSD(I-1,1),COSD(I-1,2),COSD(I-1,3),1.0,
     C      COSD(I,1),COSD(I,2),COSD(I-1,3),1.0,CFX,CFY,CFZ)
       CALL ADDV(CFX,CFY,CFZ,1.0,COSD(I+1,1),COSD(I+1,2),
     C      COSD(I+1,3),1.0,COSEK(I,1),COSEK(I,2),COSEK(I,3))
350   CONTINUE
      RETURN
      END
C
      SUBROUTINE SINGPLN
C
C*************************************************************
C
C WHEN A PLANE IS ONLY A SINGLE LINE (BUT STORED AS IF
C A PLANE) EACH LINE MUST HAVE THE SAME RESULT.
C
CALLED BY: MAIN
C
CALLS: NONE
C
C*************************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      common/comxyz/x(id,jd,kd),y(id,jd,kd),z(id,jd,kd)
      common/comq/q(id,jd,kd,ndim)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      DO 100 J=JST+1,JEND
       DO 100 I=IST,IEND
        M=IST+I-1
        X(M,J,1)=X(M,JST,1)
        Y(M,J,1)=Y(M,JST,1)
        Z(M,J,1)=Z(M,JST,1)
        DO 100 N=1,NDIM
         Q(M,J,1,N)=Q(M,JST,1,N)
100   CONTINUE
      RETURN
      END
C
      SUBROUTINE SIZING(IER)
C
C*************************************************************
C
C THIS ROUTINE IS CALLED TO TEST GRID SIZE VERSUS
```



```
C THE CURRENT DIMENSIONS IN THE PARAMETER STATEMENT.
C IF NECESSARY, SUGGESTIONS ARE PRINTED FOR MINIMUM SIZING
C
C CALLED BY INITIAL
C
CALLS NONE
C
C*************************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM13/ISTEP,JSTEP,KSTEP,IINVERSE,JINVERSE,KINVERSE,TWOD
      COMMON/COM14/IJPLANE,IKPLANE,JKPLANE,MARCH,MARCHPL,SAVE,OK
      LOGICAL ISTEP,JSTEP,KSTEP,IINVERSE,JINVERSE,KINVERSE,TWOD
      LOGICAL IJPLANE,IKPLANE,JKPLANE,MARCH,MARCHPL,SAVE,OK
      character*72 string
      IER=0
      IF(IJPLANE) THEN
       KSIZE=KMAX
       IF(ISTEP) THEN
        ISIZE=MAX(IMAX,JMAX)
        JSIZE=ISIZE
       ELSE
        ISIZE=IMAX
        JSIZE=JMAX
       END IF
      END IF
      IF(JKPLANE) THEN
       IF(JSTEP) THEN
        ISIZE=MAX(IMAX,KMAX)
        JSIZE=JMAX
        KSIZE=MAX(IMAX,KMAX)
       ELSE
        ISIZE=MAX(IMAX,JMAX)
        JSIZE=MAX(KMAX,JMAX)
        KSIZE=MAX(IMAX,KMAX)
       END IF
      END IF
      IF(IKPLANE) THEN
       IF(ISTEP) THEN
        ISIZE=MAX(IMAX,KMAX)
        JSIZE=MAX(IMAX,JMAX)
        KSIZE=MAX(JMAX,KMAX)
       ELSE
        ISIZE=IMAX
        JSIZE=MAX(JMAX,KMAX)
        KSIZE=JSIZE
       END IF
      END IF
      MSIZE=MAX(IMAX,JMAX,KMAX)
      IF ((ISIZE.GT.ID .OR.JSIZE.GT.JD .OR. KSIZE.GT.KD .OR.
     C     MSIZE.GT.IMX) THEN
c      WRITE(6,1000) ISIZE,JSIZE,KSIZE,MSIZE
c1000  FORMAT('GRID SIZE TOO LARGE FOR DIMENSION STATEMENT'
c     C' CHANGE PARAMETER STATEMENTS TO: ID=',I3,', JD=',I3,', KD=',I3,
c     C', IMX=',I3,' THESE ARE THE MINIMUM DIMENSIONS FOR THIS SET OF
c     CCONTROL PARAMETERS ONLY')
       encode(72,1000,string)
1000   format('GRID SIZE TOO LARGE FOR DIMENSION STATEMENT')
       call error(string)
       encode(72,1010,string) isize, jsize, ksize, msize
1010   format('CHANGE PARAMETERS TO: ID=',I3,', JD=',I3,', KD=',I3,
     C ', IMX=',I3)
       call warning(string)
       IER=1
      END IF
      RETURN
      END
C
      SUBROUTINE SOLUT(J,K)
C
C*************************************************************
C
C THIS ROUTINE SOLVES FOR THE NEW S(i) AT J.
C THE COEFFS OF THE TRIDIAGONAL ARE COMPUTED AND ITERATED.
C ITERATION ENDS WHEN DSnew-DSold IS SMALL
C
C CALLED BY: MAIN
C
C CALLS: EDGEMG,GETWT,INTF,FILTER,NORM
C
C*************************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      COMMON/COM2/F(IMX),FB(IMX),WEIGHT(IMX),A,B,WDS,WDE
      COMMON/COM3/SS(IMX),SMS(IMX),DS(IMX),SN(IMX),SNM(IMX),SNMR(IMX)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM6/DSMAX,DSMIN,WT(IMX),DLENGS,DLENGE,WDSS,WDES
      COMMON/COM9/SP(IMX),SPPL(IMX),DAP(IMX),DAPPL(IMX),
     C            NEDGE,MG1,MG2
      COMMON/COM10/CLAM(2),CT(2),TH(2),CNM(2),CLAMW(2),CTM(2),THM(2)
      COMMON/COM11/NFILT,INTER,MGSTEPS,MGPLS,GEOM,QFUN,NFLAG
      COMMON/COM13/ISTEP,JSTEP,KSTEP,IINVERSE,JINVERSE,KINVERSE,TWOD
      COMMON/COM14/IJPLANE,IKPLANE,JKPLANE,MARCH,MARCHPL,SAVE,OK
      COMMON/COM15/CONV,MAXITS,NOUP,LRSING,PLSING,BK1
      DIMENSION AA(IMX),BB(IMX),CC(IMX),FF(IMX)
      DIMENSION SOLD(IMX),TAU(IMX),TAUPL(IMX)
      LOGICAL GEOM,QFUN,NOUP
      LOGICAL ISTEP,JSTEP,KSTEP,IINVERSE,JINVERSE,KINVERSE,TWOD
      LOGICAL IJPLANE,IKPLANE,JKPLANE,MARCH,MARCHPL,SAVE,OK
      INTEGER PLSING
      character*70 string
C
C STORE OLD S VALUES FOR CONVERGENCE TEST
C
      DO 50 I=1,NIPTS
       SOLD(I)=SN(I)
50    CONTINUE
C
C FIND TORSION SPRING CONSTANTS, TAU
C INCLUDE ASPECT RATIO OF DAP
C
      SPC=(1.0+A)*CLAMW(1)
      SPCPL=(1.0+A)*CLAMW(2)
      DO 75 I=2,NIPTS-1
       IF(J.GT.JST) THEN
        ASPECT=.5*(SNM(I+1)-SNM(I-1))/ABS(DAP(I))
        TAU(I)=SPC*ASPECT/ABS(DAP(I))
       ELSE
        TAU(I)=.0
       END IF
       IF(K.GT.KST) THEN
        ASPECT=.5*(SNMK(I+1)-SNMK(I-1))/ABS(DAPPL(I))
        TAUPL(I)=SPCPL*ASPECT/ABS(DAPPL(I))
       ELSE
        TAUPL(I)=.0
       END IF
75    CONTINUE
```

```
C
C START ITERATION LOOP FOR SOLVING DS ALONG LINE J
C
      DO 600 LOOP=1,MAXITS
C
C FIND M. TENSION SPRING CONSTANT
C
      DO 100 I=1,NIPTS-1
       WEIGHT(I)=(1.0+A*FB(I)**B)
100   CONTINUE
C
C OVERRIDE EDGE WEIGHTS, IF NEC.
C
      IF(NEDGE.NE.0 ) THEN
       IF(WDS.EQ..0 .AND. WDE.EQ..0) CALL WTEDGE(J-1,K)
       IF(MG1.NE.0) WEIGHT(1)=WDS
       IF(MG2.NE.0) WEIGHT(NIPTS-1)=WDE
       CALL EDGEWG(WEIGHT)
      END IF
125   CONTINUE
C
C FIND MUTIPLIER OF WEIGHT FOR SPACING CONTROL
C
      IF(LOOP.NE.1) THEN
       CALL GETWT
       DO 150 I=1,NIPTS-1
        WEIGHT(I)=WEIGHT(I)*WT(I)
150    CONTINUE
      END IF
C
C FILTER IF NEC.
C
      IF(NFILT.GT.0) CALL FILTER(WEIGHT,NIPTS,NFILT)
C
C SET UP COEFFS OF S-1,S,S+1
C
      DO 200 I=2,NIPTS-1
       AA(I)=WEIGHT(I-1)
       BB(I)=-(WEIGHT(I)+WEIGHT(I-1)+TAD(I)+TAUFL(I))
       CC(I)=WEIGHT(I)
       FF(I)=-TAU(I)*SP(I)-TAUFL(I)*SPFL(I)
200   CONTINUE
C
C CORRECT FIRST AND LAST EQUATIONS TO REFLECT KNOWN END S'S
C
      FF(2)=FF(2)-AA(2)*SN(1)
      FF(NIPTS-1)=FF(NIPTS-1)-CC(NIPTS-1)*SN(NIPTS)
      AA(2)=.0
      CC(NIPTS-1)=.0
C
C TRIDIAGONAL SOLUTION
C
      DO 300 I=3,NIPTS-1
       TEMP=AA(I)/BB(I-1)
       BB(I)=BB(I)-CC(I-1)*TEMP
       FF(I)=FF(I)-FF(I-1)*TEMP
300   CONTINUE
C
C BACK SUBSTITUTE
C
      SN(NIPTS-1)=FF(NIPTS-1)/BB(NIPTS-1)
      DO 400 I=2,NIPTS-2
       IB=NIPTS-I
       SN(IB)=(FF(IB)-CC(IB)*SN(IB+1))/BB(IB)
400   CONTINUE
C
C CORRECT SN FOR NUMERIC ERROR
C
      DO 420 I=NIPTS-1,2,-1
       IF(SN(I).LT.1.0E-06) THEN
         DSN=SN(I+1)/I
         DO 410 L=2,I
          SN(L)=SN(L-1)+DSN
410      CONTINUE
         GO TO 430
       END IF
420   CONTINUE
430   CONTINUE
      DO 450 I=1,NIPTS-1
       DS(I)=SN(I+1)-SN(I)
450   CONTINUE
C
C RE-EVALUATE FB AT THESE SN
C
      CALL INTF(F,FB,SN,SMS,NIPTS-1)
      CALL NORM(FB,NIPTS-1)
C
C TEST ERROR
C
      ERR=.0
      DO 500 I=2,NIPTS-1
       ERR=ERR+ABS(SN(I)-SOLD(I))
500   CONTINUE
      DO 550 I=2,NIPTS
       SOLD(I)=SN(I)
550   CONTINUE
      ERR=ERR/SS(NIPTS)
      IF(ERR.LT.CONV) GO TO 700
600   CONTINUE
      KERR=K
      IF(KINVERSE) KERR=KMAX-K+1
      JERR=J
      IF(JINVERSE) JERR=JMAX-J+1
      IF (ERR.GT.(CONV*10.0)) then
c      WRITE(6,1000) JERR,KERR,ERR
       encode(70,1000,string) jerr, kerr, err
1000   FORMAT(' NO CONVERGENCE ON LINE ',I3,' PLANE',I3,' ERR= ',E9.3)
       call warning(string)
      endif
C
C CONVERGENCE HERE**
C
C TEST THAT S MONOTONICALLY INCREASING
C IF NOT, END CODE AND OUTPUT.
C
700   CONTINUE
      DO 800 I=2,NIPTS
       IF(SN(I).LE. SN(I-1)) THEN
        JERR=J
        KERR=K
        IF(JINVERSE) JERR=JMAX-J+1
        IF(KINVERSE) KERR=KMAX-K+1
c       WRITE(6,1001) JERR,KERR
        encode(50,1001,string) jerr, kerr
1001    FORMAT(' S NOT MONOTONIC ON LINE ',I3, ' AND PLANE ',I2)
        call warning(string)
        OK=.FALSE.
        GO TO 900
```



```
       END IF
800   CONTINUE
900   CONTINUE
      RETURN
      END
C
      SUBROUTINE SPEVAL(NT,S,V,SPF,SI,VI,VPI,VPPI)
C
C**********************************************************************
C
C INTERPOLATION ROUTINE (USED IN CONJUNCTION WITH CSPLIN).
C FIRST AND SECOND DERIVATIVES ARE COMPUTED FOR RADIUS OF CURVATURE.
C USED IN WALLS
C
C CALLED BY: INTF,INTXYZQ,WALLS
C
C CALLS: NONE
C
C**********************************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      DIMENSION S(IMX),V(IMX),SPF(IMX)
      DO 10 I  = 1,NT-1
       IF(SI.LE.S(I+1)) GO TO 20
   10 CONTINUE
   20 CONTINUE
      DSP= S(I+1)-SI
      DSM= SI-S(I)
      DEL= S(I+1)-S(I)
      VI = SPF(I)*DSP*(DSP*DSP/DEL-DEL)/6.+
     &     SPF(I+1)*DSM*(DSM*DSM/DEL-DEL)/6.+
     &     V(I)*DSP/DEL+V(I+1)*DSM/DEL
C
C COMPUTE FIRST AND SECOND DERIVATIVES
C
       CAPA=SPF(I)*DSP/6.0
       CAPB=DSP**2/DEL-DEL
       CAPC=SPF(I+1)*DSM/6.0
       CAPD=DSM**2/DEL-DEL
       DCAPA=-SPF(I)/6.0
       DCAPB=-2.0*DSP/DEL
       DCAPC=SPF(I+1)/6.0
       DCAPD=2.0*DSM/DEL
       VPI=CAPA*DCAPB+CAPB*DCAPA+CAPC*DCAPD+CAPD*DCAPC
     &     -(V(I)-V(I+1))/DEL
       VPPI=2.0*(CAPA+CAPC)/DEL+2.0*DCAPA*DCAPB+2.0*DCAPC*DCAPD
50    CONTINUE
      RETURN
      END
C
      SUBROUTINE SUBPTS
C
C************************************************
C
C THIS ROUTINE DECREASES THE NO. OF POINTS ALONG
C THE ADAPTION LINE IF  REQUESTED ON INPUT
C
C CALLED BY: INITIAL
C
C CALLS : NONE
C
C************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      common/comxyz/x(id,jd,kd),y(id,jd,kd),z(id,jd,kd)
      common/comq/q(id,jd,kd,ndim)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM13/ISTEP,JSTEP,KSTEP,IINVERSE,JINVERSE,KINVERSE,TWOD
      COMMON/COM14/IJPLANE,IKPLANE,JKPLANE,MARCH,MARCHPL,SAVE,OK
      COMMON/COM18/SUB,LSTSUB,LENDSUB
      DIMENSION XT(IMX),YT(IMX),ZT(IMX),QT(IMX,NDIM)
      LOGICAL ISTEP,JSTEP,KSTEP,IINVERSE,JINVERSE,KINVERSE,TWOD
      LOGICAL IJPLANE,IKPLANE,JKPLANE,MARCH,MARCHPL,SAVE,OK
      INTEGER SUB
      character*50 string
C
C INITIALIZE START AND END LIMITS TO BE CONSISTENT
C WITH ADAPTION DOMAIN
C
      IF(LSTSUB.EQ.0) LSTSUB=IST
      IF(LENDSUB.EQ.0) LENDSUB=IEND
      IF(IINVERSE) THEN
       L1=LENDSUB
       LENDSUB=IMAX-LSTSUB+1
       LSTSUB=IMAX-L1+1
      END IF
      IF(LSTSUB.GT.LENDSUB) THEN
c      WRITE (6,1020)
c1020   FORMAT(/' NO POINTS SUBTRACTED')
       call warning("NO POINTS SUBTRACTED")
       GO TO 999
      END IF
      NUMSUB=(LENDSUB-LSTSUB)*SUB/(SUB+1)
      IMAXB=IMAX
      IMAX=IMAX-NUMSUB
      IF(IMAX.LT.10) THEN
c      WRITE(6,1000)
c1000   FORMAT(/,' CAUTION: SUB OPTION PRODUCES TOO FEW POINTS
c     C FOR ADAPTION')
       call warning("SUB OPTION PRODUCES TOO FEW POINTS FOR ADAPTION")
      END IF
C
C MOVE OVER FIRST REGION
C
      DO 500 K=1,KMAX
      DO 500 J=1,JMAX
       DO 50 I=1,LSTSUB
        XT(I)=X(I,J,K)
        YT(I)=Y(I,J,K)
        ZT(I)=Z(I,J,K)
        DO 30 N=1,NDIM
         QT(I,N)=Q(I,J,K,N)
30      CONTINUE
50     CONTINUE
C
C SUBTRACT  POINTS IN REQUESTED REGION
C
       DO 100 I=1,NUMSUB
        M=LSTSUB+I*(SUB+1)
        L=LSTSUB+I
        IF(M.GE.LENDSUB) GO TO 100
        XT(L)=X(M,J,K)
        YT(L)=Y(M,J,K)
        ZT(L)=Z(M,J,K)
        DO 70 N=1,NDIM
         QT(L,N)=Q(M,J,K,N)
70      CONTINUE
100    CONTINUE
```

```
C
C MOVE OVER LAST REGION
C
      DO 150 I=LENDSUB,IMAXB
       L=I-NUMSUB
       XT(L)=X(I,J,K)
       YT(L)=Y(I,J,K)
       ZT(L)=Z(I,J,K)
       DO 130 N=1,NDIM
        QT(L,N)=Q(I,J,K,N)
130    CONTINUE
150   CONTINUE
C
C RETURN DATA IN ORIGINAL ARRAYS
C
      DO 200 I=1,IMAX
       X(I,J,K)=XT(I)
       Y(I,J,K)=YT(I)
       Z(I,J,K)=ZT(I)
       DO 175 N=1,NDIM
        Q(I,J,K,N)=QT(I,N)
175    CONTINUE
200   CONTINUE
500  CONTINUE
     IEND=IEND-NUMSUB
900  CONTINUE
c     WRITE(6,1010) IMAXB,IMAX
     encode(50,1010,string) imaxb, imax
1010 FORMAT(' NO. OF PTS DECREASED FROM ',I3,' TO ',I3)
     call warning(string)
999  CONTINUE
     RETURN
     END
C
     SUBROUTINE SWAPINV
C
C*****************************************************
C
C THIS ROUTINE SWAPS DATA AROUND TO PROVIDE CONSISTENT
C ORDERING WITHIN CODE, REGARDLESS OF INPUT DEMANDS:
C THE CODE REQUIRES THAT KST<KEND, JST<JEND AND IST<IEND
C HENCE DATA NEEDS TO BE SWAPPED AT BEGINNING AND
C RESWAPPED AT END
C
C CALLED BY: INITIAL,OUTPUT
C
C CALLS: NONE
C
C*****************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      common/commxyz/x(id,jd,kd),y(id,jd,kd),z(id,jd,kd)
      common/comq/q(id,jd,kd,ndim)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM11/NFILT,INTER,MGSTEPS,MGPLS,GEOM,QFUN,NFLAG
      COMMON/COM13/ISTEP,JSTEP,KSTEP,IINVERSE,JINVERSE,KINVERSE,TWOD
      COMMON/COM14/IJPLANE,IKPLANE,JKPLANE,MARCH,MARCHPL,SAVE,OK
      DIMENSION XT(IMX),YT(IMX),ZT(IMX),QT(IMX,NDIM)
      LOGICAL ISTEP,JSTEP,KSTEP,IINVERSE,JINVERSE,KINVERSE,TWOD
      LOGICAL IJPLANE,IKPLANE,JKPLANE,MARCH,MARCHPL,SAVE,OK
      LOGICAL GEOM,QFUN
C
C I SWAP
C
      IF(IINVERSE) THEN
       NFLAG=-NFLAG
       DO 120 K=1,KMAX
        DO 100 J=1,JMAX
         DO 50 I=1,IMAX
          XT(I)=X(IMAX-I+1,J,K)
          YT(I)=Y(IMAX-I+1,J,K)
          ZT(I)=Z(IMAX-I+1,J,K)
          DO 30 N=1,NDIM
           QT(I,N)=Q(IMAX-I+1,J,K,N)
30        CONTINUE
50       CONTINUE
         DO 70 I=1,IMAX
          X(I,J,K)=XT(I)
          Y(I,J,K)=YT(I)
          Z(I,J,K)=ZT(I)
          DO 60 N=1,NDIM
           Q(I,J,K,N)=QT(I,N)
60        CONTINUE
70       CONTINUE
100     CONTINUE
120    CONTINUE
c       IST=IMAX-IST+1
c       IEND=IMAX-IEND+1
       itmp = iend
       iend=imax-ist+1
       ist=imax-itmp+1
      END IF
C
C J SWAP
C
      IF(JINVERSE) THEN
       NFLAG=-NFLAG
       DO 250 K=1,KMAX
        DO 200 I=1,IMAX
         DO 150 J=1,JMAX
          XT(J)=X(I,JMAX-J+1,K)
          YT(J)=Y(I,JMAX-J+1,K)
          ZT(J)=Z(I,JMAX-J+1,K)
          DO 130 N=1,NDIM
           QT(J,N)=Q(I,JMAX-J+1,K,N)
130       CONTINUE
150      CONTINUE
         DO 170 J=1,JMAX
          X(I,J,K)=XT(J)
          Y(I,J,K)=YT(J)
          Z(I,J,K)=ZT(J)
          DO 160 N=1,NDIM
           Q(I,J,K,N)=QT(J,N)
160       CONTINUE
170      CONTINUE
200     CONTINUE
250    CONTINUE
c       JST=JMAX-JST+1
c       JEND=JMAX-JEND+1
       jtmp = jend
       jend=jmax-jst+1
       jst=jmax-jtmp+1
      END IF
      IF(KINVERSE) THEN
       NFLAG=-NFLAG
       DO 400 J=1,JMAX
        DO 380 I=1,IMAX
         DO 320 K=1,KMAX
```



```
          XT(K)=X(I,J,KMAX-K+1)
          YT(K)=Y(I,J,KMAX-K+1)
          ZT(K)=Z(I,J,KMAX-K+1)
          DO 320 N=1,NDIM
           QT(K,N)=Q(I,J,KMAX-K+1,N)
320      CONTINUE
         DO 340 K=1,KMAX
          X(I,J,K)=XT(K)
          Y(I,J,K)=YT(K)
          Z(I,J,K)=ZT(K)
          DO 340 N=1,NDIM
           Q(I,J,K,N)=QT(K,N)
340      CONTINUE
380     CONTINUE
400    CONTINUE
c       KST=KMAX-KST+1
c       KEND=KMAX-KEND+1
       ktmp=kend
       kend=kmax-kst+1
       kst=kmax-ktmp+1
      END IF
      RETURN
      END
C
      SUBROUTINE SWAPXYZ(RSWAP)
C
C*************************************************
C
C CODE REQUIRES ADAPTING IN THE I DIRECTION,
C STEPPING ALONG J LINES AND MARCHING UP K PLANES
C HENCE THIS ROUTINE INTERCHANGES X,Y AND Z IF NEC.
C Q IS ALSO SWITCHED (INEFFICIENTLY, TO SAVE TEMPORARY
C STORAGE)
C
C CALLED BY: INITIAL,OUTPUT
C
C CALLS: NONE
C
C*************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      common/commxyz/x(id,jd,kd),y(id,jd,kd),z(id,jd,kd)
      common/comq/q(id,jd,kd,ndim)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM13/ISTEP,JSTEP,KSTEP,IINVERSE,JINVERSE,KINVERSE,TWOD
      COMMON/COM14/IJPLANE,IKPLANE,JKPLANE,MARCH,MARCHPL,SAVE,OK
      DIMENSION XT(ID,JD,KD)
      DIMENSION YT(ID,JD,KD),ZT(ID,JD,KD)
      LOGICAL ISTEP,JSTEP,KSTEP,IINVERSE,JINVERSE,KINVERSE,TWOD
      LOGICAL IJPLANE,IKPLANE,JKPLANE,MARCH,MARCHPL,SAVE,OK
      LOGICAL RSWAP
C
C STORE GRID INTO TEMP ARRAYS
C
      IMAXT=IMAX
      JMAXT=JMAX
      KMAXT=KMAX
      ISTT=IST
      IENDT=IEND
      JSTT=JST
      JENDT=JEND
      KSTT=KST
      KENDT=KEND
      DO 100 K=1,KMAX
       DO 100 J=1,JMAX
        DO 100 I=1,IMAX
         XT(I,J,K)=X(I,J,K)
         YT(I,J,K)=Y(I,J,K)
         ZT(I,J,K)=Z(I,J,K)
100   CONTINUE
C
C SWAP COORDINATES
C
      IF(IJPLANE .AND. ISTEP) THEN
       DO 200 K=1,KMAX
       DO 200 J=1,IMAX
       DO 200 I=1,JMAX
        X(I,J,K)=YT(J,I,K)
        Y(I,J,K)=XT(J,I,K)
        Z(I,J,K)=ZT(J,I,K)
200    CONTINUE
       DO 250 N=1,NDIM
        DO 225 K=1,KMAX
        DO 225 J=1,JMAX
        DO 225 I=1,IMAX
         XT(I,J,K)=Q(I,J,K,N)
225     CONTINUE
        DO 240 K=1,KMAX
        DO 240 J=1,IMAX
        DO 240 I=1,JMAX
         Q(I,J,K,N)=XT(J,I,K)
240     CONTINUE
250    CONTINUE
       IMAX=JMAXT
       JMAX=IMAXT
       JST=ISTT
       JEND=IENDT
       IST=JSTT
       IEND=JENDT
      END IF
      IF(IKPLANE .AND.KSTEP) THEN
       DO 300 K=1,JMAX
       DO 300 J=1,KMAX
       DO 300 I=1,IMAX
        X(I,J,K)=XT(I,K,J)
        Y(I,J,K)=ZT(I,K,J)
        Z(I,J,K)=YT(I,K,J)
300    CONTINUE
       DO 350 N=1,NDIM
        DO 325 K=1,KMAX
        DO 325 J=1,JMAX
        DO 325 I=1,IMAX
         XT(I,J,K)=Q(I,J,K,N)
325     CONTINUE
        DO 340 K=1,JMAX
        DO 340 J=1,KMAX
        DO 340 I=1,IMAX
         Q(I,J,K,N)=XT(I,K,J)
340     CONTINUE
350    CONTINUE
       JMAX=KMAXT
       KMAX=JMAXT
       JST=KSTT
       JEND=KENDT
       KST=JSTT
       KEND=JENDT
      END IF
      IF(JKPLANE.AND.JSTEP) THEN
```

```
      DO 400 K=1,KMAX
      DO 400 J=1,JMAX
      DO 400 I=1,KMAX
       X(I,J,K)=XT(K,J,I)
       Y(I,J,K)=YT(K,J,I)
       Z(I,J,K)=ZT(K,J,I)
400   CONTINUE
      DO 440 N=1,NDIM
       DO 420 K=1,KMAX
       DO 420 J=1,JMAX
       DO 420 I=1,IMAX
        XT(I,J,K)=Q(I,J,K,N)
420    CONTINUE
       DO 430 K=1,IMAX
       DO 430 J=1,JMAX
       DO 430 I=1,KMAX
        Q(I,J,K,N)=XT(K,J,I)
430    CONTINUE
440   CONTINUE
      IMAX=KMAXT
      KMAX=IMAXT
      IST=KSTT
      IEND=KENDT
      KST=ISTT
      KEND=IENDT
     END IF
     IF(IKPLANE .AND.ISTEP .AND. .NOT. RSWAP .OR.
     C    JKPLANE.AND.KSTEP.AND. RSWAP) THEN
      DO 450 K=1,JMAX
      DO 450 J=1,IMAX
      DO 450 I=1,KMAX
       X(I,J,K)=ZT(J,K,I)
       Y(I,J,K)=XT(J,K,I)
       Z(I,J,K)=YT(J,K,I)
450   CONTINUE
      DO 490 N=1,NDIM
       DO 460 K=1,KMAX
       DO 460 J=1,JMAX
       DO 460 I=1,IMAX
        XT(I,J,K)=Q(I,J,K,N)
460    CONTINUE
       DO 480 K=1,JMAX
       DO 480 J=1,IMAX
       DO 480 I=1,KMAX
        Q(I,J,K,N)=XT(J,K,I)
480    CONTINUE
490    CONTINUE
      IMAX=KMAXT
      JMAX=IMAXT
      KMAX=JMAXT
      IST=KSTT
      IEND=KENDT
      JST=ISTT
      JEND=IENDT
      KST=JSTT
      KEND=JENDT
     END IF
     IF(JKPLANE .AND. KSTEP.AND. .NOT. RSWAP .OR.
     C    IKPLANE .AND. ISTEP.AND. RSWAP) THEN
      DO 500 K=1,IMAX
      DO 500 J=1,KMAX
      DO 500 I=1,JMAX
       X(I,J,K)=YT(K,I,J)
       Y(I,J,K)=ZT(K,I,J)
       Z(I,J,K)=XT(K,I,J)
500   CONTINUE
      DO 540 N=1,NDIM
       DO 520 K=1,KMAX
       DO 520 J=1,JMAX
       DO 520 I=1,IMAX
        XT(I,J,K)=Q(I,J,K,N)
520    CONTINUE
       DO 530 K=1,IMAX
       DO 530 J=1,KMAX
       DO 530 I=1,JMAX
        Q(I,J,K,N)=XT(K,I,J)
530    CONTINUE
540   CONTINUE
      IMAX=JMAXT
      JMAX=KMAXT
      KMAX=IMAXT
      IST=JSTT
      IEND=JENDT
      JST=KSTT
      JEND=KENDT
      KST=ISTT
      KEND=IENDT
     END IF
     RETURN
     END
C
      SUBROUTINE SWAPZD(IO)
C
C*****************************************************
C
C IF 2-D DATASETS, TURN INTO SINGLE PLANE 3-D DATASET
C
CALLED BY: INITIAL,OUTPUT
C
CALLS: NONE
C
C*****************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      common/consxyz/x(id,jd,kd),y(id,jd,kd),z(id,jd,kd)
      common/comq/q(id,jd,kd,ndim)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM13/ISTEP,JSTEP,KSTEP,IINVERSE,JINVERSE,KINVERSE,TWOD
      LOGICAL ISTEP,JSTEP,KSTEP,IINVERSE,JINVERSE,KINVERSE,TWOD
C
C ON INPUT, MOVE Q4 TO Q5 ETC
C
      IF (IO.EQ.1) THEN
       KMAX=1
       KEND=1
       IF(.NOT. JSTEP) ISTEP=.TRUE.
       DO 100 I=1,IMAX
       DO 100 J=1,JMAX
        Z(I,J,1)=.0
        DO 50 N=NDIM,5,-1
         Q(I,J,1,N)=Q(I,J,1,N-1)
50      CONTINUE
        Q(I,J,1,4)=0.0
100    CONTINUE
      END IF
C
C ON OUTPUT, MOVE Q5 TO Q4 ETC
C
```



```
      IF(IO.EQ.2) THEN
       DO 200 I=1,IMAX
       DO 200 J=1,JMAX
       DO 200 N=4,NDIM-1
        Q(I,J,1,N)=Q(I,J,1,N+1)
200    CONTINUE
      END IF
      RETURN
      END
C
      SUBROUTINE TORCOF(L,JK,JKST,JKEND,MGNOS,MARCHJK)
C
C*****************************************************
C
C THIS ROUTINE COMPUTES THE COEFFS THAT ARE USED TO
C FIND THE TWO TORSION VECTORS, L=1 IMPLIES FROM J-1
C L=2 IMPLIES FROM K-1
C
CALLED BY: MAIN
C
CALLS: NONE
C
C*****************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      COMMON/COM10/CLAM(2),CT(2),TN(2),CHN(2),CLAMN(2),CTN(2),TNN(2)
      COMMON/COM15/CONV,MAXITS,NOUP,LNSING,PLSING,BK1
      COMMON/COM16/ORTHS,ORTHE
      LOGICAL MARCHJK,NOUP,ORTHS,ORTHE
      INTEGER PLSING
C
C SET UP CONSTANTS FOR TORSION COEFF:
C
C IF MERGING REGION, COMPUTE PROPORTIONAL COEFF (CHN)
C ENFORCE MORE ORTHOGONALITY AT WALLS: (CAN BE OVERRIDDEN
C BY INPUT)
C    IF AT EITHER WALL BOUNDARY, INCREASE CLAM
C    IF AT A START WALL, INCREASE U
C    IF AT END BOUNDARY, INCREASE B
C
      CLAMN(L)=CLAM(L)
      CTN(L)=CT(L)
      TNN(L)=TN(L)
      CHN(L)=.0
      IF(JK.LT.MGNOS) THEN
       CHN(L)=FLOAT(MGNOS-JK)/FLOAT(MGNOS-JKST)
       CLAMN(L)=CLAM(L)*(1.0+CHN(L)*5.0)
      ELSE
       IF(ORTHS) THEN
        IF(JK.EQ.2) THEN
         TNN(L)=.0
         CTN(L)=.25*CT(L)
         CLAMN(L)=CLAM(L)*10.
        END IF
        IF(JK.EQ.3) THEN
         TNN(L)=.5*TN(L)
         CLAMN(L)=CLAM(L)*3.0
         CTN(L)=.5*CT(L)
        END IF
       END IF
      END IF
C
C INCREASE CLAM AT OUTER WALL
C
      IF (.NOT. MARCHJK) THEN
       IF (ORTHE) THEN
       IF(JK.EQ.JKEND-2) THEN
        CLAMN(L)=2.0*CLAM(L)
       END IF
       IF(JK.EQ.JKEND-1) THEN
        CLAMN(L)=5.0*CLAM(L)
        TNN(L)=TN(L)+(1.0-TN(L))/3.0
        CTN(L)=.5*CT(L)
       END IF
       IF(JK.EQ.JKEND) THEN
        CLAMN(L)=10.0*CLAM(L)
        TNN(L)=TN(L)+(1.0-TN(L))*2.0/3.0
        CTN(L)=.25*CT(L)
       END IF
      END IF
10    CONTINUE
      RETURN
      END
C
      SUBROUTINE TORSION(J,K)
C
C*****************************************************
C
C THIS ROUTINE FINDS BOTH T (TORSION) VECTORS
C AS A FUNCTION OF VECTORS U,B AND E,
C AND HENCE S' FOR J-1,K AND S* FOR J,K-1,
C  REQUIRED IN THE SOLUTION EQUATION.
C
C CALLED BY: MAIN
C
C CALLS: ADDV,CROSSV
C
C*****************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      COMMON/COM3/SS(IMX),SMS(IMX),DS(IMX),SN(IMX),SNM(IMX),SNMK(IMX)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM7/COSE(IMX,3),COSU(IMX,3),COSB(IMX,3)
      COMMON/COM8/XJ(IMX,3,3),YJ(IMX,3,3),ZJ(IMX,3,3)
      COMMON/COM9/SP(IMX),SPFL(IMX),DAP(IMX),DAPPL(IMX),
     C            NEDGE,MG1,MG2
      COMMON/COM10/CLAM(2),CT(2),TN(2),CHN(2),CLAMN(2),CTN(2),TNN(2)
      COMMON/COM11/NFILT,INTER,MGSTEPS,MGPLS,GEOM,QFUN,NFLAG
      COMMON/COM20/COSEK(IMX,3),COSUK(IMX,3),COSBK(IMX,3)
      DIMENSION AAP(IMX),ICROSS(IMX),COSB1K(IMX,3)
      DIMENSION COST(IMX,3),COSB1(IMX,3),DUM2(IMX),COSN(IMX,3)
      LOGICAL GEOM,QFUN
C
C FIND INDEX OF DS INTERVAL THAT INTERSECTS EACH COSB
C
      DO 60 I=1,NIPTS-1
       DO 50 M=1,3
        COSB1(I,M)=COSB(I,M)
        COSB1K(I,M)=COSBK(I,M)
50     CONTINUE
60    CONTINUE
      IF(J.EQ.JST) THEN
       DO 100 I=1,NIPTS
        SP(I)=.0
100    CONTINUE
      ELSE
       CALL CROSSV(XJ(1,2,2),YJ(1,2,2),ZJ(1,2,2),
```

```
     C    XJ(1,1,2),YJ(1,1,2),ZJ(1,1,2),DS,COSB1,AAP,DUM2,ICROSS,J)
C
C CHOOSE CORRECT UNIT NORMAL VECTOR TO USE FOR EACH I POINT AT J-1
C
      DO 150 I=2,NIPTS-1
       L=ICROSS(I)
       IF((AAP(I).LT.DS(L)/2.0 .AND. L.NE.1) .OR.L.EQ.NIPTS-1) L=L-1
       CALL ADDV(COSB1(L,1),COSB1(L,2),COSB1(L,3),.5,
     C      COSB1(L+1,1),COSB1(L+1,2),COSB1(L+1,3),.5,
     C      COSB(I,1),COSB(I,2),COSB(I,3))
150   CONTINUE
C
C ADD U AND B, TO GIVE NORMAL VECTOR, N.
C
      THM1=1.0-THM(1)
      DO 200 I=2,NIPTS-1
       CALL ADDV(COSU(I,1),COSU(I,2),COSU(I,3),THM1,
     C           COSB(I,1),COSB(I,2),COSB(I,3),THM(1),
     C           COSN(I,1),COSN(I,2),COSN(I,3))
200   CONTINUE
C
C FIND T, BY ADDING N AND E, PROPORTION BY CT.
C
      ACT=CTM(1)*(1.0-CHM(1))+CHM(1)
      ACT1=1.0-ACT
      DO 240 I=2,NIPTS-1
       CALL ADDV(COSE(I,1),COSE(I,2),COSE(I,3),ACT,
     C           COSN(I,1),COSN(I,2),COSN(I,3),ACT1,
     C           COST(I,1),COST(I,2),COST(I,3))
240   CONTINUE
C
C COMPUTE S'
C
      CALL CROSSV(XJ(1,2,2),YJ(1,2,2),ZJ(1,2,2),
     C    XJ(1,1,2),YJ(1,1,2),ZJ(1,1,2),DS,COST,AAP,DAP,ICROSS,J)
      SP(1)=.0
      SP(NIPTS)=SN(NIPTS)
      DO 250 I=2,NIPTS-1
       SP(I)=SN(ICROSS(I))+AAP(I)
250   CONTINUE
C
C CHECK FOR ANY CROSS OVER OF S'
C AND TRY TO CORRECT FOR MINOR ERRORS
C IF NOT, HOPE FOR THE BEST!
C
      DO 260 L=1,10
       SWFL=.0
       DO 255 I=1,NIPTS-1
        IF(SP(I+1).LT.SP(I)) THEN
         SPT=SP(I+1)
         SP(I+1)=SP(I)
         SP(I)=SPT
         SWFL=1.0
        END IF
255    CONTINUE
       IF (SWFL.EQ..0) GO TO 270
260   CONTINUE
270   CONTINUE
      END IF
C
C DO THE SAME FOR COMPUTING SPPL
C
      IF(K.EQ.KST) THEN
       DO 275 I=1,NIPTS
        SPPL(I)=.0
275   CONTINUE
      ELSE
       CALL CROSSV(XJ(1,2,2),YJ(1,2,2),ZJ(1,2,2),
     C    XJ(1,2,1),YJ(1,2,1),ZJ(1,2,1),DS,COSB1K,AAP,DUM2,ICROSS,J)
       DO 280 I=2,NIPTS-1
        L=ICROSS(I)
        IF((AAP(I).LT.DS(L)/2.0 .AND. L.NE.1) .OR.L.EQ.NIPTS-1) L=L-1
        CALL ADDV(COSB1K(L,1),COSB1K(L,2),COSB1K(L,3),.5,
     C      COSB1K(L+1,1),COSB1K(L+1,2),COSB1K(L+1,3),.5,
     C      COSBK(I,1),COSBK(I,2),COSBK(I,3))
280   CONTINUE
      THM2=1.0-THM(2)
      DO 300 I=2,NIPTS-1
       CALL ADDV(COSUK(I,1),COSUK(I,2),COSUK(I,3),THM2,
     C           COSBK(I,1),COSBK(I,2),COSBK(I,3),THM(2),
     C           COSN(I,1),COSN(I,2),COSN(I,3))
300   CONTINUE
      ACT2=CTM(2)*(1.0-CHM(2))+CHM(2)
      ACT21=1.0-ACT2
      DO 340 I=2,NIPTS-1
       CALL ADDV(COSEK(I,1),COSEK(I,2),COSEK(I,3),ACT2,
     C           COSN(I,1),COSN(I,2),COSN(I,3),ACT21,
     C           COST(I,1),COST(I,2),COST(I,3))
340   CONTINUE
C
C COMPUTE S* FOR THE PLANE
C
      CALL CROSSV(XJ(1,2,2),YJ(1,2,2),ZJ(1,2,2),
     C   XJ(1,2,1),YJ(1,2,1),ZJ(1,2,1),DS,COST,AAP,DAPPL,ICROSS,J)
      SPPL(1)=.0
      SPPL(NIPTS)=SN(NIPTS)
      DO 350 I=2,NIPTS-1
       SPPL(I)=SN(ICROSS(I))+AAP(I)
350   CONTINUE
C
C SIMILARLY CHECK FOR ANY CROSS OVER OF S* PL
C
      DO 360 L=1,10
       SWFL=.0
       DO 355 I=1,NIPTS-1
        IF(SPPL(I+1).LT.SPPL(I)) THEN
         SPT=SPPL(I+1)
         SPPL(I+1)=SPPL(I)
         SPPL(I)=SPT
         SWFL=1.0
        END IF
355    CONTINUE
       IF (SWFL.EQ..0) GO TO 370
360   CONTINUE
370   CONTINUE
      END IF
      RETURN
      END
C
      SUBROUTINE UNITV(X1,Y1,Z1,X2,Y2,Z2,DIRCX,DIRCY,DIRCZ)
C
C*************************************************************
C
C THIS ROUTINE FINDS THE DIRECTION COSINES OF THE
C UNIT VECTOR JOINING (X1,Y1,Z1) TOWARDS (X2,Y2,Z2)
C
C CALLED BY: CROSSV,DETERM,SETUP,TORSION
C
```



```
C CALLS: NONE
C
C*************************************************************
C
      DIRCX=X2-X1
      DIRCY=Y2-Y1
      DIRCZ=Z2-Z1
      VMOD=SQRT(DIRCX**2+DIRCY**2+DIRCZ**2)
      IF(VMOD.NE..0) THEN
       DIRCX=DIRCX/VMOD
       DIRCY=DIRCY/VMOD
       DIRCZ=DIRCZ/VMOD
      END IF
      RETURN
      END
C
      SUBROUTINE UPDATE(J,K)
C
C*************************************************************
C
C THIS ROUTINE IS CALLED AFTER LINE HAS CONVERGED,
C IT RE-EVALUATES X,Y,Z,Q FROM CONVERGED S
C
C CALLED BY: MAIN
C
C CALLS: INTXYZQ
C
C*************************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      common/comxyz/x(id,jd,kd),y(id,jd,kd),z(id,jd,kd)
      common/comq/q(id,jd,kd,ndim)
      COMMON/COM3/SS(IMX),SNS(IMX),DS(IMX),SN(IMX),SBN(IMX),SNNK(IMX)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM8/XJ(IMX,3,3),YJ(IMX,3,3),ZJ(IMX,3,3)
      COMMON/COM13/ISTEP,JSTEP,KSTEP,IINVERSE,JINVERSE,KINVERSE,TWOD
      COMMON/COM14/IJPLANE,IKPLANE,JKPLANE,MARCH,MARCHPL,SAVE,OK
      COMMON/COM15/CONV,MAXITS,NOUP,LHSING,PLSING,BK1
      DIMENSION QJ(IMX,NDIM)
      LOGICAL ISTEP,JSTEP,KSTEP,IINVERSE,JINVERSE,KINVERSE,NOUP,TWOD
      LOGICAL IJPLANE,IKPLANE,JKPLANE,MARCH,MARCHPL,SAVE,OK
      INTEGER PLSING
C
      CALL INTXYZQ(J,K,2,2,SS,SN,QJ)
C
C STORE INTO ORIGINAL GRID
C
      DO 100 I=2,NIPTS-1
       N=IST+I-1
       X(N,J,K)=XJ(I,2,2)
       Y(N,J,K)=YJ(I,2,2)
       Z(N,J,K)=ZJ(I,2,2)
       DO 100 M=1,NDIM
        Q(N,J,K,M)=QJ(I,M)
100   CONTINUE
      DO 400 I=1,NIPTS
       SNN(I)=SN(I)
400   CONTINUE
      RETURN
      END
C
      SUBROUTINE VMERGE(DIRV,LST,LEND)
C
C*************************************************************
C
C THIS ROUTINE MERGES IN END VECTORS OF DIRV ARRAY
C EG. DIRV(2)=DIRV(1)*.75+DIRV(2)*.25, USING VECTOR ADDITION
C
C CALLED BY:SETUPJ,SETUPK
C
C CALLS: ADDV
C
C*************************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      DIMENSION DIRV(IMX,3)
C
C MERGE 1ST PT
C
      IF(LST.NE.0) THEN
       L=LST
       CALL ADDV(DIRV(L,1),DIRV(L,2),DIRV(L,3),.25,
     C           DIRV(L+2,1),DIRV(L+2,2),DIRV(L+2,3),.75,
     C           DIRV(L+3,1),DIRV(L+3,2),DIRV(L+3,3))
       CALL ADDV(DIRV(L,1),DIRV(L,2),DIRV(L,3),.5,
     C           DIRV(L+2,1),DIRV(L+2,2),DIRV(L+2,3),.5,
     C           DIRV(L+2,1),DIRV(L+2,2),DIRV(L+2,3))
       CALL ADDV(DIRV(L,1),DIRV(L,2),DIRV(L,3),.75,
     C           DIRV(L+1,1),DIRV(L+1,2),DIRV(L+1,3),.25,
     C           DIRV(L+1,1),DIRV(L+1,2),DIRV(L+1,3))
      END IF
C
C MERGE END
C
      IF(LEND.NE.0) THEN
       L=LEND
       CALL ADDV(DIRV(L,1),DIRV(L,2),DIRV(L,3),.75,
     C           DIRV(L-1,1),DIRV(L-1,2),DIRV(L-1,3),.25,
     C           DIRV(L-1,1),DIRV(L-1,2),DIRV(L-1,3))
       CALL ADDV(DIRV(L,1),DIRV(L,2),DIRV(L,3),.5,
     C           DIRV(L-2,1),DIRV(L-2,2),DIRV(L-2,3),.5,
     C           DIRV(L-2,1),DIRV(L-2,2),DIRV(L-2,3))
       CALL ADDV(DIRV(L,1),DIRV(L,2),DIRV(L,3),.25,
     C           DIRV(L-3,1),DIRV(L-3,2),DIRV(L-3,3),.75,
     C           DIRV(L-3,1),DIRV(L-3,2),DIRV(L-3,3))
      END IF
      RETURN
      END
C
      SUBROUTINE WALLS(JW,K)
C
C*************************************************************
C
C THIS ROUTINE COMPUTES THE THE START AND END WALL
C GEOMETRY FUNCTIONS, BASED ON THE INITIAL GRID
C
C CALLED BY: FBAR
C
C CALLS: CSPLIN,SPEVAL
C
C*************************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      common/comxyz/x(id,jd,kd),y(id,jd,kd),z(id,jd,kd)
      common/comq/q(id,jd,kd,ndim)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM17/FQ(IMX),FC(IMX),FCS(IMX),SMSS(IMX),HXFG,FGW,FQW
      DIMENSION DSS(IMX),SSS(IMX)
```

```
      DIMENSION SPSX(IMX),SPSY(IMX)
      DIMENSION XWALLS(IMX),YWALLS(IMX)
C
C STORE WALL COORDINATES IN SINGLE DIM ARRAYS
C
      DO 25 I=1,NIPTS
       XWALLS(I)=X(IST-1+I,JW,K)
       YWALLS(I)=Y(IST-1+I,JW,K)
25    CONTINUE
C
C FIND SS ARRAY AT BOTH BOUNDARIES
C
      SSS(1)=.0
      DO 50 I=1,NIPTS-1
       I=IST+I-1
       SSS(I)=SQRT((X(I+1,JW,K)-X(I,JW,K))**2+
     C             (Y(I+1,JW,K)-Y(I,JW,K))**2+
     C             (Z(I+1,JW,K)-Z(I,JW,K))**2)
        SSS(I+1)=SSS(I)+SSS(I)
50    CONTINUE
C
C FIND MID-PTS FOR INTERPOLATION ROUTINE
C
      DO 70 I=1,NIPTS-1
       SMSS(I)=(SSS(I+1)+SSS(I))*.5
70    CONTINUE
C
C FIND SPLINE COEFFS
C
      CALL CSPLIN(NIPTS,SSS,XWALLS,SPSX)
      CALL CSPLIN(NIPTS,SSS,YWALLS,SPSY)
C
C OBTAIN DERIVATIVES FOR RADIUS OF CURVATURE
C
      DO 100 I=1,NIPTS-1
       CALL SPEVAL(NIPTS,SSS,XWALLS,SPSX,SSS(I),XI,XPI,XPPI)
       CALL SPEVAL(NIPTS,SSS,YWALLS,SPSY,SSS(I),YI,YPI,YPPI)
       FGS(I)=ABS(XPI*YPPI-YPI*XPPI)/(XPI**2+YPI**2)**1.5
100   CONTINUE
C
C FIND LOCATION OF MAX FG FOR ASPECT RATIO CALCULATION
C
      MXFG=2
      DO 200 I=3,NIPTS-1
       IF(FGS(I).GT.FGS(MXFG)) MXFG=I
200   CONTINUE
      RETURN
      END
C
      SUBROUTINE WRITOUT
C
C**********************************************************
C
C THIS ROUTINE WRITES OUT THE DATASETS
C AND RESTORES 2-D DATASETS TO THEIR ORIGINAL FORM
C
CALLED BY OUTPUT
C
CALLS: NONE
C
C**********************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      common/comxyz/x(id,jd,kd),y(id,jd,kd),z(id,jd,kd)
      common/comq/q(id,jd,kd,ndim)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM12/FSMACH,ALPHA,RE,TIME,NITG,NITQ
      COMMON/COM13/ISTEP,JSTEP,KSTEP,IINVERSE,JINVERSE,KINVERSE,TWOD
      LOGICAL ISTEP,JSTEP,KSTEP,IINVERSE,JINVERSE,KINVERSE,TWOD
      IF (.NOT. TWOD) THEN
       WRITE(NITG) IMAX,JMAX,KMAX
       WRITE(NITG) (((X(I,J,K),I=1,IMAX),J=1,JMAX),K=1,KMAX),
     C             (((Y(I,J,K),I=1,IMAX),J=1,JMAX),K=1,KMAX),
     C             (((Z(I,J,K),I=1,IMAX),J=1,JMAX),K=1,KMAX)
       WRITE(NITQ) IMAX,JMAX,KMAX
       WRITE(NITQ) FSMACH,ALPHA,RE,TIME
       WRITE(NITQ) ((((Q(I,J,K,N),I=1,IMAX),J=1,JMAX),
     C               K=1,KMAX),N=1,NDIM)
      ELSE
       WRITE(NITG) IMAX,JMAX
       WRITE(NITG) ((X(I,J,1),I=1,IMAX),J=1,JMAX),
     C             ((Y(I,J,1),I=1,IMAX),J=1,JMAX)
       WRITE(NITQ) IMAX,JMAX
       WRITE(NITQ) FSMACH,ALPHA,RE,TIME
       WRITE(NITQ) (((Q(I,J,1,N),I=1,IMAX),J=1,JMAX),N=1,NDIM-1)
      END IF
      RETURN
      END
C
      SUBROUTINE WTEDGE(J,K)
C
C**********************************************************
C
C THIS ROUTINE TRIES TO ENFORCE THE EDGE WEIGHTING FUNCTION
C TO BE A FUNCTION OF DLENGS, DLENGE, TO PROVIDE BETTER
C CONTINUITY BY OVERRIDING FB (MATCH FOR SINGULAR LINE:
C WDE,WDS NEED TO BE SAME ON NEXT LINE)
C
C CALLED BY: MAIN,LINE1
C
C CALLS: DLENG
C
C**********************************************************
C
      PARAMETER(id=150,jd=150,kd=150,imx=150,NDIM=5)
      COMMON/COM2/F(IMX),FB(IMX),WEIGHT(IMX),A,B,WDS,WDE
      COMMON/COM3/SS(IMX),SMS(IMX),DS(IMX),SH(IMX),SHM(IMX),SHMK(IMX)
      COMMON/COM4/IMAX,JMAX,KMAX,IST,IEND,JST,JEND,KST,KEND,NIPTS
      COMMON/COM6/DSMAX,DSMIN,WT(IMX),DLENGS,DLENGE,WDSS,WDES
      COMMON/COM15/CONV,MAXITS,NOUP,LNSING,PLSING,BK1
      LOGICAL NOUP
      INTEGER PLSING
      IF(J.EQ.JEND) THEN
       WDS=.0
       WDE=.0
       GO TO 900
      END IF
      IF(J.NE.0 .AND. LNSING.EQ.J .AND. K.NE.KST) THEN
       WDE=WDES
       WDS=WDSS
       GO TO 900
      END IF
C
C DETERMINE FUNCTION OF DLENG FOR NEXT J LINE
C
      CALL DLENG(J+1,K)
C
C FIND AVERAGE DS*W FOR INTERNAL POINTS
```



```
C
      WTSUM=.0
      DO 100 I=5,NIPTS-5
       WTSUM=WTSUM+DS(I)*WEIGHT(I)
100   CONTINUE
      WTSUM=WTSUM/(NIPTS-9)
C
C AT IST (PREVENT WDS FROM BEING TOO LARGE
C  WHEN DLENGS IS VERY SMALL)
C
      WDS=WTSUM/DLENGS
      WDMAX=(1.0+A)*10.0
      IF(WDS.GT. WDMAX) WDS=WDMAX
C
C AT IEND
C
      WDE=WTSUM/DLENGE
      IF(WDE.GT.WDMAX) WDE=WDMAX
      IF(LNSING.EQ.J.AND.K.EQ.KST) THEN
       WDES=WDE
       WDSS=WDS
      END IF
900   CONTINUE
      RETURN
      END



      subroutine priVar()

c  check variable initialization

      parameter(id=150,jd=150,kd=150,imx=150,ndim=5)
      common/comxyz/x(id,jd,kd),y(id,jd,kd),z(id,jd,kd)
      common/comq/q(id,jd,kd,ndim)
      common/com2/f(imx),fb(imx),weight(imx),a,b,wds,wde
      common/com3/ss(imx),sms(imx),ds(imx),sh(imx),shm(imx),shmk(imx)
      common/com4/imax,jmax,kmax,ist,iend,jst,jend,kst,kend,nipts
      common/com5/rdsmax,rdsmin,indq,iq(ndim+3)
      common/com6/dsmax,dsmin,wt(imx),dlengs,dlenge,wdss,wdes
      common/com7/cose(imx,3),cosu(imx,3),cosb(imx,3)
      common/com8/xj(imx,3,3),yj(imx,3,3),zj(imx,3,3)
      common/com9/sp(imx),appl(imx),dsp(imx),dappl(imx),
     c            nedge,mql,mq2
      common/com10/clam(2),ct(2),tn(2),cnm(2),clamw(2),ctm(2),tnm(2)
      common/com11/nfilt,inter,mgsteps,mgple,geom,qfun,nflag
      common/com12/fsmach,alpha,re,time,nitg,nitq
      common/com13/istep,jstep,kstep,iinverse,jinverse,kinverse,twod
      common/com14/ijplane,ikplane,jkplane,smrch,smrchpl,save,ok
      common/com15/conv,maxits,noup,lnsing,plsing,bk1
      common/com16/orths,orthe
      common/com17/fq(imx),fq(imx),fqs(imx),smss(imx),mxfg,fqw,fqw
      common/com18/sub,lstsub,lendsub
      common/com19/add,lstadd,lendadd
      common/com20/cosek(imx,3),cosuk(imx,3),coshk(imx,3)
      logical geom,qfun,noup,orths,orthe,nomore
      logical istep,jstep,kstep,iinverse,jinverse,kinverse,twod
      logical ijplane,ikplane,jkplane,smrch,smrchpl,save,ok
      integer add,sub,plsing

      write(6,100) ist, iend
100   formt("ist = ",i5,",     iend = ",i5)
      write(6,101) jst, jend
101   format("jst = ",i5,",     jend = ",i5)
      write(6,102) kst, kend
102   format("kst = ",i5,",     kend = ",i5)
      write(6,103) istep, jstep, kstep
103   format("istep = ",l,",     jstep = ",l,",     kstep = ",l)
      write(6,104) ijplane, jkplane, ikplane
104   format("ijplane = ",l,", jkplane = ",l,", ikplane = ",l)
      write(6,105) rdsmax, rdsmin
105   format("rdsmax = ",f10.5,", rdsmin = ",f10.5)
      write(6,106) clam(1), clam(2)
106   format("clam(1) = ",f10.5,", clam(2) = ",f10.5)
      write(6,107) nedge
107   format("nedge = ",i5)
      write(6,108) ct(1), ct(2)
108   format("ct(1) = ",f10.5,", ct(2) = ",f10.5)
      write(6,109) mgsteps, mgple
109   format("mgsteps = ",i5,", mgple = ",i5)
      write(6,110) lnsing, plsing
110   format("lnsing = ",i5,", plsing = ",i5)
      write(6,111) indq
111   format("indq = ",i5)
      write(6,112) noup
112   format("noup = ",l)
      write(6,113) smrch, smrchpl
113   format("smrch = ",l,", smrchpl = ",l)
      write(6,114) add, lstadd, lendadd
114   format("add = ",i5,", lstadd = ",i5,", lendadd = ",i5)
      write(6,115) sub, lstsub, lensub
115   format("sub = ",i5,", lstsub = ",i5,", lendsub = ",i5)
      write(6,116) save
116   format("save = ",l)
      write(6,117) orths, orthe
117   format("orths = ",l,", orthe = ",l)
      write(6,118) nfilt
118   format("nfilt = ",i5)
      write(6,119) geom
119   format("geom = ",l)
      write(6,120) qfun
120   format("qfun = ",l)
      write(6,121) inter
121   format("inter = ",i5)
      write(6,122) twod
122   format("twod = ",l)
      write(6,123) ok
123   formt("ok = ",l)
      write(6,124) fqw, fqw
124   format("fqw = ",f10.5,", fqw = ",f10.5)
      write(6,125) iinverse, jinverse, kinverse
125   format("iinverse = ",l,", jinverse = ",l,", kinverse = ",l)
      write(6,126) tn(1), tn(2)
126   format("tn(1) = ",f10.5,", tn(2) = ",f10.5)
      write(6,127) conv
127   format("conv = ",f10.5)
      write(6,128) mql, mq2
128   formt("mql = ",i5,", mq2 = ",i5)
      write(6,129) maxits
129   formt("maxits = ",i5)
      write(6,130) nflag
130   format("nflag = ",i5)
      write(6,131) wds, wde
131   format("wds = ",f10.5,", wde = ",f10.5)

      return
      end
```

```
/*------------------------------ INCLUDES ----------------------------------*/

#include        <stdio.h>       /* for NULL etc.                            */
#include        <stdlib.h>      /* for atoi()                               */
#include        <string.h>      /* for string stuff                         */
#include        <ctype.h>       /* for isdigit() etc.                       */
#include        <math.h>        /* for atof()                               */
#include        <fld_pan.h>     /* for GRID_SCALAR_VECTOR_TYPEOUT           */
#include        <panel_utils.h> /* for typedefs and FLOAT_STRING_FORMAT     */
#include        <fast_cmap.h>   /* for init_fast_cmap()                     */
#include        <Object.h>      /* for OBJECT_NAME_LENGTH, etc.             */
#include        <grid_surface.h>/* for Grid_Surface typedef                 */
#include        <fast_error.h>  /* for ERROR() macro                        */
#include        <fast_memory.h> /* for DDIM3() and DDIM4() macros           */
#include        <fld_list.h>    /* for SCALAR, NUM_VARS etc                 */
#include        <get_data.h>    /* for req_*()                              */
#include        <fast_cmap.h>   /* for MAX_SCMAPS & colormap functions      */
#include        <Module.h>      /* for scripting stuff                      */
#include        <ViewI.h>       /* for locking/unlocking                    */
#include        "funcs.h"       /* for function declarations                */
#include        "sagerSupport.h"/* for sagerSupport                         */

void initialize_all_sage_parameters_to_default() {

/*  The following are globally defined variables from SUBROUTINE INITIAL. */

/*  SET INPUT DEFAULT VALUES  */

/*  The followin variables have fixed values for sage:  */

   com13_.istep = FALSE;
   com13_.jstep = TRUE;
   com13_.kstep = FALSE;
   com13_.iinverse = FALSE;
   com13_.jinverse = FALSE;
   com13_.kinverse = FALSE;
   com14_.ijplane = TRUE;
   com14_.jkplane = FALSE;
   com14_.ikplane = FALSE;

/*  End of fixed values for sage  */

   com4_.ist = 1;
   com4_.iend = 0;
   com4_.jst = 1;
   com4_.jend = 0;
   com4_.kst = 1;
   com4_.kend = 0;
   com13_.istep = FALSE;
   com13_.jstep = TRUE;
   com13_.kstep = FALSE;
   com14_.ijplane = TRUE;
   com14_.jkplane = FALSE;
   com14_.ikplane = FALSE;
   com5_.rdsmax = RDSMAX_INI;
   com5_.rdsmin = RDSMIN_INI;
   com10_.clam[0] = CLAM1_INI;
   com10_.clam[1] = CLAM2_INI;
   com9_.nedge = NEDGE_INI;
   com10_.ct[0] = CT1_INI;
   com10_.ct[1] = CT2_INI;
   com11_.mgsteps = MGSTEPS_INI;
   com11_.mgpls = MGPLS_INI;
   com15_.lssing = LSSING_INI;
   com15_.plsing = PLSING_INI;
   com5_.indq = INDQ_INI;
   com15_.noup = FALSE;
   com14_.marchpl = MARCHPL_INI;
   com19_.add = ADD_INI;
   com19_.latadd = 0;
   com19_.lendadd = 0;
   com18_.sub = SUB_INI;
   com18_.latsub = 0;
   com18_.lendsub = 0;
   com14_.save = FALSE;
   com16_.orths = ORTHS_INI;
   com16_.orthe = ORTHE_INI;
   com11_.nfilt = NFILT_INI;
   com11_.geom = GEOM_INI;
   com11_.qfun = QFUN_INI;
   com11_.inter = INTER_INI;
   com13_.twod = FALSE;

/*  SET OTHER START VALUES  */

   com14_.ok = TRUE;
   com17_.fqw = 1.0;
   com17_.fqw = 0.0;
   com13_.iinverse = FALSE;
   com13_.jinverse = FALSE;
   com13_.kinverse = FALSE;
   com10_.tn[0] = 0.5;
   com10_.tn[1] = 0.5;
   com15_.conv = 0.001;
   com9_.mg1 = MG1_INI;
   com9_.mg2 = MG2_INI;
   com15_.maxits = 20;
   com11_.nflag = 1;
   com2_.wds = 0.0;
   com2_.wde = 0.0;
}


Panel
   *panel_sage_main;

float
   panel_sage_x,
   panel_sage_y;


void SagerPanel(p, x, y)
Panel   *p;                     /* Panel to add actuators to    */
float   x;                      /* initial x position of acts   */
float   y;                      /* initial y position of acts   */
{
   void Beg_adaptpts(Actuator*);
   void Beg_addpts(Actuator*);
   void Beg_subpts(Actuator*);
   void Beg_undo(Actuator*);
   void Beg_remove(Actuator*);
   void Intergize_add(Actuator*);
   void Intergize_mgsteps(Actuator*);
   void Intergize_mgpls(Actuator*);
   void Intergize_sub(Actuator*);
   void Intergize_nedge(Actuator*);
   void Intergize_indq(Actuator*);
```

```
void Intergize_nfilt(Actuator*);
void Intergize_inter(Actuator*);
void Intergize_mg1(Actuator*);
void Intergize_mg2(Actuator*);
void Intergize_lnsing(Actuator*);
void Intergize_plsing(Actuator*);
void Set_com14_march();
void Set_com14_marchpl();
void Set_com16_orths();
void Set_com16_orthe();
void Set_com11_geom();
void Set_com11_qfun();
void Set_com5_rdsmax();
void Set_com5_rdsmin();
void Set_com10_claml();
void Set_com10_clam2();
void Set_com10_ct1();
void Set_com10_ct2();
void Next_up_direction(Actuator*);
void Next_dn_direction(Actuator*);
void Up_nedge(Actuator*);
void Down_nedge(Actuator*);
void Reset_nedge(Actuator*);
void Up_indq(Actuator*);
void Down_indq(Actuator*);
void Reset_indq(Actuator*);
void Up_mg1(Actuator*);
void Down_mg1(Actuator*);
void Reset_mg1(Actuator*);
void Up_mg2(Actuator*);
void Down_mg2(Actuator*);
void Reset_mg2(Actuator*);
void Up_mgsteps(Actuator*);
void Down_mgsteps(Actuator*);
void Reset_mgsteps(Actuator*);
void Up_mgpls(Actuator*);
void Down_mgpls(Actuator*);
void Reset_mgpls(Actuator*);
void Up_nfilt(Actuator*);
void Down_nfilt(Actuator*);
void Reset_nfilt(Actuator*);
void Up_inter(Actuator*);
void Down_inter(Actuator*);
void Reset_inter(Actuator*);
void Up_lnsing(Actuator*);
void Down_lnsing(Actuator*);
void Reset_lnsing(Actuator*);
void Up_plsing(Actuator*);
void Down_plsing(Actuator*);
void Reset_plsing(Actuator*);
void Up_add(Actuator*);
void Down_add(Actuator*);
void Reset_add(Actuator*);
void Up_sub(Actuator*);
void Down_sub(Actuator*);
void Reset_sub(Actuator*);
void But_rdsmin(Actuator*);
void But_rdsmax(Actuator*);
void But_claml(Actuator*);
void But_clam2(Actuator*);
void But_ct1(Actuator*);
void But_ct2(Actuator*);
void Sld_rdsmin();
void Sld_rdsmax();
```

```
void Sld_claml();
void Sld_clam2();
void Sld_ct1();
void Sld_ct2();
void But_calculator();


void initialize_all_sage_parameters_to_default();

panel_sage_main = p;

initialize_all_sage_parameters_to_default();

x = x + 13.1;
y = y - 19.5;

panel_sage_x = x;
panel_sage_y = y;

/*   cur_direction = mod( 48, ++cur_direction );   */

dummy = pnl_mkact( pnl_label );
dummy -> label = LABEL_ADAPTION_LABEL;
dummy -> x = x + LABEL_ADAPTION_X;
dummy -> y = y + LABEL_ADAPTION_Y;
pnl_addact( dummy, p );

adp_dn_direction = pnl_mkact(pnl_down_arrow_button);
adp_dn_direction -> x = x + ADP_DN_DIRECTION_X;
adp_dn_direction -> y = y + ADP_DN_DIRECTION_Y;
adp_dn_direction -> upfunc = Next_dn_direction;
pnl_addact(adp_dn_direction, p);

adp_up_direction = pnl_mkact(pnl_up_arrow_button);
adp_up_direction -> x = x + ADP_UP_DIRECTION_X;
adp_up_direction -> y = y + ADP_UP_DIRECTION_Y;
adp_up_direction -> upfunc = Next_up_direction;
pnl_addact(adp_up_direction, p);

pts_direction = pnl_mkact( pnl_label );
pts_direction -> label = direction_of_ijk[ cur_direction ][ PTS_DIRECTION ];
pts_direction -> x = x + PTS_DIRECTION_X;
pts_direction -> y = y + PTS_DIRECTION_Y;
pnl_addact( pts_direction, p );

lin_direction = pnl_mkact( pnl_label );
lin_direction -> label = direction_of_ijk[ cur_direction ][ LIN_DIRECTION ];
lin_direction -> x = x + LIN_DIRECTION_X;
lin_direction -> y = y + LIN_DIRECTION_Y;
pnl_addact( lin_direction, p );

pla_direction = pnl_mkact( pnl_label );
pla_direction -> label = direction_of_ijk[ cur_direction ][ PLA_DIRECTION ];
pla_direction -> x = x + PLA_DIRECTION_X;
pla_direction -> y = y + PLA_DIRECTION_Y;
pnl_addact( pla_direction, p );

but_adaptpts = pnl_mkact(pnl_wide_button);
but_adaptpts->label = BUT_ADAPTPTS_LABEL;
but_adaptpts->x = x + BUT_ADAPTPTS_X;
but_adaptpts->y = y + BUT_ADAPTPTS_Y;
but_adaptpts->upfunc = Beg_adaptpts;
pnl_addact(but_adaptpts, p);
```



```
but_remove = pnl_mkact(pnl_wide_button);
but_remove->label = BUT_REMOVE_LABEL;
but_remove->x = x + BUT_REMOVE_X;
but_remove->y = y + BUT_REMOVE_Y;
but_remove->upfunc = Beg_remove;
pnl_addact(but_remove, p);

but_undo = pnl_mkact(pnl_wide_button);
but_undo->label = BUT_UNDO_LABEL;
but_undo->x = x + BUT_UNDO_X;
but_undo->y = y + BUT_UNDO_Y;
but_undo->upfunc = Beg_undo;
pnl_addact(but_undo, p);

but_addpts = pnl_mkact(pnl_wide_button);
but_addpts->label = BUT_ADDPTS_LABEL;
but_addpts->x = x + BUT_ADDPTS_X;
but_addpts->y = y + BUT_ADDPTS_Y;
but_addpts->upfunc = Beg_addpts;
pnl_addact(but_addpts, p);

but_subpts = pnl_mkact(pnl_wide_button);
but_subpts->label = "Delete";
but_subpts->x = x + 2.25;
but_subpts->y = y + 1.3;
but_subpts->upfunc = Beg_subpts;
pnl_addact(but_subpts, p);

but_march = pnl_mkact(pnl_toggle_button);
but_march->label = BUT_MARCH_LABEL;
but_march->x = x + BUT_MARCH_X;
but_march->y = y + BUT_MARCH_Y;
but_march->val = MARCH_INI;
but_march->upfunc = Set_com14_march;
pnl_addact(but_march, p);

but_marchpl = pnl_mkact(pnl_toggle_button);
but_marchpl->label = BUT_MARCHPL_LABEL;
but_marchpl->x = x + BUT_MARCHPL_X;
but_marchpl->y = y + BUT_MARCHPL_Y;
but_marchpl->val = MARCHPL_INI;
but_marchpl->upfunc = Set_com14_marchpl;
pnl_addact(but_marchpl, p);

but_orths = pnl_mkact(pnl_toggle_button);
but_orths->label = BUT_ORTHS_LABEL;
but_orths->x = x + BUT_ORTHS_X;
but_orths->y = y + BUT_ORTHS_Y;
but_orths->val = ORTHS_INI;
but_orths->upfunc = Set_com16_orths;
pnl_addact(but_orths, p);

but_orthe = pnl_mkact(pnl_toggle_button);
but_orthe->label = BUT_ORTHE_LABEL;
but_orthe->x = x + BUT_ORTHE_X;
but_orthe->y = y + BUT_ORTHE_Y;
but_orthe->val = ORTHE_INI;
but_orthe->upfunc = Set_com16_orthe;
pnl_addact(but_orthe, p);

but_geom = pnl_mkact(pnl_toggle_button);
but_geom->label = BUT_GEOM_LABEL;
but_geom->x = x + BUT_GEOM_X;
but_geom->y = y + BUT_GEOM_Y;
```

```
but_geom->val = GEOM_INI;
but_geom->upfunc = Set_com11_geom;
pnl_addact(but_geom, p);

but_qfun = pnl_mkact(pnl_toggle_button);
but_qfun->label = BUT_QFUN_LABEL;
but_qfun->x = x + BUT_QFUN_X;
but_qfun->y = y + BUT_QFUN_Y;
but_qfun->val = QFUN_INI;
but_qfun->upfunc = Set_com11_qfun;
pnl_addact(but_qfun, p);

Sld_rdsmin();

Sld_rdsmax();

Sld_claml();

Sld_clam2();

Sld_ct1();

Sld_ct2();

typein_lnsing = pnl_mkact(pnl_typein);
typein_lnsing->x = x + TYPEIN_LNSING_X;
typein_lnsing->y = y + TYPEIN_LNSING_Y;
typein_lnsing->labeltype = PNL_LABEL_BOTTOM;
typein_lnsing->label = TYPEIN_LNSING_LABEL;
PNL_ACCESS(Typein,typein_lnsing,len) = TYPEIN_LNSING_LENGTH;
sprintf(PNL_ACCESS(Typein,typein_lnsing,str),TYPEIN_LNSING_FORMAT,LNSING_INI);
typein_lnsing->upfunc = Intergize_lnsing;
pnl_addact(typein_lnsing,p);

typein_plsing = pnl_mkact(pnl_typein);
typein_plsing->x = x + TYPEIN_PLSING_X;
typein_plsing->y = y + TYPEIN_PLSING_Y;
typein_plsing->labeltype = PNL_LABEL_BOTTOM;
typein_plsing->label = TYPEIN_PLSING_LABEL;
PNL_ACCESS(Typein,typein_plsing,len) = TYPEIN_PLSING_LENGTH;
sprintf(PNL_ACCESS(Typein,typein_plsing,str),TYPEIN_PLSING_FORMAT,PLSING_INI);
typein_plsing->upfunc = Intergize_plsing;
pnl_addact(typein_plsing,p);

typein_mgsteps = pnl_mkact(pnl_typein);
typein_mgsteps->x = x + MGSTEPS_X;
typein_mgsteps->y = y + MGSTEPS_Y;
typein_mgsteps->labeltype = PNL_LABEL_BOTTOM;
typein_mgsteps->label = TYPEIN_MGSTEPS_LABEL;
PNL_ACCESS(Typein,typein_mgsteps,len) = TYPEIN_MGSTEPS_LENGTH;
sprintf(PNL_ACCESS(Typein,typein_mgsteps,str),TYPEIN_MGSTEPS_FORMAT,MGSTEPS_INI);
typein_mgsteps->upfunc = Intergize_mgsteps;
pnl_addact(typein_mgsteps,p);

typein_mgpls = pnl_mkact(pnl_typein);
typein_mgpls->x = x + TYPEIN_MGPLS_X;
typein_mgpls->y = y + TYPEIN_MGPLS_Y;
typein_mgpls->labeltype = PNL_LABEL_BOTTOM;
typein_mgpls->label = TYPEIN_MGPLS_LABEL;
PNL_ACCESS(Typein,typein_mgpls,len) = TYPEIN_MGPLS_LENGTH;
sprintf(PNL_ACCESS(Typein,typein_mgpls,str),TYPEIN_MGPLS_FORMAT,MGPLS_INI);
typein_mgpls->upfunc = Intergize_mgpls;
pnl_addact(typein_mgpls,p);
```

```
typein_add = pnl_mkact(pnl_typein);
typein_add->x = x + TYPEIN_ADD_X;
typein_add->y = y + TYPEIN_ADD_Y;
PNL_ACCESS(Typein,typein_add,len) = TYPEIN_ADD_LENGTH;
sprintf(PNL_ACCESS(Typein,typein_add,str),TYPEIN_ADD_FORMAT,ADD_INI);
typein_add->upfunc = Interqize_add;
pnl_addact(typein_add,p);

typein_sub = pnl_mkact(pnl_typein);
typein_sub->x = x + SUB_X;
typein_sub->y = y + SUB_Y;
PNL_ACCESS(Typein,typein_sub,len) = TYPEIN_SUB_LENGTH;
sprintf(PNL_ACCESS(Typein,typein_sub,str),TYPEIN_SUB_FORMAT,SUB_INI);
typein_sub->upfunc = Interqize_sub;
pnl_addact(typein_sub,p);

typein_nedge = pnl_mkact(pnl_typein);
typein_nedge->x = x + TYPEIN_NEDGE_X;
typein_nedge->y = y + TYPEIN_NEDGE_Y;
typein_nedge->labeltype = PNL_LABEL_BOTTOM;
typein_nedge->label = TYPEIN_NEDGE_LABEL;
PNL_ACCESS(Typein,typein_nedge,len) = TYPEIN_NEDGE_LENGTH;
sprintf(PNL_ACCESS(Typein,typein_nedge,str),TYPEIN_NEDGE_FORMAT,NEDGE_INI);
typein_nedge->upfunc = Interqize_nedge;
pnl_addact(typein_nedge,p);

typein_indq = pnl_mkact(pnl_typein);
typein_indq->x = x + TYPEIN_INDQ_X;
typein_indq->y = y + TYPEIN_INDQ_Y;
typein_indq->labeltype = PNL_LABEL_BOTTOM;
typein_indq->label = TYPEIN_INDQ_LABEL;
PNL_ACCESS(Typein,typein_indq,len) = TYPEIN_INDQ_LENGTH;
sprintf(PNL_ACCESS(Typein,typein_indq,str),TYPEIN_INDQ_FORMAT,INDQ_INI);
typein_indq->upfunc = Interqize_indq;
pnl_addact(typein_indq,p);

typein_nfilt = pnl_mkact(pnl_typein);
typein_nfilt->x = x + TYPEIN_NFILT_X;
typein_nfilt->y = y + TYPEIN_NFILT_Y;
typein_nfilt->labeltype = PNL_LABEL_BOTTOM;
typein_nfilt->label = TYPEIN_NFILT_LABEL;
PNL_ACCESS(Typein,typein_nfilt,len) = TYPEIN_NFILT_LENGTH;
sprintf(PNL_ACCESS(Typein,typein_nfilt,str),TYPEIN_NFILT_FORMAT,NFILT_INI);
typein_nfilt->upfunc = Interqize_nfilt;
pnl_addact(typein_nfilt,p);

typein_inter = pnl_mkact(pnl_typein);
typein_inter->x = x + TYPEIN_INTER_X;
typein_inter->y = y + TYPEIN_INTER_Y;
typein_inter->labeltype = PNL_LABEL_BOTTOM;
typein_inter->label = TYPEIN_INTER_LABEL;
PNL_ACCESS(Typein,typein_inter,len) = TYPEIN_INTER_LENGTH;
sprintf(PNL_ACCESS(Typein,typein_inter,str),TYPEIN_INTER_FORMAT,INTER_INI);
typein_inter->upfunc = Interqize_inter;
pnl_addact(typein_inter,p);

typein_mg1 = pnl_mkact(pnl_typein);
typein_mg1->x = x + TYPEIN_MG1_X;
typein_mg1->y = y + TYPEIN_MG2_Y;
typein_mg1->labeltype = PNL_LABEL_BOTTOM;
typein_mg1->label = TYPEIN_MG1_LABEL;
PNL_ACCESS(Typein,typein_mg1,len) = TYPEIN_MG1_LENGTH;
sprintf(PNL_ACCESS(Typein,typein_mg1,str),TYPEIN_MG1_FORMAT,MG1_INI);
typein_mg1->upfunc = Interqize_mg1;
pnl_addact(typein_mg1,p);

typein_mg2 = pnl_mkact(pnl_typein);
typein_mg2->x = x + TYPEIN_MG2_X;
typein_mg2->y = y + TYPEIN_MG2_Y;
typein_mg2->labeltype = PNL_LABEL_BOTTOM;
typein_mg2->label = TYPEIN_MG2_LABEL;
PNL_ACCESS(Typein,typein_mg2,len) = TYPEIN_MG2_LENGTH;
sprintf(PNL_ACCESS(Typein,typein_mg2,str),TYPEIN_MG2_FORMAT,MG2_INI);
typein_mg2->upfunc = Interqize_mg2;
pnl_addact(typein_mg2,p);

up_nedge = pnl_mkact(pnl_up_arrow_button);
up_nedge -> x = x + UP_NEDGE_X;
up_nedge -> y = y + UP_NEDGE_Y;
up_nedge -> upfunc = Up_nedge;
pnl_addact(up_nedge, p);

down_nedge = pnl_mkact(pnl_down_arrow_button);
down_nedge -> x = x + DOWN_NEDGE_X;
down_nedge -> y = y + DOWN_NEDGE_Y;
down_nedge -> upfunc = Down_nedge;
pnl_addact(down_nedge, p);

reset_nedge = pnl_mkact(pnl_button);
reset_nedge -> x = x + RESET_NEDGE_X;
reset_nedge -> y = y + RESET_NEDGE_Y;
reset_nedge -> labeltype = PNL_LABEL_CENTER;
reset_nedge -> label = RESET_NEDGE_LABEL;
reset_nedge -> upfunc = Reset_nedge;
pnl_addact(reset_nedge, p);

up_indq = pnl_mkact(pnl_up_arrow_button);
up_indq -> x = x + UP_INDQ_X;
up_indq -> y = y + UP_INDQ_Y;
up_indq -> upfunc = Up_indq;
pnl_addact(up_indq, p);

down_indq = pnl_mkact(pnl_down_arrow_button);
down_indq -> x = x + DOWN_INDQ_X;
down_indq -> y = y + DOWN_INDQ_Y;
down_indq -> upfunc = Down_indq;
pnl_addact(down_indq, p);

reset_indq = pnl_mkact(pnl_button);
reset_indq -> x = x + RESET_INDQ_X;
reset_indq -> y = y + RESET_INDQ_Y;
reset_indq -> labeltype = PNL_LABEL_CENTER;
reset_indq -> label = RESET_INDQ_LABEL;
reset_indq -> upfunc = Reset_indq;
pnl_addact(reset_indq, p);

up_mg1 = pnl_mkact(pnl_up_arrow_button);
up_mg1 -> x = x + UP_MG1_X;
up_mg1 -> y = y + UP_MG1_Y;
up_mg1 -> upfunc = Up_mg1;
pnl_addact(up_mg1, p);

down_mg1 = pnl_mkact(pnl_down_arrow_button);
down_mg1 -> x = x + DOWN_MG1_X;
down_mg1 -> y = y + DOWN_MG1_Y;
down_mg1 -> upfunc = Down_mg1;
pnl_addact(down_mg1, p);
```



```
reset_mg1 = pnl_mkact(pnl_button);
reset_mg1 -> x = x + RESET_MG1_X;
reset_mg1 -> y = y + RESET_MG1_Y;
reset_mg1 -> labeltype = PNL_LABEL_CENTER;
reset_mg1 -> label = RESET_MG1_LABEL;
reset_mg1 -> upfunc = Reset_mg1;
pnl_addact(reset_mg1, p);

up_mg2 = pnl_mkact(pnl_up_arrow_button);
up_mg2 -> x = x + UP_MG2_X;
up_mg2 -> y = y + UP_MG2_Y;
up_mg2 -> upfunc = Up_mg2;
pnl_addact(up_mg2, p);

down_mg2 = pnl_mkact(pnl_down_arrow_button);
down_mg2 -> x = x + DOWN_MG2_X;
down_mg2 -> y = y + DOWN_MG2_Y;
down_mg2 -> upfunc = Down_mg2;
pnl_addact(down_mg2, p);

reset_mg2 = pnl_mkact(pnl_button);
reset_mg2 -> x = x + RESET_MG2_X;
reset_mg2 -> y = y + RESET_MG2_Y;
reset_mg2 -> labeltype = PNL_LABEL_CENTER;
reset_mg2 -> label = RESET_MG2_LABEL;
reset_mg2 -> upfunc = Reset_mg2;
pnl_addact(reset_mg2, p);

up_mgsteps = pnl_mkact(pnl_up_arrow_button);
up_mgsteps -> x = x + UP_MGSTEPS_X;
up_mgsteps -> y = y + UP_MGSTEPS_Y;
up_mgsteps -> upfunc = Up_mgsteps;
pnl_addact(up_mgsteps, p);

down_mgsteps = pnl_mkact(pnl_down_arrow_button);
down_mgsteps -> x = x + DOWN_MGSTEPS_X;
down_mgsteps -> y = y + DOWN_MGSTEPS_Y;
down_mgsteps -> upfunc = Down_mgsteps;
pnl_addact(down_mgsteps, p);

reset_mgsteps = pnl_mkact(pnl_button);
reset_mgsteps -> x = x + RESET_MGSTEPS_X;
reset_mgsteps -> y = y + RESET_MGSTEPS_Y;
reset_mgsteps -> labeltype = PNL_LABEL_CENTER;
reset_mgsteps -> label = RESET_MGSTEPS_LABEL;
reset_mgsteps -> upfunc = Reset_mgsteps;
pnl_addact(reset_mgsteps, p);

up_mgpls = pnl_mkact(pnl_up_arrow_button);
up_mgpls -> x = x + UP_MGPLS_X;
up_mgpls -> y = y + UP_MGPLS_Y;
up_mgpls -> upfunc = Up_mgpls;
pnl_addact(up_mgpls, p);

down_mgpls = pnl_mkact(pnl_down_arrow_button);
down_mgpls -> x = x + DOWN_MGPLS_X;
down_mgpls -> y = y + DOWN_MGPLS_Y;
down_mgpls -> upfunc = Down_mgpls;
pnl_addact(down_mgpls, p);

reset_mgpls = pnl_mkact(pnl_button);
reset_mgpls -> x = x + RESET_MGPLS_X;
reset_mgpls -> y = y + RESET_MGPLS_Y;
reset_mgpls -> labeltype = PNL_LABEL_CENTER;
reset_mgpls -> label = RESET_MGPLS_LABEL;
reset_mgpls -> upfunc = Reset_mgpls;
pnl_addact(reset_mgpls, p);

up_nfilt = pnl_mkact(pnl_up_arrow_button);
up_nfilt -> x = x + UP_NFILT_X;
up_nfilt -> y = y + UP_NFILT_Y;
up_nfilt -> upfunc = Up_nfilt;
pnl_addact(up_nfilt, p);

down_nfilt = pnl_mkact(pnl_down_arrow_button);
down_nfilt -> x = x + DOWN_NFILT_X;
down_nfilt -> y = y + DOWN_NFILT_Y;
down_nfilt -> upfunc = Down_nfilt;
pnl_addact(down_nfilt, p);

reset_nfilt = pnl_mkact(pnl_button);
reset_nfilt -> x = x + RESET_NFILT_X;
reset_nfilt -> y = y + RESET_NFILT_Y;
reset_nfilt -> labeltype = PNL_LABEL_CENTER;
reset_nfilt -> label = RESET_NFILT_LABEL;
reset_nfilt -> upfunc = Reset_nfilt;
pnl_addact(reset_nfilt, p);

up_inter = pnl_mkact(pnl_up_arrow_button);
up_inter -> x = x + UP_INTER_X;
up_inter -> y = y + UP_INTER_Y;
up_inter -> upfunc = Up_inter;
pnl_addact(up_inter, p);

down_inter = pnl_mkact(pnl_down_arrow_button);
down_inter -> x = x + DOWN_INTER_X;
down_inter -> y = y + DOWN_INTER_Y;
down_inter -> upfunc = Down_inter;
pnl_addact(down_inter, p);

reset_inter = pnl_mkact(pnl_button);
reset_inter -> x = x + RESET_INTER_X;
reset_inter -> y = y + RESET_INTER_Y;
reset_inter -> labeltype = PNL_LABEL_CENTER;
reset_inter -> label = RESET_INTER_LABEL;
reset_inter -> upfunc = Reset_inter;
pnl_addact(reset_inter, p);

up_lnsing = pnl_mkact(pnl_up_arrow_button);
up_lnsing -> x = x + UP_LNSING_X;
up_lnsing -> y = y + UP_LNSING_Y;
up_lnsing -> upfunc = Up_lnsing;
pnl_addact(up_lnsing, p);

down_lnsing = pnl_mkact(pnl_down_arrow_button);
down_lnsing -> x = x + DOWN_LNSING_X;
down_lnsing -> y = y + DOWN_LNSING_Y;
down_lnsing -> upfunc = Down_lnsing;
pnl_addact(down_lnsing, p);

reset_lnsing = pnl_mkact(pnl_button);
reset_lnsing -> x = x + RESET_LNSING_X;
reset_lnsing -> y = y + RESET_LNSING_Y;
reset_lnsing -> labeltype = PNL_LABEL_CENTER;
reset_lnsing -> label = RESET_LNSING_LABEL;
reset_lnsing -> upfunc = Reset_lnsing;
pnl_addact(reset_lnsing, p);
```

```
   up_plsing = pnl_mkact(pnl_up_arrow_button);
   up_plsing -> x = x + UP_PLSING_X;
   up_plsing -> y = y + UP_PLSING_Y;
   up_plsing -> upfunc = Up_plsing;
   pnl_addact(up_plsing, p);

   down_plsing = pnl_mkact(pnl_down_arrow_button);
   down_plsing -> x = x + DOWN_PLSING_X;
   down_plsing -> y = y + DOWN_PLSING_Y;
   down_plsing -> upfunc = Down_plsing;
   pnl_addact(down_plsing, p);

   reset_plsing = pnl_mkact(pnl_button);
   reset_plsing -> x = x + RESET_PLSING_X;
   reset_plsing -> y = y + RESET_PLSING_Y;
   reset_plsing -> labeltype = PNL_LABEL_CENTER;
   reset_plsing -> label = RESET_PLSING_LABEL;
   reset_plsing -> upfunc = Reset_plsing;
   pnl_addact(reset_plsing, p);

   up_add = pnl_mkact(pnl_up_arrow_button);
   up_add -> x = x + UP_ADD_X;
   up_add -> y = y + UP_ADD_Y;
   up_add -> upfunc = Up_add;
   pnl_addact(up_add, p);

   down_add = pnl_mkact(pnl_down_arrow_button);
   down_add -> x = x + DOWN_ADD_X;
   down_add -> y = y + DOWN_ADD_Y;
   down_add -> upfunc = Down_add;
   pnl_addact(down_add, p);

   reset_add = pnl_mkact(pnl_button);
   reset_add -> x = x + RESET_ADD_X;
   reset_add -> y = y + RESET_ADD_Y;
   reset_add -> labeltype = PNL_LABEL_CENTER;
   reset_add -> label = RESET_ADD_LABEL;
   reset_add -> upfunc = Reset_add;
   pnl_addact(reset_add, p);

   up_sub = pnl_mkact(pnl_up_arrow_button);
   up_sub -> x = x + UP_SUB_X;
   up_sub -> y = y + UP_SUB_Y;
   up_sub -> upfunc = Up_sub;
   pnl_addact(up_sub, p);

   down_sub = pnl_mkact(pnl_down_arrow_button);
   down_sub -> x = x + DOWN_SUB_X;
   down_sub -> y = y + DOWN_SUB_Y;
   down_sub -> upfunc = Down_sub;
   pnl_addact(down_sub, p);

   reset_sub = pnl_mkact(pnl_button);
   reset_sub -> x = x + RESET_SUB_X;
   reset_sub -> y = y + RESET_SUB_Y;
   reset_sub -> labeltype = PNL_LABEL_CENTER;
   reset_sub -> label = RESET_SUB_LABEL;
   reset_sub -> upfunc = Reset_sub;
   pnl_addact(reset_sub, p);

   but_rdsmin = pnl_mkact(pnl_wide_button);
   but_rdsmin -> label = BUT_RDSMIN_LABEL;
   but_rdsmin -> x = x + BUT_RDSMIN_X;
   but_rdsmin -> y = y + BUT_RDSMIN_Y;
   but_rdsmin -> upfunc = But_rdsmin;
   pnl_addact(but_rdsmin, p);

   but_rdsmax = pnl_mkact(pnl_wide_button);
   but_rdsmax -> label = BUT_RDSMAX_LABEL;
   but_rdsmax -> x = x + BUT_RDSMAX_X;
   but_rdsmax -> y = y + BUT_RDSMAX_Y;
   but_rdsmax -> upfunc = But_rdsmax_upfunc;
   pnl_addact(but_rdsmax, p);

   but_clam1 = pnl_mkact(pnl_wide_button);
   but_clam1 ->label = BUT_CLAM1_LABEL;
   but_clam1 -> x = x + BUT_CLAM1_X;
   but_clam1 -> y = y + BUT_CLAM1_Y;
   but_clam1 -> upfunc = But_clam1;
   pnl_addact(but_clam1, p);

   but_clam2 = pnl_mkact(pnl_wide_button);
   but_clam2 ->label = BUT_CLAM2_LABEL;
   but_clam2 -> x = x + BUT_CLAM2_X;
   but_clam2 -> y = y + BUT_CLAM2_Y;
   but_clam2 -> upfunc = But_clam2;
   pnl_addact(but_clam2, p);

   but_ct1 = pnl_mkact(pnl_wide_button);
   but_ct1 ->label = BUT_CT1_LABEL;
   but_ct1 -> x = x + BUT_CT1_X;
   but_ct1 -> y = y + BUT_CT1_Y;
   but_ct1 -> upfunc = But_ct1;
   pnl_addact(but_ct1, p);

   but_ct2 = pnl_mkact(pnl_wide_button);
   but_ct2 ->label = BUT_CT2_LABEL;
   but_ct2 -> x = x + BUT_CT2_X;
   but_ct2 -> y = y + BUT_CT2_Y;
   but_ct2 -> upfunc = But_ct2;
   pnl_addact(but_ct2, p);

   but_calculator = pnl_mkact(pnl_wide_button);
   but_calculator -> label = BUT_CALCULATOR_LABEL;
   but_calculator -> x = x + BUT_CALCULATOR_X;
   but_calculator -> y = y + BUT_CALCULATOR_Y;
   but_calculator -> upfunc = But_calculator;
   pnl_addact(but_calculator, p);

}

void Beg_adaptpts(Actuator* act)
{
   void GetIJKparams();
   void MemoryManager( float*, float* );
   float *getQ();

   printf("\n\n%%  INITIALIZING ADAPTION PROCESS  %%\n\n\n");

   lock_cur_object();

   GetIJKparams();

/*  if( !ireadq )
   {
      qaddress = getQ();
      ireadq = TRUE;
   }
```



```
*/
   adapt_grid = (float *) shm_attach( grid_seger -> field_ids[GRID_ID] );
   adapt_q = (float *) shm_attach( grid_seger -> field_ids[VECTOR_ID] );
   MemoryManager( adapt_grid, adapt_q );

/*  if( !ireadq ) free( qaddress );    */

   unlock_cur_object();

/*
   write_grid_data();
   lock_q_data();
   interpolate_q_data();
   write_q_data();
   unlock_q_data();
*/
}

void Beg_addpts(Actuator* act)
{
   printf("\n\n\nInsertion of points will be in next software release.\n\n\n");
}

void Beg_subpts(Actuator* act)
{
   printf("\n\n\nDeletion of points will be in next software release.\n\n\n");
}

void Beg_undo(Actuator* act)
{
   int dimensions[3];

   printf("\n\n%%  INITIALIZING UNDO PROCESS %%\n\n\n");

   lock_cur_object();

   dimensions[0] = undo_imax;
   dimensions[1] = undo_jmax;
   dimensions[2] = undo_kmax;

   data_xyz_bytes = dimensions[0] * dimensions[1] * dimensions[2] * XYZDIM * sizeof(floa
t);
   data_q_bytes = dimensions[0] * dimensions[1] * dimensions[2] * QDIM * sizeof(float);

   if (!set_data_dims(GRID + grid_seger -> zones[GRID_TYPE][REG], grid_seger -> zones[GR
ID_TYPE][FLD], dimensions))
   {
      Error("cannot set field!\n");
      unlock_cur_object();
      exit(0);
   }

   if (!set_data_dims(SOLUTION + grid_seger -> zones[VECTOR_TYPE][REG], grid_seger -> zo
nes[VECTOR_TYPE][FLD], dimensions))
   {
      Error("cannot set field!\n");
      unlock_cur_object();
      exit(0);
   }

   memcpy((char *)adapt_grid, (char *)unadapt_grid, (size_t)data_xyz_bytes );
   memcpy((char *)adapt_q, (char *)unadapt_q, (size_t)data_q_bytes );

   update_actuators();

   update_fld_data_panel();

   unlock_cur_object();
   update_object_typeout();

   printf("\n\n%%  INITIALIZING UNDO PROCESS %%\n\n\n");
}

void Intergize_add(Actuator* act)
{
   long int
      temp_int;

   temp_int = atol(PNL_ACCESS(Typein,typein_add,str));
   if( temp_int < ADD_MIN )
   {
      com19_.add = ADD_MIN;
   }
   else if( temp_int > ADD_MAX )
   {
      com19_.add = ADD_MAX;
   }
   else
   {
      com19_.add = temp_int;
   }

   sprintf(PNL_ACCESS(Typein,typein_add,str),TYPEIN_ADD_FORMAT,com19_.add);
}

void Intergize_mgsteps(Actuator* act)
{
   long int
      temp_int;

   temp_int = atol(PNL_ACCESS(Typein,typein_mgsteps,str));
   if( temp_int < MGSTEPS_MIN )
   {
      com11_.mgsteps = MGSTEPS_MIN;
   }
   else if( temp_int > MGSTEPS_MAX )
   {
      com11_.mgsteps = MGSTEPS_MAX;
   }
   else
   {
      com11_.mgsteps = temp_int;
   }

   sprintf(PNL_ACCESS(Typein,typein_mgsteps,str),TYPEIN_MGSTEPS_FORMAT,com11_.mgsteps);
}

void Intergize_mgpls(Actuator* act)
{
   long int
      temp_int;

   temp_int = atol(PNL_ACCESS(Typein,typein_mgpls,str));
   if( temp_int < MGPLS_MIN )
   {
      com11_.mgpls = MGPLS_MIN;
   }
   else if( temp_int > MGPLS_MAX )
   {
```

```
      coml1_.mgpls = MGPLS_MAX;
   }
   else
   {
      coml1_.mgpls = temp_int;
   }

   sprintf(PWL_ACCESS(Typein,typein_mgpls,str),TYPEIN_MGPLS_FORMAT,coml1_.mgpls);
}

void Intergize_sub(Actuator* act)
{
   long int
      temp_int;

   temp_int = atol(PWL_ACCESS(Typein,typein_sub,str));
   if( temp_int < SUB_MIN )
   {
      com18_.sub = SUB_MIN;
   }
   else if( temp_int > SUB_MAX )
   {
      com18_.sub = SUB_MAX;
   }
   else
   {
      com18_.sub = temp_int;
   }

   sprintf(PWL_ACCESS(Typein,typein_sub,str),TYPEIN_SUB_FORMAT,com18_.sub);
}

void Intergize_nedge(Actuator* act)
{
   long int
      temp_int;

   temp_int = atol(PWL_ACCESS(Typein,typein_nedge,str));
   if( temp_int < NEDGE_MIN )
   {
      com9_.nedge = NEDGE_MIN;
   }
   else if( temp_int > NEDGE_MAX )
   {
      com9_.nedge = NEDGE_MAX;
   }
   else
   {
      com9_.nedge = temp_int;
   }

   sprintf(PWL_ACCESS(Typein,typein_nedge,str),TYPEIN_NEDGE_FORMAT,com9_.nedge);
}

void Intergize_indq(Actuator* act)
{
   long int
      temp_int;

   temp_int = atol(PWL_ACCESS(Typein,typein_indq,str));
   if( temp_int < INDQ_MIN )
   {
      com5_.indq = INDQ_MIN;
   }
   else if( temp_int > INDQ_MAX )
   {
      com5_.indq = INDQ_MAX;
   }
   else
   {
      com5_.indq = temp_int;
   }

   sprintf(PWL_ACCESS(Typein,typein_indq,str),TYPEIN_INDQ_FORMAT,com5_.indq);
}

void Intergize_nfilt(Actuator* act)
{
   long int
      temp_int;

   temp_int = atol(PWL_ACCESS(Typein,typein_nfilt,str));
   if( temp_int < NFILT_MIN )
   {
      coml1_.nfilt = NFILT_MIN;
   }
   else if( temp_int > NFILT_MAX )
   {
      coml1_.nfilt = NFILT_MAX;
   }
   else
   {
      coml1_.nfilt = temp_int;
   }

   sprintf(PWL_ACCESS(Typein,typein_nfilt,str),TYPEIN_NFILT_FORMAT,coml1_.nfilt);
}

void Intergize_inter(Actuator* act)
{
   long int
      temp_int;

   temp_int = atol(PWL_ACCESS(Typein,typein_inter,str));
   if( temp_int < INTER_MIN )
   {
      coml1_.inter = INTER_MIN;
   }
   else if( temp_int > INTER_MAX )
   {
      coml1_.inter = INTER_MAX;
   }
   else
   {
      coml1_.inter = temp_int;
   }

   sprintf(PWL_ACCESS(Typein,typein_inter,str),TYPEIN_INTER_FORMAT,coml1_.inter);
}

void Intergize_mg1(Actuator* act)
{
   long int
      temp_int;

   temp_int = atol(PWL_ACCESS(Typein,typein_mg1,str));
   if( temp_int < MG1_MIN )
   {
```



```
      com9_.mg1 = MG1_MIN;
   }
   else if( temp_int > MG1_MAX )
   {
      com9_.mg1 = MG1_MAX;
   }
   else
   {
      com9_.mg1 = temp_int;
   }

   sprintf(PWL_ACCESS(Typein,typein_mg1,str),TYPEIN_MG1_FORMAT,com9_.mg1);
}

void Intergize_mg2(Actuator* act)
{
   long int
      temp_int;

   temp_int = atol(PWL_ACCESS(Typein,typein_mg2,str));
   if( temp_int < MG2_MIN )
   {
      com9_.mg2 = MG2_MIN;
   }
   else if( temp_int > MG2_MAX )
   {
      com9_.mg2 = MG2_MAX;
   }
   else
   {
      com9_.mg2 = temp_int;
   }

   sprintf(PWL_ACCESS(Typein,typein_mg2,str),TYPEIN_MG2_FORMAT,com9_.mg2);
}

void Intergize_lnsing(Actuator* act)
{
   long int
      temp_int;

   temp_int = atol(PWL_ACCESS(Typein,typein_lnsing,str));
   if( temp_int < LNSING_MIN )
   {
      coml5_.lnsing = LNSING_MIN;
   }
   else if( temp_int > LNSING_MAX )
   {
      coml5_.lnsing = LNSING_MAX;
   }
   else
   {
      coml5_.lnsing = temp_int;
   }

   sprintf(PWL_ACCESS(Typein,typein_lnsing,str),TYPEIN_LNSING_FORMAT,coml5_.lnsing);
}

void Intergize_plsing(Actuator* act)
{
   long int
      temp_int;

   temp_int = atol(PWL_ACCESS(Typein,typein_plsing,str));
   if( temp_int < PLSING_MIN )
   {
      coml5_.plsing = PLSING_MIN;
   }
   else if( temp_int > PLSING_MAX )
   {
      coml5_.plsing = PLSING_MAX;
   }
   else
   {
      coml5_.plsing = temp_int;
   }

   sprintf(PWL_ACCESS(Typein,typein_plsing,str),TYPEIN_PLSING_FORMAT,coml5_.plsing);
}

void Set_coml4_march()
{
   coml4_.march = but_march->val;
}

void Set_coml4_marchpl()
{
   coml4_.marchpl = but_marchpl->val;
}

void Set_coml6_orths()
{
   coml6_.orths = but_orths->val;
}

void Set_coml6_orthe()
{
   coml6_.orthe = but_orthe->val;
}

void Set_coml1_geom()
{
   coml1_.geom = but_geom->val;
}

void Set_coml1_qfun()
{
   coml1_.qfun = but_qfun->val;
}

void Set_com5_rdsmax()
{
   com5_.rdsmax = sld_rdsmax->val;
}

void Set_com5_rdsmin()
{
   com5_.rdsmin = sld_rdsmin->val;
}

void Set_coml0_clam1()
{
   coml0_.clam[0] = sld_clam1->val;
}

void Set_coml0_clam2()
{
   coml0_.clam[1] = sld_clam2->val;
```

```
}

void Set_coml0_ct1()
{
   coml0_.ct[0] = sld_ct1->val;
}

void Set_coml0_ct2()
{
   coml0_.ct[1] = sld_ct2->val;
}

void Next_dn_direction(Actuator* act)
{
   if( --cur_direction == -1 )
      cur_direction = 47;

   pts_direction -> label = direction_of_ijk[ cur_direction ][ PTS_DIRECTION ];
   lin_direction -> label = direction_of_ijk[ cur_direction ][ LIN_DIRECTION ];
   pla_direction -> label = direction_of_ijk[ cur_direction ][ PLA_DIRECTION ];

   pnl_fixact(pts_direction);
   pnl_fixact(lin_direction);
   pnl_fixact(pla_direction);
}

void Next_up_direction(Actuator* act)
{
   if( ++cur_direction == 48 )
      cur_direction = 0;

   pts_direction -> label = direction_of_ijk[ cur_direction ][ PTS_DIRECTION ];
   lin_direction -> label = direction_of_ijk[ cur_direction ][ LIN_DIRECTION ];
   pla_direction -> label = direction_of_ijk[ cur_direction ][ PLA_DIRECTION ];

   pnl_fixact(pts_direction);
   pnl_fixact(lin_direction);
   pnl_fixact(pla_direction);
}



void GetIJKparams()
{

        int iinc, jinc, kinc;
        float *xyz;

        printf("\n");
        printf("object_type                         = %d\n",
                grid_sager -> object_type);
        printf("direction                           = %d\n",
                grid_sager -> direction);
        printf("dims[GRID_TYPE][I/J/K]              = [%d, %d, %d]\n",
                grid_sager -> dims[GRID_TYPE][I],
                grid_sager -> dims[GRID_TYPE][J],
                grid_sager -> dims[GRID_TYPE][K]);
        printf("dims[SCALAR_TYPE][I/J/K]            = [%d, %d, %d]\n",
                grid_sager -> dims[SCALAR_TYPE][I],
                grid_sager -> dims[SCALAR_TYPE][J],
                grid_sager -> dims[SCALAR_TYPE][K]);
        printf("dims[VECTOR_TYPE][I/J/K]            = [%d, %d, %d]\n",
                grid_sager -> dims[VECTOR_TYPE][I],
                grid_sager -> dims[VECTOR_TYPE][J],
                grid_sager -> dims[VECTOR_TYPE][K]);
        printf("ranges[I][START/END/INC/CUR/DIM]    = [%d, %d, %d, %d, %d]\n",
                grid_sager -> ranges[I][START],
                grid_sager -> ranges[I][END],
                grid_sager -> ranges[I][INC],
                grid_sager -> ranges[I][CUR],
                grid_sager -> ranges[I][DIM]);
        printf("ranges[J][START/END/INC/CUR/DIM]    = [%d, %d, %d, %d, %d]\n",
                grid_sager -> ranges[J][START],
                grid_sager -> ranges[J][END],
                grid_sager -> ranges[J][INC],
                grid_sager -> ranges[J][CUR],
                grid_sager -> ranges[J][DIM]);
        printf("ranges[K][START/END/INC/CUR/DIM]    = [%d, %d, %d, %d, %d]\n",
                grid_sager -> ranges[K][START],
                grid_sager -> ranges[K][END],
                grid_sager -> ranges[K][INC],
                grid_sager -> ranges[K][CUR],
                grid_sager -> ranges[K][DIM]);
        printf("minmax[CLIP][MINI/MAXI/BOTTOM/TOP] = [%f, %f, %f, %f]\n",
                grid_sager -> minmax[CLIP][MINI],
                grid_sager -> minmax[CLIP][MAXI],
                grid_sager -> minmax[CLIP][BOTTOM],
                grid_sager -> minmax[CLIP][TOP]);
        printf("minmax[NORM][MINI/MAXI/BOTTOM/TOP] = [%f, %f, %f, %f]\n",
                grid_sager -> minmax[NORM][MINI],
                grid_sager -> minmax[NORM][MAXI],
                grid_sager -> minmax[NORM][BOTTOM],
                grid_sager -> minmax[NORM][TOP]);
        printf("scalar_id                           = %d\n",
                grid_sager -> field_ids[SCALAR_ID]);
        printf("grid_id                             = %d\n",
                grid_sager -> field_ids[GRID_ID]);
        printf("vector_id                           = %d\n",
                grid_sager -> field_ids[VECTOR_ID]);
        printf("\n");
/*
   xyz = (float *) shm_attach( grid_sager -> grid_id );
   for(kinc=0; kinc<5; kinc++) {
      for(jinc=0; jinc<5; jinc++) {
         for(iinc=0; iinc<10; iinc++) {
            printf("i=%d, j=%d, k=%d, xyz=%f, %f, %f\n",iinc,jinc,kinc,
               *(xyz+kinc*10*20*3+jinc*10*3+iinc*3),
               *(xyz+kinc*10*20*3+jinc*10*3+iinc*3+1),
               *(xyz+kinc*10*20*3+jinc*10*3+iinc*3+2));
         }
      }
   }
*/

   data_imin = 1;
   data_imax = undo_imax = grid_sager -> ranges[I][DIM];
   data_jmin = 1;
   data_jmax = undo_jmax = grid_sager -> ranges[J][DIM];
   data_kmin = 1;
   data_kmax = undo_kmax = grid_sager -> ranges[K][DIM];

   data_xyz_bytes = data_imax * data_jmax * data_kmax * XYZDIM * sizeof(float);
   data_q_bytes = data_imax * data_jmax * data_kmax * QDIM * sizeof(float);

   adapt_imin = grid_sager -> ranges[I][START];
   adapt_imax = grid_sager -> ranges[I][END];
   adapt_jmin = grid_sager -> ranges[J][START];
```



```
   adapt_jmax = grid_sager -> ranges[J][END];
   adapt_kmin = grid_sager -> ranges[K][START];
   adapt_kmax = grid_sager -> ranges[K][END];
/*
   coml_.ist  = data_imin;
   coml_.iend = data_imax;
   coml_.jst  = 1;
   coml_.jend = 4;
   coml_.kst  = 1;
   coml_.kend = 4;

   idir = grid_sager -> ranges[I][INC];
   jdir = grid_sager -> ranges[J][INC];
   kdir = grid_sager -> ranges[K][INC];
*/

}

float *GetQ()
{
   extern int open( const char*, int, ... );
   extern int read( int, char*, int );

   char *buf;
   char *filename = "binq";
   int fd;
   int intsize, floatsize;
   int *idim, *jdim, *kdim, nx=5;
   float *fsmach, *alpha, *re, *time;
   int bytesread;

   if( (fd = open( filename, O_RDONLY, 0)) == -1 ) exit(0);

   intsize = sizeof( int );
   floatsize = sizeof( float );

   buf = (char *) malloc((unsigned long int) (3*intsize + 4*floatsize) );
   read( fd, buf, (3*intsize + 4*floatsize) );

   idim = (int*) buf;
   jdim = (int*) (buf + intsize);
   kdim = (int*) (buf + 2*intsize);
   printf("idim = %d, jdim = %d, kdim = %d\n", *idim, *jdim, *kdim );
   fsmach = (float*) (buf + 3*intsize);
   alpha = (float*) (buf + 3*intsize + floatsize);
   re = (float*) (buf + 3*intsize + 2*floatsize);
   time = (float*) (buf + 3*intsize + 3*floatsize);
   printf("fsmach = %f, alpha = %f, re = %f, time = %f\n", *fsmach, *alpha, *re,
          *time);
   free(buf);
   buf = (char *) malloc((unsigned long int) (*idim**jdim**kdim*nx*floatsize));
   if(buf == 0) exit(0);
   if(  (bytesread =  read( fd, buf, (unsigned long int) (*idim**jdim**kdim*nx
         *floatsize)) ) == 0 ) exit(0);

   close(fd);

   return ((float*) buf);
}



void MemoryManager( float *xyz, float *q )
{
   void Init_sage_params();
   void Ship_data( float*, float* );
   void Unship_data( float*, float* );
   void sager_();
/*   void Duplicate_from_to( float*, float*, unsigned long int);  */

   Init_sage_params();
/*   lock_cur_object();   */
   Ship_data( xyz, q );
   memcpy((char *) unadapt_grid, (char *) xyz, (size_t) data_xyz_bytes);
   memcpy((char *) unadapt_q, (char *) q, (size_t) data_q_bytes);
   sager_();
   Unship_data( xyz, q );
/*   unlock_cur_object();  */
}


void Init_sage_params()
{
/*   idir = adapt_dir[0];
   jdir = adapt_dir[1];
   kdir = adapt_dir[2];
*/   coml3_.istep = FALSE;
   coml3_.jstep = FALSE;
   coml3_.kstep = FALSE;
   coml3_.iinverse = FALSE;
   coml3_.jinverse = FALSE;
   coml3_.kinverse = FALSE;
   coml4_.ijplane = FALSE;
   coml4_.ikplane = FALSE;
   coml4_.jkplane = FALSE;
   coml4_.ok = TRUE;


   switch( cur_direction ) {
   case I_J_K:
   {
      coml3_.jstep = TRUE;
      coml4_.ijplane = TRUE;
      break;
   }
   case I_J_REVK:
   {
      coml3_.jstep = TRUE;
      coml3_.kinverse = TRUE;
      coml4_.ijplane = TRUE;
      break;
   }
   case I_REVJ_K:
   {
      coml3_.jstep = TRUE;
      coml3_.jinverse = TRUE;
      coml4_.ijplane = TRUE;
      break;
   }
   case I_REVJ_REVK:
   {
      coml3_.jstep = TRUE;
      coml3_.jinverse = TRUE;
      coml3_.kinverse = TRUE;
      coml4_.ijplane = TRUE;
      break;
   }
   case REVI_J_K:
```

```
{
   com13_.jstep = TRUE;
   com13_.iinverse = TRUE;
   com14_.ijplane = TRUE;
   break;
}
case REVI_J_REVK:
{
   com13_.jstep = TRUE;
   com13_.iinverse = TRUE;
   com13_.kinverse = TRUE;
   com14_.ijplane = TRUE;
   break;
}
case REVI_REVJ_K:
{
   com13_.jstep = TRUE;
   com13_.iinverse = TRUE;
   com13_.jinverse = TRUE;
   com14_.ijplane = TRUE;
   break;
}
case REVI_REVJ_REVK:
{
   com13_.jstep = TRUE;
   com13_.iinverse = TRUE;
   com13_.jinverse = TRUE;
   com13_.kinverse = TRUE;
   com14_.ijplane = TRUE;
   break;
}

case I_K_J:
{
   com13_.kstep = TRUE;
   com14_.ikplane = TRUE;
   break;
}
case I_K_REVJ:
{
   com13_.kstep = TRUE;
   com13_.kinverse = TRUE;
   com14_.ikplane = TRUE;
   break;
}
case I_REVK_J:
{
   com13_.kstep = TRUE;
   com13_.jinverse = TRUE;
   com14_.ikplane = TRUE;
   break;
}
case I_REVK_REVJ:
{
   com13_.kstep = TRUE;
   com13_.jinverse = TRUE;
   com13_.kinverse = TRUE;
   com14_.ikplane = TRUE;
   break;
}
case REVI_K_J:
{
   com13_.kstep = TRUE;
   com13_.iinverse = TRUE;
   com14_.ikplane = TRUE;
   break;
}
case REVI_K_REVJ:
{
   com13_.kstep = TRUE;
   com13_.iinverse = TRUE;
   com13_.kinverse = TRUE;
   com14_.ikplane = TRUE;
   break;
}
case REVI_REVK_J:
{
   com13_.kstep = TRUE;
   com13_.iinverse = TRUE;
   com13_.jinverse = TRUE;
   com14_.ikplane = TRUE;
   break;
}
case REVI_REVK_REVJ:
{
   com13_.kstep = TRUE;
   com13_.iinverse = TRUE;
   com13_.jinverse = TRUE;
   com13_.kinverse = TRUE;
   com14_.ikplane = TRUE;
   break;
}

case J_I_K:
{
   com13_.istep = TRUE;
   com14_.ijplane = TRUE;
   break;
}
case J_I_REVK:
{
   com13_.istep = TRUE;
   com13_.kinverse = TRUE;
   com14_.ijplane = TRUE;
   break;
}
case J_REVI_K:
{
   com13_.istep = TRUE;
   com13_.jinverse = TRUE;
   com14_.ijplane = TRUE;
   break;
}
case J_REVI_REVK:
{
   com13_.istep = TRUE;
   com13_.jinverse = TRUE;
   com13_.kinverse = TRUE;
   com14_.ijplane = TRUE;
   break;
}
case REVJ_I_K:
{
   com13_.istep = TRUE;
   com13_.iinverse = TRUE;
   com14_.ijplane = TRUE;
   break;
}
```



```
case REVJ_I_REVK:
{
   com13_.istep = TRUE;
   com13_.iinverse = TRUE;
   com13_.kinverse = TRUE;
   com14_.ijplane = TRUE;
   break;
}
case REVJ_REVI_K:
{
   com13_.istep = TRUE;
   com13_.iinverse = TRUE;
   com13_.jinverse = TRUE;
   com14_.ijplane = TRUE;
   break;
}
case REVJ_REVI_REVK:
{
   com13_.istep = TRUE;
   com13_.iinverse = TRUE;
   com13_.jinverse = TRUE;
   com13_.kinverse = TRUE;
   com14_.ijplane = TRUE;
   break;
}

case J_K_I:
{
   com13_.kstep = TRUE;
   com14_.jkplane = TRUE;
   break;
}
case J_K_REVI:
{
   com13_.kstep = TRUE;
   com13_.kinverse = TRUE;
   com14_.jkplane = TRUE;
   break;
}
case J_REVK_I:
{
   com13_.kstep = TRUE;
   com13_.jinverse = TRUE;
   com14_.jkplane = TRUE;
   break;
}
case J_REVK_REVI:
{
   com13_.kstep = TRUE;
   com13_.jinverse = TRUE;
   com13_.kinverse = TRUE;
   com14_.jkplane = TRUE;
   break;
}
case REVJ_K_I:
{
   com13_.kstep = TRUE;
   com13_.iinverse = TRUE;
   com14_.jkplane = TRUE;
   break;
}
case REVJ_K_REVI:
{
   com13_.kstep = TRUE;
   com13_.iinverse = TRUE;
   com13_.kinverse = TRUE;
   com14_.jkplane = TRUE;
   break;
}
case REVJ_REVK_I:
{
   com13_.kstep = TRUE;
   com13_.iinverse = TRUE;
   com13_.jinverse = TRUE;
   com14_.jkplane = TRUE;
   break;
}
case REVJ_REVK_REVI:
{
   com13_.kstep = TRUE;
   com13_.iinverse = TRUE;
   com13_.jinverse = TRUE;
   com13_.kinverse = TRUE;
   com14_.jkplane = TRUE;
   break;
}

case K_I_J:
{
   com13_.istep = TRUE;
   com14_.ikplane = TRUE;
   break;
}
case K_I_REVJ:
{
   com13_.istep = TRUE;
   com13_.kinverse = TRUE;
   com14_.ikplane = TRUE;
   break;
}
case K_REVI_J:
{
   com13_.istep = TRUE;
   com13_.jinverse = TRUE;
   com14_.ikplane = TRUE;
   break;
}
case K_REVI_REVJ:
{
   com13_.istep = TRUE;
   com13_.jinverse = TRUE;
   com13_.kinverse = TRUE;
   com14_.ikplane = TRUE;
   break;
}
case REVK_I_J:
{
   com13_.istep = TRUE;
   com13_.iinverse = TRUE;
   com14_.ikplane = TRUE;
   break;
}
case REVK_I_REVJ:
{
   com13_.istep = TRUE;
   com13_.iinverse = TRUE;
   com13_.kinverse = TRUE;
   com14_.ikplane = TRUE;
```

```
      break;
   }
   case REVK_REVI_J:
   {
      com13_.istep = TRUE;
      com13_.iinverse = TRUE;
      com13_.jinverse = TRUE;
      com14_.ikplane = TRUE;
      break;
   }
   case REVK_REVI_REVJ:
   {
      com13_.istep = TRUE;
      com13_.iinverse = TRUE;
      com13_.jinverse = TRUE;
      com13_.kinverse = TRUE;
      com14_.ikplane = TRUE;
      break;
   }

   case K_J_I:
   {
      com13_.jstep = TRUE;
      com14_.jkplane = TRUE;
      break;
   }
   case K_J_REVI:
   {
      com13_.jstep = TRUE;
      com13_.kinverse = TRUE;
      com14_.jkplane = TRUE;
      break;
   }
   case K_REVJ_I:
   {
      com13_.jstep = TRUE;
      com13_.jinverse = TRUE;
      com14_.jkplane = TRUE;
      break;
   }
   case K_REVJ_REVI:
   {
      com13_.jstep = TRUE;
      com13_.jinverse = TRUE;
      com13_.kinverse = TRUE;
      com14_.jkplane = TRUE;
      break;
   }
   case REVK_J_I:
   {
      com13_.jstep = TRUE;
      com13_.iinverse = TRUE;
      com14_.jkplane = TRUE;
      break;
   }
   case REVK_J_REVI:
   {
      com13_.jstep = TRUE;
      com13_.iinverse = TRUE;
      com13_.kinverse = TRUE;
      com14_.jkplane = TRUE;
      break;
   }
   case REVK_REVJ_I:
   {
      com13_.jstep = TRUE;
      com13_.iinverse = TRUE;
      com13_.jinverse = TRUE;
      com14_.jkplane = TRUE;
      break;
   }
   case REVK_REVJ_REVI:
   {
      com13_.jstep = TRUE;
      com13_.iinverse = TRUE;
      com13_.jinverse = TRUE;
      com13_.kinverse = TRUE;
      com14_.jkplane = TRUE;
      break;
   }

   default:
      break;
   }

   com4_.imax = grid_seger -> ranges[I][DIM];
   com4_.jmax = grid_seger -> ranges[J][DIM];
   com4_.kmax = grid_seger -> ranges[K][DIM];
   com4_.ist  = grid_seger -> ranges[I][START];
   com4_.iend = grid_seger -> ranges[I][END];
   com4_.jst  = grid_seger -> ranges[J][START];
   com4_.jend = grid_seger -> ranges[J][END];
   com4_.kst  = grid_seger -> ranges[K][START];
   com4_.kend = grid_seger -> ranges[K][END];

}


#define XYZ(k_inc,j_inc,i_inc,xyz_inc) *(xyz+k_inc*data_jmax*data_imax*3+j_inc*data_imax*3+i_inc*3+xyz_inc)

#define Q(k_inc,j_inc,i_inc,n_inc) *(q+k_inc*data_jmax*data_imax*NDIM+j_inc*data_imax*NDIM+i_inc*NDIM+n_inc)

void Ship_data(
   float *xyz,
   float *q
)
{
   int
      i_inc, j_inc, k_inc, n_inc;

   for( k_inc = 0; k_inc < data_kmax; k_inc++ ) {
      for( j_inc = 0; j_inc < data_jmax; j_inc++ ) {
         for( i_inc = 0; i_inc < data_imax; i_inc++ ) {
            comxyz_.x[k_inc][j_inc][i_inc] = XYZ(k_inc,j_inc,i_inc,0);
            comxyz_.y[k_inc][j_inc][i_inc] = XYZ(k_inc,j_inc,i_inc,1);
            comxyz_.z[k_inc][j_inc][i_inc] = XYZ(k_inc,j_inc,i_inc,2);
            for( n_inc = 0; n_inc < 5; n_inc++ ) {
               comq_.q[n_inc][k_inc][j_inc][i_inc]
                     = Q(k_inc,j_inc,i_inc,n_inc);
            }
         }
      }
   }
}
```



```
void Unship_data(
   float *xyz,
   float *q
)
{
   int
      i_inc, j_inc, k_inc, n_inc;

   for( k_inc = 0; k_inc < data_kmax; k_inc++ ) {
      for( j_inc = 0; j_inc < data_jmax; j_inc++ ) {
         for( i_inc = 0; i_inc < data_imax; i_inc++ ) {
            XYZ(k_inc,j_inc,i_inc,0) = comxyz_.x[k_inc][j_inc][i_inc];
            XYZ(k_inc,j_inc,i_inc,1) = comxyz_.y[k_inc][j_inc][i_inc];
            XYZ(k_inc,j_inc,i_inc,2) = comxyz_.z[k_inc][j_inc][i_inc];
            for( n_inc = 0; n_inc < 5; n_inc++ ) {
               Q(k_inc,j_inc,i_inc,n_inc)
                     = comq_.q[n_inc][k_inc][j_inc][i_inc];
            }
         }
      }
   }
}

#undef XYZ
#undef Q


void Up_nedge(Actuator *act)
{
   if( com9_.nedge < NEDGE_MAX ) {
      com9_.nedge ++;
      sprintf(PNL_ACCESS(Typein,typein_nedge,str),TYPEIN_NEDGE_FORMAT,com9_.nedge);
      pnl_fixact(typein_nedge);
   }
}

void Down_nedge(Actuator *act)
{
   if( com9_.nedge > NEDGE_MIN ) {
      com9_.nedge --;
      sprintf(PNL_ACCESS(Typein,typein_nedge,str),TYPEIN_NEDGE_FORMAT,com9_.nedge);
      pnl_fixact(typein_nedge);
   }
}

void Reset_nedge(Actuator *act)
{
   com9_.nedge = NEDGE_INI;
   sprintf(PNL_ACCESS(Typein,typein_nedge,str),TYPEIN_NEDGE_FORMAT,com9_.nedge);
   pnl_fixact(typein_nedge);
}

void Up_indq(Actuator *act)
{
   if( com5_.indq < INDQ_MAX) {
      com5_.indq ++;
      sprintf(PNL_ACCESS(Typein,typein_indq,str),TYPEIN_INDQ_FORMAT,com5_.indq);
      pnl_fixact(typein_indq);
   }
}

void Down_indq(Actuator *act)
{
   if( com5_.indq > INDQ_MIN ) {
      com5_.indq --;
      sprintf(PNL_ACCESS(Typein,typein_indq,str),TYPEIN_INDQ_FORMAT,com5_.indq);
      pnl_fixact(typein_indq);
   }
}

void Reset_indq(Actuator *act)
{
   com5_.indq = INDQ_INI;
   sprintf(PNL_ACCESS(Typein,typein_indq,str),TYPEIN_INDQ_FORMAT,com5_.indq);
   pnl_fixact(typein_indq);
}

void Up_mg1(Actuator *act)
{
   if( com9_.mg1 < MG1_MAX) {
      com9_.mg1 ++;
      sprintf(PNL_ACCESS(Typein,typein_mg1,str),TYPEIN_MG1_FORMAT,com9_.mg1);
      pnl_fixact(typein_mg1);
   }
}

void Down_mg1(Actuator *act)
{
   if( com9_.mg1 > MG1_MIN ) {
      com9_.mg1 --;
      sprintf(PNL_ACCESS(Typein,typein_mg1,str),TYPEIN_MG1_FORMAT,com9_.mg1);
      pnl_fixact(typein_mg1);
   }
}

void Reset_mg1(Actuator *act)
{
   com9_.mg1 = MG1_INI;
   sprintf(PNL_ACCESS(Typein,typein_mg1,str),TYPEIN_MG1_FORMAT,com9_.mg1);
   pnl_fixact(typein_mg1);
}

void Up_mg2(Actuator *act)
{
   if( com9_.mg2 < MG2_MAX) {
      com9_.mg2 ++;
      sprintf(PNL_ACCESS(Typein,typein_mg2,str),TYPEIN_MG2_FORMAT,com9_.mg2);
      pnl_fixact(typein_mg2);
   }
}

void Down_mg2(Actuator *act)
{  if( com9_.mg2 > MG2_MIN ) {
      com9_.mg2 --;
      sprintf(PNL_ACCESS(Typein,typein_mg2,str),TYPEIN_MG2_FORMAT,com9_.mg2);
      pnl_fixact(typein_mg2);
   }
}

void Reset_mg2(Actuator *act)
{
   com9_.mg2 = MG2_INI;
   sprintf(PNL_ACCESS(Typein,typein_mg2,str),TYPEIN_MG2_FORMAT,com9_.mg2);
   pnl_fixact(typein_mg2);
}

void Up_mgsteps(Actuator *act)
```

```
{
   if( com11_.mgsteps < MGSTEPS_MAX) {
      com11_.mgsteps ++;
      sprintf(PNL_ACCESS(Typein,typein_mgsteps,str),TYPEIN_MGSTEPS_FORMAT,com11_.mgsteps
);
      pnl_fixact(typein_mgsteps);
   }
}

void Down_mgsteps(Actuator *act)
{
   if( com11_.mgsteps > MGSTEPS_MIN ) {
      com11_.mgsteps --;
      sprintf(PNL_ACCESS(Typein,typein_mgsteps,str),TYPEIN_MGSTEPS_FORMAT,com11_.mgsteps
);
      pnl_fixact(typein_mgsteps);
   }
}

void Reset_mgsteps(Actuator *act)
{
   com11_.mgsteps = MGSTEPS_INI;
   sprintf(PNL_ACCESS(Typein,typein_mgsteps,str),TYPEIN_MGSTEPS_FORMAT,com11_.mgsteps);
   pnl_fixact(typein_mgsteps);
}

void Up_mgpls(Actuator *act)
{
   if( com11_.mgpls < MGPLS_MAX) {
      com11_.mgpls ++;
      sprintf(PNL_ACCESS(Typein,typein_mgpls,str),TYPEIN_MGPLS_FORMAT,com11_.mgpls);
      pnl_fixact(typein_mgpls);
   }
}

void Down_mgpls(Actuator *act)
{
   if( com11_.mgpls > MGPLS_MIN ) {
      com11_.mgpls --;
      sprintf(PNL_ACCESS(Typein,typein_mgpls,str),TYPEIN_MGPLS_FORMAT,com11_.mgpls);
      pnl_fixact(typein_mgpls);
   }
}

void Reset_mgpls(Actuator *act)
{
   com11_.mgpls = MGPLS_INI;
   sprintf(PNL_ACCESS(Typein,typein_mgpls,str),TYPEIN_MGPLS_FORMAT,com11_.mgpls);
   pnl_fixact(typein_mgpls);
}

void Up_nfilt(Actuator *act)
{
   if( com11_.nfilt < NFILT_MAX) {
      com11_.nfilt ++;
      sprintf(PNL_ACCESS(Typein,typein_nfilt,str),TYPEIN_NFILT_FORMAT,com11_.nfilt);
      pnl_fixact(typein_nfilt);
   }
}

void Down_nfilt(Actuator *act)
{
   if( com11_.nfilt > NFILT_MIN ) {
      com11_.nfilt --;
      sprintf(PNL_ACCESS(Typein,typein_nfilt,str),TYPEIN_NFILT_FORMAT,com11_.nfilt);
      pnl_fixact(typein_nfilt);
   }
}

void Reset_nfilt(Actuator *act)
{
   com11_.nfilt = NFILT_INI;
   sprintf(PNL_ACCESS(Typein,typein_nfilt,str),TYPEIN_NFILT_FORMAT,com11_.nfilt);
   pnl_fixact(typein_nfilt);
}

void Up_inter(Actuator *act)
{
   if( com11_.inter < INTER_MAX) {
      com11_.inter ++;
      sprintf(PNL_ACCESS(Typein,typein_inter,str),TYPEIN_INTER_FORMAT,com11_.inter);
      pnl_fixact(typein_inter);
   }
}

void Down_inter(Actuator *act)
{
   if( com11_.inter > INTER_MIN ) {
      com11_.inter --;
      sprintf(PNL_ACCESS(Typein,typein_inter,str),TYPEIN_INTER_FORMAT,com11_.inter);
      pnl_fixact(typein_inter);
   }
}

void Reset_inter(Actuator *act)
{
   com11_.inter = INTER_INI;
   sprintf(PNL_ACCESS(Typein,typein_inter,str),TYPEIN_INTER_FORMAT,com11_.inter);
   pnl_fixact(typein_inter);

}

void Up_lnsing(Actuator *act)
{
   if( com15_.lnsing < LNSING_MAX) {
      com15_.lnsing ++;
      sprintf(PNL_ACCESS(Typein,typein_lnsing,str),TYPEIN_LNSING_FORMAT,com15_.lnsing);
      pnl_fixact(typein_lnsing);
   }
}

void Down_lnsing(Actuator *act)
{
   if( com15_.lnsing > LNSING_MIN ) {
      com15_.lnsing --;
      sprintf(PNL_ACCESS(Typein,typein_lnsing,str),TYPEIN_LNSING_FORMAT,com15_.lnsing);
      pnl_fixact(typein_lnsing);
   }
}

void Reset_lnsing(Actuator *act)
{
   com15_.lnsing = LNSING_INI;
   sprintf(PNL_ACCESS(Typein,typein_lnsing,str),TYPEIN_LNSING_FORMAT,com15_.lnsing);
   pnl_fixact(typein_lnsing);
}

void Up_plsing(Actuator *act)
```



```
{
   if( com15_.plsing < PLSING_MAX) {
      com15_.plsing ++;
      sprintf(PNL_ACCESS(Typein,typein_plsing,str),TYPEIN_PLSING_FORMAT,com15_.plsing);
      pnl_fixact(typein_plsing);
   }
}

void Down_plsing(Actuator *act)
{
   if( com15_.plsing > PLSING_MIN ) {
      com15_.plsing --;
      sprintf(PNL_ACCESS(Typein,typein_plsing,str),TYPEIN_PLSING_FORMAT,com15_.plsing);
      pnl_fixact(typein_plsing);
   }
}

void Reset_plsing(Actuator *act)
{
   com15_.plsing = PLSING_INI;
   sprintf(PNL_ACCESS(Typein,typein_plsing,str),TYPEIN_PLSING_FORMAT,com15_.plsing);
   pnl_fixact(typein_plsing);
}

void Up_add(Actuator *act)
{
   if( com19_.add < ADD_MAX) {
      com19_.add ++;
      sprintf(PNL_ACCESS(Typein,typein_add,str),TYPEIN_ADD_FORMAT,com19_.add);
      pnl_fixact(typein_add);
   }
}

void Down_add(Actuator *act)
{
   if( com19_.add > ADD_MIN ) {
      com19_.add --;
      sprintf(PNL_ACCESS(Typein,typein_add,str),TYPEIN_ADD_FORMAT,com19_.add);
      pnl_fixact(typein_add);
   }
}

void Reset_add(Actuator *act)
{
   com19_.add = ADD_INI;
   sprintf(PNL_ACCESS(Typein,typein_add,str),TYPEIN_ADD_FORMAT,com19_.add);
   pnl_fixact(typein_add);

}

void Up_sub(Actuator *act)
{
   if( com18_.sub < SUB_MAX) {
      com18_.sub ++;
      sprintf(PNL_ACCESS(Typein,typein_sub,str),TYPEIN_SUB_FORMAT,com18_.sub);
      pnl_fixact(typein_sub);
   }
}

void Down_sub(Actuator *act)
{
   if( com18_.sub > SUB_MIN ) {
      com18_.sub --;
      sprintf(PNL_ACCESS(Typein,typein_sub,str),TYPEIN_SUB_FORMAT,com18_.sub);
      pnl_fixact(typein_sub);
   }
}

void Reset_sub(Actuator *act)
{
   com18_.sub = SUB_INI;
   sprintf(PNL_ACCESS(Typein,typein_sub,str),TYPEIN_SUB_FORMAT,com18_.sub);
   pnl_fixact(typein_sub);
}

void But_rdsmin(Actuator *act)
{
   void Sld_rdsmin();
   void Typein_rdsmin();
   void pnl_delact(Actuator*);

   if( but_rdsmin_flag )
   {
      pnl_delact(sld_rdsmin);
      pnl_fixpanel(panel_sage_main);
      Typein_rdsmin();
   }
   else
   {
      pnl_delact(typein_rdsmin);
      pnl_fixpanel(panel_sage_main);
      Sld_rdsmin();
      sld_rdsmin->val = com5_.rdsmin;
      pnl_fixact(sld_rdsmin);
   }
   but_rdsmin_flag = !but_rdsmin_flag;
}

void Sld_rdsmin()
{
   void Set_com5_rdsmin(Actuator*);

   sld_rdsmin = pnl_mkact(pnl_slideroid);
   sld_rdsmin->x = panel_sage_x + SLD_RDSMIN_X;
   sld_rdsmin->y = panel_sage_y + SLD_RDSMIN_Y;
   sld_rdsmin->minval = SLD_RDSMIN_MIN;
   sld_rdsmin->maxval = SLD_RDSMIN_MAX;
   sld_rdsmin->val = SLD_RDSMIN_INI;
   sld_rdsmin->upfunc = Set_com5_rdsmin;
   pnl_addact(sld_rdsmin, panel_sage_main);
}

void Typein_rdsmin()
{
   void Set_com5_rdsmin_typein();

   typein_rdsmin = pnl_mkact(pnl_typein);
   typein_rdsmin->x = panel_sage_x + TYPEIN_RDSMIN_X;
   typein_rdsmin->y = panel_sage_y + TYPEIN_RDSMIN_Y;
   PNL_ACCESS(Typein,typein_rdsmin,len) = TYPEIN_RDSMIN_LENGTH;
   sprintf(PNL_ACCESS(Typein,typein_rdsmin,str),TYPEIN_RDSMIN_FORMAT,com5_.rdsmin);
   typein_rdsmin->upfunc = Set_com5_rdsmin_typein;
   pnl_addact(typein_rdsmin,panel_sage_main);
}

void Set_com5_rdsmin_typein()
{
   float
```

```
    temp_float;

  temp_float = atof(PNL_ACCESS(Typein,typein_rdsmin,str));
  if( temp_float < RDSMIN_MIN )
  {
     com5_.rdsmin = RDSMIN_MIN;
  }
  else if( temp_float > RDSMIN_MAX )
  {
     com5_.rdsmin = RDSMIN_MAX;
  }
  else
  {
     com5_.rdsmin = temp_float;
  }

  sprintf(PNL_ACCESS(Typein,typein_rdsmin,str),TYPEIN_RDSMIN_FORMAT,com5_.rdsmin);
}

/************************************** rdsmax *********************************/

void But_rdsmax_upfunc(Actuator *act)
{
   void Sld_rdsmax();
   void Typein_rdsmax();
   void pnl_delact(Actuator*);

   if( but_rdsmax_flag )
   {
      pnl_delact(sld_rdsmax);
      pnl_fixpanel(panel_sage_main);
      Typein_rdsmax();
   }
   else
   {
      pnl_delact(typein_rdsmax);
      pnl_fixpanel(panel_sage_main);
      Sld_rdsmax();
      sld_rdsmax->val = com5_.rdsmax;
      pnl_fixact(sld_rdsmax);
   }
   but_rdsmax_flag = !but_rdsmax_flag;
}

void Sld_rdsmax()
{
   void Set_com5_rdsmax();

   sld_rdsmax = pnl_mkact(pnl_slideroid);
   sld_rdsmax->x = panel_sage_x + SLD_RDSMAX_X;
   sld_rdsmax->y = panel_sage_y + SLD_RDSMAX_Y;
   sld_rdsmax->minval = SLD_RDSMAX_MIN;
   sld_rdsmax->maxval = SLD_RDSMAX_MAX;
   sld_rdsmax->val = SLD_RDSMAX_INI;
   sld_rdsmax->upfunc = Set_com5_rdsmax;
   pnl_addact(sld_rdsmax, panel_sage_main);
}

void Typein_rdsmax()
{
   void Set_com5_rdsmax_typein();

   typein_rdsmax = pnl_mkact(pnl_typein);
   typein_rdsmax->x = panel_sage_x + TYPEIN_RDSMAX_X;
   typein_rdsmax->y = panel_sage_y + TYPEIN_RDSMAX_Y;
   PNL_ACCESS(Typein,typein_rdsmax,len) = TYPEIN_RDSMAX_LENGTH;
   sprintf(PNL_ACCESS(Typein,typein_rdsmax,str),TYPEIN_RDSMAX_FORMAT,com5_.rdsmax);
   typein_rdsmax->upfunc = Set_com5_rdsmax_typein;
   pnl_addact(typein_rdsmax,panel_sage_main);
}

void Set_com5_rdsmax_typein()
{
   float
      temp_float;

   temp_float = atof(PNL_ACCESS(Typein,typein_rdsmax,str));
   if( temp_float < RDSMAX_MIN )
   {
      com5_.rdsmax = RDSMAX_MIN;
   }
   else if( temp_float > RDSMAX_MAX )
   {
      com5_.rdsmax = RDSMAX_MAX;
   }
   else
   {
      com5_.rdsmax = temp_float;
   }

   sprintf(PNL_ACCESS(Typein,typein_rdsmax,str),TYPEIN_RDSMAX_FORMAT,com5_.rdsmax);
}

/************************************** clam1 **********************************/

void But_clam1()
{
   void Sld_clam1();
   void Typein_clam1();
   void pnl_delact(Actuator*);

   if( but_clam1_flag )
   {
      pnl_delact(sld_clam1);
      pnl_fixpanel(panel_sage_main);
      Typein_clam1();
   }
   else
   {
      pnl_delact(typein_clam1);
      pnl_fixpanel(panel_sage_main);
      Sld_clam1();
      sld_clam1->val = com10_.clam[0];
      pnl_fixact(sld_clam1);
   }
   but_clam1_flag = !but_clam1_flag;
}

void Sld_clam1()
{
   void set_com10_clam1();

   sld_clam1 = pnl_mkact(pnl_slideroid);
   sld_clam1->x = panel_sage_x + SLD_CLAM1_X;
   sld_clam1->y = panel_sage_y + SLD_CLAM1_Y;
   sld_clam1->minval = SLD_CLAM1_MIN;
   sld_clam1->maxval = SLD_CLAM1_MAX;
   sld_clam1->val = SLD_CLAM1_INI;
```



```
   sld_clam1->upfunc = Set_com10_clam1;
   pnl_addact(sld_clam1, panel_sage_main);
}

void Typein_clam1()
{
   void Set_com10_clam1_typein();

   typein_clam1 = pnl_mkact(pnl_typein);
   typein_clam1->x = panel_sage_x + TYPEIN_CLAM1_X;
   typein_clam1->y = panel_sage_y + TYPEIN_CLAM1_Y;
   PNL_ACCESS(Typein,typein_clam1,len) = TYPEIN_CLAM1_LENGTH;
   sprintf(PNL_ACCESS(Typein,typein_clam1,str),TYPEIN_CLAM1_FORMAT,com10_.clam[0]);
   typein_clam1->upfunc = Set_com10_clam1_typein;
   pnl_addact(typein_clam1,panel_sage_main);
}

void Set_com10_clam1_typein()
{
   float
      temp_float;

   temp_float = atof(PNL_ACCESS(Typein,typein_clam1,str));
   if( temp_float < CLAM1_MIN )
   {
      com10_.clam[0] = CLAM1_MIN;
   }
   else if( temp_float > CLAM1_MAX )
   {
      com10_.clam[0] = CLAM1_MAX;
   }
   else
   {
      com10_.clam[0] = temp_float;
   }

   sprintf(PNL_ACCESS(Typein,typein_clam1,str),TYPEIN_CLAM1_FORMAT,com10_.clam[0]);
}

void But_clam2()
{
   void Sld_clam2();
   void Typein_clam2();
   void pnl_delact(Actuator*);

   if( but_clam2_flag )
   {
      pnl_delact(sld_clam2);
      pnl_fixpanel(panel_sage_main);
      Typein_clam2();
   }
   else
   {
      pnl_delact(typein_clam2);
      pnl_fixpanel(panel_sage_main);
      Sld_clam2();
      sld_clam2->val = com10_.clam[1];
      pnl_fixact(sld_clam2);
   }
   but_clam2_flag = !but_clam2_flag;
}

void Sld_clam2()
{
   void set_com10_clam2();

   sld_clam2 = pnl_mkact(pnl_slideroid);
   sld_clam2->x = panel_sage_x + SLD_CLAM2_X;
   sld_clam2->y = panel_sage_y + SLD_CLAM2_Y;
   sld_clam2->minval = SLD_CLAM2_MIN;
   sld_clam2->maxval = SLD_CLAM2_MAX;
   sld_clam2->val = SLD_CLAM2_INI;
   sld_clam2->upfunc = Set_com10_clam2;
   pnl_addact(sld_clam2, panel_sage_main);
}

void Typein_clam2()
{
   void Set_com10_clam2_typein();

   typein_clam2 = pnl_mkact(pnl_typein);
   typein_clam2->x = panel_sage_x + TYPEIN_CLAM2_X;
   typein_clam2->y = panel_sage_y + TYPEIN_CLAM2_Y;
   PNL_ACCESS(Typein,typein_clam2,len) = TYPEIN_CLAM2_LENGTH;
   sprintf(PNL_ACCESS(Typein,typein_clam2,str),TYPEIN_CLAM2_FORMAT,com10_.clam[1]);
   typein_clam2->upfunc = Set_com10_clam2_typein;
   pnl_addact(typein_clam2,panel_sage_main);
}

void Set_com10_clam2_typein()
{
   float
      temp_float;

   temp_float = atof(PNL_ACCESS(Typein,typein_clam2,str));
   if( temp_float < CLAM2_MIN )
   {
      com10_.clam[1] = CLAM2_MIN;
   }
   else if( temp_float > CLAM2_MAX )
   {
      com10_.clam[1] = CLAM2_MAX;
   }
   else
   {
      com10_.clam[1] = temp_float;
   }

   sprintf(PNL_ACCESS(Typein,typein_clam2,str),TYPEIN_CLAM2_FORMAT,com10_.clam[1]);
}

void But_ctl(Actuator *act)
{
   void Sld_ctl();
   void Typein_ctl();
   void pnl_delact(Actuator*);

   if( but_ctl_flag )
   {
      pnl_delact(sld_ctl);
      pnl_fixpanel(panel_sage_main);
      Typein_ctl();
   }
   else
   {
      pnl_delact(typein_ctl);
      pnl_fixpanel(panel_sage_main);
      Sld_ctl();
```

```
      sld_ct1->val = com10_.ct[0];
      pnl_fixact(sld_ct1);
   }
   but_ct1_flag = !but_ct1_flag;
}

void Sld_ct1()
{
   void set_com10_ct1(Actuator);

   sld_ct1 = pnl_mkact(pnl_slideroid);
   sld_ct1->x = panel_sage_x + SLD_CT1_X;
   sld_ct1->y = panel_sage_y + SLD_CT1_Y;
   sld_ct1->minval = CT1_MIN;
   sld_ct1->maxval = CT1_MAX;
   sld_ct1->val = CT1_INI;
   sld_ct1->upfunc = Set_com10_ct1;
   pnl_addact(sld_ct1, panel_sage_main);
}

void Typein_ct1()
{
   void Set_com10_ct1_typein();

   typein_ct1 = pnl_mkact(pnl_typein);
   typein_ct1->x = panel_sage_x + TYPEIN_CT1_X;
   typein_ct1->y = panel_sage_y + TYPEIN_CT1_Y;
   PNL_ACCESS(Typein,typein_ct1,len) = TYPEIN_CT1_LENGTH;
   sprintf(PNL_ACCESS(Typein,typein_ct1,str),TYPEIN_CT1_FORMAT,com10_.ct[0]);
   typein_ct1->upfunc = Set_com10_ct1_typein;
   pnl_addact(typein_ct1,panel_sage_main);
}

void Set_com10_ct1_typein()
{
   float
      temp_float;

   temp_float = atof(PNL_ACCESS(Typein,typein_ct1,str));
   if( temp_float < CT1_MIN )
   {
      com10_.ct[0] = CT1_MIN;
   }
   else if( temp_float > CT1_MAX )
   {
      com10_.ct[0] = CT1_MAX;
   }
   else
   {
      com10_.ct[0] = temp_float;
   }

   sprintf(PNL_ACCESS(Typein,typein_ct1,str),TYPEIN_CT1_FORMAT,com10_.ct[0]);
}

void But_ct2(Actuator *act)
{
   void Sld_ct2();
   void Typein_ct2();
   void pnl_delact(Actuator*);

   if( but_ct2_flag )
   {
      pnl_delact(sld_ct2);
      pnl_fixpanel(panel_sage_main);
      Typein_ct2();
   }
   else
   {
      pnl_delact(typein_ct2);
      pnl_fixpanel(panel_sage_main);
      Sld_ct2();
      sld_ct2->val = com10_.ct[1];
      pnl_fixact(sld_ct2);
   }
   but_ct2_flag = !but_ct2_flag;
}

void Sld_ct2()
{
   void set_com10_ct2(Actuator);

   sld_ct2 = pnl_mkact(pnl_slideroid);
   sld_ct2->x = panel_sage_x + SLD_CT2_X;
   sld_ct2->y = panel_sage_y + SLD_CT2_Y;
   sld_ct2->minval = SLD_CT2_MIN;
   sld_ct2->maxval = SLD_CT2_MAX;
   sld_ct2->val = SLD_CT2_INI;
   sld_ct2->upfunc = Set_com10_ct2;
   pnl_addact(sld_ct2, panel_sage_main);
}

void Typein_ct2()
{
   void Set_com10_ct2_typein();

   typein_ct2 = pnl_mkact(pnl_typein);
   typein_ct2->x = panel_sage_x + TYPEIN_CT2_X;
   typein_ct2->y = panel_sage_y + TYPEIN_CT2_Y;
   PNL_ACCESS(Typein,typein_ct2,len) = TYPEIN_CT2_LENGTH;
   sprintf(PNL_ACCESS(Typein,typein_ct2,str),TYPEIN_CT2_FORMAT,com10_.ct[1]);
   typein_ct2->upfunc = Set_com10_ct2_typein;
   pnl_addact(typein_ct2,panel_sage_main);
}

void Set_com10_ct2_typein()
{
   float
      temp_float;

   temp_float = atof(PNL_ACCESS(Typein,typein_ct2,str));
   if( temp_float < CT2_MIN )
   {
      com10_.ct[1] = CT2_MIN;
   }
   else if( temp_float > CT2_MAX )
   {
      com10_.ct[1] = CT2_MAX;
   }
   else
   {
      com10_.ct[1] = temp_float;
   }

   sprintf(PNL_ACCESS(Typein,typein_ct2,str),TYPEIN_CT2_FORMAT,com10_.ct[1]);
}

void Beg_remove(Actuator* act)
```



```
{
   void GetRmIJKparams();
   void MemoryRmManager( float*, float* );

   lock_cur_object();

   GetRmIJKparams();

   adapt_grid = (float *) shm_attach( grid_sager -> field_ids[GRID_ID] );
   adapt_q = (float *) shm_attach( grid_sager -> field_ids[VECTOR_ID] );
   MemoryRmManager( adapt_grid, adapt_q );

   update_actuators();
   update_fld_data_panel();

   unlock_cur_object();
   update_object_typeout();

}


void GetRmIJKparams()
{

   int iinc, jinc, kinc;
   float *xyz;
               grid_sager -> dims[GRID_TYPE][I],

   data_imin = 1;
   data_imax = undo_imax = grid_sager -> dims[GRID_TYPE][I];
   data_jmin = 1;
   data_jmax = undo_jmax = grid_sager -> dims[GRID_TYPE][J];
   data_kmin = 1;
   data_kmax = undo_kmax = grid_sager -> dims[GRID_TYPE][K];

   data_xyz_bytes = data_imax * data_jmax * data_kmax * XYZDIM * sizeof(float);
   data_q_bytes = data_imax * data_jmax * data_kmax * QDIM * sizeof(float);

   rmset_imin = grid_sager -> ranges[I][START];
   rmset_imax = grid_sager -> ranges[I][END];
   rmset_jmin = grid_sager -> ranges[J][START];
   rmset_jmax = grid_sager -> ranges[J][END];
   rmset_kmin = grid_sager -> ranges[K][START];
   rmset_kmax = grid_sager -> ranges[K][END];

}

void SetIJK_rmdata()
{
   int dimensions[3];

   dimensions[0] = rmset_imax - rmset_imin + 1;
   dimensions[1] = rmset_jmax - rmset_jmin + 1;
   dimensions[2] = rmset_kmax - rmset_kmin + 1;

   if (!set_data_dims(GRID + grid_sager -> zones[GRID_TYPE][REG], grid_sager -> zones[GRID_TYPE][FLD], dimensions))
   {
      Error("cannot set field!\n");
      unlock_cur_object();
      exit(0);
   }

   if (!set_data_dims(SOLUTION + grid_sager -> zones[VECTOR_TYPE][REG], grid_sager -> zones[VECTOR_TYPE][FLD], dimensions))
   {
      Error("cannot set field!\n");
      unlock_cur_object();
      exit(0);
   }

}


void MemoryRmManager( float *xyz, float *q )
{
   void Ship_rmdata( float*, float*, float*, float* );
   void SetIJK_rmdata();

   memcpy((char *) unadapt_grid, (char *) xyz, (size_t) data_xyz_bytes);
   memcpy((char *) unadapt_q, (char *) q, (size_t) data_q_bytes);
   Ship_rmdata( xyz, q, (float*) unadapt_grid, (float*) unadapt_q );
   SetIJK_rmdata();
}

#define XYZNEW(k_inc,j_inc,i_inc,xyz_inc) *(xyznew+k_inc*rmset_jmax*rmset_imax*3+j_inc*rmset_imax*3+i_inc*3+xyz_inc)
#define QNEW(k_inc,j_inc,i_inc,n_inc) *(qnew+k_inc*rmset_jmax*rmset_imax*NDIM+j_inc*rmset_imax*NDIM+i_inc*NDIM+n_inc)
#define XYZOLD(k_inc,j_inc,i_inc,xyz_inc) *(xyzold+(k_inc+rmset_kmin-1)*data_jmax*data_imax*3+(j_inc+rmset_jmin-1)*data_imax*3+(i_inc+rmset_imin-1)*3+xyz_inc)
#define QOLD(k_inc,j_inc,i_inc,n_inc) *(qold+(k_inc+rmset_kmin-1)*data_jmax*data_imax*NDIM+(j_inc+rmset_jmin-1)*data_imax*NDIM+(i_inc+rmset_imin-1)*NDIM+n_inc)

void Ship_rmdata(float* xyznew, float* qnew, float *xyzold, float *qold)
{
   int
      i_inc, j_inc, k_inc, n_inc;

   printf("imax = %d, jmax = %d, kmax = %d\n",rmset_imax,rmset_jmax,rmset_kmax);
   printf("imin = %d, jmin = %d, kmin = %d\n",rmset_imin,rmset_jmin,rmset_kmin);
   for( k_inc = 0; k_inc < rmset_kmax-rmset_kmin+1; k_inc++ ) {
      for( j_inc = 0; j_inc < rmset_jmax-rmset_jmin+1; j_inc++ ) {
         for( i_inc = 0; i_inc < rmset_imax-rmset_imin+1; i_inc++ ) {
            XYZNEW(k_inc,j_inc,i_inc,0) = XYZOLD(k_inc,j_inc,i_inc,0);
            XYZNEW(k_inc,j_inc,i_inc,1) = XYZOLD(k_inc,j_inc,i_inc,1);
            XYZNEW(k_inc,j_inc,i_inc,2) = XYZOLD(k_inc,j_inc,i_inc,2);
            for( n_inc = 0; n_inc < 5; n_inc++ ) {
               QNEW(k_inc,j_inc,i_inc,n_inc) = QOLD(k_inc,j_inc,i_inc,n_inc);
            }
         }
      }
   }
}


#undef XYZNEW
#undef QNEW
#undef XYZOLD
#undef QOLD

extern int* gArgc;
extern char** gArgv;

void But_calculator()
{
   int Calculator( int gArgc, char** gArgv );
```

```
/*   Calculator( *gArgc, gArgv );    */

   Error("Currently not implemented");
}


void error_(char* sentence, int length)
{
   void Error(char*);

   sentence[length-1] = '\0';
   Error(sentence);
}

void warning_(char* sentence, int length)
{
   void Waring(char*);

   sentence[length-1] = '\0';
   Warning(sentence);
}

/************************** Sage files end here ****************************/


/*--------------------------- END OF FILE panels.c --------------------------*/
```



```
/*
  The following structures are globally defined common blocks from the
main fortran routine.
*/

/*
  IMPORTANT NOTE::  In fortran arrays, the first index runs fastest, whereas,
  in C, the last index runs fastest.  In the following structures, the array
  order has been reversed to reflect this.
*/

#ifndef FALSE
#define FALSE                   0
#endif

#ifndef TRUE
#define TRUE                    1
#endif

#define ID                      150
#define JD                      150
#define KD                      150
#define IMX                     150
#define NDIM                    5
#define QDIM                    5
#define XYZDIM                  3

extern Grid_Surface*            grid_sager;

extern int Calculator( int argc, char* argv[] );

struct {
   float
      x[KD][JD][ID],
      y[KD][JD][ID],
      z[KD][JD][ID];
} commyx_;

struct {
   float
      q[NDIM][KD][JD][ID];
} com1_;

struct {
   float
      f[IMX],
      fb[IMX],
      weight[IMX],
      a,
      b,
      wds,
      wde;
} com2_;

struct {
   float
      ss[IMX],
      sms[IMX],
      ds[IMX],
      sn[IMX],
      snm[IMX],
      snmk[IMX];
} com3_;

struct {
   int
      imax,
      jmax,
      kmax,
      ist,
      iend,
      jst,
      jend,
      kst,
      kend,
      nipts;
} com4_;

struct {
   float
      rdsmax,
      rdsmin;
   int
      indq,
      iq[NDIM+3];
} com5_;

struct {
   float
      damax,
      damin,
      wt[IMX],
      dlengs,
      dlenge,
      wdss,
      wdes;
} COM6_;

struct {
   float
      cose[3][IMX],
      cosu[3][IMX],
      cosb[3][IMX];
} com7_;

struct {
   float
      xj[3][3][IMX],
      yj[3][3][IMX],
      zj[3][3][IMX];
} com8_;

struct {
   float
      sp[IMX],
      spol[IMX],
      dx[IMX],
      dappl[IMX];
   int
      nedge,
      mg1,
      mg2;
} com9_;

struct {
   float
      clam[2],
```

```
      ct[2],
      tn[2],
      cnm[2],
      clamm[2],
      ctm[2],
      tnm[2];
} com10_;

struct {
   int
      nfilt,
      inter,
      mgsteps,
      mgpls,
      gscm,
      qfun,
      nflag;
} com11_;

struct {
   float
      fsmach,
      alpha,
      re,
      time;
   int
      nitg,
      nitq;
} com12_;

struct {
   int
      istep,
      jstep,
      kstep,
      iinverse,
      jinverse,
      kinverse,
      twod;
} com13_;

struct {
   int
      ijplane,
      ikplane,
      jkplane,
      march,
      marchpl,
      save,
      ok;
} com14_;

struct {
   float
      conv;
   int
      maxits,
      noup,
      lnsing,
      plsing;
   float
      kk1;
} com15_;

struct {
   int
      orthe,
      orthe;
} com16_;

struct {
   float
      fq[IMX],
      fq[IMX],
      fqs[IMX],
      smss[IMX];
   int
      mxfg;
   float
      fqw,
      fqw;
} com17_;

struct {
   int
      sub,
      lstsub,
      lendsub;
} com18_;

struct {
   int
      add,
      lstadd,
      lendadd;
} com19_;

struct {
   float
      cosak[3][IMX],
      cosuk[3][IMX],
      cosbk[3][IMX];
} com20_;

/************************************ rdsmax ************************/

void But_rdsmax_upfunc( Actuator *act );

#define RDSMAX_X                2.35
#define RDSMAX_Y                11.0
#define RDSMAX_X_OFFSET         -0.05
#define RDSMAX_Y_OFFSET         -0.5
#define RDSMAX_MIN              1.0
#define RDSMAX_MAX              200.0
#define RDSMAX_INI              2.0

Actuator* sld_rdsmax;
#define SLD_RDSMAX_MIN          RDSMAX_MIN
#define SLD_RDSMAX_MAX          RDSMAX_MAX
#define SLD_RDSMAX_INI          RDSMAX_INI
#define SLD_RDSMAX_X            RDSMAX_X
#define SLD_RDSMAX_Y            RDSMAX_Y

Actuator* but_rdsmax;
#define BUT_RDSMAX_LABEL        "RDSMAX"
#define BUT_RDSMAX_X            RDSMAX_X + RDSMAX_X_OFFSET
#define BUT_RDSMAX_Y            RDSMAX_Y + RDSMAX_Y_OFFSET
```



```
Actuator* typein_rdsmax;
#define TYPEIN_RDSMAX_X         RDSMAX_X
#define TYPEIN_RDSMAX_Y         RDSMAX_Y
#define TYPEIN_RDSMAX_LENGTH    8
#define TYPEIN_RDSMAX_FORMAT    "%3.4f"

/************************************ rdsmin ************************/

#define RDSMIN_X                0.0
#define RDSMIN_Y                11.0
#define RDSMIN_X_OFFSET         -0.05
#define RDSMIN_Y_OFFSET         -0.5
#define RDSMIN_MIN              0.0
#define RDSMIN_MAX              1.0
#define RDSMIN_INI              0.5

Actuator* sld_rdsmin;
#define SLD_RDSMIN_MIN          RDSMIN_MIN
#define SLD_RDSMIN_MAX          RDSMIN_MAX
#define SLD_RDSMIN_INI          RDSMIN_INI
#define SLD_RDSMIN_X            RDSMIN_X
#define SLD_RDSMIN_Y            RDSMIN_Y

Actuator* but_rdsmin;
#define BUT_RDSMIN_LABEL        "RDSMIN"
#define BUT_RDSMIN_X            RDSMIN_X + RDSMIN_X_OFFSET
#define BUT_RDSMIN_Y            RDSMIN_Y + RDSMIN_Y_OFFSET

Actuator* typein_rdsmin;
#define TYPEIN_RDSMIN_X         RDSMIN_X
#define TYPEIN_RDSMIN_Y         RDSMIN_Y
#define TYPEIN_RDSMIN_LENGTH    8
#define TYPEIN_RDSMIN_FORMAT    "%2.6f"

/************************************ clam1 *************************/

#define CLAM1_X                 6.0
#define CLAM1_Y                 11.0
#define CLAM1_X_OFFSET          -0.0
#define CLAM1_Y_OFFSET          -0.5
#define CLAM1_MIN               0.0
#define CLAM1_MAX               0.5
#define CLAM1_INI               0.01

Actuator* sld_clam1;
#define SLD_CLAM1_MIN           CLAM1_MIN
#define SLD_CLAM1_MAX           CLAM1_MAX
#define SLD_CLAM1_INI           CLAM1_INI
#define SLD_CLAM1_X             CLAM1_X
#define SLD_CLAM1_Y             CLAM1_Y

Actuator* but_clam1;
#define BUT_CLAM1_LABEL         "CLAM1"
#define BUT_CLAM1_X             CLAM1_X + CLAM1_X_OFFSET
#define BUT_CLAM1_Y             CLAM1_Y + CLAM1_Y_OFFSET

Actuator* typein_clam1;
#define TYPEIN_CLAM1_X          CLAM1_X
#define TYPEIN_CLAM1_Y          CLAM1_Y
#define TYPEIN_CLAM1_LENGTH     8
#define TYPEIN_CLAM1_FORMAT     "%2.6f"

/************************************ clam2 *************************/

#define CLAM2_X                 8.35
#define CLAM2_Y                 11.0
#define CLAM2_X_OFFSET          -0.0
#define CLAM2_Y_OFFSET          -0.5
#define CLAM2_MIN               0.0
#define CLAM2_MAX               0.5
#define CLAM2_INI               0.0001

Actuator* sld_clam2;
#define SLD_CLAM2_MIN           CLAM2_MIN
#define SLD_CLAM2_MAX           CLAM2_MAX
#define SLD_CLAM2_INI           CLAM2_INI
#define SLD_CLAM2_X             CLAM2_X
#define SLD_CLAM2_Y             CLAM2_Y

Actuator* but_clam2;
#define BUT_CLAM2_LABEL         "CLAM2"
#define BUT_CLAM2_X             CLAM2_X + CLAM2_X_OFFSET
#define BUT_CLAM2_Y             CLAM2_Y + CLAM2_Y_OFFSET

Actuator* typein_clam2;
#define TYPEIN_CLAM2_X          CLAM2_X
#define TYPEIN_CLAM2_Y          CLAM2_Y
#define TYPEIN_CLAM2_LENGTH     8
#define TYPEIN_CLAM2_FORMAT     "%2.6f"

/************************************ ct1 ***************************/

#define CT1_X                   0.0
#define CT1_Y                   6.4
#define CT1_X_OFFSET            -0.0
#define CT1_Y_OFFSET            -0.5
#define CT1_MIN                 0.0
#define CT1_MAX                 1.0
#define CT1_INI                 0.5

Actuator* sld_ct1;
#define SLD_CT1_MIN             CT1_MIN
#define SLD_CT1_MAX             CT1_MAX
#define SLD_CT1_INI             CT1_INI
#define SLD_CT1_X               CT1_X
#define SLD_CT1_Y               CT1_Y

Actuator* but_ct1;
#define BUT_CT1_LABEL           "CT1"
#define BUT_CT1_X               CT1_X + CT1_X_OFFSET
#define BUT_CT1_Y               CT1_Y + CT1_Y_OFFSET

Actuator* typein_ct1;
#define TYPEIN_CT1_X            CT1_X
#define TYPEIN_CT1_Y            CT1_Y
#define TYPEIN_CT1_LENGTH       8
#define TYPEIN_CT1_FORMAT       "%2.6f"

/************************************ ct2 ***************************/

#define CT2_X                   2.35
#define CT2_Y                   6.4
#define CT2_X_OFFSET            -0.0
#define CT2_Y_OFFSET            -0.5
#define CT2_MIN                 0.0
#define CT2_MAX                 1.0
#define CT2_INI                 0.5
```

```
Actuator* sld_ct2;
#define SLD_CT2_MIN             CT2_MIN
#define SLD_CT2_MAX             CT2_MAX
#define SLD_CT2_INI             CT2_INI
#define SLD_CT2_X               CT2_X
#define SLD_CT2_Y               CT2_Y

Actuator* but_ct2;
#define BUT_CT2_LABEL           "CT2"
#define BUT_CT2_X               CT2_X + CT2_X_OFFSET
#define BUT_CT2_Y               CT2_Y + CT2_Y_OFFSET

Actuator* typein_ct2;
#define TYPEIN_CT2_X            CT2_X
#define TYPEIN_CT2_Y            CT2_Y
#define TYPEIN_CT2_LENGTH       8
#define TYPEIN_CT2_FORMAT       "%2.6f"

/************************************ mgsteps ****************************/

#define MGSTEPS_X               6.0
#define MGSTEPS_Y               6.4
#define MGSTEPS_Y_OFFSET        0.65
#define MGSTEPS_MIN             0
#define MGSTEPS_MAX             20
#define MGSTEPS_INI             0

Actuator* typein_mgsteps;
#define TYPEIN_MGSTEPS_X        MGSTEPS_X
#define TYPEIN_MGSTEPS_Y        MGSTEPS_Y
#define TYPEIN_MGSTEPS_LABEL    "MGSTEPS"
#define TYPEIN_MGSTEPS_LENGTH   6
#define TYPEIN_MGSTEPS_FORMAT   "%6d"

Actuator* up_mgsteps;
#define UP_MGSTEPS_X            MGSTEPS_X + 0.0
#define UP_MGSTEPS_Y            MGSTEPS_Y + MGSTEPS_Y_OFFSET

Actuator* down_mgsteps;
#define DOWN_MGSTEPS_X          MGSTEPS_X + 1.155
#define DOWN_MGSTEPS_Y          MGSTEPS_Y + MGSTEPS_Y_OFFSET

Actuator* reset_mgsteps;
#define RESET_MGSTEPS_X         MGSTEPS_X + 0.575
#define RESET_MGSTEPS_Y         MGSTEPS_Y + MGSTEPS_Y_OFFSET
#define RESET_MGSTEPS_LABEL     "R"

/************************************ mgpls ****************************/

#define MGPLS_X                 8.35
#define MGPLS_Y                 6.4
#define MGPLS_Y_OFFSET          0.65
#define MGPLS_MIN               0
#define MGPLS_MAX               20
#define MGPLS_INI               0

Actuator* typein_mgpls;
#define TYPEIN_MGPLS_X          MGPLS_X
#define TYPEIN_MGPLS_Y          MGPLS_Y
#define TYPEIN_MGPLS_LABEL      "MGPLS"
#define TYPEIN_MGPLS_LENGTH     6
#define TYPEIN_MGPLS_FORMAT     "%6d"

Actuator* up_mgpls;
#define UP_MGPLS_X              MGPLS_X + 0.0
#define UP_MGPLS_Y              MGPLS_Y + MGPLS_Y_OFFSET

Actuator* down_mgpls;
#define DOWN_MGPLS_X            MGPLS_X  + 1.155
#define DOWN_MGPLS_Y            MGPLS_Y  + MGPLS_Y_OFFSET

Actuator* reset_mgpls;
#define RESET_MGPLS_X           MGPLS_X + 0.575
#define RESET_MGPLS_Y           MGPLS_Y + MGPLS_Y_OFFSET
#define RESET_MGPLS_LABEL       "R"

/************************************ add ****************************/

#define ADD_X                   0.0
#define ADD_Y                   1.8
#define ADD_Y_OFFSET            0.65
#define ADD_MIN                 0
#define ADD_MAX                 100
#define ADD_INI                 0

Actuator* typein_add;
#define TYPEIN_ADD_X            ADD_X
#define TYPEIN_ADD_Y            ADD_Y
#define TYPEIN_ADD_LABEL        "ADD"
#define TYPEIN_ADD_LENGTH       6
#define TYPEIN_ADD_FORMAT       "%6d"

Actuator* up_add;
#define UP_ADD_X                ADD_X + 0.0
#define UP_ADD_Y                ADD_Y + ADD_Y_OFFSET

Actuator* down_add;
#define DOWN_ADD_X              ADD_X + 1.155
#define DOWN_ADD_Y              ADD_Y + ADD_Y_OFFSET

Actuator* reset_add;
#define RESET_ADD_X             ADD_X + 0.575
#define RESET_ADD_Y             ADD_Y + ADD_Y_OFFSET
#define RESET_ADD_LABEL         "R"

/************************************ sub ****************************/

#define SUB_X                   2.35
#define SUB_Y                   1.8
#define SUB_Y_OFFSET            0.65
#define SUB_MIN                 0
#define SUB_MAX                 100
#define SUB_INI                 0

Actuator* typein_sub;
#define TYPEIN_SUB_X            SUB_X
#define TYPEIN_SUB_Y            SUB_Y
#define TYPEIN_SUB_LABEL        "SUB"
#define TYPEIN_SUB_LENGTH       6
#define TYPEIN_SUB_FORMAT       "%6d"

Actuator* up_sub;
#define UP_SUB_X                SUB_X + 0.0
#define UP_SUB_Y                SUB_Y + SUB_Y_OFFSET

Actuator* down_sub;
#define DOWN_SUB_X              SUB_X + 1.155
#define DOWN_SUB_Y              SUB_Y + SUB_Y_OFFSET
```



```
Actuator* reset_sub;
#define RESET_SUB_X             SUB_X + 0.575
#define RESET_SUB_Y             SUB_Y + SUB_Y_OFFSET
#define RESET_SUB_LABEL         "R"

/************************************ nedge ****************************/

#define NEDGE_X                 0.0
#define NEDGE_Y                 8.7
#define NEDGE_Y_OFFSET          0.65
#define NEDGE_MIN               0
#define NEDGE_MAX               3
#define NEDGE_INI               0

Actuator* typein_nedge;
#define TYPEIN_NEDGE_X          NEDGE_X
#define TYPEIN_NEDGE_Y          NEDGE_Y
#define TYPEIN_NEDGE_LABEL      "NEDGE"
#define TYPEIN_NEDGE_LENGTH     6
#define TYPEIN_NEDGE_FORMAT     "%6d"

Actuator* up_nedge;
#define UP_NEDGE_X              NEDGE_X + 0.0
#define UP_NEDGE_Y              NEDGE_Y + NEDGE_Y_OFFSET

Actuator* down_nedge;
#define DOWN_NEDGE_X            NEDGE_X + 1.155
#define DOWN_NEDGE_Y            NEDGE_Y + NEDGE_Y_OFFSET

Actuator* reset_nedge;
#define RESET_NEDGE_X           NEDGE_X + 0.575
#define RESET_NEDGE_Y           NEDGE_Y + NEDGE_Y_OFFSET
#define RESET_NEDGE_LABEL       "R"

/************************************ indq ****************************/

#define INDQ_X                  2.35
#define INDQ_Y                  8.7
#define INDQ_Y_OFFSET           0.65
#define INDQ_MIN                1
#define INDQ_MAX                8
#define INDQ_INI                1

Actuator* typein_indq;
#define TYPEIN_INDQ_X           INDQ_X
#define TYPEIN_INDQ_Y           INDQ_Y
#define TYPEIN_INDQ_LABEL       "INDQ"
#define TYPEIN_INDQ_LENGTH      6
#define TYPEIN_INDQ_FORMAT      "%6d"

Actuator* up_indq;
#define UP_INDQ_X               INDQ_X + 0.0
#define UP_INDQ_Y               INDQ_Y + INDQ_Y_OFFSET

Actuator* down_indq;
#define DOWN_INDQ_X             INDQ_X + 1.155
#define DOWN_INDQ_Y             INDQ_Y + INDQ_Y_OFFSET

Actuator* reset_indq;
#define RESET_INDQ_X            INDQ_X + 0.575
#define RESET_INDQ_Y            INDQ_Y + INDQ_Y_OFFSET
#define RESET_INDQ_LABEL        "R"
```

```
/************************************ nfilt ****************************/

#define NFILT_X                 0.0
#define NFILT_Y                 4.1
#define NFILT_Y_OFFSET          0.65
#define NFILT_MIN               0
#define NFILT_MAX               10
#define NFILT_INI               2

Actuator* typein_nfilt;
#define TYPEIN_NFILT_X          NFILT_X
#define TYPEIN_NFILT_Y          NFILT_Y
#define TYPEIN_NFILT_LABEL      "NFILT"
#define TYPEIN_NFILT_LENGTH     6
#define TYPEIN_NFILT_FORMAT     "%6d"

Actuator* up_nfilt;
#define UP_NFILT_X              NFILT_X + 0.0
#define UP_NFILT_Y              NFILT_Y + NFILT_Y_OFFSET

Actuator* down_nfilt;
#define DOWN_NFILT_X            NFILT_X + 1.155
#define DOWN_NFILT_Y            NFILT_Y + NFILT_Y_OFFSET

Actuator* reset_nfilt;
#define RESET_NFILT_X           NFILT_X + 0.575
#define RESET_NFILT_Y           NFILT_Y + NFILT_Y_OFFSET
#define RESET_NFILT_LABEL       "R"

/************************************ inter ****************************/

#define INTER_X                 2.35
#define INTER_Y                 4.1
#define INTER_Y_OFFSET          0.65
#define INTER_MIN               0
#define INTER_MAX               4
#define INTER_INI               2

Actuator* typein_inter;
#define TYPEIN_INTER_X          INTER_X
#define TYPEIN_INTER_Y          INTER_Y
#define TYPEIN_INTER_LABEL      "INTER"
#define TYPEIN_INTER_LENGTH     6
#define TYPEIN_INTER_FORMAT     "%6d"

Actuator* up_inter;
#define UP_INTER_X              INTER_X + 0.0
#define UP_INTER_Y              INTER_Y + INTER_Y_OFFSET

Actuator* down_inter;
#define DOWN_INTER_X            INTER_X + 1.155
#define DOWN_INTER_Y            INTER_Y + INTER_Y_OFFSET

Actuator* reset_inter;
#define RESET_INTER_X           INTER_X + 0.575
#define RESET_INTER_Y           INTER_Y + INTER_Y_OFFSET
#define RESET_INTER_LABEL       "R"

/************************************ mg1 ****************************/

#define MG1_X                   6.0
#define MG1_Y                   8.7
#define MG1_Y_OFFSET            0.65
#define MG1_MIN                 0
```

```
#define MG1_MAX                 20
#define MG1_INI                 0

Actuator* typein_mg1;
#define TYPEIN_MG1_X            MG1_X
#define TYPEIN_MG1_Y            MG1_Y
#define TYPEIN_MG1_LABEL        "MG1"
#define TYPEIN_MG1_LENGTH        6
#define TYPEIN_MG1_FORMAT       "%6d"

Actuator* up_mg1;
#define UP_MG1_X                MG1_X + 0.0
#define UP_MG1_Y                MG1_Y + MG1_Y_OFFSET

Actuator* down_mg1;
#define DOWN_MG1_X              MG1_X + 1.155
#define DOWN_MG1_Y              MG1_Y + MG1_Y_OFFSET

Actuator* reset_mg1;
#define RESET_MG1_X             MG1_X + 0.575
#define RESET_MG1_Y             MG1_Y + MG1_Y_OFFSET
#define RESET_MG1_LABEL         "R"

/************************************ mg2 ************************************/

#define MG2_X                   8.35
#define MG2_Y                   8.7
#define MG2_Y_OFFSET            0.65
#define MG2_MIN                 0
#define MG2_MAX                 20
#define MG2_INI                 0

Actuator* typein_mg2;
#define TYPEIN_MG2_X            MG2_X
#define TYPEIN_MG2_Y            MG2_Y
#define TYPEIN_MG2_LABEL        "MG2"
#define TYPEIN_MG2_LENGTH       6
#define TYPEIN_MG2_FORMAT       "%6d"

Actuator* up_mg2;
#define UP_MG2_X                MG2_X + 0.0
#define UP_MG2_Y                MG2_Y + MG2_Y_OFFSET

Actuator* down_mg2;
#define DOWN_MG2_X              MG2_X + 1.155
#define DOWN_MG2_Y              MG2_Y + MG2_Y_OFFSET

Actuator* reset_mg2;
#define RESET_MG2_X             MG2_X + 0.575
#define RESET_MG2_Y             MG2_Y + MG2_Y_OFFSET
#define RESET_MG2_LABEL         "R"

/************************************ lnsing ************************************/

#define LNSING_X                6.0
#define LNSING_Y                4.1
#define LNSING_Y_OFFSET         0.65
#define LNSING_MIN              0
#define LNSING_MAX              500
#define LNSING_INI              0

Actuator* typein_lnsing;
#define TYPEIN_LNSING_X         LNSING_X
#define TYPEIN_LNSING_Y         LNSING_Y
#define TYPEIN_LNSING_LABEL     "LNSING"
#define TYPEIN_LNSING_LENGTH    6
#define TYPEIN_LNSING_FORMAT    "%6d"

Actuator* up_lnsing;
#define UP_LNSING_X             LNSING_X + 0.0
#define UP_LNSING_Y             LNSING_Y + LNSING_Y_OFFSET

Actuator* down_lnsing;
#define DOWN_LNSING_X           LNSING_X + 1.155
#define DOWN_LNSING_Y           LNSING_Y + LNSING_Y_OFFSET

Actuator* reset_lnsing;
#define RESET_LNSING_X          LNSING_X + 0.575
#define RESET_LNSING_Y          LNSING_Y + LNSING_Y_OFFSET
#define RESET_LNSING_LABEL      "R"

/************************************ plsing ************************************/

#define PLSING_X                8.35
#define PLSING_Y                4.1
#define PLSING_Y_OFFSET         0.65
#define PLSING_MIN              0
#define PLSING_MAX              500
#define PLSING_INI              0

Actuator* typein_plsing;
#define TYPEIN_PLSING_X         PLSING_X
#define TYPEIN_PLSING_Y         PLSING_Y
#define TYPEIN_PLSING_LABEL     "PLSING"
#define TYPEIN_PLSING_LENGTH    6
#define TYPEIN_PLSING_FORMAT    "%6d"

Actuator* up_plsing;
#define UP_PLSING_X             PLSING_X + 0.0
#define UP_PLSING_Y             PLSING_Y + PLSING_Y_OFFSET

Actuator* down_plsing;
#define DOWN_PLSING_X           PLSING_X + 1.155
#define DOWN_PLSING_Y           PLSING_Y + PLSING_Y_OFFSET

Actuator* reset_plsing;
#define RESET_PLSING_X          PLSING_X + 0.575
#define RESET_PLSING_Y          PLSING_Y + PLSING_Y_OFFSET
#define RESET_PLSING_LABEL      "R"

/************************************ adaption label ************************************/

#define LABEL_ADAPTION_LABEL    "ADAPTION DIRECTIONS"
#define LABEL_ADAPTION_X        1.5
#define LABEL_ADAPTION_Y        19.5

/************************************ adaption down direction ************************************/

#define ADP_DN_DIRECTION_X      0.8
#define ADP_DN_DIRECTION_Y      19.5

/************************************ adaption up direction ************************************/

#define ADP_UP_DIRECTION_X      6.5
#define ADP_UP_DIRECTION_Y      19.5

/************************************ point direction ************************************/
```



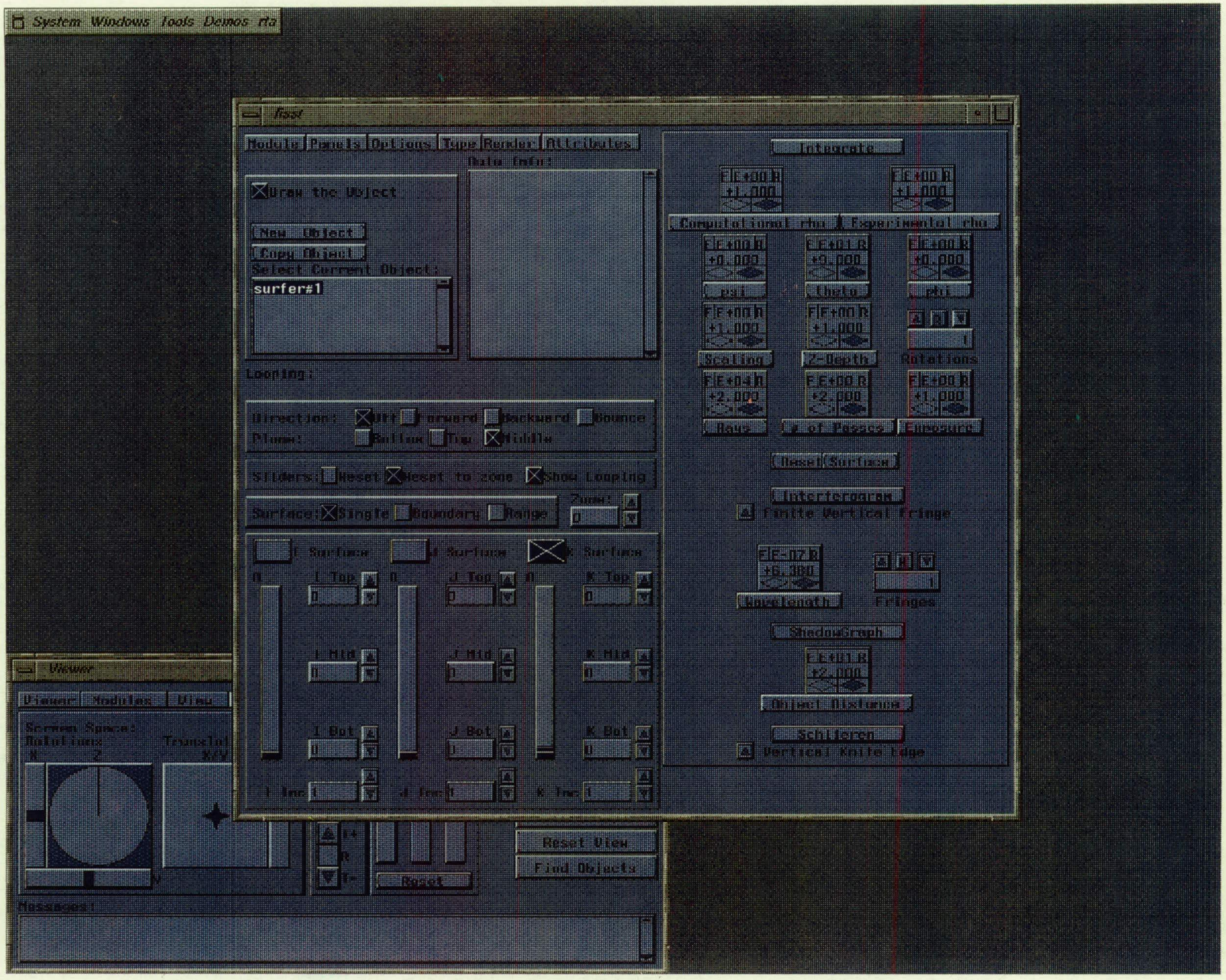
```
#define PTS_DIRECTION_X         1.5
#define PTS_DIRECTION_Y         18.5

/************************************ line direction ************************************/

#define LIN_DIRECTION_X         1.5
#define LIN_DIRECTION_Y         18.0

/************************************ plane direction ************************************/

#define PLA_DIRECTION_X         1.5
#define PLA_DIRECTION_Y         17.5

/************************************ button addapt points ************************************/

#define BUT_ADAPTPTS_LABEL      "Adapt Grid"
#define BUT_ADAPTPTS_X          0.2
#define BUT_ADAPTPTS_Y          0.6

/************************************ button remove points ************************************/

#define BUT_REMOVE_LABEL        " Remove "
#define BUT_REMOVE_X            6.0
#define BUT_REMOVE_Y            0.6

/************************************ button undo ************************************/

#define BUT_UNDO_LABEL          "Undo"
#define BUT_UNDO_X              3.7
#define BUT_UNDO_Y              0.6

/************************************ button addpts ************************************/

#define BUT_ADDPTS_LABEL        "Insert"
#define BUT_ADDPTS_X            -0.1
#define BUT_ADDPTS_Y            1.3

/************************************ button subpts ************************************/

#define BUT_SUBPTS_LABEL        "Delete"
#define BUT_SUBPTS_X            2.25
#define BUT_SUBPTS_Y            1.3

/************************************ button march ************************************/

#define BUT_MARCH_LABEL         ""
#define BUT_MARCH_X             0.0
#define BUT_MARCH_Y             14.0

/************************************ button marchpl ************************************/

#define BUT_MARCHPL_LABEL       " MARCH, MARCHPL"
#define BUT_MARCHPL_X           0.5
#define BUT_MARCHPL_Y           14.0

/************************************ button orths ************************************/

#define BUT_ORTHS_LABEL         ""
#define BUT_ORTHS_X             0.0
#define BUT_ORTHS_Y             14.5

/************************************ button orthe ************************************/

#define BUT_ORTHE_LABEL         " ORTHS, ORTHE"
#define BUT_ORTHE_X             0.5
#define BUT_ORTHE_Y             14.5

/************************************ button geom ************************************/

#define BUT_GEOM_LABEL          ""
#define BUT_GEOM_X              0.0
#define BUT_GEOM_Y              13.5

/************************************ button qfun ************************************/

#define BUT_QFUN_LABEL          " GEOM, QFUN"
#define BUT_QFUN_X              0.5
#define BUT_QFUN_Y              13.5

/************************************ button calculator ************************************/

#define BUT_CALCULATOR_LABEL    "Calculator"
#define BUT_CALCULATOR_X        0.0
#define BUT_CALCULATOR_Y        15.5

/*******************************************************************************************/

Actuator
   *but_march, *but_marchpl,
   *but_orths, *but_orthe,
   *but_geom,
   *but_qfun,
   *but_addpts, *but_subpts,
   *but_adaptpts,
   *but_undo,
   *adp_up_direction,
   *adp_dn_direction,
   *but_remove,
   *but_calculator;

Actuator
   *dummy;

/*******************************************************************************************/

#define MARCH_INI               FALSE
#define MARCHPL_INI             FALSE
#define ORTHS_INI               TRUE
#define ORTHE_INI               TRUE
#define GEOM_INI                FALSE
#define QFUN_INI                TRUE
#define LINEINVERSE_INI         FALSE
#define POINTINVERSE_INI        FALSE
#define PLANEINVERSE_INI        FALSE

#define I_STEP_ADAPT_LABEL      "1\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"
#define J_STEP_ADAPT_LABEL      "2\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"
#define K_STEP_ADAPT_LABEL      "3\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"

int
   ireadq = FALSE;

int   /* adaption range */
   adapt_imin,
   adapt_imax,
   adapt_jmin,
   adapt_jmax,
   adapt_kmin,
```

```
   adapt_kmax,
   adapt_dir[] = { 1, 2, 3 };

int   /* direction range */
   idir,
   jdir,
   kdir;

int   /* dataset range */
   data_imin,
   data_imax,
   data_jmin,
   data_jmax,
   data_kmin,
   data_kmax;

long int /* undo range */
   undo_imax,
   undo_jmax,
   undo_kmax;

long int /* remove subset range */
   rmset_imin,
   rmset_imax,
   rmset_jmin,
   rmset_jmax,
   rmset_kmin,
   rmset_kmax;

long int
   data_xyz_bytes,
   data_q_bytes;

float
   *qaddress,
   unadapt_grid[XYZDIM][ED][JD][ID],
   unadapt_q[QDIM][ED][JD][ID],
   *adapt_grid=NULL,
   *adapt_q=NULL;

char
   sage_claml_str[8];

int
   but_rdsmin_flag = TRUE,
   but_rdsmax_flag = TRUE,
   but_clam1_flag = TRUE,
   but_clam2_flag = TRUE,
   but_ct1_flag = TRUE,
   but_ct2_flag = TRUE;

#define I_J_K          0
#define I_J_REVK       1
#define I_REVJ_K       2
#define I_REVJ_REVK    3
#define REVI_J_K       4
#define REVI_J_REVK    5
#define REVI_REVJ_K    6
#define REVI_REVJ_REVK 7

#define I_K_J          8
#define I_K_REVJ       9
#define I_REVK_J       10
#define I_REVK_REVJ    11
#define REVI_K_J       12
#define REVI_K_REVJ    13
#define REVI_REVK_J    14
#define REVI_REVK_REVJ 15

#define J_I_K          16
#define J_I_REVK       17
#define J_REVI_K       18
#define J_REVI_REVK    19
#define REVJ_I_K       20
#define REVJ_I_REVK    21
#define REVJ_REVI_K    22
#define REVJ_REVI_REVK 23

#define J_K_I          24
#define J_K_REVI       25
#define J_REVK_I       26
#define J_REVK_REVI    27
#define REVJ_K_I       28
#define REVJ_K_REVI    29
#define REVJ_REVK_I    30
#define REVJ_REVK_REVI 31

#define K_I_J          32
#define K_I_REVJ       33
#define K_REVI_J       34
#define K_REVI_REVJ    35
#define REVK_I_J       36
#define REVK_I_REVJ    37
#define REVK_REVI_J    38
#define REVK_REVI_REVJ 39

#define K_J_I          40
#define K_J_REVI       41
#define K_REVJ_I       42
#define K_REVJ_REVI    43
#define REVK_J_I       44
#define REVK_J_REVI    45
#define REVK_REVJ_I    46
#define REVK_REVJ_REVI 47

#define FOR_I_PTS      "FORWARD IN I POINTS"
#define REV_I_PTS      "REVERSE IN I POINTS"
#define FOR_I_LIN      "FORWARD IN I LINES"
#define REV_I_LIN      "REVERSE IN I LINES"
#define FOR_I_PLA      "FORWARD IN I PLANES"
#define REV_I_PLA      "REVERSE IN I PLANES"

#define FOR_J_PTS      "FORWARD IN J POINTS"
#define REV_J_PTS      "REVERSE IN J POINTS"
#define FOR_J_LIN      "FORWARD IN J LINES"
#define REV_J_LIN      "REVERSE IN J LINES"
#define FOR_J_PLA      "FORWARD IN J PLANES"
#define REV_J_PLA      "REVERSE IN J PLANES"

#define FOR_K_PTS      "FORWARD IN K POINTS"
#define REV_K_PTS      "REVERSE IN K POINTS"
#define FOR_K_LIN      "FORWARD IN K LINES"
#define REV_K_LIN      "REVERSE IN K LINES"
#define FOR_K_PLA      "FORWARD IN K PLANES"
#define REV_K_PLA      "REVERSE IN K PLANES"

char
   for_i_pts[] = FOR_I_PTS,
```



```
   rev_i_pts[] = REV_I_PTS,
   for_i_lin[] = FOR_I_LIN,
   rev_i_lin[] = REV_I_LIN,
   for_i_pla[] = FOR_I_PLA,
   rev_i_pla[] = REV_I_PLA,

   for_j_pts[] = FOR_J_PTS,
   rev_j_pts[] = REV_J_PTS,
   for_j_lin[] = FOR_J_LIN,
   rev_j_lin[] = REV_J_LIN,
   for_j_pla[] = FOR_J_PLA,
   rev_j_pla[] = REV_J_PLA,

   for_k_pts[] = FOR_K_PTS,
   rev_k_pts[] = REV_K_PTS,
   for_k_lin[] = FOR_K_LIN,
   rev_k_lin[] = REV_K_LIN,
   for_k_pla[] = FOR_K_PLA,
   rev_k_pla[] = REV_K_PLA,

   *direction_of_ijk[48][3] = {
       for_i_pts, for_j_lin, for_k_pla,
       for_i_pts, for_j_lin, rev_k_pla,
       for_i_pts, rev_j_lin, for_k_pla,
       for_i_pts, rev_j_lin, rev_k_pla,

       rev_i_pts, for_j_lin, for_k_pla,
       rev_i_pts, for_j_lin, rev_k_pla,
       rev_i_pts, rev_j_lin, for_k_pla,
       rev_i_pts, rev_j_lin, rev_k_pla,

       for_i_pts, for_k_lin, for_j_pla,
       for_i_pts, for_k_lin, rev_j_pla,
       for_i_pts, rev_k_lin, for_j_pla,
       for_i_pts, rev_k_lin, rev_j_pla,

       rev_i_pts, for_k_lin, for_j_pla,
       rev_i_pts, for_k_lin, rev_j_pla,
       rev_i_pts, rev_k_lin, for_j_pla,
       rev_i_pts, rev_k_lin, rev_j_pla,

       for_j_pts, for_i_lin, for_k_pla,
       for_j_pts, for_i_lin, rev_k_pla,
       for_j_pts, rev_i_lin, for_k_pla,
       for_j_pts, rev_i_lin, rev_k_pla,

       rev_j_pts, for_i_lin, for_k_pla,
       rev_j_pts, for_i_lin, rev_k_pla,
       rev_j_pts, rev_i_lin, for_k_pla,
       rev_j_pts, rev_i_lin, rev_k_pla,

       for_j_pts, for_k_lin, for_i_pla,
       for_j_pts, for_k_lin, rev_i_pla,
       for_j_pts, rev_k_lin, for_i_pla,
       for_j_pts, rev_k_lin, rev_i_pla,

       rev_j_pts, for_k_lin, for_i_pla,
       rev_j_pts, for_k_lin, rev_i_pla,
       rev_j_pts, rev_k_lin, for_i_pla,
       rev_j_pts, rev_k_lin, rev_i_pla,

       for_k_pts, for_i_lin, for_j_pla,
       for_k_pts, for_i_lin, rev_j_pla,
       for_k_pts, rev_i_lin, for_j_pla,
       for_k_pts, rev_i_lin, rev_j_pla,

       rev_k_pts, for_i_lin, for_j_pla,
       rev_k_pts, for_i_lin, rev_j_pla,
       rev_k_pts, rev_i_lin, for_j_pla,
       rev_k_pts, rev_i_lin, rev_j_pla,

       for_k_pts, for_j_lin, for_i_pla,
       for_k_pts, for_j_lin, rev_i_pla,
       for_k_pts, rev_j_lin, for_i_pla,
       for_k_pts, rev_j_lin, rev_i_pla,

       rev_k_pts, for_j_lin, for_i_pla,
       rev_k_pts, for_j_lin, rev_i_pla,
       rev_k_pts, rev_j_lin, for_i_pla,
       rev_k_pts, rev_j_lin, rev_i_pla
   };

int cur_direction = 0;

#define PTS_DIRECTION 0
#define LIN_DIRECTION 1
#define PLA_DIRECTION 2

Actuator
   *pts_direction,
   *lin_direction,
   *pla_direction;
```

Appendix B: CISS GUI with Source Code



```
#
#       Makefile for fisst module
#
#       instructions:
#
#               make            compiles code to update executable file
#               make clean      removes object files
#               make clobber    removes object files and the executable file
#               make dirty      touches source files
#               make link       removes executable file
#               make lint       runs lint on each module source file
#

#
#       include global definitions
#

include ../../Makefile.defs

#
#       must define CFLAGS here to be equal to:
#
#               ${IFLAGS}               -I search path for include files
#               ${GLOBALDEFINES}        -D global defines (e.g., -DDEBUG)
#               ${GLOBALCFLAGS}         e.g., -g, -O2 and other optional flags
#
#       (MYFLAGS is for additional flags you may want to add after these)
#

MYCFLAGS =
CFLAGS =        ${IFLAGS} ${GLOBALDEFINES} ${GLOBALCFLAGS} ${MYCFLAGS}
FFLAGS =        -static -mp -pfa -WK,-O=5,-UR=2,-UR2=200,-LM=20000,-LM2=50000

#
#       external library link flags
#

LFLAGS = -lgl_s -lbsd -lm -lF77 -lU77  \
        -lI77_mp -lF77_mp_G0 -lI77 -lI77_G0     \
        -lU77 -lU77_G0 -lisam -lisam_G0 -lm     \
        -lmpc


#
#       module application executable file
#

MOD =           ${INSTALLDIR}fisst

#
#       object files
#

MODOBJS =       main.o          \
                panels.o        \
                fisst.o         \
                fisstSupport.o

#
#       source files
#

MODSRC =        main.c          \
                panels.c        \
                fisstSupport.c  \
                fisst.f         \
                fisstSupport.h

#
#       FAST library archive files
#

FASTLIBS =      ${LIBDIR}libfldpan.a    \
                ${LIBDIR}libfldddata.a  \
                ${LIBDIR}libviewI.a     \
                ${LIBDIR}libpanu.a      \
                ${LIBDIR}libcmap.a      \
                ${LIBDIR}libfpanel.a    \
                ${LIBDIR}libmem.a       \
                ${LIBDIR}libmodule.a    \
                ${LIBDIR}liboobj.a      \
                ${LIBDIR}liblist.a      \
                ${LIBDIR}libcps.a       \
                ${LIBDIR}libFgl.a

#
#       include module definitions
#

include ../Makefile.defs
```



```
      subroutine imagef(r,q)

C=======================================================================
C
C   Author:
C      Leslie A. Yates
C      Eloret Institute
C      NASA Ames Research Center
C      Moffett Field, CA 94035
C      (415) 604-3436
C
C   Date:
C      December, 1991
C
C=======================================================================

      character*1 image, knife
      character*12 filenam, gname, qname
      character*1 mulsin,ctype
      logical mod(600001)
C      dimension q(51*51*61*3), r(51*51*61*3)
      dimension q(1), r(1)
      dimension jdim(10), kdim(10), ldim(10)

      integer imagerErr

      common/data/ nsub(10), jsub1(10,10), jsub2(10,10), ksub1(10,10),
     .  ksub2(10,10), lsub1(10,10), lsub2(10,10)
      common/image/Ixmax,Iymax,mod,p(600001,4)

      common/subjkl/jdim(10),kdim(10),ldim(10)
      common/inteq/ngrid,npts,rho0,rhoinf,al,pass,psi,theta,phi,dz,nphi,
     &            gstrch,strch,ctype,mulsin
      common/shadow/camera
      common/schl/knife
      common/interf/awl,cons,vert,horiz,type
      character*1 type

C      equivalence (q(1), p(1,4))
      print*,"imagef"
C      write(*,*)
C      write(*,*)' How many points would you like to use to define'
C      write(*,*)' the image plane (up to 600,001)?'
C      read(*,*)npts
C      if(npts.gt.600001)npts=600001
      Ixmax=npts
      Iymax=1
C      write(*,*)

c Input parameters: grid name, etc.

C      Write(*,*)' What is the grid name?'
C      Read(*,'(a12)')qname
C      Write(*,*)' What is the flow field name?'
C      Read(*,'(a12)')qname
C      write(*,*)
C      write(*,*)' Input the type of calculation.'
C      write(*,*)'       2 for 2-dimensional'
C      write(*,*)'       A for axi-symmetric'
C      write(*,*)'       3 for 3-dimensional'
C      Read(*,'(a1)')ctype
C      if(ctype.eq.'a')ctype='A'
C      write(*,*)
C      Write(*,*)' Input m if there are are multiple grids.'
C      Write(*,*)'       s if there is a single grid.'
C      Read(*,'(a1)')mulsin
C      Write(*,*)
C      write(*,*)' Input the computational freestream density.'
C      read(*,*) rho0
C      write(*,*)' Input the experimental freestream density. '
C      write(*,*)'          (fraction of STD value).'
C      read(*,*) rhoinf
C      write(*,*)' Input the length used for nondimensionalization.'
C      read(*,*) al
C      write(*,*)' Input the number of passes. If the symmetry plane is '
C      write(*,*)'  normal to the image plane and if symmetry conditions'
C      write(*,*)'  were used in the computation, double the number.'
C      read(*,*) pass
C      write(*,*)
C      if(ctype.eq.'3')then
C           Write(*,*)' What are psi, theta, and phi for the rotation?'
C           read(*,*)psi,theta,phi
C      end if
C      if(ctype.eq.'2')then
C           Write(*,*)' What is the thickness of the 2-D flow?'
C          read(*,*)dz
C      end if
C      if((ctype.eq.'A').or.(ctype.eq.'a'))then
C           Write(*,*)' How many planes from 0 to 90 degrees (180)?'
C           read(*,*)nphi
C           if(nphi.gt.180)nphi=180
C      end if

      GD = 0.000292/1.293

c Get stretch and centers of the multiple grid. This information will be
c used to place the image in the center of the grid.
      print*,"ctype=",ctype
      if(ctype.eq.'3')Call Multi3(mulsin,gname,ngrid,jdim,kdim,ldim,
     .      psi,theta,phi,Ixmax,Iymax,strch,xcg,ycg,z)
      if(ctype.ne.'3')Call Multi2(ctype,mulsin,gname,ngrid,jdim,kdim,
     .      Ixmax,Iymax,strch,xcg,ycg,z)

c Initialize the integration planes and the model mask.

      Do 50 j=1,3
       Do 50 i=1,Ixmax*Iymax
        p(i,j)=0.0
 50   Continue

      Do 60 i=1,Ixmax*Iymax
        mod(i)=.FALSE.
 60   Continue
      print *,"ngrid=",ngrid
      Do 100 ng=1,ngrid

       nj = jdim(ng)
       nk = kdim(ng)
       nl = ldim(ng)
      print *,"Entering Getrq"
       if(ctype.eq.'3')
     .       Call Getrq3(gname,qname,mulsin,ng,ngrid,r,q,nj,nk,nl)
       if(ctype.ne.'3')
     .       Call Getrq2(gname,qname,mulsin,ng,ngrid,r,q,nj,nk)

c Get the index of refraction minus one
      print *,"Entering deln"
       Call deln(q,nj,nk,nl,rho0,rhoinf,GD,deln0)
```

```
c Rotate and stretch the grid.
      print *,"Entering view"
       if(ctype.eq.'3')Call view3(r,nj,nk,nl,psi,
     .                    theta,phi,strch,xcg,ycg)
       if(ctype.ne.'3')Call view2(r,nj,nk,strch,xcg,ycg)

c Zero out points associated with the model for 3-D flows or for points
c not covered by the grid for 2-D and axially symmetric flows.
      print *,"Entering model"
       if(ctype.eq.'3')Call model(ng,r,nj,nk,nl)
      print *,"Entering Grdovr"
       if(ctype.ne.'3')Call Grdovr(ng,r,nj,nk)

c Integrate through the flow field
      print *,"Entering Integ, ctype=",ctype
       if(ctype.eq.'3')Call Integ3(ng,r,q,nj,nk,nl,deln0)
       if(ctype.eq.'A')Call IntegA(ng,r,q,nj,nk,deln0,nphi)
       if(ctype.eq.'2')Call Integ2(ng,r,q,nj,nk,deln0,dz)
      print *,"Left Integ"
 100  continue

      do 110 j=1,4
       do 110 i=1,Ixmax*Iymax
         if(mod(i))p(i,j)=0.0
 110  continue

c Query as to the desired image
C  5   Write(*,*)
C      Write(*,*)' Input the type of image you wish to create'
C      Write(*,*)'             i for interferogram'
C      Write(*,*)'             s for shadowgraph   '
C      Write(*,*)'             k for schlieren     '
C      Read(*,'(a1)')image
C      if(image.ne.'i')go to 10

c Form infinite fringe and finite fringe interferograms.

C      write(*,*)' Input the wave length of light (same scale as length)'
C      read(*,*) awl
C      cons = 8.0*atan(1.0)*al*pass/(awl*strch)
C      call fringes(cons,filenam)
C      go to 20

C  10  if(image.ne.'s')go to 15
C      write(*,*)
C      write(*,*)' Input distance of focus plane from the flow field.'
C      read(*,*)camera
C      camera=camera*strch*al
C      call shadowg(camera,filenam)
C      go to 20

C  15  write(*,*)
C      write(*,*)' Input the direction of the knife blade.'
C      write(*,*)'             h for horizontal '
C      write(*,*)'             v for vertical   '
C      read(*,'(A1)')knife
C      call schlieren(knife,filenam)

  20  do 120 i=1,Ixmax*Iymax
        if(mod(i))p(i,4)=0.0
 120  continue

C      open(unit=8,file=filenam,status='unknown')
C      write(8,'(4014)') (int(p(i,4)),i=1,Ixmax*Iymax)
C      close(8)

C      write(*,*)
C      write(*,*)' Do you want another image (y/n)?'
C       read(*,'(a1)')image
C      if(image.eq.'y')go to 5

C      write(*,*)
C      write(*,*)' The adjusted image sizes are '
C      write(*,*)  Ixmax,' by ',Iymax,' pixels.'
C      x1 = - xcg/strch
C      x2 = x1 + float(Ixmax)/strch
C      y1 = - ycg/strch
C      y2 = y1 + float(Iymax)/strch
C      write(*,*)
C      write(*,*)' The values for the corners of the image are:'
C      write(*,*)'       x1, x2: ',x1,x2
C      write(*,*)'       y1, y2: ',y1,y2

C      imageErr = drawimage(Ixmax,Iymax,p(1,4))

      print *, "leaving imagef"

      return
      end


*******************************************************************************

      Subroutine Multi3(mulsin,gname,ngrid,jdim,kdim,ldim,
     . psi,theta,phi,Ixmax,Iymax,strch,xcg,ycg,r)

C=============================================================
C
C   Author:
C      Leslie A. Yates
C      Eloret Institute
C      NASA Ames Research Center
C      Moffett Field, CA 94035
C      (415) 604-3436
C
C   Date:
C      December, 1991
C
C=============================================================

      character*12 gname
      character*1 mulsin
      dimension jdim(10), kdim(10), ldim(10)
      dimension r(1)

      common/date/ nsub(10), jsub1(10,10), jsub2(10,10), ksub1(10,10),
     .  ksub2(10,10), lsub1(10,10), lsub2(10,10)

      twopi=8.0*atan(1.0)
      d2rad = twopi/360.
      psi=psi*d2rad+twopi
      theta=theta*d2rad+twopi
      phi=phi*d2rad+twopi

      cospsi=cos(psi)
      sinpsi=sin(psi)
      costhe=cos(theta)
```



```
      sinthe=sin(theta)
      cosphi=cos(phi)
      sinphi=sin(phi)

      rot11= costhe*cospsi
      rot12= costhe*sinpsi
      rot13=-sinthe

      rot21= sinphi*sinthe*cospsi - cosphi*sinpsi
      rot22= sinphi*sinthe*sinpsi + cosphi*cospsi
      rot23= sinphi*costhe

      rot31= cosphi*sinthe*cospsi + sinphi*sinpsi
      rot32= cosphi*sinthe*sinpsi - sinphi*cospsi
      rot33= cosphi*costhe

C      open(unit=15,file=gname,status='old',form='unformatted')
      ngrid=1
C     if(mulsin.eq.'m')read(15)ngrid
C     read(15)(jdim(n),kdim(n),ldim(n), n=1,ngrid)
C
C     do 20 n=1,ngrid
C      write(*,*)
C      write(*,*)' For grid ', n,', the nj, nk, and nl are ',
C     .          jdim(n), kdim(n), ldim(n)
C      write(*,*)' How many subsets of this grid would you like?'
C      read(*,*)nsub(n)
C      do 30 m=1,nsub(n)
C        write(*,*)' For subset ', m, ' input jmin, jmax, kmin, kmax,'
C        write(*,*)'     lmin, and lmax.'
C        read(*,*)jsub1(n,m),jsub2(n,m),ksub1(n,m),ksub2(n,m),
C     .           lsub1(n,m),lsub2(n,m)
C
C         jsub1(n,m) = max0(1      ,jsub1(n,m))
C        jsub1(n,m) = min0(jdim(n),jsub1(n,m))
C         jsub2(n,m) = max0(1,      jsub2(n,m))
C        jsub2(n,m) = min0(jdim(n),jsub2(n,m))
C
C         ksub1(n,m) = max0(1,      ksub1(n,m))
C        ksub1(n,m) = min0(kdim(n),ksub1(n,m))
C         ksub2(n,m) = max0(1,      ksub2(n,m))
C        ksub2(n,m) = min0(kdim(n),ksub2(n,m))
C
C         lsub1(n,m) = max0(1,      lsub1(n,m))
C        lsub1(n,m) = min0(ldim(n),lsub1(n,m))
C         lsub2(n,m) = max0(1      ,lsub2(n,m))
C        lsub2(n,m) = min0(ldim(n),lsub2(n,m))
C
C 30    continue
C 20   continue

      n=1
C      do 50 n=1,ngrid
C
C       READ(15) (r(j,1),j=1,jdim(n)*kdim(n)*ldim(n)),
C     &          (r(j,2),j=1,jdim(n)*kdim(n)*ldim(n)),
C     &          (r(j,3),j=1,jdim(n)*kdim(n)*ldim(n))
       if(n.eq.1)then
          xmin = rot11*r(jsub1(1,1)*ksub1(1,1)*lsub1(1,1))
     .         + rot12*r(jdim(n)*kdim(n)*ldim(n)
     .                  +jsub1(1,1)*ksub1(1,1)*lsub1(1,1))
     .         + rot13*r(jdim(n)*kdim(n)*ldim(n)*2
     .                  +jsub1(1,1)*ksub1(1,1)*lsub1(1,1))
          xmax = xmin
          ymin = rot21*r(jsub1(1,1)*ksub1(1,1)*lsub1(1,1))
     .         + rot22*r(jdim(n)*kdim(n)*ldim(n)
     .                  +jsub1(1,1)*ksub1(1,1)*lsub1(1,1))
     .         + rot23*r(jdim(n)*kdim(n)*ldim(n)*3
     .                  +jsub1(1,1)*ksub1(1,1)*lsub1(1,1))
          ymax=ymin
       end if

       do 100 m=1,nsub(n)
        do 100 l=lsub1(n,m),lsub2(n,m)
         do 100 k=ksub1(n,m),ksub2(n,m)
          do 100 j=jsub1(n,m),jsub2(n,m)
           npt = j + (k-1)*jdim(n) + (l-1)*jdim(n)*kdim(n)
           anewx = rot11*r(npt)
     .           + rot12*r(jdim(n)*kdim(n)*ldim(n) + npt)
     .           + rot13*r(jdim(n)*kdim(n)*ldim(n)*2 + npt)
           anewy = rot21*r(npt)
     .           + rot22*r(jdim(n)*kdim(n)*ldim(n) + npt)
     .           + rot23*r(jdim(n)*kdim(n)*ldim(n)*2 + npt)
           xmin=amin1(xmin,anewx)
           xmax=amax1(xmax,anewx)
           ymin=amin1(ymin,anewy)
           ymax=amax1(ymax,anewy)
 100     continue
C  50   continue
C       close(15)
       print*,"xmin,xmax,ymin,ymax",xmin,xmax,ymin,ymax
       print*,"jdim,kdim,ldim",jdim(1),kdim(1),ldim(1)
        if((xmin.eq.xmax).or.(ymin.eq.ymax))then
          write(*,*)' Data in this view all lies along a single line.'
          write(*,*)' No further computations will be performed.'
          stop
       end if
c
c To minimize the size of the image, Ixmax and Iymax are adjusted so that the image
c does not extend beyond the computational domain.
c
        ratio = (xmax-xmin)/(ymax-ymin)
        Iymax = int(sqrt(float(Ixmax)/ratio))
        Ixmax = int(sqrt(ratio*float(Ixmax)))
        Ixmax = int(((Ixmax+1)/2)*2-1
        xstrch=Ixmax/(xmax-xmin)
        ystrch=Iymax/(ymax-ymin)
        strch = amin1(xstrch,ystrch)
        xcg = (Ixmax - strch*(xmax+xmin))/2.
        ycg = (Iymax - strch*(ymax+ymin))/2.
       return
       end

*******************************************************************************
**
      Subroutine Multi2(ctype,mulsin,gname,ngrid,jdim,kdim,
     .     Ixmax,Iymax,strch,xcg,ycg,r)

C=============================================================
C
C   Author:
C      Leslie A. Yates
C      Eloret Institute
C      NASA Ames Research Center
C      Moffett Field, CA 94035
C      (415) 604-3436
C
C   Date:
```

```
C       December, 1991
C
C=======================================================================

      character*12 gname
      character*1 mulsin, ctype
      dimension jdim(10), kdim(10), ldim(10)
      dimension r(10,2)

      common/data/ nsub(10), jsubl(10,10), jsub2(10,10), ksubl(10,10),
     .  ksub2(10,10), lsubl(10,10), lsub2(10,10)
C     open(unit=15,file=gname,status='old',form='unformatted')

      ngrid=1
C     if(mulsin.eq.'m')read(15)ngrid
C     read(15)(jdim(n),kdim(n), n=1,ngrid)
C
C     do 20 n=1,ngrid
C      ldim(n)=1
C      write(*,*)
C      write(*,*)' For grid ', n,',  nj and nk ',
C    .           jdim(n), kdim(n)
C      write(*,*)' How many subsets of this grid would you like?'
C      read(*,*)nsub(n)
C      do 30 m=1,nsub(n)
C       write(*,*)' For subset ', m, ' input jmin, jmax, kmin, kmax'
C       read(*,*)jsubl(n,m),jsub2(n,m),ksubl(n,m),ksub2(n,m)
C
C        jsubl(n,m) = max0(1      ,jsubl(n,m))
C       jsubl(n,m) = min0(jdim(n),jsubl(n,m))
C        jsub2(n,m) = max0(1,      jsub2(n,m))
C       jsub2(n,m) = min0(jdim(n),jsub2(n,m))
C
C        ksubl(n,m) = max0(1,      ksubl(n,m))
C       ksubl(n,m) = min0(kdim(n),ksubl(n,m))
C        ksub2(n,m) = max0(1,      ksub2(n,m))
C       ksub2(n,m) = min0(kdim(n),ksub2(n,m))
C
C        lsubl(n,m)=1
C       lsub2(n,m)=1
C
C 30    continue
C 20   continue

C      do 50 n=1,ngrid
C
C       READ(15)  (r(j,1),j=1,jdim(n)*kdim(n)),
C     &           (r(j,2),j=1,jdim(n)*kdim(n))
C
       if(n.eq.1)then
          xmin = r(jsubl(n,1)+(ksubl(n,1)-1)*jdim(n),1)
          xmax = xmin
          ymin = r(jsubl(n,1)+(ksubl(n,1)-1)*jdim(n),2)
          ymax=ymin
        end if

       do 100 m=1,nsub(n)
         do 100 k=ksubl(n,m),ksub2(n,m)
          do 100 j=jsubl(n,m),jsub2(n,m)
            xmin=aminl(xmin,r(j+jdim(n)*(k-1),1))
            xmax=amaxl(xmax,r(j+jdim(n)*(k-1),1))
            ymin=aminl(ymin,r(j+jdim(n)*(k-1),2))
            ymax=amaxl(ymax,r(j+jdim(n)*(k-1),2))
 100     continue
C 50    continue
C       close(15)

       if((xmin.eq.xmax).or.(ymin.eq.ymax))then
          write(*,*)' Data in this view all lies along a single line.'
          write(*,*)' No further computations will be performed.'
          stop
       end if
c
c To minimize the size of the image, Ixmax and Iymax are adjusted so that the image
c does not extend beyond the computational domain.
c
        ratio = (xmax-xmin)/(ymax-ymin)
        Iymax = int(sqrt(float(Ixmax)/ratio))
        Ixmax = int(sqrt(ratio*float(Ixmax)))
        Ixmax = int((Ixmax+1)/2)*2-1
        xstrch=Ixmax/(xmax-xmin)
        ystrch=Iymax/(ymax-ymin)
        strch = amin1(xstrch,ystrch)
        xcg = (Ixmax - strch*(xmax+xmin))/2.
        ycg = (Iymax - strch*(ymax+ymin))/2.
        if(ctype.eq.'A')ycg=-strch*ymin
      return
      end

*******************************************************************************
      Subroutine Getrq3(qname,qname,mulsin,ng,ngrid,r,q,nj,nk,nl)

C=======================================================================
C
C   Author:
C      Leslie A. Yates
C      Eloret Institute
C      NASA Ames Research Center
C      Moffett Field, CA 94035
C      (415) 604-3436
C
C   Date:
C      December, 1991
C
C=======================================================================

      character*12 qname, qname
      dimension r(nj*nk*nl*3), q(nj*nk*nl)

C      if(ng.eq.1)then
C        OPEN(UNIT=15,FILE=qname,FORM='UNFORMATTED',STATUS='OLD')
C        OPEN(UNIT=16,FILE=qname,FORM='UNFORMATTED',STATUS='OLD')
C        if(mulsin.eq.'m')Read(15) k
C        Read(15) (j,j,j,k=1,ngrid)
C        if(mulsin.eq.'m')Read(16) k
C        Read(16) (j,j,j,k=1,ngrid)
C      end if
C
C      READ(15) (r(j),j=1,nj*nk*nl*3)
C
C      READ(16) a1, a2, a3, a4
C      READ(16) (q(j),j=1,nj*nk*nl),
C     .          (a1,j=1,nj*nk*nl*4)
C
C      if(ng.eq.ngrid)then
C        CLOSE(UNIT=15)
C        CLOSE(UNIT=16)
C      end if
```



```
      return
      end

*******************************************************************************
***
      Subroutine Getrq2(qname,qname,mulsin,ng,ngrid,r,q,nj,nk)

C=======================================================================
C
C   Author:
C      Leslie A. Yates
C      Eloret Institute
C      NASA Ames Research Center
C      Moffett Field, CA 94035
C      (415) 604-3436
C
C   Date:
C      December, 1991
C
C=======================================================================

      character*12 qname, qname
      dimension r(nj*nk*2), q(nj*nk)

C      if(ng.eq.1)then
C        OPEN(UNIT=15,FILE=qname,FORM='UNFORMATTED',STATUS='OLD')
C        OPEN(UNIT=16,FILE=qname,FORM='UNFORMATTED',STATUS='OLD')
C        if(mulsin.eq.'m')Read(15) k
C        Read(15) (j,j,k=1,ngrid)
C        if(mulsin.eq.'m')Read(16) k
C        Read(16) (j,j,k=1,ngrid)
C      end if
C
C      READ(15) (r(j),j=1,nj*nk*2)
c
C      READ(16) a1, a2, a3, a4
C      READ(16) (q(j),j=1,nj*nk),
C     .          (a1,j=1,nj*nk*3)
C
C      if(ng.eq.ngrid)then
C        CLOSE(UNIT=15)
C        CLOSE(UNIT=16)
C      end if
      return
      end

*******************************************************************************
***
      Subroutine view3( r, nj, nk, nl, psi,theta, phi, strch, xcg, ycg)
      dimension r(nj*nk*nl,3)

C=======================================================================
C
C   Author:
C      Leslie A. Yates
C      Eloret Institute
C      NASA Ames Research Center
C      Moffett Field, CA 94035
C      (415) 604-3436
C
C   Date:
C      December, 1991
C
C
C=======================================================================

      cospsi=cos(psi)
      sinpsi=sin(psi)
      costhe=cos(theta)
      sinthe=sin(theta)
      cosphi=cos(phi)
      sinphi=sin(phi)

      rot11= costhe*cospsi
      rot12= costhe*sinpsi
      rot13=-sinthe

      rot21= sinphi*sinthe*cospsi - cosphi*sinpsi
      rot22= sinphi*sinthe*sinpsi + cosphi*cospsi
      rot23= sinphi*costhe

      rot31= cosphi*sinthe*cospsi + sinphi*sinpsi
      rot32= cosphi*sinthe*sinpsi - sinphi*cospsi
      rot33= cosphi*costhe

      do 100 j = 1, nj*nk*nl
          anewx=rot11*r(j,1) + rot12*r(j,2) + rot13*r(j,3)
          anewy=rot21*r(j,1) + rot22*r(j,2) + rot23*r(j,3)
          anewz=rot31*r(j,1) + rot32*r(j,2) + rot33*r(j,3)
          r(j,1)=strch*anewx+xcg
          r(j,2)=strch*anewy+ycg
          r(j,3)=strch*anewz
 100  continue

      return
      end

*******************************************************************************

      Subroutine view2( r, nj, nk, strch, xcg, ycg)

C=======================================================================
C
C   Author:
C      Leslie A. Yates
C      Eloret Institute
C      NASA Ames Research Center
C      Moffett Field, CA 94035
C      (415) 604-3436
C
C   Date:
C      December, 1991
C
C=======================================================================

      dimension r(nj*nk,2)

      do 100 j = 1, nj*nk
          r(j,1)=strch*r(j,1)+xcg
          r(j,2)=strch*r(j,2)+ycg
 100  continue
      return
      end

*******************************************************************************

      subroutine inteq3(ng,r,q,nj,nk,nl,deln0)
```

```
C=======================================================================
C
C   Author:
C      Leslie A. Yates
C      Eloret Institute
C      NASA Ames Research Center
C      Moffett Field, CA 94035
C      (415) 604-3436
C
C   Date:
C      December, 1991
C
C=======================================================================

       dimension x(2), f(2,3), x(nj,nk,nl,3),q(nj,nk,nl,3)
       logical mode(600001)

       common/data/ nsub(10), jsub1(10,10), jsub2(10,10), ksub1(10,10),
     .   ksub2(10,10),lsub1(10,10), lsub2(10,10)
       common/image/Ixmax,Iymax,mode,p(600001,4)

       call ngrad3(ng,q,x,nj,nk,nl)

       do 100 ns = 1, nsub(ng)
         do 101 l=lsub1(ng,ns),lsub2(ng,ns)-1
           do 101 k=ksub1(ng,ns),ksub2(ng,ns)-1
            do 101 j=jsub1(ng,ns),jsub2(ng,ns)-1

              x1 = x(j  , k  , l, 1)
              x2 = x(j+1, k  , l, 1)
              x3 = x(j  , k+1, l, 1)
              x4 = x(j+1, k+1, l, 1)
              x5 = x(j  , k  , l+1, 1)
              x6 = x(j+1, k  , l+1, 1)
              x7 = x(j  , k+1, l+1, 1)
              x8 = x(j+1, k+1, l+1, 1)

              y1 = x(j  , k  , l, 2)
              y2 = x(j+1, k  , l, 2)
              y3 = x(j  , k+1, l, 2)
              y4 = x(j+1, k+1, l, 2)
              y5 = x(j  , k  , l+1, 2)
              y6 = x(j+1, k  , l+1, 2)
              y7 = x(j  , k+1, l+1, 2)
              y8 = x(j+1, k+1, l+1, 2)

              minx = int(aminl(x1,x2,x3,x4,x5,x6,x7,x8) + 0.99)
              maxx = int(amaxl(x1,x2,x3,x4,x5,x6,x7,x8))
              miny = int(aminl(y1,y2,y3,y4,y5,y6,y7,y8) + 0.99)
              maxy = int(amaxl(y1,y2,y3,y4,y5,y6,y7,y8))

              do 220 m=minx,maxx
               am=float(m)
               do 220 n=miny,maxy
                if(.NOT.mode(m+(n-1)*Ixmax)) then
                an=float(n)
                ipoint = 0

c  Does xm, xn lie in surface 1?
                jk = 0

                A1 = (x2-am)*(y3-an) - (x3-am)*(y2-an)
                A2 = (x3-am)*(y1-an) - (x1-am)*(y3-an)
                A3 = (x1-am)*(y2-an) - (x2-am)*(y1-an)

                if(abs(A1+A2+A3).ne.abs(A1)+abs(A2)+abs(A3))then
                   jk = 1

                   A2 = (x3-am)*(y4-an) - (x4-am)*(y3-an)
                   A3 = (x4-am)*(y2-an) - (x2-am)*(y4-an)
                end if

                if(abs(A1+A2+A3).eq.abs(A1)+abs(A2)+abs(A3))then
                   ipoint = ipoint + 1
                   denom = A1 + A2 + A3
                   if(denom.eq.0.0)denom = 1.0E+24
                   denom = 1./denom
                   z(ipoint) = (A1*z(j+jk,k+jk,1,3) + A2*z(j+1,k,1,3)
     .                        + A3*z(j,k+1,1,3))*denom
                   f1 = (A1*q(j+jk,k+jk,1,1) + A2*q(j+1,k,1,1)
     .                        + A3*q(j,k+1,1,1))*denom
                   f(ipoint,1) = f1 - deln0
                   f(ipoint,2) = (A1*q(j+jk,k+jk,1,2)
     .                        + A2*q(j+1,k,1,2)
     .                        + A3*q(j,k+1,1,2))*denom/(1. + f1)
                   f(ipoint,3) = (A1*q(j+jk,k+jk,1,3)
     .                        + A2*q(j+1,k,1,3)
     .                        + A3*q(j,k+1,1,3))*denom/(1. + f1)
                 end if

c  Does xm, xn lie in surface 2?
                jk = 0

                A1 = (x6-am)*(y7-an) - (x7-am)*(y6-an)
                A2 = (x7-am)*(y5-an) - (x5-am)*(y7-an)
                A3 = (x5-am)*(y6-an) - (x6-am)*(y5-an)

                if(abs(A1+A2+A3).ne.abs(A1)+abs(A2)+abs(A3))then
                   jk = 1

                   A2 = (x7-am)*(y8-an) - (x8-am)*(y7-an)
                   A3 = (x8-am)*(y6-an) - (x6-am)*(y8-an)
                end if

                if(abs(A1+A2+A3).eq.abs(A1)+abs(A2)+abs(A3))then
                   ipoint = ipoint + 1
                   denom = A1 + A2 + A3
                   if(denom.eq.0.0)denom = 1.0E+24
                   denom = 1./denom
                   z(ipoint) = (A1*z(j+jk,k+jk,l+1,3) + A2*z(j+1,k,l+1,3)
     .                        + A3*z(j,k+1,l+1,3))*denom
                   f1 = (A1*q(j+jk,k+jk,l+1,1) + A2*q(j+1,k,l+1,1)
     .                        + A3*q(j,k+1,l+1,1))*denom
                   f(ipoint,1) = f1 - deln0
                   f(ipoint,2) = (A1*q(j+jk,k+jk,l+1,2)
     .                        + A2*q(j+1,k,l+1,2)
     .                        + A3*q(j,k+1,l+1,2))*denom/(1. + f1)
                   f(ipoint,3) = (A1*q(j+jk,k+jk,l+1,3)
     .                        + A2*q(j+1,k,l+1,3)
     .                        + A3*q(j,k+1,l+1,3))*denom/(1. + f1)
                 end if

c  Does xm, xn lie in surface 3?
                if(ipoint.gt.1) goto 225
                 kl = 0

                 A1 = (x3-am)*(y5-an) - (x5-am)*(y3-an)
                 A2 = (x5-am)*(y1-an) - (x1-am)*(y5-an)
```



```
                 A3 = (x1-am)*(y3-an) - (x3-am)*(y1-an)

                if(abs(A1+A2+A3).ne.abs(A1)+abs(A2)+abs(A3))then
                   kl = 1

                   A2 = (x5-am)*(y7-an) - (x7-am)*(y5-an)
                   A3 = (x7-am)*(y3-an) - (x3-am)*(y7-an)
                end if

                if(abs(A1+A2+A3).eq.abs(A1)+abs(A2)+abs(A3))then
                   denom = A1 + A2 + A3
                   if(denom.eq.0.0)denom = 1.0e+24
                   denom = 1./denom
                   ipoint = ipoint + 1
                   z(ipoint) = (A1*z(j,k+kl,l+kl,3) + A2*z(j,k+1,l,3)
     .                        + A3*z(j,k,l+1,3))*denom
                   f1 = (A1*q(j,k+kl,l+kl,1) + A2*q(j,k+1,l,1)
     .                        + A3*q(j,k,l+1,1))*denom
                   f(ipoint,1) = f1 - deln0
                   f(ipoint,2) = (A1*q(j,k+kl,l+kl,2)
     .                        + A2*q(j,k+1,l,2)
     .                        + A3*q(j,k,l+1,2))*denom/(1.+f1)
                   f(ipoint,3) = (A1*q(j,k+kl,l+kl,3)
     .                        + A2*q(j,k+1,l,3)
     .                        + A3*q(j,k,l+1,3))*denom/(1.+f1)
                 end if

c  Does xm, xn lie in surface 4?
                if(ipoint.gt.1) goto 225
                 kl = 0

                 A1 = (x4-am)*(y6-an) - (x6-am)*(y4-an)
                 A2 = (x6-am)*(y2-an) - (x2-am)*(y6-an)
                 A3 = (x2-am)*(y4-an) - (x4-am)*(y2-an)

                if(abs(A1+A2+A3).ne.abs(A1)+abs(A2)+abs(A3))then
                   kl = 1

                   A2 = (x6-am)*(y8-an) - (x8-am)*(y6-an)
                   A3 = (x8-am)*(y4-an) - (x4-am)*(y8-an)
                end if

                if(abs(A1+A2+A3).eq.abs(A1)+abs(A2)+abs(A3))then
                    denom = A1 + A2 + A3
                    if(denom.eq.0.0)denom = 1.0e+24
                    denom = 1./denom
                    ipoint = ipoint + 1
                    z(ipoint) = (A1*z(j+1,k+kl,l+kl,3) + A2*z(j+1,k+1,l,3)
     .                         + A3*z(j+1,k,l+1,3))*denom
                    f1 = (A1*q(j+1,k+kl,l+kl,1) + A2*q(j+1,k+1,l,1)
     .                         + A3*q(j+1,k,l+1,1))*denom
                    f(ipoint,1) = f1 - deln0
                    f(ipoint,2) = (A1*q(j+1,k+kl,l+kl,2)
     .                         + A2*q(j+1,k+1,l,2)
     .                         + A3*q(j+1,k,l+1,2))*denom/(1.+f1)
                    f(ipoint,3) = (A1*q(j+1,k+kl,l+kl,3)
     .                         + A2*q(j+1,k+1,l,3)
     .                         + A3*q(j+1,k,l+1,3))*denom/(1.+f1)
                  end if

c  Does xm, xn lie in surface 5?
                 if(ipoint.gt.1) goto 225
                  jl = 0

                  A1 = (x2-am)*(y5-an) - (x5-am)*(y2-an)
                  A2 = (x5-am)*(y1-an) - (x1-am)*(y5-an)
                  A3 = (x1-am)*(y2-an) - (x2-am)*(y1-an)

                  if(abs(A1+A2+A3).ne.abs(A1)+abs(A2)+abs(A3))then
                    jl = 1

                    A2 = (x5-am)*(y6-an) - (x6-am)*(y5-an)
                    A3 = (x6-am)*(y2-an) - (x2-am)*(y6-an)
                  end if

                  if(abs(A1+A2+A3).eq.abs(A1)+abs(A2)+abs(A3))then
                    denom = A1 + A2 + A3
                    if(denom.eq.0.0)denom=1.0E+24
                    denom = 1./denom
                    ipoint = ipoint + 1
                    z(ipoint) = (A1*z(j+jl,k,l+jl,3) + A2*z(j+1,k,l,3)
     .                         + A3*z(j,k,l+1,3))*denom
                    f1 = (A1*q(j+jl,k,l+jl,1) + A2*q(j+1,k,l,1)
     .                         + A3*q(j,k,l+1,1))*denom
                    f(ipoint,1) = f1 - deln0
                    f(ipoint,2) = (A1*q(j+jl,k,l+jl,2)
     .                         + A2*q(j+1,k,l,2)
     .                         + A3*q(j,k,l+1,2))*denom/(1.+f1)
                    f(ipoint,3) = (A1*q(j+jl,k,l+jl,3)
     .                         + A2*q(j+1,k,l,3)
     .                         + A3*q(j,k,l+1,3))*denom/(1.+f1)
                  end if

c  Does xm, xn lie in surface 6?
                 if(ipoint.gt.1) goto 225
                  jl = 0

                  A1 = (x4-am)*(y7-an) - (x7-am)*(y4-an)
                  A2 = (x7-am)*(y3-an) - (x3-am)*(y7-an)
                  A3 = (x3-am)*(y4-an) - (x4-am)*(y3-an)

                  if(abs(A1+A2+A3).ne.abs(A1)+abs(A2)+abs(A3))then
                    jl = 1

                    A2 = (x7-am)*(y8-an) - (x8-am)*(y7-an)
                    A3 = (x8-am)*(y4-an) - (x4-am)*(y8-an)
                  end if

                  if(abs(A1+A2+A3).eq.abs(A1)+abs(A2)+abs(A3))then
                    denom = A1 + A2 + A3
                    if(denom.eq.0.0)denom = 1.0E+24
                    denom = 1./denom
                    ipoint = ipoint + 1
                    z(ipoint) = (A1*z(j+jl,k+1,l+jl,3) + A2*z(j+1,k+1,l,3)
     .                         + A3*z(j,k+1,l+1,3))*denom
                    f1 = (A1*q(j+jl,k+1,l+jl,1) + A2*q(j+1,k+1,l,1)
     .                         + A3*q(j,k+1,l+1,1))*denom
                    f(ipoint,1) = f1 - deln0
                    f(ipoint,2) = (A1*q(j+jl,k+1,l+jl,2)
     .                         + A2*q(j+1,k+1,l,2)
     .                         + A3*q(j,k+1,l+1,2))*denom/(1.+f1)
                    f(ipoint,3) = (A1*q(j+jl,k+1,l+jl,3)
     .                         + A2*q(j+1,k+1,l,3)
     .                         + A3*q(j,k+1,l+1,3))*denom/(1.+f1)
                  end if
 225            continue
                if(ipoint.eq.2)then
                  imn=m+(n-1)*Ixmax
```

```
            p(imn,1)=p(imn,1) + abs(r(2)-r(1))*(f(2,1)+f(1,1))/2.0
            p(imn,2)=p(imn,2) + abs(r(2)-r(1))*(f(2,2)+f(1,2))/2.0
            p(imn,3)=p(imn,3) + abs(r(2)-r(1))*(f(2,3)+f(1,3))/2.0
          end if
 220      continue

 101    continue
 100    continue

        return
        end

*****************************************************************************
        subroutine ngrad3(ng,q,r,nj,nk,nl)

C=============================================================
C
C   Author:
C      Leslie A. Yates
C      Eloret Institute
C      NASA Ames Research Center
C      Moffett Field, CA 94035
C      (415) 604-3436
C
C   Date:
C      December, 1991
C
C=============================================================

        dimension q(nj,nk,nl,3), r(nj,nk,nl,3)

        common/data/ nsub(10), jsub1(10,10), jsub2(10,10), ksub1(10,10),
     .   ksub2(10,10),lsub1(10,10), lsub2(10,10)

        do 10 m=1,nsub(ng)
         do 10 l=lsub1(ng,m),lsub2(ng,m)
          l1 = 0
          if(l.eq.lsub1(ng,m))l1 =  1
          if(l.eq.lsub2(ng,m))l1 = -1

          do 20 k = ksub1(ng,m), ksub2(ng,m)
           k1 = 0
           if(k.eq.ksub1(ng,m))k1 =  1
           if(k.eq.ksub2(ng,m))k1 = -1

           do 30 j = jsub1(ng,m), jsub2(ng,m)
            j1 = 0
            if(j.eq.jsub1(ng,m))j1 =  1
            if(j.eq.jsub2(ng,m))j1 = -1
            if(j1.eq.0)then
              a11 = r(j+1,k,l,1) - r(j-1,k,l,1)
              a21 = r(j+1,k,l,2) - r(j-1,k,l,2)
              a31 = r(j+1,k,l,3) - r(j-1,k,l,3)
              dp1 = q(j+1,k,l,1) - q(j-1,k,l,1)
            else
              a11 = -3.0*r(j,k,l,1) + 4.0*r(j+j1,k,l,1) - r(j+2*j1,k,l,1)
              a21 = -3.0*r(j,k,l,2) + 4.0*r(j+j1,k,l,2) - r(j+2*j1,k,l,2)
              a31 = -3.0*r(j,k,l,3) + 4.0*r(j+j1,k,l,3) - r(j+2*j1,k,l,3)
              dp1 = -3.0*q(j,k,l,1) + 4.0*q(j+j1,k,l,1) - q(j+2*j1,k,l,1)
            end if
            if(k1.eq.0)then
              a12 = r(j,k+1,l,1) - r(j,k-1,l,1)
              a22 = r(j,k+1,l,2) - r(j,k-1,l,2)
              a32 = r(j,k+1,l,3) - r(j,k-1,l,3)
              dp2 = q(j,k+1,l,1) - q(j,k-1,l,1)
            else
              a12 = -3.0*r(j,k,l,1) + 4.0*r(j,k+k1,l,1) - r(j,k+2*k1,l,1)
              a22 = -3.0*r(j,k,l,2) + 4.0*r(j,k+k1,l,2) - r(j,k+2*k1,l,2)
              a32 = -3.0*r(j,k,l,3) + 4.0*r(j,k+k1,l,3) - r(j,k+2*k1,l,3)
              dp2 = -3.0*q(j,k,l,1) + 4.0*q(j,k+k1,l,1) - q(j,k+2*k1,l,1)
            end if
            if(l1.eq.0)then
              a13 = r(j,k,l+1,1) - r(j,k,l-1,1)
              a23 = r(j,k,l+1,2) - r(j,k,l-1,2)
              a33 = r(j,k,l+1,3) - r(j,k,l-1,3)
              dp3 = q(j,k,l+1,1) - q(j,k,l-1,1)
            else
              a13 = -3.0*r(j,k,l,1) + 4.0*r(j,k,l+l1,1) - r(j,k,l+2*l1,1)
              a23 = -3.0*r(j,k,l,2) + 4.0*r(j,k,l+l1,2) - r(j,k,l+2*l1,2)
              a33 = -3.0*r(j,k,l,3) + 4.0*r(j,k,l+l1,3) - r(j,k,l+2*l1,3)
              dp3 = -3.0*q(j,k,l,1) + 4.0*q(j,k,l+l1,1) - q(j,k,l+2*l1,1)
            end if
            rJ = a11*(a22*a33 - a23*a32) + a12*(a23*a31 - a21*a33)
     .                                   + a13*(a21*a32 - a22*a31)
            if(rJ.ne.0.0)rJ = 1./rJ
            q(j,k,l,1) = rJ*((a33*a22 - a32*a23)*dp1
     .                    + (a23*a31 - a21*a33)*dp2 + (a21*a32 - a31*a22)*dp3)
            q(j,k,l,2) = rJ*((a13*a32 - a12*a33)*dp1
     .                    + (a33*a11 - a31*a13)*dp2 + (a31*a12 - a11*a32)*dp3)
 30       continue
 20     continue
 10     continue

        return
        end


*****************************************************************************
*
        subroutine IntegA(ng,r,q,nj,nk,deln0,nphi)

C=============================================================
C
C   Author:
C      Leslie A. Yates
C      Eloret Institute
C      NASA Ames Research Center
C      Moffett Field, CA 94035
C      (415) 604-3436
C
C   Date:
C      December, 1991
C
C=============================================================

        dimension r(3),f(3,3),r(nj,nk,2),q(nj,nk,3),vcos(181),vsin(181)
        logical mod(600001)

        common/data/ nsub(10), jsub1(10,10), jsub2(10,10), ksub1(10,10),
     .   ksub2(10,10),lsub1(10,10), lsub2(10,10)
        common/image/Ixmax,Iymax,mod,p(600001,4)

        Call ngrad2(ng,q,r,nj,nk)

        dphi = 2.0*atan(1.0)/float(nphi)
        do 50 n=1,nphi+1
          vcos(n)=cos(float(n-1)*dphi)
```



```
          vsin(n)=sin(float(n-1)*dphi)
 50     continue

        do 100 no=1,nsub(ng)
         do 100 j=jsub1(ng,no),jsub2(ng,no)-1
          do 100 k=ksub1(ng,no),ksub2(ng,no)-1

            x1 = r(j  , k  , 1)
            x2 = r(j+1, k  , 1)
            x3 = r(j  , k+1, 1)
            x4 = r(j+1, k+1, 1)

            y1 = r(j,  k,  2)
            y2 = r(j+1,k,  2)
            y3 = r(j,  k+1,2)
            y4 = r(j+1,k+1,2)

            minx = int(aminl(x1,x2,x3,x4) + 1.0)
            maxx = int(amaxl(x1,x2,x3,x4))
            aminy = aminl(y1,y2,y3,y4)
            amaxy = amaxl(y1,y2,y3,y4)

            do 120 m=minx,maxx
             am=float(m)
             B1  = (x2-am)*y3 - (x3-am)*y2
             B12 = (x3-am)*y1 - (x1-am)*y3
             B13 = (x1-am)*y2 - (x2-am)*y1
             B22 = (x3-am)*y4 - (x4-am)*y3
             B23 = (x4-am)*y2 - (x2-am)*y4

             do 130 iphi = 1, nphi
              miny = int(aminy*vcos(iphi+1) + 1.49)
              maxy = int(amaxy*vcos(iphi) + 0.5)

              do 140 n=miny,maxy
               if(.not.mod(m + (n-1)*Ixmax)) then
                 an=float(n) - 0.5
                 ipoint = 0

c  Does xm, xn lie in surface 1?
                 jk = 0

                 A1 = vcos(iphi)*B1 - an*(x2 - x3)
                 A2 = vcos(iphi)*B12 - an*(x3 - x1)
                 A3 = vcos(iphi)*B13 - an*(x1 - x2)

                 if(abs(A1+A2+A3).ne.abs(A1)+abs(A2)+abs(A3))then
                   jk = 1

                   A2 = vcos(iphi)*B22 - an*(x3 - x4)
                   A3 = vcos(iphi)*B23 - an*(x4 - x2)
                 end if

                 if(abs(A1+A2+A3).eq.abs(A1)+abs(A2)+abs(A3))then
                   ipoint = ipoint + 1
                   denom = A1 + A2 + A3
                   if(denom.eq.0.0)denom = 1.0E+24
                   z(ipoint) = an*vsin(iphi)/vcos(iphi)
                   f1 = (A1*q(j+jk,k+jk,1) + A2*q(j+1,k,1)
     .                       + A3*q(j,k+1,1))/denom
                   f(ipoint,1) = f1 - deln0
                   f(ipoint,2) = (A1*q(j+jk,k+jk,2) + A2*q(j+1,k,2)
     .                       + A3*q(j,k+1,2))/denom/(1. + f1)
                   f(ipoint,3) = (A1*q(j+jk,k+jk,3) + A2*q(j+1,k,3)
     .                       + A3*q(j,k+1,3))*vcos(iphi)/denom/(1. + f1)
                 end if

c  Does xm, xn lie in surface 2?

                 if(vcos(iphi+1).ne.0)then
                   jk = 0

                   A1 = vcos(iphi+1)*B1  - an*(x2 - x3)
                   A2 = vcos(iphi+1)*B12 - an*(x3 - x1)
                   A3 = vcos(iphi+1)*B13 - an*(x1 - x2)

                  if(abs(A1+A2+A3).ne.abs(A1)+abs(A2)+abs(A3))then
                    jk = 1

                    A2 = vcos(iphi+1)*B22 - an*(x3 - x4)
                    A3 = vcos(iphi+1)*B23 - an*(x4 - x2)
                  end if


                  if(abs(A1+A2+A3).eq.abs(A1)+abs(A2)+abs(A3))then
                    ipoint = ipoint + 1
                    denom = A1 + A2 + A3
                    if(denom.eq.0.0)denom = 1.0E+24
                    z(ipoint) = an*vsin(iphi+1)/vcos(iphi+1)
                    f1 = (A1*q(j+jk,k+jk,1) + A2*q(j+1,k,1)
     .                       + A3*q(j,k+1,1))/denom
                    f(ipoint,1) = f1 - deln0
                    f(ipoint,2) = (A1*q(j+jk,k+jk,2) + A2*q(j+1,k,2)
     .                       + A3*q(j,k+1,2))/denom/(1. + f1)
                    f(ipoint,3) = (A1*q(j+jk,k+jk,3) + A2*q(j+1,k,3)
     .                       + A3*q(j,k+1,3))*vcos(iphi+1)/denom/(1. + f1)
                  end if
                 end if

c  Does xm, xn lie in surface 3? (Axial symmetry is used)
                 if((ipoint.lt.2).and.((x2-am)*(am-x1).ge.0.0))then
                   denom=x2-x1
                   if(denom.eq.0.0)denom = 1.0E+24
                   ar = (y1*(x2-am) + y2*(am-x1))/denom
                   if((ar*vcos(iphi)-an)*(an-ar*vcos(iphi+1)).gt.0.0)then
                     ipoint=ipoint+1
                     z(ipoint) = sqrt(abs(ar*ar - an*an))
                     f1 = (q(j,k,1)*(x2-am)+q(j+1,k,1)*(am-x1))/denom
                     f(ipoint,1) = f1 - deln0
                     f(ipoint,2)=(q(j,k,2)*(x2-am)
     .                          +q(j+1,k,2)*(am-x1))/denom/(1.+f1)
                     f(ipoint,3)=(q(j,k,3)*(x2-am)
     .                          +q(j+1,k,3)*(am-x1))*an/ar/denom/(1.+f1)
                   end if
                 end if


c  Does xm, xn lie in surface 4?
                 if((ipoint.lt.2).and.((x4-am)*(am-x3).ge.0.0))then
                   denom=x4-x3
                   if(denom.eq.0.0)denom = 1.0E+24
                   ar = (y3*(x4-am) + y4*(am-x3))/denom
                   if((ar*vcos(iphi)-an)*(an-ar*vcos(iphi+1)).gt.0.0)then
                     ipoint=ipoint+1
                     z(ipoint) = sqrt(abs(ar*ar - an*an))
                     f1 =        (q(j,k+1,1)*(x4-am)
     .                          + q(j+1,k+1,1)*(am-x3))/denom
                     f(ipoint,1) = f1 - deln0
```

```
                f(ipoint,2)=(q(j,k+1,2)*(x4-am)
     .                      +q(j+1,k+1,2)*(am-x3))/denom/(1.+f1)
                f(ipoint,3)=(q(j,k+1,3)*(x4-am)
     .                     +q(j+1,k+1,3)*(am-x3))*an/ar/denom/(1.+f1)
              end if
            end if

c  Does xm, xn lie in surface 5?
            if((ipoint.lt.2).and.((x1-am)*(am-x3).ge.0.0))then
              ar = 0.0
              if(x3-x1.ne.0.0)ar = (y1*(x3-am) + y3*(am-x1))/(x3-x1)
              if((ar*vcos(iphi)-an)*(an-ar*vcos(iphi+1)).gt.0.0)then
                denom = y3 - y1
                if(denom.eq.0.0)denom=1.0E+24
                ipoint = ipoint + 1
                z(ipoint) = sqrt(abs(ar*ar-an*an))
                f1 = (q(j,k,1)*(y3-ar) + q(j,k+1,1)*(ar-y1))/denom
                f(ipoint,1) = f1 - deln0
                f(ipoint,2) = (q(j,k,2)*(y3-ar)
     .                       + q(j,k+1,2)*(ar-y1))/denom/(1.+f1)
                f(ipoint,3) = (q(j,k,3)*(y3-ar)
     .                       + q(j,k+1,3)*(ar-y1))*an/ar/denom/(1.+f1)
              end if
            end if

c  Does xm, xn lie in surface 6?
            if((ipoint.lt.2).and.((x2-am)*(am-x4).ge.0.0))then
              ar = 0.0
              if(x4-x2.ne.0.0)ar = (y2*(x4-am) + y4*(am-x2))/(x4-x2)
              if((ar*vcos(iphi)-an)*(an-ar*vcos(iphi+1)).gt.0.0)then
                denom = y4-y2
                if(denom.eq.0.0)denom=1.0E+24
                ipoint = ipoint + 1
                z(ipoint) = sqrt(abs(ar*ar-an*an))
                f1      =      (q(j+1,k,1)*(y4-ar)
     .                        + q(j+1,k+1,1)*(ar-y2))/denom
                f(ipoint,1) = f1 - deln0
                f(ipoint,2) = (q(j+1,k,2)*(y4-ar)
     .                       + q(j+1,k+1,2)*(ar-y2))/denom/(1.+f1)
                f(ipoint,3) = (q(j+1,k,3)*(y4-ar)
     .                       + q(j+1,k+1,3)*(ar-y2))*an/ar/denom/(1.+f1)
              end if
            end if


            if(ipoint.ge.2)then
              imn = m + (n-1)*Ixmax
              p(imn,1)=p(imn,1) + abs(z(2)-z(1))*(f(2,1)+f(1,1))/2.0
              p(imn,2)=p(imn,2) + abs(z(2)-z(1))*(f(2,2)+f(1,2))/2.0
              p(imn,3)=p(imn,3) + abs(z(2)-z(1))*(f(2,3)+f(1,3))/2.0
            end if
          end if

 140      continue
 130     continue
 120    continue

 100   continue

       return
       end
*******************************************************************************
```

```
      subroutine inteq2(ng,r,q,nj,nk,deln0,dz)

C=============================================================================
C
C  Author:
C     Leslie A. Yates
C     Eloret Institute
C     NASA Ames Research Center
C     Moffett Field, CA 94035
C     (415) 604-3436
C
C  Date:
C     December, 1991
C
C=============================================================================

      dimension r(nj,nk,2), q(nj,nk,3)
      logical mod(600001)

      common/data/ nsub(10), jsub1(10,10), jsub2(10,10), ksub1(10,10),
     .   ksub2(10,10),lsub1(10,10), lsub2(10,10)
      common/image/Ixmax,Iymax,mod,p(600001,4)

      Call ngrad2(ng,q,r,nj,nk)

      do 100 no=1,nsub(ng)
       do 100 j=jsub1(ng,no),jsub2(ng,no)-1
        do 100 k=ksub1(ng,no),ksub2(ng,no)-1

          x1 = r(j  , k  , 1)
          x2 = r(j+1, k  , 1)
          x3 = r(j  , k+1, 1)
          x4 = r(j+1, k+1, 1)

          y1 = r(j  , k  , 2)
          y2 = r(j+1, k  , 2)
          y3 = r(j  , k+1, 2)
          y4 = r(j+1, k+1, 2)

          minx = int(amin1(x1,x2,x3,x4) + 0.99)
          maxx = int(amax1(x1,x2,x3,x4))
          miny = int(amin1(y1,y2,y3,y4) + 0.99)
          maxy = int(amax1(y1,y2,y3,y4))

          do 120 m=minx,maxx
           am = float(m)
           do 120 n=miny,maxy
             an = float(n)

c  Does xm, xn lie in the grid?

            jk = 0

            A1 = (x2-am)*(y3-an) - (x3-am)*(y2-an)
            A2 = (x3-am)*(y1-an) - (x1-am)*(y3-an)
            A3 = (x1-am)*(y2-an) - (x2-am)*(y1-an)


            if(abs(A1+a2+A3) .ne. abs(A1)+abs(A2)+abs(A3))then
               jk = 1

               A2 = (x3-am)*(y4-an) - (x4-am)*(y3-an)
               A3 = (x4-am)*(y2-an) - (x2-am)*(y4-an)
            end if
```



```
            if(abs(A1+a2+A3) .eq. abs(A1)+abs(A2)+abs(A3))then
               ipoint = ipoint + 1
               denom = A1 + A2 + A3
               if(denom.eq.0.0)denom = 1.0E+24
               f0 = (A1*q(j+jk,k+jk,1) + A2*q(j+1,k,1)
     .                + A3*q(j,k+1,1))/denom
               f1 = f0 - deln0
               f2 = (A1*q(j+jk,k+jk,2) + A2*q(j+1,k,2)
     .                + A3*q(j,k+1,2))/denom/(1.+f0)
               f3 = (A1*q(j+jk,k+jk,3) + A2*q(j+1,k,3)
     .                + A3*q(j,k+1,3))/denom/(1.+f0)

               imn = m + (n-1)*Ixmax

               p(imn,1) = f1*dz
               p(imn,2) = f2*dz
               p(imn,3) = f3*dz
             end if

 120       continue

 100   continue

       return
       end
*******************************************************************************
****
       subroutine ngrad2(ng,q,r,nj,nk)

C=============================================================================
C
C  Author:
C     Leslie A. Yates
C     Eloret Institute
C     NASA Ames Research Center
C     Moffett Field, CA 94035
C     (415) 604-3436
C
C  Date:
C     December, 1991
C
C=============================================================================

       dimension r(nj,nk,2),q(nj,nk,3)

       common/data/ nsub(10), jsub1(10,10), jsub2(10,10), ksub1(10,10),
     .   ksub2(10,10),lsub1(10,10), lsub2(10,10)

       do 10 n=1,nsub(ng)

        do 10 k = ksub1(ng,n), ksub2(ng,n)
         kl = 0
         if(k.eq.ksub1(ng,n))kl =  1
         if(k.eq.ksub2(ng,n))kl = -1

         do 20 j = jsub1(ng,n),jsub2(ng,n)
           jl = 0
           if(j.eq.jsub1(ng,n))jl =  1
           if(j.eq.jsub2(ng,n))jl = -1

           if(jl.ne.0)then
             a11 = -3.0*r(j,k,1) + 4.0*r(j+jl,k,1) - r(j+2*jl,k,1)
             a21 = -3.0*r(j,k,2) + 4.0*r(j+jl,k,2) - r(j+2*jl,k,2)
             dp1 = -3.0*q(j,k,1) + 4.0*q(j+jl,k,1) - q(j+2*jl,k,1)
           end if

           if(jl.eq.0)then
             a11 = r(j+1,k,1) - r(j-1,k,1)
             a21 = r(j+1,k,2) - r(j-1,k,2)
             dp1 = q(j+1,k,1) - q(j-1,k,1)
           end if

           if(kl.ne.0)then
             a12 = -3.0*r(j,k,1) + 4.0*r(j,k+kl,1) - r(j,k+2*kl,1)
             a22 = -3.0*r(j,k,2) + 4.0*r(j,k+kl,2) - r(j,k+2*kl,2)
             dp2 = -3.0*q(j,k,1) + 4.0*q(j,k+kl,1) - q(j,k+2*kl,1)
           end if

           if(kl.eq.0)then
             a12 = r(j,k+1,1) - r(j,k-1,1)
             a22 = r(j,k+1,2) - r(j,k-1,2)
             dp2 = q(j,k+1,1) - q(j,k-1,1)
           end if

           rJ = a11*a22 - a12*a21
           if(rJ.ne.0)rJ = 1./rJ

           q(j,k,2) =  rJ*( a22*dp1 - a21*dp2)
           q(j,k,3) =  rJ*(-a12*dp1 + a11*dp2)
 20      continue
 10     continue
  5    continue

       return
       end
*******************************************************************************
*
       subroutine model(ng,r,nj,nk,nl)

C=============================================================================
C
C  Author:
C     Leslie A. Yates
C     Eloret Institute
C     NASA Ames Research Center
C     Moffett Field, CA 94035
C     (415) 604-3436
C
C  Date:
C     December, 1991
C
C=============================================================================

       character*1 plane
       logical mod(600001)
       dimension r(nj,nk,nl,3)

       common/image/Ixmax,Iymax,mod,p(600001,4)
      common/surfset/nseg,jminset(10),jmaxset(10),kminset(10),
     .               kmaxset(10),lminset(10),lmaxset(10)

      print*,"in model, nseg=",nseg
C       write(*,*)
C       write(*,*)' In grid ',ng,' how many surface segments are there?'
C      read(*,*)nseg
       if(nseg.eq.0)return
```

```
      do 50 no=1,nseg
     print*,"no = ",no
     print*,"jminset, jmaxset = ",jminset(no),jmaxset(no)
     print*,"kminset, kmaxset = ",kminset(no),kmaxset(no)
     print*,"lminset, lmaxset = ",lminset(no),lmaxset(no)
C       write(*,*)
C       write(*,*)' For surfaces defined by constant j,k,l planes,'
C       write(*,*)' input j,k, or l'
C       read(*,'(a1)') plane

C       if(plane.eq.'j')then
      if(jminset(no).eq.jmaxset(no)) then
         j = jminset(no)
         kmin = kminset(no)
         kmax = kmaxset(no)
         lmin = lminset(no)
         lmax = lmaxset(no)
C          write(*,*)' Input j for the model surface.'
C          read(*,*)j

C           write(*,*)' Input kmin, kmax for the model surface.'
C          read(*,*)kmin, kmax

C          write(*,*)' Input lmin, lmax for the model surface.'
C          read(*,*)lmin, lmax

           if((j-1)*(nj-j).le.0)j=1

           kmin = max0(1, kmin)
           kmin = min0(nk,kmin)
           kmax = max0(1, kmax)
           kmax = min0(nk,kmax)

           lmin = max0(1, lmin)
           lmin = min0(nl,lmin)
           lmax = max0(1, lmax)
           lmax = min0(nl,lmax)

           do 10 l= lmin, lmax-1
            do 10 k = kmin, kmax-1

              x1 = r(j, k  , l  , 1)
              x2 = r(j, k+1, l  , 1)
              x3 = r(j, k  , l+1, 1)
              x4 = r(j, k+1, l+1, 1)

              y1 = r(j, k  , l  , 2)
              y2 = r(j, k+1, l  , 2)
              y3 = r(j, k  , l+1, 2)
              y4 = r(j, k+1, l+1, 2)

              minx = int(aminl(x1,x2,x3,x4) + 0.99)
              maxx = int(amaxl(x1,x2,x3,x4))
              miny = int(aminl(y1,y2,y3,y4) + 0.99)
              maxy = int(amaxl(y1,y2,y3,y4))

              do 11 n=miny, maxy
               an=float(n)
               do 12 m=minx, maxx
                 am=float(m)

                 A1  = sign(1.0,(x2-am)*(y3-an) - (x3-am)*(y2-an))
                 A12 = sign(1.0,(x3-am)*(y1-an) - (x1-am)*(y3-an))
                 A13 = sign(1.0,(x1-am)*(y2-an) - (x2-am)*(y1-an))

                 A22 = sign(1.0,(x3-am)*(y4-an) - (x4-am)*(y3-an))
                 A23 = sign(1.0,(x4-am)*(y2-an) - (x2-am)*(y4-an))
                 test = amaxl(abs(A1 + A12 + A13),abs(A1 + A22 + A23))

                 if(test .gt. 2.5)mod(m+(n-1)*lxmax)=.TRUE.
12             continue
11            continue
10          continue

          end if

C         if(plane.eq.'k')then
      if(kminset(no).eq.kmaxset(no)) then
         k = kminset(no)
         jmin = jminset(no)
         jmax = jmaxset(no)
         lmin = lminset(no)
         lmax = lmaxset(no)
C          write(*,*)
C          write(*,*)' Input jmin, jmax for the model surface.'
C          read(*,*)jmin, jmax

C           write(*,*)' Input k for the model surface.'
C          read(*,*)k

C          write(*,*)' Input lmin, lmax for the model surface.'
C          read(*,*)lmin, lmax

           jmin = max0(1, jmin)
           jmin = min0(nj,jmin)
           jmax = max0(1, jmax)
           jmax = min0(nj,jmax)

           if((k-1)*(nk-k).le.0)k=1

           lmin = max0(1, lmin)
           lmin = min0(nl,lmin)
           lmax = max0(1, lmax)
           lmax = min0(nl,lmax)

           do 20 j = jmin, jmax-1
            do 20 l = lmin, lmax-1

              x1 = r(j  , k, l  , 1)
              x2 = r(j+1, k, l  , 1)
              x3 = r(j  , k, l+1, 1)
              x4 = r(j+1, k, l+1, 1)

              y1 = r(j  , k, l  , 2)
              y2 = r(j+1, k, l  , 2)
              y3 = r(j  , k, l+1, 2)
              y4 = r(j+1, k, l+1, 2)

              minx = int(aminl(x1,x2,x3,x4) + 0.99)
              maxx = int(amaxl(x1,x2,x3,x4))
              miny = int(aminl(y1,y2,y3,y4) + 0.99)
              maxy = int(amaxl(y1,y2,y3,y4))

              do 21 n=miny, maxy
               an = float(n)
               do 22 m=minx, maxx
```



```
                 am = float(m)

                 A1  = sign(1.0,(x2-am)*(y3-an) - (x3-am)*(y2-an))
                 A12 = sign(1.0,(x3-am)*(y1-an) - (x1-am)*(y3-an))
                 A13 = sign(1.0,(x1-am)*(y2-an) - (x2-am)*(y1-an))

                 A22 = sign(1.0,(x3-am)*(y4-an) - (x4-am)*(y3-an))
                 A23 = sign(1.0,(x4-am)*(y2-an) - (x2-am)*(y4-an))
                 test = amaxl(abs(A1 + A12 + A13),abs(A1 + A22 + A23))

                 if(test .gt. 2.5)mod(m+(n-1)*lxmax)=.TRUE.
22             continue
21            continue
20          continue

          end if

C         if(plane.eq.'l')then
      if(lminset(no).eq.lmaxset(no)) then
         l = lminset(no)
         jmin = jminset(no)
         jmax = jmaxset(no)
         kmin = kminset(no)
         kmax = kmaxset(no)
C          write(*,*)
C          write(*,*)' Input jmin, jmax for the model surface.'
C          read(*,*)jmin, jmax

C           write(*,*)' Input kmin, kmax for the model surface.'
C          read(*,*)kmin, kmax

C          write(*,*)' Input l for the model surface.'
C          read(*,*)l


           jmin = max0(1, jmin)
           jmin = min0(nj,jmin)
           jmax = max0(1, jmax)
           jmax = min0(nj,jmax)

           kmin = max0(1, kmin)
           kmin = min0(nk,kmin)
           kmax = max0(1, kmax)
           kmax = min0(nk,kmax)

           if((l-1)*(nl-1).le.0)l=1

           do 30 j = jmin, jmax-1
            do 30 k = kmin, kmax-1

              x1 = r(j  , k  , l, 1)
              x2 = r(j+1, k  , l, 1)
              x3 = r(j  , k+1, l, 1)
              x4 = r(j+1, k+1, l, 1)

              y1 = r(j  , k  , l, 2)
              y2 = r(j+1, k  , l, 2)
              y3 = r(j  , k+1, l, 2)
              y4 = r(j+1, k+1, l, 2)

              minx = int(aminl(x1,x2,x3,x4) + 0.99)
              maxx = int(amaxl(x1,x2,x3,x4))
              miny = int(aminl(y1,y2,y3,y4) + 0.99)
              maxy = int(amaxl(y1,y2,y3,y4))

              do 31 n=miny, maxy
               an = float(n)
               do 32 m=minx,maxx
                 am = float(m)

                 A1  = sign(1.0,(x2-am)*(y3-an) - (x3-am)*(y2-an))
                 A12 = sign(1.0,(x3-am)*(y1-an) - (x1-am)*(y3-an))
                 A13 = sign(1.0,(x1-am)*(y2-an) - (x2-am)*(y1-an))

                 A22 = sign(1.0,(x3-am)*(y4-an) - (x4-am)*(y3-an))
                 A23 = sign(1.0,(x4-am)*(y2-an) - (x2-am)*(y4-an))
                 test = amaxl(abs(A1 + A12 + A13),abs(A1 + A22 + A23))

                 if(test .gt. 2.5)mod(m+(n-1)*lxmax)=.TRUE.
32             continue
31            continue
30          continue

          end if

50      continue

        return
        end

*****************************************************************************
        subroutine Grdovr(ng,r,nj,nk)

C=============================================================================
C
C  Author:
C     Leslie A. Yates
C     Eloret Institute
C     NASA Ames Research Center
C     Moffett Field, CA 94035
C     (415) 604-3436
C
C  Date:
C     December, 1991
C
C=============================================================================

        dimension r(nj,nk,2)
        logical mod(600001)

        common/data/ nsub(10), jsub1(10,10), jsub2(10,10), ksub1(10,10),
     .    ksub2(10,10), lsub1(10,10), lsub2(10,10)
        common/image/lxmax,lymax,mod,p(600001,4)

        do 5 i=1,lxmax*lymax
          mod(i)=.TRUE.
5       continue

        do 10 no = 1, nsub(ng)
         do 10 j= jsub1(ng,no), jsub2(ng,no)-1
          do 10 k = ksub1(ng,no), ksub2(ng,no)-1

             x1 = r(j,  k  ,1)
             x2 = r(j,  k+1,1)
             x3 = r(j+1,k  ,1)
             x4 = r(j+1,k+1,1)

             y1 = r(j,  k,  2)
```

```
            y2 = r(j,  k+1,2)
            y3 = r(j+1,k  ,2)
            y4 = r(j+1,k+1,2)

            minx = int(aminl(x1,x2,x3,x4) + 0.99)
            maxx = int(amaxl(x1,x2,x3,x4))
            miny = int(aminl(y1,y2,y3,y4) + 0.99)
            maxy = int(amaxl(y1,y2,y3,y4))

            do 11 n=miny, maxy
             an=float(n)
             do 12 m=minx, maxx
               am=float(m)

               A1 = (x2-am)*(y3-an) - (x3-am)*(y2-an)
               A2 = (x3-am)*(y1-an) - (x1-am)*(y3-an)
               A3 = (x1-am)*(y2-an) - (x2-am)*(y1-an)

               if(abs(A1+A2+A3).ne.abs(A1)+abs(A2)+abs(A3))then
                 A2 = (x3-am)*(y4-an) - (x4-am)*(y3-an)
                 A3 = (x4-am)*(y2-an) - (x2-am)*(y4-an)
               end if

               if(abs(A1+A2+A3).eq.abs(A1)+abs(A2)+abs(A3))
                                  mod(m+(n-1)*Ixmax)=.FALSE.

 12           continue
 11          continue

 10       continue

          return
          end

*******************************************************************************
          subroutine fringes(cons)

C=============================================================
C
C   Author:
C      Leslie A. Yates
C      Eloret Institute
C      NASA Ames Research Center
C      Moffett Field, CA 94035
C      (415) 604-3436
C
C   Date:
C      December, 1991
C
C=============================================================

          character*1 type
          character*12 file
          logical mod(600001)

          common/image/Ixmax,Iymax,mod,p(600001,4)

         common/interf/swl,cjunk,vert,horiz,type

         print *,"in fringes"
         print *,"cons = ",cons
         print *,"horiz, vert",horiz,vert
           file='IFringe.im'
           twopi = 8.0*atan(1.0)
C        horiz = 0.0
C        vert = 0.0

C         write(*,*)' Input type of interferogram'
C        write(*,*)'      i for infinite fringe '
C        write(*,*)'      v for finite fringe with vertical fringes'
C        write(*,*)'      h for finite fringe with horizontal fringes'
C        read(*,'(a1)')type
C        write(*,*)
C         if(type.eq.'v') then
C          write(*,*)' Input the number of vertical finite fringes.'
C           read(*,*)vert
C           file='VFFringe.im'
C        end if
C        if(type.eq.'h') then
C          write(*,*)' Input the number of horizontal finite fringes.'
C           read(*,*)horiz
C           file='HFFringe.im'
C        end if

           f1 = twopi*vert/Ixmax
           f2 = twopi*horiz/Iymax
           do 10 j=1, Iymax
            dphi2 = float(j-1)*f2
            do 10 i=1,Ixmax
              dphi1 = float(i-1)*f1
              a = cons*p(i+(j-1)*Ixmax,1) + twopi
              p(i+(j-1)*Ixmax,4) = 128. + 127.*cos(a + dphi1 + dphi2)
 10        continue

          do 120 i=1,Ixmax*Iymax
             if(mod(i))p(i,4)=0.0
 120      continue

           return
           end


*******************************************************************************
*
          subroutine shadowg(camera)

C=============================================================
C
C   Author:
C      Leslie A. Yates
C      Eloret Institute
C      NASA Ames Research Center
C      Moffett Field, CA 94035
C      (415) 604-3436
C
C   Date:
C      December, 1991
C
C=============================================================

          character*12 filenam
          logical mod(600001)

          common/image/Ixmax,Iymax,mod,p(600001,4)

         common/integ/ngrid,npts,rho0,rhoinf,al,pass,psi,theta,phi,dz,nphi,
        &             gstrch,strch,ctype,mulsin
         character*1 mulsin,ctype
```



```
         print*,"in shadowg"
         print*,"camera,gstrch",camera,gstrch
C         filenam='Shadow.im'
          do 10 i=1,Ixmax*Iymax
             p(i,4) = 0.0
 10       continue

          do 11 i=1,Ixmax
             p(i,4) = 1.0
             p(i + (Iymax-1)*Ixmax, 4) = 1.0
 11       continue

          do 12 j=1,Iymax
             p(1+(j-1)*Ixmax,4) = 1.0
             p(j*Ixmax,4) = 1.0
 12       continue


          do 15 j= 2, Iymax-1
           do 15 i = 2,  Ixmax-1
             il = i + (j-1)*Ixmax
             x1 = float(i) - 0.5 + 0.5*camera*(p(il-1,2) + p(il,2))
             x2 = float(i) + 0.5 + 0.5*camera*(p(il+1,2) + p(il,2))
             y1 = float(j) - 0.5 + 0.5*camera*(p(il-Ixmax,3) + p(il,3))
             y2 = float(j) + 0.5 + 0.5*camera*(p(il+Ixmax,3) + p(il,3))
             if(x2.lt.x1)then
               dx = x1
               x1 = x2
               x2 = dx
             end if
             if(y2.lt.y1)then
               dy = y1
               y1 = y2
               y2 = dy
             end if
             dx = x2 - x1
             dy = y2 - y1

             darea = 100.
             if(dx*dy .ne. 0) darea = 1./(dx*dy)

             imin = max0(1, int(x1 + 0.5))
             imin = min0(Ixmax, imin)

             imax = max0(1, int(x2 + 0.49999999))
             imax = min0(Ixmax , imax)

             jmin = max0(1, int(y1 + 0.5))
             jmin = min0(Iymax, jmin)

             jmax = max0(1, int(y2 + 0.49999999))
             jmax = min0(Iymax, jmax)

             do 112 m = jmin,jmax
               wy = aminl(float(m)+0.5,y2) - amaxl(float(m)-0.5,y1)
               do 114 n = imin,imax
                 wx = aminl(float(n)+0.5,x2) - amaxl(float(n)-0.5,x1)
                 p(n + (m-1)*Ixmax,4) = p(n + (m-1)*Ixmax,4) + wx*wy*darea
 114           continue
 112         continue

 15       continue
C         write(*,*)' Input linear stretch factor for gray-scale.'
C        read(*,*)gstrch

          do 30 i=1,Ixmax*Iymax
              val = 85.0 + 84.0*gstrch*( p(i,4)*p(i,4) - 1.0)
              val = amaxl(1.0, val)
              p(i,4) = aminl(val,255.0)
 30       continue

          do 120 i=1,Ixmax*Iymax
             if(mod(i))p(i,4)=0.0
 120      continue

          return
          end
*******************************************************************************
          subroutine schlieren(knife)

C=============================================================
C
C   Author:
C      Leslie A. Yates
C      Eloret Institute
C      NASA Ames Research Center
C      Moffett Field, CA 94035
C      (415) 604-3436
C
C   Date:
C      December, 1991
C
C=============================================================

          character*1 knife
          character*12 filenam
          logical mod(600001)

          common/image/Ixmax,Iymax,mod,p(600001,4)

         common/integ/ngrid,npts,rho0,rhoinf,al,pass,psi,theta,phi,dz,nphi,
        &             gstrch,strch,ctype,mulsin
         character*1 mulsin,ctype

         print*,"in schlieren, knife=",knife
           n = 2
C          filenam='VSchlier.im'
         print*,"Ixmax,Iymax=",Ixmax,Iymax
           if(knife.eq.'h')then
             n = 3
C            filenam='HSchlier.im'
           end if

           pmax = 0.0
           pmin = 0.0

           do 10 i=1, Ixmax*Iymax
              pmin = aminl(p(i,n),pmin)
              pmax = amaxl(p(i,n),pmax)
 10        continue
         print*,"pmin,pmax",pmin,pmax
           pmax = amaxl(-pmin,pmax)

C         write(*,*)' Input linear stretch factor for gray-scale.'
C        read(*,*)gstrch
         print*,"gstrch",gstrch
```

```
      if(pmax.ne.0)amul = gstrch/pmax

      do 20 i=1,Ixmax*Iymax
         a = amax1(0.0,1.0 + amul*p(i,n))
         a = amin1(a,2.0)
         p(i,4) = 1.0 + 63.5*a*a
 20   continue

      print*,"leaving schlieren"

      do 120 i=1,Ixmax*Iymax
        if(mod(i))p(i,4)=0.0
 120  continue

      return
      end

*************************************************************************************
*
      subroutine deln(q,nj,nk,nl,rho0,rhoinf,GD,deln0)

C=====================================================================
C
C   Author:
C      Leslie A. Yates
C      Eloret Institute
C      NASA Ames Research Center
C      Moffett Field, CA 94035
C      (415) 604-3436
C
C   Date:
C      December, 1991
C
C=====================================================================

      dimension q(nj*nk*nl)

      do 10 j=1,nj*nk*nl
          q(j) = GD*rhoinf*q(j)/rho0
 10   continue
      deln0 = GD*rhoinf

      return
      end
*************************************************************************
```



```
/* c routines to support the fisst fortran routines for fast */

/*------------------------------ INCLUDES ----------------------------------*/

#include      <stdio.h>       /* for printf() etc.                    */
#include      <string.h>      /* for string stuff                     */
#include      <ctype.h>       /* for isdigit() etc.                   */
#include      <math.h>        /* for atof()                           */
#include      <fld_pan.h>     /* for GRID_SCALAR_VECTOR_TYPEOUT       */
#include      <panel_utils.h> /* for typedefs and FLOAT_STRING_FORMAT */
#include      <grid_surface.h>/* for Grid_Surface typedef             */
/*#include       <View.h>         for Viewing Library #defines        */
#include      <fast_error.h>  /* for ERROR() macro                    */
#include      <fast_memory.h> /* for DDIM3() and DDIM4() macros       */
#include      <fld_list.h>    /* for SCALAR, NUM_VARS etc             */
#include      <fast_cmap.h>   /* for MAX_SCMAPS & colormap functions  */
#include      <Module.h>      /* for scripting stuff                  */
#include      <ViewI.h>       /* for locking/unlocking                */
#include      "funcs.h"       /* for function declarations            */
#include      <ql/gl.h>       /* for gl variables                     */
#include      <device.h>      /* for device variables                 */
#include      "fisstSupport.h"/* for fisst variables                  */
#include      <malloc.h>      /* for memory allocation                */


/*************************************************************************/
/*************************************************************************/


void fisstPanel(Panel* mainPanel, float startX, float startY)
{
   Frame* frameData;

   fisstFrame = pnl_mkact(pnl_frame);
   fisstFrame -> x = startX + FISSTFRAMEX;
   fisstFrame -> y = startY + FISSTFRAMEY;
   pnl_addact(fisstFrame,mainPanel);

   butIntegrate = pnl_mkact(pnl_wide_button);
   butIntegrate -> label = BUTINTEGRATELABEL;
   butIntegrate -> x = BUTINTEGRATEX;
   butIntegrate -> y = BUTINTEGRATEY;
   butIntegrate -> upfunc = butIntegrateUpfunc;
   pnl_addsubact(butIntegrate,fisstFrame);

   butInterferogram = pnl_mkact(pnl_wide_button);
   butInterferogram -> label = BUTINTERFEROGRAMLABEL;
   butInterferogram -> x = BUTINTERFEROGRAMX;
   butInterferogram -> y = BUTINTERFEROGRAMY;
   butInterferogram -> upfunc = butInterferogramUpfunc;
   pnl_addsubact(butInterferogram,fisstFrame);

   butShadowGraph = pnl_mkact(pnl_wide_button);
   butShadowGraph -> label = BUTSHADOWGRAPHLABEL;
   butShadowGraph -> x = BUTSHADOWGRAPHX;
   butShadowGraph -> y = BUTSHADOWGRAPHY;
   butShadowGraph -> upfunc = butShadowGraphUpfunc;
   pnl_addsubact(butShadowGraph,fisstFrame);

   butSchlieren = pnl_mkact(pnl_wide_button);
   butSchlieren -> label = BUTSCHLIERENLABEL;
   butSchlieren -> x = BUTSCHLIERENX;
   butSchlieren -> y = BUTSCHLIERENY;
   butSchlieren -> upfunc = butSchlierenUpfunc;
   pnl_addsubact(butSchlieren,fisstFrame);

/************************ npts ***********************************/

   butNumberOfPoints = pnl_mkact(pnl_wide_button);
   butNumberOfPoints -> label = BUT_NUMBER_OF_POINTS_LABEL;
   butNumberOfPoints -> x = BUT_NUMBER_OF_POINTS_X;
   butNumberOfPoints -> y = BUT_NUMBER_OF_POINTS_Y;
   butNumberOfPoints -> upfunc = butNumberOfPointsUpfunc;
   pnl_addsubact(butNumberOfPoints,fisstFrame);

   displaySldNumberOfPoints();

/******************** rho0 ***********************************/

   butComputationalRho = pnl_mkact(pnl_wide_button);
   butComputationalRho -> label = BUT_COMPUTATIONAL_RHO_LABEL;
   butComputationalRho -> x = BUT_COMPUTATIONAL_RHO_X;
   butComputationalRho -> y = BUT_COMPUTATIONAL_RHO_Y;
   butComputationalRho -> upfunc = butComputationalRhoUpfunc;
   pnl_addsubact(butComputationalRho,fisstFrame);

   displaySldComputationalRho();

/************************ rhoinf ***********************************/

   butExperimentalRho = pnl_mkact(pnl_wide_button);
   butExperimentalRho -> label = BUT_EXPERIMENTAL_RHO_LABEL;
   butExperimentalRho -> x = BUT_EXPERIMENTAL_RHO_X;
   butExperimentalRho -> y = BUT_EXPERIMENTAL_RHO_Y;
   butExperimentalRho -> upfunc = butExperimentalRhoUpfunc;
   pnl_addsubact(butExperimentalRho,fisstFrame);

   displaySldExperimentalRho();

/************************ el ***********************************/

   butNondimensionalizedLength = pnl_mkact(pnl_wide_button);
   butNondimensionalizedLength -> label = BUT_NONDIMENSIONALIZED_LENGTH_LABEL;
   butNondimensionalizedLength -> x = BUT_NONDIMENSIONALIZED_LENGTH_X;
   butNondimensionalizedLength -> y = BUT_NONDIMENSIONALIZED_LENGTH_Y;
   butNondimensionalizedLength -> upfunc = butNondimensionalizedLengthUpfunc;
   pnl_addsubact(butNondimensionalizedLength,fisstFrame);

   displaySldNondimensionalizedLength();

/************************ pass ***********************************/

   butNumberOfPasses = pnl_mkact(pnl_wide_button);
   butNumberOfPasses -> label = BUT_NUMBER_OF_PASSES_LABEL;
   butNumberOfPasses -> x = BUT_NUMBER_OF_PASSES_X;
   butNumberOfPasses -> y = BUT_NUMBER_OF_PASSES_Y;
   butNumberOfPasses -> upfunc = butNumberOfPassesUpfunc;
   pnl_addsubact(butNumberOfPasses,fisstFrame);

   displaySldNumberOfPasses();

/************************ psi ***********************************/

   butPsiAngle = pnl_mkact(pnl_wide_button);
   butPsiAngle -> label = BUT_PSI_ANGLE_LABEL;
   butPsiAngle -> x = BUT_PSI_ANGLE_X;
   butPsiAngle -> y = BUT_PSI_ANGLE_Y;
   butPsiAngle -> upfunc = butPsiAngleUpfunc;
```

```
   pnl_addsubact(butPsiAngle,fisstFrame);

   displaySldPsiAngle();

/************************* theta *********************************/

   butThetaAngle = pnl_mkact(pnl_wide_button);
   butThetaAngle -> label = BUT_THETA_ANGLE_LABEL;
   butThetaAngle -> x = BUT_THETA_ANGLE_X;
   butThetaAngle -> y = BUT_THETA_ANGLE_Y;
   butThetaAngle -> upfunc = butThetaAngleUpfunc;
   pnl_addsubact(butThetaAngle,fisstFrame);

   displaySldThetaAngle();

/************************* phi *********************************/

   butPhiAngle = pnl_mkact(pnl_wide_button);
   butPhiAngle -> label = BUT_PHI_ANGLE_LABEL;
   butPhiAngle -> x = BUT_PHI_ANGLE_X;
   butPhiAngle -> y = BUT_PHI_ANGLE_Y;
   butPhiAngle -> upfunc = butPhiAngleUpfunc;
   pnl_addsubact(butPhiAngle,fisstFrame);

   displaySldPhiAngle();

/************************* dz *********************************/

   butDz2D = pnl_mkact(pnl_wide_button);
   butDz2D -> label = BUT_DZ_2D_LABEL;
   butDz2D -> x = BUT_DZ_2D_X;
   butDz2D -> y = BUT_DZ_2D_Y;
   butDz2D -> upfunc = butDz2DUpfunc;
   pnl_addsubact(butDz2D,fisstFrame);

   displaySldDz2D();

/************************* awl *********************************/

   butWaveLengthOfLight = pnl_mkact(pnl_wide_button);
   butWaveLengthOfLight -> label = BUT_WAVE_LENGTH_OF_LIGHT_LABEL;
   butWaveLengthOfLight -> x = BUT_WAVE_LENGTH_OF_LIGHT_X;
   butWaveLengthOfLight -> y = BUT_WAVE_LENGTH_OF_LIGHT_Y;
   butWaveLengthOfLight -> upfunc = butWaveLengthOfLightUpfunc;
   pnl_addsubact(butWaveLengthOfLight,fisstFrame);

   displaySldWaveLengthOfLight();

/************************* camera *********************************/

   butObjectDistance = pnl_mkact(pnl_wide_button);
   butObjectDistance -> label = BUT_OBJECT_DISTANCE_LABEL;
   butObjectDistance -> x = BUT_OBJECT_DISTANCE_X;
   butObjectDistance -> y = BUT_OBJECT_DISTANCE_Y;
   butObjectDistance -> upfunc = butObjectDistanceUpfunc;
   pnl_addsubact(butObjectDistance,fisstFrame);

   displaySldObjectDistance();

/************************* exposure *********************************/

   butExposure = pnl_mkact(pnl_wide_button);
   butExposure -> label = BUT_EXPOSURE_LABEL;
   butExposure -> x = BUT_EXPOSURE_X;
   butExposure -> y = BUT_EXPOSURE_Y;
   butExposure -> upfunc = butExposureUpfunc;
   pnl_addsubact(butExposure,fisstFrame);

   displaySldExposure();

/************************* nphi *********************************/

   typeinNumberOfPlanes = pnl_mkact(pnl_typein);
   typeinNumberOfPlanes -> x = TYPEIN_NUMBER_OF_PLANES_X;
   typeinNumberOfPlanes -> y = TYPEIN_NUMBER_OF_PLANES_Y;
   typeinNumberOfPlanes -> labeltype = PNL_LABEL_BOTTOM;
   typeinNumberOfPlanes -> label = TYPEIN_NUMBER_OF_PLANES_LABEL;
   PNL_ACCESS(Typein,typeinNumberOfPlanes,len) = NUMBER_OF_PLANES_STRING_LENGTH;
   sprintf(PNL_ACCESS(Typein,typeinNumberOfPlanes,str),NUMBER_OF_PLANES_FORMAT,INI_NUMBER
_OF_PLANES);
   typeinNumberOfPlanes -> upfunc = typeinNumberOfPlanesUpfunc;
   pnl_addsubact(typeinNumberOfPlanes,fisstFrame);

   upArrowNumberOfPlanes = pnl_mkact(pnl_up_arrow_button);
   upArrowNumberOfPlanes -> x = UPARROW_NUMBER_OF_PLANES_X;
   upArrowNumberOfPlanes -> y = UPARROW_NUMBER_OF_PLANES_Y;
   upArrowNumberOfPlanes -> upfunc = upArrowNumberOfPlanesUpfunc;
   pnl_addsubact(upArrowNumberOfPlanes, fisstFrame);

   downArrowNumberOfPlanes = pnl_mkact(pnl_down_arrow_button);
   downArrowNumberOfPlanes -> x = DOWNARROW_NUMBER_OF_PLANES_X;
   downArrowNumberOfPlanes -> y = DOWNARROW_NUMBER_OF_PLANES_Y;
   downArrowNumberOfPlanes -> upfunc = downArrowNumberOfPlanesUpfunc;
   pnl_addsubact(downArrowNumberOfPlanes, fisstFrame);

   resetNumberOfPlanes = pnl_mkact(pnl_button);
   resetNumberOfPlanes -> x = RESET_NUMBER_OF_PLANES_X;
   resetNumberOfPlanes -> y = RESET_NUMBER_OF_PLANES_Y;
   resetNumberOfPlanes -> labeltype = PNL_LABEL_CENTER;
   resetNumberOfPlanes -> label = "R";
   resetNumberOfPlanes -> upfunc = resetNumberOfPlanesUpfunc;
   pnl_addsubact(resetNumberOfPlanes, fisstFrame);

/************************* surface *****************************/

   butSurfaceIJK = pnl_mkact(pnl_wide_button);
   butSurfaceIJK -> label = BUT_SURFACE_IJK_LABEL;
   butSurfaceIJK -> x = BUT_SURFACE_IJK_X;
   butSurfaceIJK -> y = BUT_SURFACE_IJK_Y;
   butSurfaceIJK -> upfunc = butSurfaceIJKUpfunc;
   pnl_addsubact(butSurfaceIJK,fisstFrame);

   butResetSurfaceIJK = pnl_mkact(pnl_wide_button);
   butResetSurfaceIJK -> label = BUT_RESET_SURFACE_IJK_LABEL;
   butResetSurfaceIJK -> x = BUT_RESET_SURFACE_IJK_X;
   butResetSurfaceIJK -> y = BUT_RESET_SURFACE_IJK_Y;
   butResetSurfaceIJK -> upfunc = butResetSurfaceIJKUpfunc;
   pnl_addsubact(butResetSurfaceIJK,fisstFrame);

/************************* knifeEdge *************************/

   displayUpArrowKnifeEdge();

/************************* fringes *************************/

   typeinFringes = pnl_mkact(pnl_typein);
   typeinFringes -> x = TYPEIN_FRINGES_X;
   typeinFringes -> y = TYPEIN_FRINGES_Y;
```



```
   typeinFringes -> labeltype = PNL_LABEL_BOTTOM;
   typeinFringes -> label = TYPEIN_FRINGES_LABEL;
   PNL_ACCESS(Typein,typeinFringes,len) = FRINGES_STRING_LENGTH;
   sprintf(PNL_ACCESS(Typein,typeinFringes,str),FRINGES_FORMAT,INI_FRINGES);
   typeinFringes -> upfunc = typeinFringesUpfunc;
   pnl_addsubact(typeinFringes,fisstFrame);

   upArrowFringes = pnl_mkact(pnl_up_arrow_button);
   upArrowFringes -> x = UPARROW_FRINGES_X;
   upArrowFringes -> y = UPARROW_FRINGES_Y;
   upArrowFringes -> upfunc = upArrowFringesUpfunc;
   pnl_addsubact(upArrowFringes, fisstFrame);

   downArrowFringes = pnl_mkact(pnl_down_arrow_button);
   downArrowFringes -> x = DOWNARROW_FRINGES_X;
   downArrowFringes -> y = DOWNARROW_FRINGES_Y;
   downArrowFringes -> upfunc = downArrowFringesUpfunc;
   pnl_addsubact(downArrowFringes, fisstFrame);

   resetFringes = pnl_mkact(pnl_button);
   resetFringes -> x = RESET_FRINGES_X;
   resetFringes -> y = RESET_FRINGES_Y;
   resetFringes -> labeltype = PNL_LABEL_CENTER;
   resetFringes -> label = "R";
   resetFringes -> upfunc = resetFringesUpfunc;
   pnl_addsubact(resetFringes, fisstFrame);

/************************* FringeType *************************/

   displayUpArrowFringeType();

/**************************************************************/

   pnl_fixact(fisstFrame);
}



void butIntegrateUpfunc(Actuator* act)
{

   if(iniIntegrateParams())
   {
      if(xyzData == NULL)
      {
         if(!(xyzData = (float*) malloc( xyzByteSize )))
         {
            printf("%% Unable to allocate memory %%\n");
            exit(0);
         }
         if(!(rhoData = (float*) malloc( rhoByteSize )))
         {
            printf("%% Unable to allocate memory %%\n");
            exit(0);
         }
      }
   }
   else if(ijkSize<newijkSize)
   {
      if(!(xyzData = (float*) realloc((void*) xyzData,  xyzByteSize )))
      {
         printf("%% Unable to allocate memory for xyz %%\n");
         exit(0);
      }
      if(!(rhoData = (float*) realloc((void*) rhoData,  rhoByteSize )))
      {
         printf("%% Unable to allocate memory for rho %%\n");
         exit(0);
      }
   }

   inixyzRho( xyzData, rhoData );

   imageF_(xyzData,rhoData);

   imageDrawn = FALSE;
}


void inixyzRho( float* xyz, float* rho )
{
   int
      inc ,i_inc, j_inc, k_inc, xyz_inc;
   int
      data_imax, data_jmax, data_kmax;

   lock_cur_object();

   data_imax = grid_fisst -> ranges[I][DIM];
   data_jmax = grid_fisst -> ranges[J][DIM];
   data_kmax = grid_fisst -> ranges[K][DIM];
   for( inc=0, xyz_inc = 0; xyz_inc < 3; xyz_inc++ ) {
      for( k_inc = grid_fisst -> ranges[K][START]-1;
           k_inc < grid_fisst -> ranges[K][END]; k_inc++ ) {
         for( j_inc = grid_fisst -> ranges[J][START]-1;
              j_inc < grid_fisst -> ranges[J][END]; j_inc++ ) {
            for( i_inc = grid_fisst -> ranges[I][START]-1;
                 i_inc < grid_fisst -> ranges[I][END]; i_inc++ ) {
               *(xyz+inc) = *(xyzFisst+k_inc*data_jmax*data_imax*3
                            +j_inc*data_imax*3+i_inc*3+xyz_inc);
               if(xyz_inc==0)
                  *(rho+inc) = *(rhoFisst+k_inc*data_jmax*data_imax*NDIM
                               +j_inc*data_imax*NDIM+i_inc*NDIM);
               inc++;
            }
         }
      }
   }
   unlock_cur_object();
}


int iniIntegrateParams()
{
   callIntegrate = TRUE;
   lock_cur_object();
   newijkSize= (grid_fisst -> ranges[I][END] - grid_fisst -> ranges[I][START]+1)
      * (grid_fisst -> ranges[J][END] - grid_fisst -> ranges[J][START] + 1)
      * (grid_fisst -> ranges[K][END] - grid_fisst -> ranges[K][START] + 1);
   xyzFisst = (float *) shm_attach( grid_fisst -> field_ids[GRID_ID] );
   rhoFisst = (float *) shm_attach( grid_fisst -> field_ids[VECTOR_ID] );
   integ_.ngrid = 0;
   integ_.npts = numberOfPoints;
   integ_.rho0 = computationalRho;
   integ_.rhoinf = experimentalRho;
   integ_.al = nondimensionalizedLength;
   integ_.pass = numberOfPasses;
```

```
   integ_.psi = psiAngle;
   integ_.theta = thetaAngle;
   integ_.phi = phiAngle;
   integ_.dz = dz2D;
   integ_.nphi = numberOfPlanes;
   integ_.gstrch = exposure;
   integ_.ctype = '3';
   integ_.mulsin = 's';

   if(!integ_.ngrid++)
   {
      subjkl_.jdim[0] = grid_fisst -> ranges[I][DIM];
      subjkl_.kdim[0] = grid_fisst -> ranges[J][DIM];
      subjkl_.ldim[0] = grid_fisst -> ranges[K][DIM];
   }

   surfset_.nseg = surfaceIJK;

/* assume a single grid */
   data_.nsub[0] = 1;
   data_.jsub1[0][0] = 1;
   data_.jsub2[0][0] = subjkl_.jdim[0];
   data_.ksub1[0][0] = 1;
   data_.ksub2[0][0] = subjkl_.kdim[0];
   data_.lsub1[0][0] = 1;
   data_.lsub2[0][0] = subjkl_.ldim[0];

   unlock_cur_object();

   if( ijkSize >= newijkSize )
   {
      return FALSE;
   }
   else
   {
      ijkSize = newijkSize;
      xyzByteSize = 3 * 4 * ijkSize;
      rhoByteSize = 4 * ijkSize * 5;   /* times 5 temporarily */
      return TRUE;
   }
}

void butInterferogramUpfunc(Actuator* act)
{
   if(!callIntegrate)
      butIntegrateUpfunc(act);

   interf_.awl = waveLengthOfLight;
   if(fringeType==FRINGE_TYPE_HORIZONTAL)
   {
      interf_.vert  = 0.0;
      interf_.horiz = (float) fringes;
   }
   else if(fringeType==FRINGE_TYPE_VERTICAL)
   {
      interf_.horiz = 0.0;
      interf_.vert  = (float) fringes;
   }
   else
   {
      interf_.vert  = 0.0;
      interf_.horiz = 0.0;
   }
   interf_.type = fringeType;

printf("vert=%f, horiz=%f\n",interf_.vert,interf_.horiz);

   interf_.cons= 8.0*atan(1.0)*integ_.al*integ_.pass/(interf_.awl*integ_.strch);
   if(integ_.ctype=='2')interf_.cons= 8.0*atan(1.0)*integ_.al*integ_.pass/interf_.awl;
printf("cons =%f, al=%e, pass=%f, awl=%e, strch=%f\n",interf_.cons,
integ_.al,integ_.pass,interf_.awl,integ_.strch);
   fringes_(&interf_.cons);

   image_title = image_interferogram;
   drawimage_(&image_.ixmax,&image_.iymax,&image_.p[3][0]);
}

void butShadowGraphUpfunc(Actuator* act)
{
   if(!callIntegrate)
      butIntegrateUpfunc(act);
   shadow_.camera = objectDistance*integ_.strch*integ_.al;
   shadowg_(&shadow_.camera);

   image_title = image_shadowgraph;
   drawimage_(&image_.ixmax,&image_.iymax,&image_.p[3][0]);
}

void butSchlierenUpfunc(Actuator* act)
{
   if(!callIntegrate)
      butIntegrateUpfunc(act);
   printf("calling schlieren\n");
   schlieren_(&knifeEdge);

   image_title = image_schlieren;
   drawimage_(&image_.ixmax,&image_.iymax,&(image_.p[3][0]));
}


int drawimage_( int *xdim, int *ydim, float *pix )
{
   void InitWindow(int,int);
   void DumpPixs(int,int,float*);

   if(!imageDrawn)
   {
      if(image_gid) winclose(image_gid);
      imageDrawn = TRUE;
      InitWindow(*xdim,*ydim);
   }
   winset(image_gid);
   wintitle(image_title);
   DumpPixs(*xdim,*ydim,pix);
}

void InitWindow(int xdim, int ydim)
{
   extern void prefsize(long, long);
   extern long int winopen(String);

   prefsize(xdim,ydim);
   image_gid = winopen("image");
   gconfig();
}

void DumpPixs( int xdim, int ydim, float *pix )
{
   extern void rectwrite(Screencoord, Screencoord, Screencoord, Screencoord, const Colori
```



```
ndex[]);
   extern float ftrunc(float);

   static unsigned short int *pixs = NULL;
   long int incr;

   if(!imageDrawn)
   {
      if(pixs)
         pixs = (unsigned short int*) malloc( (unsigned long int) xdim*ydim*2 );
   }
   else
   {
      free(pixs);
         pixs = (unsigned short int*) malloc( (unsigned long int) xdim*ydim*2 );
   }
   for(incr=0; incr<xdim*ydim; incr++)
   {
      *(pixs+incr) = ftrunc( (*(pix+incr)/256)*1024 + 64.4999 );
   }

   rectwrite(0,0,xdim-1,ydim-1,pixs);
}


/****************** NumberOfPointsUpfunc *********************/

void typeinNumberOfPointsUpfunc( Actuator *act )
{
   int tmp = atoi(PNL_ACCESS(Typein,typeinNumberOfPoints,str));

   if( tmp < NUMBER_OF_POINTS_MIN )
   {
      numberOfPoints = NUMBER_OF_POINTS_MIN;
   }
   else if( tmp > NUMBER_OF_POINTS_MAX )
   {
      numberOfPoints = NUMBER_OF_POINTS_MAX;
   }
   else
   {
      numberOfPoints = tmp;
   }

   sprintf(PNL_ACCESS(Typein,typeinNumberOfPoints,str),NUMBER_OF_POINTS_FORMAT,numberOfP
oints);
}

void butNumberOfPointsUpfunc( Actuator *act )
{
   if( flagNumberOfPoints = !flagNumberOfPoints )
   {
      pnl_delact(typeinNumberOfPoints);
      pnl_fixact(fisstFrame);
      displaySldNumberOfPoints();
   }
   else
   {
      pnl_delact(sldNumberOfPoints);
      pnl_fixact(fisstFrame);
      displayTypeinNumberOfPoints();
   }
}
void sldNumberOfPointsUpfunc( Actuator *act )
{
   numberOfPoints = sldNumberOfPoints -> val;
}

/****************** ComputationalRho *************************/

void typeinComputationalRhoUpfunc( Actuator* act )
{
   float tmp = atof(PNL_ACCESS(Typein,typeinComputationalRho,str));
   if( tmp < COMPUTATIONAL_RHO_MIN )
   {
      computationalRho = COMPUTATIONAL_RHO_MIN;
   }
   else if( tmp > COMPUTATIONAL_RHO_MAX )
   {
      computationalRho = COMPUTATIONAL_RHO_MAX;
   }
   else
   {
      computationalRho = tmp;
   }

   sprintf(PNL_ACCESS(Typein,typeinComputationalRho,str),COMPUTATIONAL_RHO_FORMAT,computa
tionalRho);
}

void butComputationalRhoUpfunc( Actuator* act )
{
   if( flagComputationalRho = !flagComputationalRho )
   {
      pnl_delact(typeinComputationalRho);
      pnl_fixact(fisstFrame);
      displaySldComputationalRho();
   }
   else
   {
      pnl_delact(sldComputationalRho);
      pnl_fixact(fisstFrame);
      displayTypeinComputationalRho();
   }
}

void sldComputationalRhoUpfunc( Actuator* act )
{
   computationalRho = sldComputationalRho -> val;
}

/********************* ExperimentalRho ***********************************/

void typeinExperimentalRhoUpfunc( Actuator* act )
{
   float tmp = atof(PNL_ACCESS(Typein,typeinExperimentalRho,str));
   if( tmp < EXPERIMENTAL_RHO_MIN )
   {
      experimentalRho = EXPERIMENTAL_RHO_MIN;
   }
   else if( tmp > EXPERIMENTAL_RHO_MAX )
   {
      experimentalRho = EXPERIMENTAL_RHO_MAX;
   }
   else
   {
      experimentalRho = tmp;
   }
```

```
   sprintf(PNL_ACCESS(Typein,typeinExperimentalRho,str),EXPERIMENTAL_RHO_FORMAT,experime
ntalRho);
}

void butExperimentalRhoUpfunc( Actuator* act )
{
   if( flagExperimentalRho = !flagExperimentalRho )
   {
      pnl_delact(typeinExperimentalRho);
      pnl_fixact(fisstFrame);
      displaySldExperimentalRho();
   }
   else
   {
      pnl_delact(sldExperimentalRho);
      pnl_fixact(fisstFrame);
      displayTypeinExperimentalRho();
   }
}

void sldExperimentalRhoUpfunc( Actuator* act )
{
   experimentalRho = sldExperimentalRho -> val;
}

/********************* NondimensionalizedLength  *************************/

void typeinNondimensionalizedLengthUpfunc( Actuator* act )
{
   float tmp = atof(PNL_ACCESS(Typein,typeinNondimensionalizedLength,str));
   if( tmp < NONDIMENSIONALIZED_LENGTH_MIN )
   {
      nondimensionalizedLength = NONDIMENSIONALIZED_LENGTH_MIN;
   }
   else if( tmp > NONDIMENSIONALIZED_LENGTH_MAX )
   {
      nondimensionalizedLength = NONDIMENSIONALIZED_LENGTH_MAX;
   }
   else
   {
      nondimensionalizedLength = tmp;
   }

   sprintf(PNL_ACCESS(Typein,typeinNondimensionalizedLength,str),NONDIMENSIONALIZED_LENG
TH_FORMAT,nondimensionalizedLength);
}

void butNondimensionalizedLengthUpfunc( Actuator* act )
{
   if( flagNondimensionalizedLength = !flagNondimensionalizedLength )
   {
      pnl_delact(typeinNondimensionalizedLength);
      pnl_fixact(fisstFrame);
      displaySldNondimensionalizedLength();
   }
   else
   {
      pnl_delact(sldNondimensionalizedLength);
      pnl_fixact(fisstFrame);
      displayTypeinNondimensionalizedLength();
   }
}

void sldNondimensionalizedLengthUpfunc( Actuator* act )
{
   nondimensionalizedLength = sldNondimensionalizedLength -> val;
}

/************************* NumberOfPasses *********************************/

void typeinNumberOfPassesUpfunc( Actuator* act )
{
   float tmp = atof(PNL_ACCESS(Typein,typeinNumberOfPasses,str));
   if( tmp < NUMBER_OF_PASSES_MIN )
   {
      numberOfPasses = NUMBER_OF_PASSES_MIN;
   }
   else if( tmp > NUMBER_OF_PASSES_MAX )
   {
      numberOfPasses = NUMBER_OF_PASSES_MAX;
   }
   else
   {
      numberOfPasses = tmp;
   }

   sprintf(PNL_ACCESS(Typein,typeinNumberOfPasses,str),NUMBER_OF_PASSES_FORMAT,numberOfPa
sses);
}

void butNumberOfPassesUpfunc( Actuator* act )
{
   if( flagNumberOfPasses = !flagNumberOfPasses )
   {
      pnl_delact(typeinNumberOfPasses);
      pnl_fixact(fisstFrame);
      displaySldNumberOfPasses();
   }
   else
   {
      pnl_delact(sldNumberOfPasses);
      pnl_fixact(fisstFrame);
      displayTypeinNumberOfPasses();
   }
}

void sldNumberOfPassesUpfunc( Actuator* act )
{
   numberOfPasses = sldNumberOfPasses -> val;
}

/************************* PsiAngle *********************************/

void typeinPsiAngleUpfunc( Actuator* act )
{
   float tmp = atof(PNL_ACCESS(Typein,typeinPsiAngle,str));
   if( tmp < PSI_ANGLE_MIN )
   {
      psiAngle = PSI_ANGLE_MIN;
   }
   else if( tmp > PSI_ANGLE_MAX )
   {
      psiAngle = PSI_ANGLE_MAX;
   }
   else
   {
      psiAngle = tmp;
```



```
   }

   sprintf(PNL_ACCESS(Typein,typeinPsiAngle,str),PSI_ANGLE_FORMAT,psiAngle);
}

void butPsiAngleUpfunc( Actuator* act )
{
   if( flagPsiAngle = !flagPsiAngle )
   {
      pnl_delact(typeinPsiAngle);
      pnl_fixact(fisstFrame);
      displaySldPsiAngle();
   }
   else
   {
      pnl_delact(sldPsiAngle);
      pnl_fixact(fisstFrame);
      displayTypeinPsiAngle();
   }
}

void sldPsiAngleUpfunc( Actuator* act )
{
   psiAngle = sldPsiAngle -> val;
}

/************************* ThetaAngle *********************************/

void typeinThetaAngleUpfunc( Actuator* act )
{
   float tmp = atof(PNL_ACCESS(Typein,typeinThetaAngle,str));
   if( tmp < THETA_ANGLE_MIN )
   {
      thetaAngle = THETA_ANGLE_MIN;
   }
   else if( tmp > THETA_ANGLE_MAX )
   {
      thetaAngle = THETA_ANGLE_MAX;
   }
   else
   {
      thetaAngle = tmp;
   }

   sprintf(PNL_ACCESS(Typein,typeinThetaAngle,str),THETA_ANGLE_FORMAT,thetaAngle);
}

void butThetaAngleUpfunc( Actuator* act )
{
   if( flagThetaAngle = !flagThetaAngle )
   {
      pnl_delact(typeinThetaAngle);
      pnl_fixact(fisstFrame);
      displaySldThetaAngle();
   }
   else
   {
      pnl_delact(sldThetaAngle);
      pnl_fixact(fisstFrame);
      displayTypeinThetaAngle();
   }
}

void sldThetaAngleUpfunc( Actuator* act )
{
   thetaAngle = sldThetaAngle -> val;
}

/************************* PhiAngle *********************************/

void typeinPhiAngleUpfunc( Actuator* act )
{
   float tmp = atof(PNL_ACCESS(Typein,typeinPhiAngle,str));
   if( tmp < PHI_ANGLE_MIN )
   {
      phiAngle = PHI_ANGLE_MIN;
   }
   else if( tmp > PHI_ANGLE_MAX )
   {
      phiAngle = PHI_ANGLE_MAX;
   }
   else
   {
      phiAngle = tmp;
   }

   sprintf(PNL_ACCESS(Typein,typeinPhiAngle,str),PHI_ANGLE_FORMAT,phiAngle);
}

void butPhiAngleUpfunc( Actuator* act )
{
   if( flagPhiAngle = !flagPhiAngle )
   {
      pnl_delact(typeinPhiAngle);
      pnl_fixact(fisstFrame);
      displaySldPhiAngle();
   }
   else
   {
      pnl_delact(sldPhiAngle);
      pnl_fixact(fisstFrame);
      displayTypeinPhiAngle();
   }
}

void sldPhiAngleUpfunc( Actuator* act )
{
   phiAngle = sldPhiAngle -> val;
}

/************************* Dz2D *********************************/

void typeinDz2DUpfunc( Actuator* act )
{
   float tmp = atof(PNL_ACCESS(Typein,typeinDz2D,str));
   if( tmp < DZ_2D_MIN )
   {
      dz2D = DZ_2D_MIN;
   }
   else if( tmp > DZ_2D_MAX )
   {
      dz2D = DZ_2D_MAX;
   }
   else
   {
      dz2D = tmp;
   }
```

```
   sprintf(PNL_ACCESS(Typein,typeinDz2D,str),DZ_2D_FORMAT,dz2D);
}

void butDz2DUpfunc( Actuator* act )
{
   if( flagDz2D = !flagDz2D )
   {
      pnl_delact(typeinDz2D);
      pnl_fixact(fisstFrame);
      displaySldDz2D();
   }
   else
   {
      pnl_delact(sldDz2D);
      pnl_fixact(fisstFrame);
      displayTypeinDz2D();
   }
}

void sldDz2DUpfunc( Actuator* act )
{
   dz2D = sldDz2D -> val;
}

/************************* NumberOfPlanes *********************************/

void typeinNumberOfPlanesUpfunc( Actuator* act )
{
   float tmp = atof(PNL_ACCESS(Typein,typeinNumberOfPlanes,str));
   if( tmp < NUMBER_OF_PLANES_MIN )
   {
      numberOfPlanes = NUMBER_OF_PLANES_MIN;
   }
   else if( tmp > NUMBER_OF_PLANES_MAX )
   {
      numberOfPlanes = NUMBER_OF_PLANES_MAX;
   }
   else
   {
      numberOfPlanes = tmp;
   }

   sprintf(PNL_ACCESS(Typein,typeinNumberOfPlanes,str),NUMBER_OF_PLANES_FORMAT,numberOfP
lanes);
}

void upArrowNumberOfPlanesUpfunc( Actuator* act )
{
   if( numberOfPlanes < NUMBER_OF_PLANES_MAX ) {
      ++numberOfPlanes;
      sprintf(PNL_ACCESS(Typein,typeinNumberOfPlanes,str),NUMBER_OF_PLANES_FORMAT,number
OfPlanes);
      pnl_fixact(typeinNumberOfPlanes);
   }
}

void downArrowNumberOfPlanesUpfunc( Actuator* act )
{
   if( numberOfPlanes > NUMBER_OF_PLANES_MIN ) {
      --numberOfPlanes;
      sprintf(PNL_ACCESS(Typein,typeinNumberOfPlanes,str),NUMBER_OF_PLANES_FORMAT,number
OfPlanes);
      pnl_fixact(typeinNumberOfPlanes);
   }
}

void resetNumberOfPlanesUpfunc( Actuator* act )
{
   numberOfPlanes = INI_NUMBER_OF_PLANES;
   sprintf(PNL_ACCESS(Typein,typeinNumberOfPlanes,str),NUMBER_OF_PLANES_FORMAT,numberOfPl
anes);
   pnl_fixact(typeinNumberOfPlanes);
}

/************************* WaveLengthOfLight *********************************/

void typeinWaveLengthOfLightUpfunc( Actuator* act )
{
   float tmp = atof(PNL_ACCESS(Typein,typeinWaveLengthOfLight,str));
   if( tmp < WAVE_LENGTH_OF_LIGHT_MIN )
   {
      waveLengthOfLight = WAVE_LENGTH_OF_LIGHT_MIN;
   }
   else if( tmp > WAVE_LENGTH_OF_LIGHT_MAX )
   {
      waveLengthOfLight = WAVE_LENGTH_OF_LIGHT_MAX;
   }
   else
   {
      waveLengthOfLight = tmp;
   }

   sprintf(PNL_ACCESS(Typein,typeinWaveLengthOfLight,str),WAVE_LENGTH_OF_LIGHT_FORMAT,wav
eLengthOfLight);
}

void butWaveLengthOfLightUpfunc( Actuator* act )
{
   if( flagWaveLengthOfLight = !flagWaveLengthOfLight )
   {
      pnl_delact(typeinWaveLengthOfLight);
      pnl_fixact(fisstFrame);
      displaySldWaveLengthOfLight();
   }
   else
   {
      pnl_delact(sldWaveLengthOfLight);
      pnl_fixact(fisstFrame);
      displayTypeinWaveLengthOfLight();
   }
}

void sldWaveLengthOfLightUpfunc( Actuator* act )
{
   waveLengthOfLight = sldWaveLengthOfLight -> val;
}

/************************* ObjectDistance *********************************/

void typeinObjectDistanceUpfunc( Actuator* act )
{
   float tmp = atof(PNL_ACCESS(Typein,typeinObjectDistance,str));
   if( tmp < OBJECT_DISTANCE_MIN )
   {
      objectDistance = OBJECT_DISTANCE_MIN;
   }
   else if( tmp > OBJECT_DISTANCE_MAX )
   {
      objectDistance = OBJECT_DISTANCE_MAX;
   }
   else
   {
      objectDistance = tmp;
   }

   sprintf(PNL_ACCESS(Typein,typeinObjectDistance,str),OBJECT_DISTANCE_FORMAT,objectDist
ance);
}

void butObjectDistanceUpfunc( Actuator* act )
{
   if( flagObjectDistance = !flagObjectDistance )
   {
      pnl_delact(typeinObjectDistance);
      pnl_fixact(fisstFrame);
      displaySldObjectDistance();
   }
   else
   {
      pnl_delact(sldObjectDistance);
      pnl_fixact(fisstFrame);
      displayTypeinObjectDistance();
   }
}

void sldObjectDistanceUpfunc( Actuator* act )
{
   objectDistance = sldObjectDistance -> val;
}

/************************* Exposure *********************************/

void typeinExposureUpfunc( Actuator* act )
{
   float tmp = atof(PNL_ACCESS(Typein,typeinExposure,str));
   if( tmp < EXPOSURE_MIN )
   {
      exposure = EXPOSURE_MIN;
   }
   else if( tmp > EXPOSURE_MAX )
   {
      exposure = EXPOSURE_MAX;
   }
   else
   {
      exposure = tmp;
   }

   sprintf(PNL_ACCESS(Typein,typeinExposure,str),EXPOSURE_FORMAT,exposure);
   integ_.gstrch = exposure;
}

void butExposureUpfunc( Actuator* act )
{
   if( flagExposure = !flagExposure )
   {
      pnl_delact(typeinExposure);
      pnl_fixact(fisstFrame);
      displaySldExposure();
   }
   else
   {
      pnl_delact(sldExposure);
      pnl_fixact(fisstFrame);
      displayTypeinExposure();
   }
}

void sldExposureUpfunc( Actuator* act )
{
   exposure = sldExposure -> val;
   integ_.gstrch = exposure;
}

/************************* SurfaceIJK *********************************/

void butSurfaceIJKUpfunc( Actuator* act )
{
   lock_cur_object();

   surfset_.iminset[surfaceIJK] = grid_fisst -> ranges[I][START];
   surfset_.imaxset[surfaceIJK] = grid_fisst -> ranges[I][END];
   surfset_.jminset[surfaceIJK] = grid_fisst -> ranges[J][START];
   surfset_.jmaxset[surfaceIJK] = grid_fisst -> ranges[J][END];
   surfset_.kminset[surfaceIJK] = grid_fisst -> ranges[K][START];
   surfset_.kmaxset[surfaceIJK] = grid_fisst -> ranges[K][END];
   surfaceIJK++;

   unlock_cur_object();
}

void butResetSurfaceIJKUpfunc( Actuator* act )
{
   surfaceIJK = 0;
}

/************************* npts *********************************/

void displayTypeinNumberOfPoints()
{
   typeinNumberOfPoints = pnl_mkact(pnl_typein);
   typeinNumberOfPoints -> x = TYPEIN_NUMBER_OF_POINTS_X;
   typeinNumberOfPoints -> y = TYPEIN_NUMBER_OF_POINTS_Y;
   PNL_ACCESS(Typein,typeinNumberOfPoints,len) = NUMBER_OF_POINTS_STRING_LENGTH;
   sprintf(PNL_ACCESS(Typein,typeinNumberOfPoints,str),NUMBER_OF_POINTS_FORMAT,numberOfPo
ints);
   typeinNumberOfPoints -> upfunc = typeinNumberOfPointsUpfunc;
   pnl_addsubact(typeinNumberOfPoints,fisstFrame);
}

void displaySldNumberOfPoints()
{
   sldNumberOfPoints = pnl_mkact(pnl_slideroid);
   sldNumberOfPoints -> x = SLD_NUMBER_OF_POINTS_X;
   sldNumberOfPoints -> y = SLD_NUMBER_OF_POINTS_Y;
   sldNumberOfPoints -> minval = NUMBER_OF_POINTS_MIN;
   sldNumberOfPoints -> maxval = NUMBER_OF_POINTS_MAX;
   sldNumberOfPoints -> val = (float) numberOfPoints;
   sldNumberOfPoints -> upfunc = sldNumberOfPointsUpfunc;
   pnl_addsubact(sldNumberOfPoints,fisstFrame);
}
```

```
/****************** rho0 ****************************************/

void displayTypeinComputationalRho()
{
   typeinComputationalRho = pnl_mkact(pnl_typein);
   typeinComputationalRho -> x = TYPEIN_COMPUTATIONAL_RHO_X;
   typeinComputationalRho -> y = TYPEIN_COMPUTATIONAL_RHO_Y;
   PNL_ACCESS(Typein,typeinComputationalRho,len) = COMPUTATIONAL_RHO_STRING_LENGTH;
   sprintf(PNL_ACCESS(Typein,typeinComputationalRho,str),COMPUTATIONAL_RHO_FORMAT,comput
ationalRho);
   typeinComputationalRho -> upfunc = typeinComputationalRhoUpfunc;
   pnl_addsubact(typeinComputationalRho,fisstFrame);
}

void displaySldComputationalRho()
{
   sldComputationalRho = pnl_mkact(pnl_slideroid);
   sldComputationalRho -> x = SLD_COMPUTATIONAL_RHO_X;
   sldComputationalRho -> y = SLD_COMPUTATIONAL_RHO_Y;
   sldComputationalRho -> minval = COMPUTATIONAL_RHO_MIN;
   sldComputationalRho -> maxval = COMPUTATIONAL_RHO_MAX;
   sldComputationalRho -> val = computationalRho;
   sldComputationalRho -> upfunc = sldComputationalRhoUpfunc;
   pnl_addsubact(sldComputationalRho,fisstFrame);
}

/********************** rhoinf **********************************/

void displayTypeinExperimentalRho()
{
   typeinExperimentalRho = pnl_mkact(pnl_typein);
   typeinExperimentalRho -> x = TYPEIN_EXPERIMENTAL_RHO_X;
   typeinExperimentalRho -> y = TYPEIN_EXPERIMENTAL_RHO_Y;
   PNL_ACCESS(Typein,typeinExperimentalRho,len) = EXPERIMENTAL_RHO_STRING_LENGTH
;
   sprintf(PNL_ACCESS(Typein,typeinExperimentalRho,str),EXPERIMENTAL_RHO_FORMAT,experime
ntalRho);
   typeinExperimentalRho -> upfunc = typeinExperimentalRhoUpfunc;
   pnl_addsubact(typeinExperimentalRho,fisstFrame);
}

void displaySldExperimentalRho()
{
   sldExperimentalRho = pnl_mkact(pnl_slideroid);
   sldExperimentalRho -> x = SLD_EXPERIMENTAL_RHO_X;
   sldExperimentalRho -> y = SLD_EXPERIMENTAL_RHO_Y;
   sldExperimentalRho -> minval = EXPERIMENTAL_RHO_MIN;
   sldExperimentalRho -> maxval = EXPERIMENTAL_RHO_MAX;
   sldExperimentalRho -> val = experimentalRho;
   sldExperimentalRho -> upfunc = sldExperimentalRhoUpfunc;
   pnl_addsubact(sldExperimentalRho,fisstFrame);
}

/********************** al **********************************/

void displayTypeinNondimensionalizedLength()
{
   typeinNondimensionalizedLength = pnl_mkact(pnl_typein);
   typeinNondimensionalizedLength -> x = TYPEIN_NONDIMENSIONALIZED_LENGTH_X;
   typeinNondimensionalizedLength -> y = TYPEIN_NONDIMENSIONALIZED_LENGTH_Y;
   PNL_ACCESS(Typein,typeinNondimensionalizedLength,len) = NONDIMENSIONALIZED_LENGTH_STR
ING_LENGTH;
   sprintf(PNL_ACCESS(Typein,typeinNondimensionalizedLength,str),NONDIMENSIONALIZED_LENG
TH_FORMAT,nondimensionalizedLength);
   typeinNondimensionalizedLength -> upfunc = typeinNondimensionalizedLengthUpfunc;
   pnl_addsubact(typeinNondimensionalizedLength,fisstFrame);
}

void displaySldNondimensionalizedLength()
{
   sldNondimensionalizedLength = pnl_mkact(pnl_slideroid);
   sldNondimensionalizedLength -> x = SLD_NONDIMENSIONALIZED_LENGTH_X;
   sldNondimensionalizedLength -> y = SLD_NONDIMENSIONALIZED_LENGTH_Y;
   sldNondimensionalizedLength -> minval = NONDIMENSIONALIZED_LENGTH_MIN;
   sldNondimensionalizedLength -> maxval = NONDIMENSIONALIZED_LENGTH_MAX;
   sldNondimensionalizedLength -> val = nondimensionalizedLength;
   sldNondimensionalizedLength -> upfunc = sldNondimensionalizedLengthUpfunc;
   pnl_addsubact(sldNondimensionalizedLength,fisstFrame);
}

/************************ pass **********************************/

void displayTypeinNumberOfPasses()
{
   typeinNumberOfPasses = pnl_mkact(pnl_typein);
   typeinNumberOfPasses -> x = TYPEIN_NUMBER_OF_PASSES_X;
   typeinNumberOfPasses -> y = TYPEIN_NUMBER_OF_PASSES_Y;
   PNL_ACCESS(Typein,typeinNumberOfPasses,len) = NUMBER_OF_PASSES_STRING_LENGTH;
   sprintf(PNL_ACCESS(Typein,typeinNumberOfPasses,str),NUMBER_OF_PASSES_FORMAT,numberOfPa
sses);
   typeinNumberOfPasses -> upfunc = typeinNumberOfPassesUpfunc;
   pnl_addsubact(typeinNumberOfPasses,fisstFrame);
}

void displaySldNumberOfPasses()
{
   sldNumberOfPasses = pnl_mkact(pnl_slideroid);
   sldNumberOfPasses -> x = SLD_NUMBER_OF_PASSES_X;
   sldNumberOfPasses -> y = SLD_NUMBER_OF_PASSES_Y;
   sldNumberOfPasses -> minval = NUMBER_OF_PASSES_MIN;
   sldNumberOfPasses -> maxval = NUMBER_OF_PASSES_MAX;
   sldNumberOfPasses -> val = numberOfPasses;
   sldNumberOfPasses -> upfunc = sldNumberOfPassesUpfunc;
   pnl_addsubact(sldNumberOfPasses,fisstFrame);
}

/************************ psi **********************************/

void displayTypeinPsiAngle()
{
   typeinPsiAngle = pnl_mkact(pnl_typein);
   typeinPsiAngle -> x = TYPEIN_PSI_ANGLE_X;
   typeinPsiAngle -> y = TYPEIN_PSI_ANGLE_Y;
   PNL_ACCESS(Typein,typeinPsiAngle,len) = PSI_ANGLE_STRING_LENGTH;
   sprintf(PNL_ACCESS(Typein,typeinPsiAngle,str),PSI_ANGLE_FORMAT,psiAngle)
;
   typeinPsiAngle -> upfunc = typeinPsiAngleUpfunc;
   pnl_addsubact(typeinPsiAngle,fisstFrame);
}

void displaySldPsiAngle()
{
   sldPsiAngle = pnl_mkact(pnl_slideroid);
   sldPsiAngle -> x = SLD_PSI_ANGLE_X;
   sldPsiAngle -> y = SLD_PSI_ANGLE_Y;
   sldPsiAngle -> minval = PSI_ANGLE_MIN;
   sldPsiAngle -> maxval = PSI_ANGLE_MAX;
```



```
   sldPsiAngle -> val = psiAngle;
   sldPsiAngle -> upfunc = sldPsiAngleUpfunc;
   pnl_addsubact(sldPsiAngle,fisstFrame);
}

/************************ theta **********************************/

void displayTypeinThetaAngle()
{
   typeinThetaAngle = pnl_mkact(pnl_typein);
   typeinThetaAngle -> x = TYPEIN_THETA_ANGLE_X;
   typeinThetaAngle -> y = TYPEIN_THETA_ANGLE_Y;
   PNL_ACCESS(Typein,typeinThetaAngle,len) = THETA_ANGLE_STRING_LENGTH;
   sprintf(PNL_ACCESS(Typein,typeinThetaAngle,str),THETA_ANGLE_FORMAT,thetaAngle);
   typeinThetaAngle -> upfunc = typeinThetaAngleUpfunc;
   pnl_addsubact(typeinThetaAngle,fisstFrame);
}

void displaySldThetaAngle()
{
   sldThetaAngle = pnl_mkact(pnl_slideroid);
   sldThetaAngle -> x = SLD_THETA_ANGLE_X;
   sldThetaAngle -> y = SLD_THETA_ANGLE_Y;
   sldThetaAngle -> minval = THETA_ANGLE_MIN;
   sldThetaAngle -> maxval = THETA_ANGLE_MAX;
   sldThetaAngle -> val = thetaAngle;
   sldThetaAngle -> upfunc = sldThetaAngleUpfunc;
   pnl_addsubact(sldThetaAngle,fisstFrame);
}

/************************ phi **********************************/

void displayTypeinPhiAngle()
{
   typeinPhiAngle = pnl_mkact(pnl_typein);
   typeinPhiAngle -> x = TYPEIN_PHI_ANGLE_X;
   typeinPhiAngle -> y = TYPEIN_PHI_ANGLE_Y;
   PNL_ACCESS(Typein,typeinPhiAngle,len) = PHI_ANGLE_STRING_LENGTH;
   sprintf(PNL_ACCESS(Typein,typeinPhiAngle,str),PHI_ANGLE_FORMAT,phiAngle);
   typeinPhiAngle -> upfunc = typeinPhiAngleUpfunc;
   pnl_addsubact(typeinPhiAngle,fisstFrame);
}

void displaySldPhiAngle()
{
   sldPhiAngle = pnl_mkact(pnl_slideroid);
   sldPhiAngle -> x = SLD_PHI_ANGLE_X;
   sldPhiAngle -> y = SLD_PHI_ANGLE_Y;
   sldPhiAngle -> minval = PHI_ANGLE_MIN;
   sldPhiAngle -> maxval = PHI_ANGLE_MAX;
   sldPhiAngle -> val = phiAngle;
   sldPhiAngle -> upfunc = sldPhiAngleUpfunc;
   pnl_addsubact(sldPhiAngle,fisstFrame);
}

/************************ dz **********************************/

void displayTypeinDz2D()
{
   typeinDz2D = pnl_mkact(pnl_typein);
   typeinDz2D -> x = TYPEIN_DZ_2D_X;
   typeinDz2D -> y = TYPEIN_DZ_2D_Y;
   PNL_ACCESS(Typein,typeinDz2D,len) = DZ_2D_STRING_LENGTH;
   sprintf(PNL_ACCESS(Typein,typeinDz2D,str),DZ_2D_FORMAT,dz2D);
   typeinDz2D -> upfunc = typeinDz2DUpfunc;
   pnl_addsubact(typeinDz2D,fisstFrame);
}

void displaySldDz2D()
{
   sldDz2D = pnl_mkact(pnl_slideroid);
   sldDz2D -> x = SLD_DZ_2D_X;
   sldDz2D -> y = SLD_DZ_2D_Y;
   sldDz2D -> minval = DZ_2D_MIN;
   sldDz2D -> maxval = DZ_2D_MAX;
   sldDz2D -> val = dz2D;
   sldDz2D -> upfunc = sldDz2DUpfunc;
   pnl_addsubact(sldDz2D,fisstFrame);
}

/************************ awl **********************************/

void displayTypeinWaveLengthOfLight()
{
   typeinWaveLengthOfLight = pnl_mkact(pnl_typein);
   typeinWaveLengthOfLight -> x = TYPEIN_WAVE_LENGTH_OF_LIGHT_X;
   typeinWaveLengthOfLight -> y = TYPEIN_WAVE_LENGTH_OF_LIGHT_Y;
   PNL_ACCESS(Typein,typeinWaveLengthOfLight,len) = WAVE_LENGTH_OF_LIGHT_STRING_LENGTH;
   sprintf(PNL_ACCESS(Typein,typeinWaveLengthOfLight,str),WAVE_LENGTH_OF_LIGHT_FORMAT,wav
eLengthOfLight);
   typeinWaveLengthOfLight -> upfunc = typeinWaveLengthOfLightUpfunc;
   pnl_addsubact(typeinWaveLengthOfLight,fisstFrame);
}

void displaySldWaveLengthOfLight()
{
   sldWaveLengthOfLight = pnl_mkact(pnl_slideroid);
   sldWaveLengthOfLight -> x = SLD_WAVE_LENGTH_OF_LIGHT_X;
   sldWaveLengthOfLight -> y = SLD_WAVE_LENGTH_OF_LIGHT_Y;
   sldWaveLengthOfLight -> minval = WAVE_LENGTH_OF_LIGHT_MIN;
   sldWaveLengthOfLight -> maxval = WAVE_LENGTH_OF_LIGHT_MAX;
   sldWaveLengthOfLight -> val = waveLengthOfLight;
   sldWaveLengthOfLight -> upfunc = sldWaveLengthOfLightUpfunc;
   pnl_addsubact(sldWaveLengthOfLight,fisstFrame);
}

/********************** camera **********************************/

void displayTypeinObjectDistance()
{
   typeinObjectDistance = pnl_mkact(pnl_typein);
   typeinObjectDistance -> x = TYPEIN_OBJECT_DISTANCE_X;
   typeinObjectDistance -> y = TYPEIN_OBJECT_DISTANCE_Y;
   PNL_ACCESS(Typein,typeinObjectDistance,len) = OBJECT_DISTANCE_STRING_LENGTH;
   sprintf(PNL_ACCESS(Typein,typeinObjectDistance,str),OBJECT_DISTANCE_FORMAT,objectDista
nce);
   typeinObjectDistance -> upfunc = typeinObjectDistanceUpfunc;
   pnl_addsubact(typeinObjectDistance,fisstFrame);
}

void displaySldObjectDistance()
{
   sldObjectDistance = pnl_mkact(pnl_slideroid);
   sldObjectDistance -> x = SLD_OBJECT_DISTANCE_X;
   sldObjectDistance -> y = SLD_OBJECT_DISTANCE_Y;
   sldObjectDistance -> minval = OBJECT_DISTANCE_MIN;
   sldObjectDistance -> maxval = OBJECT_DISTANCE_MAX;
   sldObjectDistance -> val = objectDistance;
```

```
   sldObjectDistance -> upfunc = sldObjectDistanceUpfunc;
   pnl_addsubact(sldObjectDistance,fisstFrame);
}

/************************ exposure ********************************/

void displayTypeinExposure()
{
   typeinExposure = pnl_mkact(pnl_typein);
   typeinExposure -> x = TYPEIN_EXPOSURE_X;
   typeinExposure -> y = TYPEIN_EXPOSURE_Y;
   PNL_ACCESS(Typein,typeinExposure,len) = EXPOSURE_STRING_LENGTH;
   sprintf(PNL_ACCESS(Typein,typeinExposure,str),EXPOSURE_FORMAT,exposure);
   typeinExposure -> upfunc = typeinExposureUpfunc;
   pnl_addsubact(typeinExposure,fisstFrame);
}

void displaySldExposure()
{
   sldExposure = pnl_mkact(pnl_slideroid);
   sldExposure -> x = SLD_EXPOSURE_X;
   sldExposure -> y = SLD_EXPOSURE_Y;
   sldExposure -> minval = EXPOSURE_MIN;
   sldExposure -> maxval = EXPOSURE_MAX;
   sldExposure -> val = exposure;
   sldExposure -> upfunc = sldExposureUpfunc;
   pnl_addsubact(sldExposure,fisstFrame);
}

/************************ knifeEdge ********************************/

void displayUpArrowKnifeEdge()
{
   upArrowKnifeEdge = pnl_mkact(pnl_up_arrow_button);
   upArrowKnifeEdge -> x = UPARROW_KNIFE_EDGE_X;
   upArrowKnifeEdge -> y = UPARROW_KNIFE_EDGE_Y;
   upArrowKnifeEdge -> label = VERTICAL_KNIFE_EDGE;
   upArrowKnifeEdge -> labeltype = PNL_LABEL_RIGHT;
   upArrowKnifeEdge -> upfunc = upArrowKnifeEdgeUpfunc;
   pnl_addsubact(upArrowKnifeEdge, fisstFrame);
}

void upArrowKnifeEdgeUpfunc(Actuator* act)
{
   if( flagKnifeEdge = !flagKnifeEdge )
   {
      upArrowKnifeEdge -> label = VERTICAL_KNIFE_EDGE;
      knifeEdge = KNIFE_EDGE_VERTICAL;
   }
   else
   {
      upArrowKnifeEdge -> label = HORIZONTAL_KNIFE_EDGE;
      knifeEdge = KNIFE_EDGE_HORIZONTAL;
   }
   pnl_fixact(upArrowKnifeEdge);
}

/************************ Fringes ********************************/

void typeinFringesUpfunc( Actuator* act )
{
   float tmp = atof(PNL_ACCESS(Typein,typeinFringes,str));
   if( tmp < FRINGES_MIN )
   {
      fringes = FRINGES_MIN;
   }
   else if( tmp > FRINGES_MAX )
   {
      fringes = FRINGES_MAX;
   }
   else
   {
      fringes = tmp;
   }
   sprintf(PNL_ACCESS(Typein,typeinFringes,str),FRINGES_FORMAT,fringes);
}

void upArrowFringesUpfunc( Actuator* act )
{
   if( fringes < FRINGES_MAX ) {
      ++fringes;
      sprintf(PNL_ACCESS(Typein,typeinFringes,str),FRINGES_FORMAT,fringes);
      pnl_fixact(typeinFringes);
   }
}

void downArrowFringesUpfunc( Actuator* act )
{
   if( fringes > FRINGES_MIN ) {
      --fringes;
      sprintf(PNL_ACCESS(Typein,typeinFringes,str),FRINGES_FORMAT,fringes);
      pnl_fixact(typeinFringes);
   }
}

void resetFringesUpfunc( Actuator* act )

{
   fringes = INI_FRINGES;
   sprintf(PNL_ACCESS(Typein,typeinFringes,str),FRINGES_FORMAT,fringes);
   pnl_fixact(typeinFringes);
}

/************************ fringeType ********************************/

void displayUpArrowFringeType()
{
   upArrowFringeType = pnl_mkact(pnl_up_arrow_button);
   upArrowFringeType -> x = UPARROW_FRINGE_TYPE_X;
   upArrowFringeType -> y = UPARROW_FRINGE_TYPE_Y;
   upArrowFringeType -> label = VERTICAL_FRINGE_TYPE;
   upArrowFringeType -> labeltype = PNL_LABEL_RIGHT;
   upArrowFringeType -> upfunc = upArrowFringeTypeUpfunc;
   pnl_addsubact(upArrowFringeType, fisstFrame);
}

void upArrowFringeTypeUpfunc(Actuator* act)
{
   if( fringeType == FRINGE_TYPE_INFINITE )
   {
      upArrowFringeType -> label = VERTICAL_FRINGE_TYPE;
      fringeType = FRINGE_TYPE_VERTICAL;
   }
   else if( fringeType == FRINGE_TYPE_VERTICAL )
   {
      upArrowFringeType -> label = HORIZONTAL_FRINGE_TYPE;
      fringeType = FRINGE_TYPE_HORIZONTAL;
   }
```



```
   else if( fringeType == FRINGE_TYPE_HORIZONTAL )
   {
      upArrowFringeType -> label = INFINITE_FRINGE_TYPE;
      fringeType = FRINGE_TYPE_INFINITE;
   }

   pnl_fixact(upArrowFringeType);
}
```

```
/* Supporting variables for both fisst and fisstSupport */

extern void imagef_(float*, float*);

int iniIntegrateParams();
void inixyzRho( float*, float* );
extern Grid_Surface* grid_fisst;
extern int lock_cur_object();
extern int unlock_cur_object();
float* xyzData;
float* rhoData;
float* xyzFisst;
float* rhoFisst;
int drawimage_( int *xdim, int *ydim, float *pix );
extern tGraphicObject  cur_object;
extern int locked;
extern void dump_state();
#define NDIM    5

void displayTypeinNumberOfPoints();
void displaySldNumberOfPoints();
void displayTypeinComputationalRho();
void displaySldComputationalRho();
void displayTypeinExperimentalRho();
void displaySldExperimentalRho();
void displayTypeinNondimensionalizedLength();
void displaySldNondimensionalizedLength();
void displayTypeinNumberOfPasses();
void displaySldNumberOfPasses();
void displayTypeinPsiAngle();
void displaySldPsiAngle();
void displayTypeinThetaAngle();
void displaySldThetaAngle();
void displayTypeinPhiAngle();
void displaySldPhiAngle();
void displayTypeinDz2D();
void displaySldDz2D();
void displayTypeinWaveLengthOfLight();
void displaySldWaveLengthOfLight();
void displayTypeinObjectDistance();
void displaySldObjectDistance();
void displayTypeinExposure();
void displaySldExposure();
void displayUpArrowKnifeEdge();
void displayUpArrowFringeType();

Actuator* fisstFrame;
#define FISSTFRAMEX             12.2
#define FISSTFRAMEY             -18.05

#define OFFSET_Y_SLD_TYPEIN     0.5

Actuator* butIntegrate;
#define BUTINTEGRATELABEL       "  Integrate  "
#define BUTINTEGRATEX           3.0
#define BUTINTEGRATEY           20.2
void butIntegrateUpfunc( Actuator* act );

Actuator* butInterferogram;
#define BUTINTERFEROGRAMLABEL   "Interferogram"
#define BUTINTERFEROGRAMX       3.0
#define BUTINTERFEROGRAMY       10.0
void butInterferogramUpfunc( Actuator* act );

Actuator* butShadowGraph;
#define BUTSHADOWGRAPHLABEL     " ShadowGraph "
#define BUTSHADOWGRAPHX         3.0
#define BUTSHADOWGRAPHY         6.0
void butShadowGraphUpfunc( Actuator* act );

Actuator* butSchlieren;
#define BUTSCHLIERENLABEL       "  Schlieren  "
#define BUTSCHLIERENX           3.0
#define BUTSCHLIERENY           3.0
void butSchlierenUpfunc( Actuator* act );

/********************** npts ***********************************/

#define BUT_NUMBER_OF_POINTS_LABEL      "Rays"
#define BUT_NUMBER_OF_POINTS_X          1.0
#define BUT_NUMBER_OF_POINTS_Y          12.0

#define TYPEIN_NUMBER_OF_POINTS_X       BUT_NUMBER_OF_POINTS_X - 0.3
#define TYPEIN_NUMBER_OF_POINTS_Y       BUT_NUMBER_OF_POINTS_Y + OFFSET_Y_SLD_TYPEIN

#define SLD_NUMBER_OF_POINTS_X          BUT_NUMBER_OF_POINTS_X
#define SLD_NUMBER_OF_POINTS_Y          BUT_NUMBER_OF_POINTS_Y + OFFSET_Y_SLD_TYPEIN

#define INI_NUMBER_OF_POINTS            20000
#define NUMBER_OF_POINTS_MIN            10000
#define NUMBER_OF_POINTS_MAX            600000

#define NUMBER_OF_POINTS_FORMAT         "%9d"
#define NUMBER_OF_POINTS_STRING_LENGTH  9

void butNumberOfPointsUpfunc( Actuator* act );
void typeinNumberOfPointsUpfunc( Actuator* act );
void sldNumberOfPointsUpfunc( Actuator* act );

Actuator* butNumberOfPoints;
Actuator* typeinNumberOfPoints;
Actuator* sldNumberOfPoints;

int numberOfPoints          =       INI_NUMBER_OF_POINTS;
short int flagNumberOfPoints =      TRUE;

/******************** rho0 ***********************************/

#define BUT_COMPUTATIONAL_RHO_LABEL     "Computational rho"
#define BUT_COMPUTATIONAL_RHO_X         0.0
#define BUT_COMPUTATIONAL_RHO_Y         18.0

#define TYPEIN_COMPUTATIONAL_RHO_X      BUT_COMPUTATIONAL_RHO_X + 1.5
#define TYPEIN_COMPUTATIONAL_RHO_Y      BUT_COMPUTATIONAL_RHO_Y + OFFSET_Y_SLD_TYPEIN

#define SLD_COMPUTATIONAL_RHO_X         BUT_COMPUTATIONAL_RHO_X + 1.5
#define SLD_COMPUTATIONAL_RHO_Y         BUT_COMPUTATIONAL_RHO_Y + OFFSET_Y_SLD_TYPEIN

#define INI_COMPUTATIONAL_RHO           1.0
#define COMPUTATIONAL_RHO_MIN           0.000001
#define COMPUTATIONAL_RHO_MAX           100.0

#define COMPUTATIONAL_RHO_FORMAT        "%7.4f"
#define COMPUTATIONAL_RHO_STRING_LENGTH 7

void butComputationalRhoUpfunc( Actuator* act );
void typeinComputationalRhoUpfunc( Actuator* act );
void sldComputationalRhoUpfunc( Actuator* act );
```



```
Actuator* butComputationalRho;
Actuator* typeinComputationalRho;
Actuator* sldComputationalRho;

float computationalRho          =       INI_COMPUTATIONAL_RHO;
short int flagComputationalRho =        TRUE;

/************************ rhoinf ***********************************/

#define BUT_EXPERIMENTAL_RHO_LABEL      "Experimental rho"
#define BUT_EXPERIMENTAL_RHO_X          5.0
#define BUT_EXPERIMENTAL_RHO_Y          18.0

#define TYPEIN_EXPERIMENTAL_RHO_X       BUT_EXPERIMENTAL_RHO_X + 1.5
#define TYPEIN_EXPERIMENTAL_RHO_Y       BUT_EXPERIMENTAL_RHO_Y + OFFSET_Y_SLD_TYPEIN

#define SLD_EXPERIMENTAL_RHO_X          BUT_EXPERIMENTAL_RHO_X + 1.5
#define SLD_EXPERIMENTAL_RHO_Y          BUT_EXPERIMENTAL_RHO_Y + OFFSET_Y_SLD_TYPEIN

#define INI_EXPERIMENTAL_RHO            1.0
#define EXPERIMENTAL_RHO_MIN            0.000001
#define EXPERIMENTAL_RHO_MAX            100.0

#define EXPERIMENTAL_RHO_FORMAT         "%7.4f"
#define EXPERIMENTAL_RHO_STRING_LENGTH  7

void butExperimentalRhoUpfunc( Actuator* act );
void typeinExperimentalRhoUpfunc( Actuator* act );
void sldExperimentalRhoUpfunc( Actuator* act );

Actuator* butExperimentalRho;
Actuator* typeinExperimentalRho;
Actuator* sldExperimentalRho;

float experimentalRho           =       INI_EXPERIMENTAL_RHO;
short int flagExperimentalRho =         TRUE;

/************************ al ***********************************/

#define BUT_NONDIMENSIONALIZED_LENGTH_LABEL     "Scaling"
#define BUT_NONDIMENSIONALIZED_LENGTH_X         0.85
#define BUT_NONDIMENSIONALIZED_LENGTH_Y         14.0

#define TYPEIN_NONDIMENSIONALIZED_LENGTH_X      BUT_NONDIMENSIONALIZED_LENGTH_X + 0.15
#define TYPEIN_NONDIMENSIONALIZED_LENGTH_Y      BUT_NONDIMENSIONALIZED_LENGTH_Y + OFFSET
_Y_SLD_TYPEIN

#define SLD_NONDIMENSIONALIZED_LENGTH_X         BUT_NONDIMENSIONALIZED_LENGTH_X + 0.15
#define SLD_NONDIMENSIONALIZED_LENGTH_Y         BUT_NONDIMENSIONALIZED_LENGTH_Y + OFFSET
_Y_SLD_TYPEIN

#define INI_NONDIMENSIONALIZED_LENGTH           1.0
#define NONDIMENSIONALIZED_LENGTH_MIN           0.000001
#define NONDIMENSIONALIZED_LENGTH_MAX           1000.0

#define NONDIMENSIONALIZED_LENGTH_FORMAT        "%7.4f"
#define NONDIMENSIONALIZED_LENGTH_STRING_LENGTH 7

void butNondimensionalizedLengthUpfunc( Actuator* act );
void typeinNondimensionalizedLengthUpfunc( Actuator* act );
void sldNondimensionalizedLengthUpfunc( Actuator* act );

Actuator* butNondimensionalizedLength;
Actuator* typeinNondimensionalizedLength;
Actuator* sldNondimensionalizedLength;

float nondimensionalizedLength          =       INI_NONDIMENSIONALIZED_LENGTH;
short int flagNondimensionalizedLength =        TRUE;

/************************ pass ***********************************/

#define BUT_NUMBER_OF_PASSES_LABEL      "# of Passes"
#define BUT_NUMBER_OF_PASSES_X          3.3
#define BUT_NUMBER_OF_PASSES_Y          12.0

#define TYPEIN_NUMBER_OF_PASSES_X       BUT_NUMBER_OF_PASSES_X + 0.7
#define TYPEIN_NUMBER_OF_PASSES_Y       BUT_NUMBER_OF_PASSES_Y + OFFSET_Y_SLD_TYPEIN

#define SLD_NUMBER_OF_PASSES_X          BUT_NUMBER_OF_PASSES_X +  0.7
#define SLD_NUMBER_OF_PASSES_Y          BUT_NUMBER_OF_PASSES_Y + OFFSET_Y_SLD_TYPEIN

#define INI_NUMBER_OF_PASSES            2.0
#define NUMBER_OF_PASSES_MIN            1.0
#define NUMBER_OF_PASSES_MAX            10000.0

#define NUMBER_OF_PASSES_FORMAT         "%7.4f"
#define NUMBER_OF_PASSES_STRING_LENGTH  7

void butNumberOfPassesUpfunc( Actuator* act );
void typeinNumberOfPassesUpfunc( Actuator* act );
void sldNumberOfPassesUpfunc( Actuator* act );

Actuator* butNumberOfPasses;
Actuator* typeinNumberOfPasses;
Actuator* sldNumberOfPasses;

float numberOfPasses            =       INI_NUMBER_OF_PASSES;
short int flagNumberOfPasses =          TRUE;

/************************ psi ***********************************/

#define BUT_PSI_ANGLE_LABEL     "psi"
#define BUT_PSI_ANGLE_X         1.0
#define BUT_PSI_ANGLE_Y         16.0

#define TYPEIN_PSI_ANGLE_X      BUT_PSI_ANGLE_X
#define TYPEIN_PSI_ANGLE_Y      BUT_PSI_ANGLE_Y + OFFSET_Y_SLD_TYPEIN

#define SLD_PSI_ANGLE_X         BUT_PSI_ANGLE_X
#define SLD_PSI_ANGLE_Y         BUT_PSI_ANGLE_Y + OFFSET_Y_SLD_TYPEIN

#define INI_PSI_ANGLE           0.0
#define PSI_ANGLE_MIN           -360.0
#define PSI_ANGLE_MAX           360.0

#define PSI_ANGLE_FORMAT        "%7.2f"
#define PSI_ANGLE_STRING_LENGTH 7

void butPsiAngleUpfunc( Actuator* act );
void typeinPsiAngleUpfunc( Actuator* act );
void sldPsiAngleUpfunc( Actuator* act );

Actuator* butPsiAngle;
Actuator* typeinPsiAngle;
Actuator* sldPsiAngle;

float psiAngle          =       INI_PSI_ANGLE;
```

```
short int flagPsiAngle =        TRUE;

/************************* theta *********************************/

#define BUT_THETA_ANGLE_LABEL   "theta"
#define BUT_THETA_ANGLE_X       4.0
#define BUT_THETA_ANGLE_Y       16.0

#define TYPEIN_THETA_ANGLE_X    BUT_THETA_ANGLE_X
#define TYPEIN_THETA_ANGLE_Y    BUT_THETA_ANGLE_Y + OFFSET_Y_SLD_TYPEIN

#define SLD_THETA_ANGLE_X       BUT_THETA_ANGLE_X
#define SLD_THETA_ANGLE_Y       BUT_THETA_ANGLE_Y + OFFSET_Y_SLD_TYPEIN

#define INI_THETA_ANGLE         90.0
#define THETA_ANGLE_MIN         -360.0
#define THETA_ANGLE_MAX         360.0

#define THETA_ANGLE_FORMAT      "%7.2f"
#define THETA_ANGLE_STRING_LENGTH 7

void butThetaAngleUpfunc( Actuator* act );
void typeinThetaAngleUpfunc( Actuator* act );
void sldThetaAngleUpfunc( Actuator* act );

Actuator* butThetaAngle;
Actuator* typeinThetaAngle;
Actuator* sldThetaAngle;

float thetaAngle        =       INI_THETA_ANGLE;
short int flagThetaAngle =      TRUE;

/************************* phi *********************************/

#define BUT_PHI_ANGLE_LABEL     "phi"
#define BUT_PHI_ANGLE_X         7.0
#define BUT_PHI_ANGLE_Y         16.0

#define TYPEIN_PHI_ANGLE_X      BUT_PHI_ANGLE_X
#define TYPEIN_PHI_ANGLE_Y      BUT_PHI_ANGLE_Y + OFFSET_Y_SLD_TYPEIN

#define SLD_PHI_ANGLE_X         BUT_PHI_ANGLE_X
#define SLD_PHI_ANGLE_Y         BUT_PHI_ANGLE_Y + OFFSET_Y_SLD_TYPEIN

#define INI_PHI_ANGLE           0.0
#define PHI_ANGLE_MIN           -360.0
#define PHI_ANGLE_MAX           360.0

#define PHI_ANGLE_FORMAT        "%7.2f"
#define PHI_ANGLE_STRING_LENGTH 7

void butPhiAngleUpfunc( Actuator* act );
void typeinPhiAngleUpfunc( Actuator* act );
void sldPhiAngleUpfunc( Actuator* act );

Actuator* butPhiAngle;
Actuator* typeinPhiAngle;
Actuator* sldPhiAngle;

float phiAngle      =           INI_PHI_ANGLE;
short int flagPhiAngle =        TRUE;

/************************* dz *********************************/

#define BUT_DZ_2D_LABEL         "Z-Depth"
#define BUT_DZ_2D_X             3.9
#define BUT_DZ_2D_Y             14.0

#define TYPEIN_DZ_2D_X          BUT_DZ_2D_X + 0.15
#define TYPEIN_DZ_2D_Y          BUT_DZ_2D_Y + OFFSET_Y_SLD_TYPEIN

#define SLD_DZ_2D_X             BUT_DZ_2D_X + 0.15
#define SLD_DZ_2D_Y             BUT_DZ_2D_Y + OFFSET_Y_SLD_TYPEIN

#define INI_DZ_2D               1.0
#define DZ_2D_MIN               0.000001
#define DZ_2D_MAX               10000.0

#define DZ_2D_FORMAT            "%7.3f"
#define DZ_2D_STRING_LENGTH     7

void butDz2DUpfunc( Actuator* act );
void typeinDz2DUpfunc( Actuator* act );
void sldDz2DUpfunc( Actuator* act );

Actuator* butDz2D;
Actuator* typeinDz2D;
Actuator* sldDz2D;

float dz2D          =           INI_DZ_2D;
short int flagDz2D =            TRUE;

/************************* nphi *********************************/

#define TYPEIN_NUMBER_OF_PLANES_LABEL   "Rotations"
#define TYPEIN_NUMBER_OF_PLANES_X       7.0
#define TYPEIN_NUMBER_OF_PLANES_Y       14.5
#define NUMBER_OF_PLANES_FORMAT         "%7d"
#define NUMBER_OF_PLANES_STRING_LENGTH  7
#define OFFSET_TYPEIN_ARROW             0.65

#define UPARROW_NUMBER_OF_PLANES_X      TYPEIN_NUMBER_OF_PLANES_X
#define UPARROW_NUMBER_OF_PLANES_Y      TYPEIN_NUMBER_OF_PLANES_Y + OFFSET_TYPEIN_ARROW

#define DOWNARROW_NUMBER_OF_PLANES_X    TYPEIN_NUMBER_OF_PLANES_X + 1.305
#define DOWNARROW_NUMBER_OF_PLANES_Y    TYPEIN_NUMBER_OF_PLANES_Y + OFFSET_TYPEIN_ARROW

#define RESET_NUMBER_OF_PLANES_X        TYPEIN_NUMBER_OF_PLANES_X + 0.65
#define RESET_NUMBER_OF_PLANES_Y        TYPEIN_NUMBER_OF_PLANES_Y + OFFSET_TYPEIN_ARROW

#define INI_NUMBER_OF_PLANES            1
#define NUMBER_OF_PLANES_MIN            1
#define NUMBER_OF_PLANES_MAX            10000

void typeinNumberOfPlanesUpfunc( Actuator* act );
void upArrowNumberOfPlanesUpfunc( Actuator* act );
void downArrowNumberOfPlanesUpfunc( Actuator* act );
void resetNumberOfPlanesUpfunc( Actuator* act );

Actuator* typeinNumberOfPlanes;
Actuator* upArrowNumberOfPlanes;
Actuator* downArrowNumberOfPlanes;
Actuator* resetNumberOfPlanes;

int numberOfPlanes          =       INI_NUMBER_OF_PLANES;
short int flagNumberOfPlanes =      TRUE;

/************************* awl *********************************/
```



```
#define BUT_WAVE_LENGTH_OF_LIGHT_LABEL  "Wavelength"
#define BUT_WAVE_LENGTH_OF_LIGHT_X      2.0
#define BUT_WAVE_LENGTH_OF_LIGHT_Y      6.9

#define TYPEIN_WAVE_LENGTH_OF_LIGHT_X   BUT_WAVE_LENGTH_OF_LIGHT_X + 0.3
#define TYPEIN_WAVE_LENGTH_OF_LIGHT_Y   BUT_WAVE_LENGTH_OF_LIGHT_Y + OFFSET_Y_SLD_TYPEIN

#define SLD_WAVE_LENGTH_OF_LIGHT_X      BUT_WAVE_LENGTH_OF_LIGHT_X + 0.6
#define SLD_WAVE_LENGTH_OF_LIGHT_Y      BUT_WAVE_LENGTH_OF_LIGHT_Y + OFFSET_Y_SLD_TYPEIN

#define INI_WAVE_LENGTH_OF_LIGHT        6.38E-07
#define WAVE_LENGTH_OF_LIGHT_MIN        1.0E-15
#define WAVE_LENGTH_OF_LIGHT_MAX        1.0E-03

#define WAVE_LENGTH_OF_LIGHT_FORMAT             "%6.3e"
#define WAVE_LENGTH_OF_LIGHT_STRING_LENGTH      9

void butWaveLengthOfLightUpfunc( Actuator* act );
void typeinWaveLengthOfLightUpfunc( Actuator* act );
void sldWaveLengthOfLightUpfunc( Actuator* act );

Actuator* butWaveLengthOfLight;
Actuator* typeinWaveLengthOfLight;
Actuator* sldWaveLengthOfLight;

float waveLengthOfLight         =       INI_WAVE_LENGTH_OF_LIGHT;
short int flagWaveLengthOfLight =       TRUE;

extern void fringes_(float* cons);

/********************* camera *********************************/

#define BUT_OBJECT_DISTANCE_LABEL       "Object Distance"
#define BUT_OBJECT_DISTANCE_X           2.7
#define BUT_OBJECT_DISTANCE_Y           3.9

#define TYPEIN_OBJECT_DISTANCE_X        BUT_OBJECT_DISTANCE_X + 1.3
#define TYPEIN_OBJECT_DISTANCE_Y        BUT_OBJECT_DISTANCE_Y + OFFSET_Y_SLD_TYPEIN

#define SLD_OBJECT_DISTANCE_X           BUT_OBJECT_DISTANCE_X + 1.3
#define SLD_OBJECT_DISTANCE_Y           BUT_OBJECT_DISTANCE_Y + OFFSET_Y_SLD_TYPEIN

#define INI_OBJECT_DISTANCE             20.0
#define OBJECT_DISTANCE_MIN             0.000001
#define OBJECT_DISTANCE_MAX             100000.0

#define OBJECT_DISTANCE_FORMAT          "%7.3f"
#define OBJECT_DISTANCE_STRING_LENGTH   7

extern void shadowg_(float* objectDistance);

void butObjectDistanceUpfunc( Actuator* act );
void typeinObjectDistanceUpfunc( Actuator* act );
void sldObjectDistanceUpfunc( Actuator* act );

Actuator* butObjectDistance;
Actuator* typeinObjectDistance;
Actuator* sldObjectDistance;

float objectDistance        =       INI_OBJECT_DISTANCE;
short int flagObjectDistance =      TRUE;

/********************* gstrch *********************************/

#define BUT_EXPOSURE_LABEL      "Exposure"
#define BUT_EXPOSURE_X          6.7
#define BUT_EXPOSURE_Y          12.0

#define TYPEIN_EXPOSURE_X       BUT_EXPOSURE_X + 0.3
#define TYPEIN_EXPOSURE_Y       BUT_EXPOSURE_Y + OFFSET_Y_SLD_TYPEIN

#define SLD_EXPOSURE_X          BUT_EXPOSURE_X + 0.3
#define SLD_EXPOSURE_Y          BUT_EXPOSURE_Y + OFFSET_Y_SLD_TYPEIN

#define INI_EXPOSURE            1.0
#define EXPOSURE_MIN            0.000001
#define EXPOSURE_MAX            100000.0

#define EXPOSURE_FORMAT         "%7.2f"
#define EXPOSURE_STRING_LENGTH  7

void butExposureUpfunc( Actuator* act );
void typeinExposureUpfunc( Actuator* act );
void sldExposureUpfunc( Actuator* act );

Actuator* butExposure;
Actuator* typeinExposure;
Actuator* sldExposure;

float exposure      =           INI_EXPOSURE;
short int flagExposure =        TRUE;

/********************* surface *********************************/

#define BUT_SURFACE_IJK_LABEL       "Surface"
#define BUT_SURFACE_IJK_X           4.5
#define BUT_SURFACE_IJK_Y           11.0

#define BUT_RESET_SURFACE_IJK_LABEL "Reset"
#define BUT_RESET_SURFACE_IJK_X     3.0
#define BUT_RESET_SURFACE_IJK_Y     11.0

void butSurfaceIJKUpfunc();
void butResetSurfaceIJKUpfunc();

Actuator* butSurfaceIJK;
Actuator* butResetSurfaceIJK;

int surfaceIJK = 0;

/************************* knife *********************************/

#define UPARROW_KNIFE_EDGE_X    2.0
#define UPARROW_KNIFE_EDGE_Y    2.5

#define KNIFE_EDGE_HORIZONTAL   'h'
#define KNIFE_EDGE_VERTICAL     'v'
#define INI_KNIFE_EDGE          KNIFE_EDGE_VERTICAL
#define VERTICAL_KNIFE_EDGE     " Vertical Knife Edge"
#define HORIZONTAL_KNIFE_EDGE   " Horizontal Knife Edge"

void upArrowKnifeEdgeUpfunc( Actuator* act );
extern void schlieren_(char* knifeEdge);

Actuator* upArrowKnifeEdge;
```

```
char knifeEdge          =       INI_KNIFE_EDGE;
short int flagKnifeEdge =       TRUE;

/***************************** vert, horiz ***************************/

#define TYPEIN_FRINGES_LABEL            "Fringes"
#define TYPEIN_FRINGES_X                6.0
#define TYPEIN_FRINGES_Y                7.4
#define FRINGES_FORMAT                  "%7d"
#define FRINGES_STRING_LENGTH            7

#define UPARROW_FRINGES_X       TYPEIN_FRINGES_X
#define UPARROW_FRINGES_Y       TYPEIN_FRINGES_Y + OFFSET_TYPEIN_ARROW

#define DOWNARROW_FRINGES_X     TYPEIN_FRINGES_X + 1.305
#define DOWNARROW_FRINGES_Y     TYPEIN_FRINGES_Y + OFFSET_TYPEIN_ARROW

#define RESET_FRINGES_X         TYPEIN_FRINGES_X + 0.65
#define RESET_FRINGES_Y         TYPEIN_FRINGES_Y + OFFSET_TYPEIN_ARROW

#define INI_FRINGES             1
#define FRINGES_MIN             1
#define FRINGES_MAX             10000

void typeinFringesUpfunc( Actuator* act );
void upArrowFringesUpfunc( Actuator* act );
void downArrowFringesUpfunc( Actuator* act );
void resetFringesUpfunc( Actuator* act );

Actuator* typeinFringes;
Actuator* upArrowFringes;
Actuator* downArrowFringes;
Actuator* resetFringes;

int fringes             =       INI_FRINGES;
short int flagFringes   =       TRUE;

/************************** type *********************************/

#define UPARROW_FRINGE_TYPE_X   2.0
#define UPARROW_FRINGE_TYPE_Y   9.5

#define INI_FRINGE_TYPE         'h'
#define FRINGE_TYPE_HORIZONTAL  INI_FRINGE_TYPE
#define FRINGE_TYPE_VERTICAL    'v'
#define FRINGE_TYPE_INFINITE    'I'
#define INFINITE_FRINGE_TYPE    " Infinite Fringe"
#define VERTICAL_FRINGE_TYPE    " Finite Vertical Fringe"
#define HORIZONTAL_FRINGE_TYPE  " Finite Horizontal Fringe"

void upArrowFringeTypeUpfunc( Actuator* act );

Actuator* upArrowFringeType;

char fringeType         =       INI_FRINGE_TYPE;

/*****************************************************************/

struct
{
   int nsub[10];
   int jsub1[10][10];
   int jsub2[10][10];
   int ksub1[10][10];
   int ksub2[10][10];
   int lsub1[10][10];
   int lsub2[10][10];
} data_;

struct
{
   int jdim[10];
   int kdim[10];
   int ldim[10];
} subjkl_;


struct
{
   int ngrid;
   int npts;
   float rho0;
   float rhoinf;
   float al;
   float pss;
   float psi;
   float theta;
   float phi;
   float dr;
   int nphi;
   float gstrch;
   float strch;
   char ctype;
   char mulsin;
} integ_;

struct
{
   float camera;
} shadow_;

struct
{
   float awl;
   float cons;
   float vert;
   float horiz;
   char type;
} interf_;

struct
{
   int ixmax;
   int iymax;
   int mod[600001];
   float p[4][600001];
} image_;

#define SURFACE_SETS    10

struct
{
   int nseg;
   int iminset[SURFACE_SETS];
   int imaxset[SURFACE_SETS];
   int jminset[SURFACE_SETS];
   int jmaxset[SURFACE_SETS];
```



```
   int kminset[SURFACE_SETS];
   int kmaxset[SURFACE_SETS];
} surfset_;


float *xyzData;
float *rhoData;

unsigned long int ijkSize     = 0;
unsigned long int newijkSize  = 0;
unsigned long int xyzByteSize = 0;
unsigned long int rhoByteSize = 0;

int callIntegrate = FALSE;


/*************************************************************/

#define STRING_12_0 {'\0','\0','\0','\0','\0','\0','\0','\0','\0','\0','\0','\0'}
#define MAX_NUMBER_CHAR         12
#define INI_GRID_FILENAME STRING_12_0
char gridFilename[MAX_NUMBER_CHAR] = INI_GRID_FILENAME;   /* qname */
#define INI_Q_FILENAME STRING_12_0
char qFilename[MAX_NUMBER_CHAR] = INI_Q_FILENAME;       /* qname */

#define INI_GRID_TYPE 3
char gridType = INI_GRID_TYPE;   /* ctype */

#define INI_MULTIPLE_GRIDS 's'
char multipleGrids = INI_MULTIPLE_GRIDS;   /* mulsin */

/*************************************************************/

#define INI_IMAGE 'i'
char generateImage = INI_IMAGE;   /* image */

/*************************************************************/

int imageDrawn = FALSE;
int image_gid = NULL;
char* image_title;
char* image_shadowgraph = "Shadowgraph";
char* image_schlieren = "Schlieren";
char* image_interferogram = "Interferogram";

/*************************************************************/
```

```
#ifndef FUNCS_H
#define FUNCS_H

/*++ funcs.h
 *
 * PURPOSE :
 *
 *              File contains prototyped declarations of public (non-static)
 *              functions of this module.
 *
 * I/O:
 *
 *              None.
 *
 * STANDARDS VIOLATIONS:
 *
 *              None.
 *
 * AUTHORS :
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY :
 *
 *              7/91
 *
 * NOTES :
 *
 --*/

#define MODULE_NAME     "fisst"
#define OBJECT_NAME     "surfer$"

/* panels.c */

extern void init_panels( int panel_x, int panel_y );
extern int get_looping( void );
extern void update_looping( void );
extern void update_minmax( void );

extern void wait_until_object_is_redrawn( void );
extern int redraw_object_when_unlocked;
extern int call_update_minmax_after_drawing;

#endif /* FUNCS_H */
```



```
/*++ main.c
 *
 * PURPOSE :
 *
 *              File contains the function main() for the fisst program
 *              (a FAST module).
 *
 * I/O:
 *
 *              None.
 *
 * STANDARDS VIOLATIONS:
 *
 *              None.
 *
 * AUTHORS :
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY :
 *
 *              4/89
 *
 * NOTE :
 *
 *      Functions declared in this file :
 *
 *      int     main()
 *
 *      External Funtions called by functions within this file :
 *
 *      extern int      get_no_redraw()                 libpanu
 *      extern int      get_quit_module()               libmodule
 *      extern void     process_hub_command()           libmodule
 *
 *      extern void     init_module()                   file init.c
 *      extern void     exit_module()                   file init.c
 *      extern int      get_looping()                   file panels.c
 *      extern void     update_looping()                file panels.c
 *      extern void     update_minmax()                 file panels.c
 *
 *      External variables used by functions within this file :
 *
 *
 --*/

/*------------------------------ INCLUDES ----------------------------------*/

#include        <stdio.h>       /* for NULL etc.                        */
#include        <Fgl.h>         /* for required by panel.h              */
#include        <Fast_panel.h>  /* for pnl_napimm & pnl_block           */
#include        <panel_utils.h> /* for get_no_redraw()                  */
#include        <Module.h>      /* for command passing                  */
#include        "funcs.h"       /* for function declarations            */

/*------------------------------ FUNCTIONS ---------------------------------*/
```

```
/*++ int main( int argc, char* argv[] )
 *
 * PURPOSE :
 *
 *              Invokes the fisst program and handles the main event loop.
 *
 * AUTHORS :
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY :
 *
 *              4/89
 *
 * INPUT PARAMETERS :
 *
 *              int     argc;           argument count from the hub
 *              char    *argv[];        argument value strings from the hub
 *
 * OUTPUT PARAMETERS :
 *
 *              None
 *
 * FUNCTION RETURN :
 *
 *              int 0
 *
 * GLOBAL VARIABLES USED :
 *
 *              None
 *
 * FILES USED :
 *
 *              None
 *
 * NOTES :
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              extern int      get_no_redraw()         libpanu
 *              extern int      get_quit_module()       libmodule
 *              extern void     process_hub_command()   libmodule
 *
 *              extern void     init_module()           file init.c
 *              extern void     exit_module()           file init.c
 *              extern int      get_looping()           file panels.c
 *              extern void     update_looping()        file panels.c
 *              extern void     update_minmax()         file panels.c
 *
 --*/

/*---------------------------------- main ------------------------------------*/
```

```
int main( int argc, char* argv[] )
{
        /* set initial panel nap time */

        pnl_naptime = HZ / 5;

        /*
         * Initialize the module:
         * read arguments from the hub and establish a connection then
         * build the panels
         */

        init_module( argc, argv );

        /* Initialize the panels */

        init_panels( get_module_panel_x(), get_module_panel_y() );

        /* Tell the hub we are done initializing */

        send_hub_command( "DONE_INITIALIZING" );

        /* Enter main event loop */

        while ( get_quit_module() == 0 )
        {
                if ( pnl_dopanel() || get_looping() )
                {
                        if ( ! get_no_redraw() )
                        {
                                update_looping();
                                update_minmax();
                        }
                }
                else if ( call_update_minmax_after_drawing )
                {
                        wait_until_object_is_redrawn();
                        redraw_object_when_unlocked = 0;
                        update_looping();
                        update_minmax();
                        redraw_object_when_unlocked = 1;
                        call_update_minmax_after_drawing = 0;
                }
                else pnl_naptime = HZ / 5; /* reset panel nap time */

                process_hub_command();

                /* process the next command */

                command_buffer_process( MODULE );
        }

        /* exit and clean up */

        exit_module();
        return 0;
}

/*---------------------------- END OF main ---------------------------------*/
```

```
/*------------------------ END OF FILE main.c ------------------------------*/
```



```
/*++ panels.c
 *
 * PURPOSE:
 *
 *              File contains the functions used for creating the panels.
 *
 *
 * I/O:
 *
 *              None.
 *
 * STANDARDS VIOLATIONS:
 *
 *              None.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              6/89
 *              7/91    prototyped functions
 *
 * NOTE :
 *
 *      Functions declared in this file :
 *
 *              void    init_panels()
 *              Panel*  main_panel();
 *              int     get_looping();
 *
 *      Support functions called by the above panel routines:
 *
 *      External Funtions called by functions within this file :
 *
 *      extern float    Norm()                          Fgl
 *      extern float    Denorm()                        Fgl
 *      extern int      view_start_write_lock()         libviewI
 *      extern int      view_end_write_unlock()         libviewI
 *      extern int      view_single_buffer()            libviewI
 *      extern int      view_set_draw_mode()            libviewI
 *      extern int      shm_get_local_address()         libviewI
 *      extern int      shm_destroy_local_address()     libviewI
 *      extern int      init_fast_cmap()                libcmap
 *      extern int      module_menu()                   libpanu
 *      extern void     draw_palettes()                 libpanu
 *      extern Panel*   color_panel()                   libpanu
 *      extern void     mark_menu()                     libpanu
 *      extern void     close_parent_panel()            libpanu
 *      extern void     clear_typeout()                 libpanu
 *      extern int      set_selection_num()             libpanu
 *      extern void     fix_color_panel()               libpanu
 *      extern void     set_typein_ival()               libpanu
 *      extern void     set_typein_fval()               libpanu
 *      extern float    get_typein_fval()               libpanu
 *      extern int      get_data_minmax()               libfldpan
 *      extern int      get_data_list_minmax()          libfldpan
 *      extern int      print_field_node_info();        libfldpan
 *      extern int      set_fld_data_selection()        libfldpan
 *      extern void     update_fld_data_panel()         libfldpan
 *      extern Panel*   fld_data_panel()                libfldpan
 *      extern void     Error()                         libmodule
 *      extern void     Warning()                       libmodule
 *
 *      Panel Library functions
 *
 --*/

/*-------------------------------- INCLUDES ---------------------------------*/

#include        <stdio.h>       /* for NULL etc.                            */
#include        <stdlib.h>      /* for atoi()                               */
#include        <string.h>      /* for string stuff                         */
#include        <ctype.h>       /* for isdigit() etc.                       */
#include        <math.h>        /* for atof()                               */
#include        <fld_pan.h>     /* for GRID_SCALAR_VECTOR_TYPEOUT           */
#include        <panel_utils.h> /* for typedefs and FLOAT_STRING_FORMAT     */
#include        <fast_cmap.h>   /* for init_fast_cmap()                     */
#include        <Object.h>      /* for OBJECT_NAME_LENGTH, etc.             */
#include        <grid_surface.h>/* for Grid_Surface typedef                 */
#include        <fast_error.h>  /* for ERROR() macro                        */
#include        <fast_memory.h> /* for DDIM3() and DDIM4() macros           */
#include        <fld_list.h>    /* for SCALAR, NUM_VARS etc                 */
#include        <get_data.h>    /* for req_*()                              */
#include        <fast_cmap.h>   /* for MAX_SCMAPS & colormap functions      */
#include        <Module.h>      /* for scripting stuff                      */
#include        <ViewI.h>       /* for locking/unlocking                    */
#include        "funcs.h"       /* for function declarations                */

/*-------------- FORWARD DECLARATIONS OF PRIVATE FUNCTIONS ------------------*/

static tGraphicObject make_new_object( char* name );
static Grid_Surface* lock_object( tGraphicObject object );
int lock_cur_object( void );
int unlock_cur_object( void );
static int inc_lock_count( void );
static int get_object_id( char* object_name );
static tGraphicObject attach_object( int object_id );
static void detach_cur_object( void );
static void delete_an_object( char* script_command );
static void deallocate_object_data( tGraphicObject object );

static void copy_color( int attribute );
static void copy_colors( void );
static void update_colors( void );
static void set_default_colors( void );

static Panel* main_panel( char* title, int win_x, int win_y );
static Panel* minmax_panel( char* title, int win_x, int win_y );

static Panel* contour_panel( char* title, int win_x, int win_y );
static void set_contour_legend( char* str );

static void file_io_func( char* file_name, int mode );
static void panels_func( int group, int item );
static void attributes_func( int group, int item );
static void set_attributes_func( char* script_command );
static void type_func( int group, int item );
static void set_type_func( char* script_command );
static void render_func( int group, int item );
static void set_render_func( char* script_command );
static void options_func( int group, int item );
static void set_options_func( char* script_command );

static void dump_state( void );
```

```
static void reset_state( int reset_actuators );

static void select_object(char* script_command );
static void new_object( char* script_command );
static void copy_object( char* script_command );
static void update_objects( Actuator* a );

static void surface_buttons_func( int row, int col, int state );
static void set_surface_buttons_func( char* script_command );
static void slider_buttons_func( int row, int col, int state );
static void set_slider_buttons_func( char* script_command );
static void loop_buttons_func( int row, int col, int state );
static void set_loop_buttons_func( char* script_command );
static void zone_func( int new_zone );
static void set_zone_func( char* script_command );
static void toggle_draw_func( char* script_command );
static void ijk_ranges_func( int dir, float ranges[5] );
static void set_ijk_ranges_func( char* script_command );
static void direction_func( int dir );
static void set_direction_func( char* script_command );
static void boundary_surfaces_func( int selections[3][3] );
static void set_boundary_surfaces_func( char* script_command );

static void reset_ijk_ranges( void );

static void data_select( int type, int reg_num, int fld_num,
            FLDDataPtr fld_data_ptr );
static void reset_data( int type );

static void vector_scale( int index, float new_value );
static void set_vector_scale( char* script_command );

static void set_minmax_func( char* script_command );
static void reset_minmax( char* script_command );
static void adjust_minmax_func( char* script_command );
static void set_minmax_modes( char* script_command );
static void invert_clip_test( char* script_command );

static void clip_and_norm( int state );
static int copy_normals( void );
static void delete_normals( void );
static void check_delete_normals( int new_dir, int new_ranges[3][5] );

static int copy_contours( void );
static void delete_contours( void );

static void first_object_name( char* name );
static void update_object_typeout( void );

static void update_actuators( void );
static void update_data_info( void );
static void update_dims( int type, int dims[3] );

static void update_minmax_sliders( float minmax[2][4], int type );
static void update_legend( float min_val, float max_val, Actuator** act );
static void update_palettes( int clip_or_norm );

/*------------------------------- DEFINES ----------------------------------*/

#ifndef MINIMUM
#define MINIMUM         0
#endif

#ifndef MAXIMUM
#define MAXIMUM         1
#endif

#ifndef MIN
#define MIN(a, b)       ((a) < (b) ? (a) : (b))
#endif

#ifndef MAX
#define MAX(a, b)       ((a) > (b) ? (a) : (b))
#endif

#define ROUND(x)        ((x) >= 0.0 ? ((int) ((x) + 0.5)) : ((int) ((x) - 0.5)))

#define LIMIT_TO_MIN(a, min)    if ((a) < (min)) (a) = (min)
#define LIMIT_TO_MAX(a, max)    if ((a) > (max)) (a) = (max)
#define LIMIT_TO(a, min, max)   {LIMIT_TO_MIN(a, min); LIMIT_TO_MAX(a, max);}

#define MATCH_DIMS(dims1, dims2)                                    \
                                                                    \
    (dims1[I] == dims2[I] && dims1[J] == dims2[J] && dims1[K] == dims2[K])

/*------------------------- Panel Attributes -----------------------------*/

/* main panel */

#define MAIN_TITLE              MODULE_NAME
#define MAIN_WIN_X              660
#define MAIN_WIN_Y              800

/* data panel */

#define DATA_TITLE              MODULE_NAME
#define DATA_WIN_X              750
#define DATA_WIN_Y              10
#define MATCH_GRID_BUTTON       1

static int data_formats[3] = { ALL_FORMATS, STRUCTURED, STRUCTURED };

/* contour panel */

#define FISST_CONTOUR_LABEL     "Surfer: Contours"
#define FISST_CONTOUR_WIN_X     800
#define FISST_CONTOUR_WIN_Y     50

/* scalar minmax panel */

#define MINMAX_TITLE            "Surfer: Scalar Minmax"
#define MINMAX_WIN_X            735
#define MINMAX_WIN_Y            20


/*--------------------------- Vector Scale Panel --------------------------*/

#define VECTOR_TITLE            "Surfer: Vector Scale"
#define VECTOR_WIN_X            1125
#define VECTOR_WIN_Y            582

/*----------------------- Vector Panel's Scale Group ----------------------*/

#define SCALE_GROUP_X           0.0
#define SCALE_GROUP_Y           (-1.0 * SCALE_GROUP_HEIGHT)
#define SCALE_GROUP_FRAMED      0
#define SCALE_GROUP_NUM_VALUES  2
```



```
static char* scale_group_labels[SCALE_GROUP_NUM_VALUES] =
        { "Vectors", "Frame Lines" };

/* initial vector and frame scale factors */

static float scale_group_values[SCALE_GROUP_NUM_VALUES] = { 1.0, 0.1 };

/* add 100% to vector and 1% to frame while the slider is pegged */

static float scale_group_slider_rates[SCALE_GROUP_NUM_VALUES] = { 1.0, 0.01 };

/* multiply the vector by 2 and the frame by 1.1 on each up button click */

static float scale_group_button_rates[SCALE_GROUP_NUM_VALUES] = { 2.0, 1.1 };

static char* scale_group_script_commands[SCALE_GROUP_NUM_VALUES] =
        {
                "VECTOR_SCALE %f",
                "FRAME_SCALE %f"
        };



/*----------------------------- Color Panel ------------------------------*/

#define COLOR_TITLE             MODULE_NAME
#define COLOR_WIN_X             1030
#define COLOR_WIN_Y             300


static char* color_labels[NUM_GS_COLORS] =
        {
                "Line",
                "Point",
                "Contour",
                "Vector",
                "Polygon",
                "Outline",
                "Glyph"
        };

static int color_indices[NUM_GS_COLORS];
static int default_color_indices[NUM_GS_COLORS] =
        {
                /* LINE_COLOR           */      RED,
                /* POINT_COLOR          */      YELLOW,
                /* CONTOUR_COLOR        */      BLUE,
                /* VECTOR_COLOR         */      CYAN,
                /* POLYGON_COLOR        */      WHITE,
                /* OUTLINE_COLOR        */      WHITE,
                /* GLYPH_COLOR          */      WHITE
        };

static float color_rgbs[NUM_GS_COLORS][3];
static float default_color_rgbs[NUM_GS_COLORS][3] =
        {
                /* LINE_COLOR           */      {1.0, 0.0, 0.0},
                /* POINT_COLOR          */      {1.0, 1.0, 0.0},
                /* CONTOUR_COLOR        */      {0.0, 0.0, 1.0},
                /* VECTOR_COLOR         */      {0.0, 1.0, 1.0},
                /* POLYGON_COLOR        */      {1.0, 1.0, 1.0},
                /* OUTLINE_COLOR        */      {1.0, 1.0, 1.0},
                /* GLYPH_COLOR          */      {1.0, 1.0, 1.0}
        };


/*---------------------------- Panels Menu -----------------------------*/

#define PANELS_MENU_ITEMS       5
#define PANELS_MENU_GROUPS      1
#define PANELS_MENU_MARKABLE    0
#define PANELS_MENU_JUSTIFY     PNL_LEFT_JUSTIFY

static int panels_menu_items_per_group[PANELS_MENU_GROUPS]={PANELS_MENU_ITEMS};
static int panels_menu_selections[PANELS_MENU_ITEMS];
static int panels_menu_markable[PANELS_MENU_GROUPS] = { 0 };
static char* panels_menu_labels[1 + PANELS_MENU_ITEMS] =
        {
                "Panels",
                "Data...",
                "Colors...",
                "Vector Scale...",
                "Scalar Minmax...",
                "Contours..."
        };

static char* panels_menu_script_commands[1 + PANELS_MENU_ITEMS] =
        {
                "OPEN_PANEL %s",
                "DATA",
                "COLORS",
                "VECTOR_SCALE",
                "SCALAR_MINMAX",
                "CONTOURS"
        };



/*------------ panels ----------*/

#define MAIN_PANEL              0
#define DATA_PANEL              1
#define COLOR_PANEL             2
#define VECTOR_PANEL            3
#define MINMAX_PANEL            4
#define CONTOUR_PANEL           5
#define NUM_PANELS              6


/*---------------------------- Options Menu ----------------------------*/

#define OPTIONS_MENU_ITEMS      5
#define OPTIONS_MENU_GROUPS     5
#define OPTIONS_MENU_MARKABLE   1
#define OPTIONS_MENU_JUSTIFY    PNL_LEFT_JUSTIFY

static int options_menu_items_per_group[OPTIONS_MENU_GROUPS] =
        { 1, 1, 1, 1, 1 };
static int options_menu_selections[OPTIONS_MENU_ITEMS] =
        { 0, 0, 0, 0, 0 };
static int options_menu_markable[OPTIONS_MENU_GROUPS] =
        { 1, 1, 0, 0, 0 };
static char* options_menu_labels[1 + OPTIONS_MENU_ITEMS] =
        {
                "Options",
                "Draw Outline",
                "Draw Glyph",
```

```
            "Recompute Normals",
            "Reset",
            "Debug"
        };


/*------------------------------ Type Menu ------------------------------*/

#define TYPE_MENU_ITEMS 3
#define TYPE_MENU_GROUPS        1
#define TYPE_MENU_MARKABLE      1
#define TYPE_MENU_JUSTIFY       PNL_LEFT_JUSTIFY

static int type_menu_items_per_group[TYPE_MENU_GROUPS] =
        { TYPE_MENU_ITEMS };
static int type_menu_selections[TYPE_MENU_ITEMS] = { 1, 0, 0 };
static int type_menu_markable[TYPE_MENU_GROUPS] = { 1 };
static char* type_menu_labels[1 + TYPE_MENU_ITEMS] =
        {
                "Type",
                "Grid",
                "Scalar",
                "Vector"
        };

static char* type_menu_script_commands[1 + TYPE_MENU_ITEMS] =
        {
                "TYPE %s",
                "GRID",
                "SCALAR",
                "VECTOR"
        };


/*------------------------------ Render Menu ------------------------------*/

#define RENDER_MENU_ITEMS       12
#define RENDER_MENU_GROUPS      1
#define RENDER_MENU_MARKABLE    1
#define RENDER_MENU_JUSTIFY     PNL_CENTER_JUSTIFY

static int render_menu_items_per_group[RENDER_MENU_GROUPS] =
        { RENDER_MENU_ITEMS };
static int render_menu_markable[RENDER_MENU_GROUPS] =
        { 1 };
static int render_menu_selections[RENDER_MENU_ITEMS] =
        { 0,0,0,0,0,0,1,0,0,0,0,0 };
static char* render_menu_labels[1 + RENDER_MENU_ITEMS] =
        {
                "Render",
                "Points",
                "Dots",
                "Bubbles",
                "Stars",
                "Grid Lines 1",
                "Grid Lines 2",
                "Grid Lines 1 and 2",
                "Contour Lines",
                "Plain Vectors",
                "Normalized Vectors",
                "Flat Polygons",
                "Smooth Polygons"
        };

static char* render_menu_script_commands[1 + RENDER_MENU_ITEMS] =
        {
                "RENDER %s",
                "POINTS",
                "DOTS",
                "BUBBLES",
                "STARS",
                "GRID_LINES_1",
                "GRID_LINES_2",
                "GRID_LINES_1_AND_2",
                "CONTOUR_LINES",
                "PLAIN_VECTORS",
                "NORMALIZED_VECTORS",
                "FLAT_POLYGONS",
                "SMOOTH_POLYGONS"
        };


/*------------------------------ Attributes Menu ------------------------------*/

#define ATTRIBUTES_MENU_ITEMS           17
#define ATTRIBUTES_MENU_GROUPS          8
#define ATTRIBUTES_MENU_MARKABLE        1
#define ATTRIBUTES_MENU_JUSTIFY         PNL_RIGHT_JUSTIFY

static int attributes_menu_items_per_group[ATTRIBUTES_MENU_GROUPS] =
        { 2, 2, 2, 2, 3, 2, 2, 2 };
static int attributes_menu_selections[ATTRIBUTES_MENU_ITEMS] =
        { 0,1, 1,0, 1,0, 1,0, 1,0,0, 1,0, 1,0, 1,0 };
static int attributes_menu_markable[ATTRIBUTES_MENU_GROUPS] =
        { 1, 1, 1, 1, 1, 1, 1, 1 };
static char* attributes_menu_labels[1 + ATTRIBUTES_MENU_ITEMS] =
        {
                "Attributes",
                "Constant Color Contours",
                "Scalar   Color Contours",
                "Constant Color Vectors",
                "Scalar   Color Vectors",
                "No       Vector Tips",
                "Arrowed Vector Tips",
                "Straight Clip",
                "Faded    Clip",
                "Default Shading",
                "Shading On",
                "Shading Off",
                "Frame Lines Off",
                "Frame Lines On",
                "Standard Normals",
                "Reversed Normals",
                "Surface Normals",
                "Zone Normals"
        };

static char* attributes_menu_script_commands[1 + ATTRIBUTES_MENU_ITEMS] =
        {
                "ATTRIBUTES %s",
                "CONSTANT_COLOR_CONTOURS",
                "SCALAR_COLOR_CONTOURS",
                "CONSTANT_COLOR_VECTORS",
                "SCALAR_COLOR_VECTORS",
                "NO_VECTOR_TIPS",
                "ARROWED_VECTOR_TIPS",
                "STRAIGHT_CLIP",
                "FADED_CLIP",
```



```
                "DEFAULT_SHADING",
                "SHADING_ON",
                "SHADING_OFF",
                "FRAME_LINES_OFF",
                "FRAME_LINES_ON",
                "STANDARD_NORMALS",
                "REVERSED_NORMALS",
                "SURFACE_NORMALS",
                "ZONE_NORMALS"
        };


/*------------------------- Main Panel Buttons -------------------------*/

#define DRAW_BUTTON_LABEL               "Draw the Object"
#define UPDATE_OBJECTS_BUTTON_LABEL     "Update All Objects"
#define NEW_OBJECT_BUTTON_LABEL         "New  Object"
#define COPY_OBJECT_BUTTON_LABEL        "Copy Object"

#define OBJECT_TYPEOUT_LABEL            "Select Current Object:"
#define OBJECT_TYPEOUT_LINES    5
#define OBJECT_TYPEOUT_COLS     OBJECT_NAME_LENGTH

/* data info typeout */

#define DATA_INFO_LABEL         "Data Info:"
#define DATA_INFO_COLS          20
#define DATA_INFO_LINES         13
#define DATA_INFO_BUF_SIZE      (2 * DATA_INFO_COLS * DATA_INFO_LINES)


/*------------------------- Loop Buttons Group -------------------------*/

#define LOOPING_LABEL   "Looping:"

#define NUM_LOOP_BUTTONS                9
#define LOOP_BUTTONS_FRAMED             1
#define LOOP_BUTTONS_RADIO_GROUPING     RADIO_ROW
#define LOOP_BUTTONS_ROWS               3

static int loop_buttons_per_row[LOOP_BUTTONS_ROWS] = { 4, 3, 2 };
static int loop_buttons_types[NUM_LOOP_BUTTONS] =
        { PNL_RADIO_BUTTON, PNL_RADIO_BUTTON, PNL_RADIO_BUTTON, PNL_RADIO_BUTTON,
          PNL_RADIO_BUTTON, PNL_RADIO_BUTTON, PNL_RADIO_BUTTON,
          PNL_RADIO_BUTTON, PNL_RADIO_BUTTON };
static int loop_buttons_selections[NUM_LOOP_BUTTONS] =
        { 1, 0, 0, 0,  0, 0, 1,  1, 0 };
static char* loop_buttons_labels[NUM_LOOP_BUTTONS] =
        {
                "Off", "Forward", "Backward", "Bounce",
                "Bottom", "Top ", "Middle",
                "Single", "Multi"
        };
static char* loop_buttons_row_labels[LOOP_BUTTONS_ROWS] =
        {
                "Direction: ",
                "Plane:     ",
                "Zones:     "
        };
static char* loop_buttons_script_load_commands[LOOP_BUTTONS_ROWS]=
        {
                "LOOP %s",
                "PLANE %s",
                "LOOP_ZONE %s"
        };
static char* loop_buttons_script_param_commands[LOOP_BUTTONS_ROWS][4]=
        {
                { "OFF", "FORWARD", "BACKWARD", "BOUNCE" },
                { "BOTTOM", "TOP", "MIDDLE", " " },
                { "SINGLE", "MULTI", " ", " " }
        };


/*------------------------- Slider Buttons Group -------------------------*/

#define NUM_SLIDER_BUTTONS              3
#define SLIDER_BUTTONS_FRAMED           1
#define SLIDER_BUTTONS_RADIO_GROUPING   RADIO_ROW
#define SLIDER_BUTTONS_ROWS             1

static int slider_buttons_per_row[SLIDER_BUTTONS_ROWS] = {NUM_SLIDER_BUTTONS};
static int slider_buttons_types[NUM_SLIDER_BUTTONS] =
        { PNL_BUTTON, PNL_TOGGLE_BUTTON, PNL_TOGGLE_BUTTON };
static int slider_buttons_selections[NUM_SLIDER_BUTTONS] =
        { 0, 1, 1 };
static char* slider_buttons_labels[NUM_SLIDER_BUTTONS] =
        { "Reset", "Reset to zone", "Show Looping" };
static char* slider_buttons_row_labels[1] = { "Sliders:" };


/*------------------------- Surface Buttons Group -------------------------*/

#define NUM_SURFACE_BUTTONS             3
#define SURFACE_BUTTONS_FRAMED          1
#define SURFACE_BUTTONS_RADIO_GROUPING  RADIO_ROW
#define SURFACE_BUTTONS_ROWS            1

static int surface_buttons_per_row[SURFACE_BUTTONS_ROWS] =
        { NUM_SURFACE_BUTTONS };
static int surface_buttons_types[NUM_SURFACE_BUTTONS] =
        { PNL_RADIO_BUTTON, PNL_RADIO_BUTTON, PNL_RADIO_BUTTON };
static int surface_buttons_selections[NUM_SURFACE_BUTTONS] =
        { 1, 0, 0 };
static char* surface_buttons_labels[NUM_SURFACE_BUTTONS] =
        { "Single", "Boundary", "Range" };
static char* surface_buttons_row_labels[1] = { "Surface:" };
static char* surface_buttons_script_commands[1 + NUM_SURFACE_BUTTONS] =
        {
                "SURFACE %s",
                "SINGLE",
                "BOUNDARY",
                "SURFACE_RANGE"
        };

/*------------------------- Zone Typein Group -------------------------*/

#define ZONE_TYPEIN_TYPE                INT_TYPE
#define ZONE_TYPEIN_FORMAT              "%d"
#define ZONE_TYPEIN_LABEL               "Zone:"
#define ZONE_TYPEIN_LABEL_TYPE          PNL_LABEL_TOP_LEFT
#define ZONE_TYPEIN_WIDTH               5
#define ZONE_TYPEIN_LIMITED             0

static float zone_typein_values[3] = { 0.0, 0.0, 0.0 };
```

```
/*--------------------------- Slider Group ----------------------------*/

#define SLIDER_GROUP_FRAMED             1
#define SLIDER_GROUP_TYPE               INT_TYPE
#define SLIDER_GROUP_SELECTION_BUTTONS  1

static char slider_group_label_letters[3] = { 'I', 'J', 'K' };
static char* slider_group_direction_label = " Surface";

static float slider_group_values[3][5] = /* [I/J/K][START/END/INC/CUR/DIM] */
        {
                { 0.0, 0.0, 1.0, 0.0, 0.0 },
                { 0.0, 0.0, 1.0, 0.0, 0.0 },
                { 0.0, 0.0, 1.0, 0.0, 0.0 }
        };

static int slider_group_selections[3][3] = /* [I/J/K][START/END/MID] */
        {
                { 1, 1, 1 },
                { 1, 1, 1 },
                { 1, 1, 1 }
        };




/*--------------------------- Actuator Attributes ----------------------*/


/* top left corner of the panel */

#define X_ORIGIN                0.0
#define Y_ORIGIN                0.0
#define SPACE                   0.1

/* pull-down menus are at the top */

#define X_MENU                  (X_ORIGIN + 1.25)
#define Y_MENU                  Y_ORIGIN
#define X_MENU_INC              3.0
#define Y_MENU_INC              0.5

/* panel buttons */

#define X_BUTTON_INC            3.0
#define Y_BUTTON_INC            0.6


#define INT_LABEL_FORMAT        "%d"



/*--------------------------- Scalar Minmax Stuff ----------------------*/

/* frame coordinates */

#define SMM_FRAME_HEIGHT        15.0

#define X_FRAME_ACT             0.0
#define Y_FRAME_ACT             6.0

/* for string typeins & labels */

#define ZERO_STRING_16          "0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"

/* multislider dimensions */

#define MULTISLIDER_WIDTH       0.5
#define MULTISLIDER_HEIGHT      6.0

/* palette dimensions */

#define PALETTE_WIDTH           1.0
#define PALETTE_HEIGHT          6.0


#define INVERT_CLIP_TEST_LABEL  "Clip Inside"

#define CLIP_LABEL              "  Clip:   "
#define CLIP_TOP_LABEL          "Top:"
#define CLIP_BOT_LABEL          "Bot:"
#define CLIP_TOP_NUMBER_STR     ZERO_STRING_16
#define CLIP_BOT_NUMBER_STR     ZERO_STRING_16
#define RESET_CLIP_LABEL        " Reset "

#define NORM_LABEL              "Normalize:"
#define NORM_TOP_LABEL          "Top:"
#define NORM_BOT_LABEL          "Bot:"
#define NORM_TOP_NUMBER_STR     ZERO_STRING_16
#define NORM_BOT_NUMBER_STR     ZERO_STRING_16
#define RESET_NORM_LABEL        " Reset "

#define LEGEND_LABEL            " Legend:  "
#define LEGEND_MAX_LABEL        "Max:"
#define LEGEND_MIN_LABEL        "Min:"
#define LEGEND_MAX_NUMBER_STR   ZERO_STRING_16
#define LEGEND_MIN_NUMBER_STR   ZERO_STRING_16
#define RESET_LEGEND_LABEL      " Reset "
#define NUM_LEGEND_VALUES       11

#define UPDATE_MMSLIDERS_LABEL  "Update Minmax Sliders"
#define AUTO_MINMAX_UPDATE_LABEL        "Auto Minmax Update"
#define MULTI_ZONE_MINMAX_LABEL         "Multi Zone Minmax"
#define SINGLE_ZONE_MINMAX_LABEL        "Single Zone Minmax"
#define SUBSET_MINMAX_LABEL             "Zone Subset Minmax"
#define SURFACE_MINMAX_LABEL            "Surface Minmax"
#define SURFACE_SUBSET_MINMAX_LABEL     "Surface Subset Minmax"


#define LEGEND                  2
#define LOW                     0
#define MED                     1
#define HI                      2
#define NUM_PALETTE_SETS        3       /* CLIP, NORM, LEGEND */
#define NUM_PALETTE_TYPES       3       /* LOW, MED, HI */

#define MIN_MAP                 ((float) get_function_index(0.0))
#define MAX_MAP                 ((float) get_function_index(1.0))

#define PALETTE_X_INC           5.0
```



```
#define RESET_CLIP_ID           "1"
#define RESET_NORM_ID           "2"
#define RESET_LEGEND_ID         "3"
#define CLIP_MSLIDER_ID         "4"
#define NORM_MSLIDER_ID         "5"
#define CLIP_TOP_ID             "6"
#define CLIP_BOT_ID             "7"
#define NORM_TOP_ID             "8"
#define NORM_BOT_ID             "9"
#define LEGEND_MAX_ID           "10"
#define LEGEND_MIN_ID           "11"
#define AUTO_MINMAX_UPDATE_ID   "12"
#define UPDATE_MMSLIDERS_ID     "13"
#define MULTI_ZONE_MINMAX_ID    "14"
#define SINGLE_ZONE_MINMAX_ID   "15"
#define SUBSET_MINMAX_ID        "16"
#define SURFACE_MINMAX_ID       "17"
#define SURFACE_SUBSET_MINMAX_ID "18"
#define CONTOUR_MIN_ID          "19"
#define CONTOUR_MAX_ID          "20"
#define CONTOUR_NUM_ID          "21"
#define CONTOUR_INC_ID          "22"



/* for routines that lock and unlock the object */

#ifdef DEBUG
#define DEBUG_LOCKING   1
#else
#define DEBUG_LOCKING   0
#endif

#if DEBUG_LOCKING
#define DPRINT(s, v)    printf(s, v)
#else
#define DPRINT(s, v)
#endif


/*--------------------------------- TYPEDEFS ---------------------------------*/

/*
 * define structures that will contain pointers to the important actuators
 * and groups of actuators that will be updated by various action funcs
 */

struct main_actuators
{
        MenuGroup*      panels_menu_group;
        MenuGroup*      attributes_menu_group;
        MenuGroup*      type_menu_group;
        MenuGroup*      render_menu_group;
        MenuGroup*      options_menu_group;
        Actuator*       object_frame;
        Actuator*       draw_object_button;
        Actuator*       update_objects_button;
        Actuator*       new_object_button;
        Actuator*       copy_object_button;
        Actuator*       object_typeout;
        Actuator*       looping_label;
        ButtonGroup*    loop_buttons_group;
        ButtonGroup*    slider_buttons_group;
        ButtonGroup*    surface_buttons_group;
        TypeinGroup*    zone_typein_group;
        SliderGroup*    slider_group;
        Actuator*       data_info_typeout;
};

typedef struct  main_actuators  Main_Acts;

struct minmax_actuators
{
        Actuator*       minmax_frame;
        Actuator*       invert_clip_test_button;
        Actuator*       clip_label;
        Actuator*       norm_label;
        Actuator*       legend_label;
        Actuator*       clip_top_typein;
        Actuator*       norm_top_typein;
        Actuator*       legend_max_typein;
        Actuator*       clip_multislider;
        Actuator*       norm_multislider;
        Actuator*       legend_multislider;
        Actuator*       clip_bot_typein;
        Actuator*       norm_bot_typein;
        Actuator*       legend_min_typein;
        Actuator*       reset_clip_button;
        Actuator*       reset_norm_button;
        Actuator*       reset_legend_button;
        Actuator*       update_minmax_sliders_button;
        Actuator*       auto_minmax_update_button;
        Actuator*       mode_buttons[NUM_SCALAR_MINMAX_MODES];
        Actuator*       palettes[NUM_PALETTE_SETS][NUM_PALETTE_TYPES];
        Actuator*       legend_labels[NUM_LEGEND_VALUES];
};

typedef struct  minmax_actuators        Minmax_Acts;

struct contour_actuators
{
        Actuator*       palette;
        Actuator*       contour_min_typein;
        Actuator*       contour_max_typein;
        Actuator*       contour_num_typein;
        Actuator*       contour_inc_typein;
        Actuator*       contour_labels[NUM_LEGEND_VALUES];
};

typedef struct  contour_actuators       Contour_Acts;

/*---------------------------------- GLOBALS ----------------------------------*/

/*
 * Cur_object is a pointer to the current graphical object which contains a
 * grid surface as part of its structure. We allocate and attach to an object
 * during initialization. We remain attached to this object until another one
 * is selected. Cur_object is locked each time before it is accessed in an
 * action func and before exiting the action func it is unlocked (however
 * we still remain attached to it). This enforces mutual exclusion between
 * this process and others (e.g., viewer). Sometimes one action func
 * will call another in which case we do not actually lock the object
 * more than once but rather increment a locked count. This locked count is
 * also decremented after each unlock. However deeply nested the lock are,
 * there must eventually be a matching unlock so that when we are done
 * accessing the object other proceses may do so. Sometimes we must read or
```

```
 * write to viewer (indirectly via the hub). In these cases we must be
 * sure that the object is unlocked otherwise we could become deadlocked.
 * (If we are waiting for the hub, the hub is waiting for viewer and
 * viewer is waiting for us to unlock our object so it can draw.)
 */

static tGraphicObject   cur_object;

static char     cur_object_name[OBJECT_NAME_LENGTH]; /* name of object */

/*
 * Grid_surf is a pointer to the current grid surface structure that is part
 * of cur_object. We must call lock_cur_object() each time before accessing
 * this pointer and call unlock_cur_object() each time after accessing it.
 * For example, we must lock at start of every action func that accesses
 * this pointer (nearly all do) and unlock before returning (or exiting).
 */

Grid_Surface*   grid_fisst;     /* define globally so fisstSupport can use it */

/* array holds pointers to all Surfer panels */

static Panel*   panels[NUM_PANELS];

/* structures hold pointers to all important actuators */

static Main_Acts        main_acts;
static Minmax_Acts      minmax_acts;
static ScaleGroup*      scale_group;
static Contour_Acts     contour_acts;

/* special flags and modes */

static int      locked;                 /* current object lock count    */
static int      loop_mode;              /* looping status               */
static int      show_looping = 1;       /* redraw sliders when looping? */
static int      update_actuators_mode;  /* set when selecting a new obj */
static int      update_all_objects_mode;/* apply to all objects?        */
static int      interactive;            /* interactive or scripted      */

int     redraw_object_when_unlocked = 1;
int     call_update_minmax_after_drawing = 0;


static User_token_def   tokens[] =
  {
    { "SELECT_OBJECT",          select_object,              REPEAT_YES },
    { "NEW_OBJECT",             new_object,                 REPEAT_YES },
    { "COPY_OBJECT",            copy_object,                REPEAT_YES },
    { "DELETE_OBJECT",          delete_an_object,           REPEAT_YES },
    { "DRAW",                   toggle_draw_func,           REPEAT_YES },

    { "TYPE",                   set_type_func,              REPEAT_YES },
    { "RENDER",                 set_render_func,            REPEAT_YES },
    { "ATTRIBUTES",             set_attributes_func,        REPEAT_YES },
    { "OPTIONS",                set_options_func,           REPEAT_YES },
    { "DIRECTION",              set_direction_func,         REPEAT_YES },
    { "SLIDER",                 set_ijk_ranges_func,        REPEAT_NO  },
    { "SURFACE",                set_surface_buttons_func,   REPEAT_YES },
    { "BOUNDARY",               set_boundary_surfaces_func, REPEAT_NO },
    { "LOOP",                   set_loop_buttons_func,      REPEAT_YES },
    { "PLANE",                  set_loop_buttons_func,      REPEAT_YES },
    { "LOOP_ZONE",              set_loop_buttons_func,      REPEAT_YES },
    { "SLIDER_ACTION",          set_slider_buttons_func,    REPEAT_YES },
    { "MINMAX_MODE",            set_minmax_modes,           REPEAT_YES },
    { "AUTO_MINMAX",            set_minmax_modes,           REPEAT_YES },
    { "UPDATE_MINMAX",          set_minmax_modes,           REPEAT_YES },
    { "CLIP",                   invert_clip_test,           REPEAT_YES },
    { "MINMAX",                 set_minmax_func,            REPEAT_NO  },
    { "RESET_MINMAX",           reset_minmax,               REPEAT_YES },
    { "ZONE",                   set_zone_func,              REPEAT_YES },
    { "VECTOR_SCALE",           set_vector_scale,           REPEAT_NO  },
    { "FRAME_SCALE",            set_vector_scale,           REPEAT_NO  },
    { "CONTOUR",                set_contour_legend,         REPEAT_YES },
    { NULL, NULL, REPEAT_NO }
  };



/*****************************************************************/
/************************** PUBLIC FUNCTIONS *********************/
/*****************************************************************/



/*++ int get_looping( void )
 *
 * PURPOSE:
 *
 *              Return the looping status of fisst.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              12/89
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
```

92/06/10 09:06:13 panels.c

```
 *
 *              int             looping status
 *
 * GLOBAL VARIABLES USED:
 *
 *              extern int      loop_mode       defined in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              None
 *
 --*/

/*---------------------------- get_looping ----------------------------------*/

int get_looping( void )
{
        return loop_mode;
}

/*------------------------- END OF get_looping ------------------------------*/









/*++ void update_looping( void )
 *
 * PURPOSE:
 *
 *              Updates the grid surface structure to specify the next
 *              surface to display based on the current state and the
 *              looping control variables. Note: update sliders button
 *              must be on for this to occur.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              7/89
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Main_Acts        main_acts;              declared in this file
 *      extern Grid_Surface*    grid_fisst;             declared in this file
 *      extern int              loop_mode               declared in this file
 *      extern int              show_looping            declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *      void            update_date_info()      defined in this file
 *
 --*/

/*---------------------------- update_looping -------------------------------*/

void update_looping( void )
{
        int             d;                      /* I, J, or K           */
        int*            ranges;                 /* ranges for d         */
        int             cur_zone;               /* current grid zone    */

#ifdef DEBUG
printf("inside update_looping() with loop_mode = %d, show_looping = %d\n",
loop_mode, show_looping);
#endif

        if ( redraw_object_when_unlocked == 1 )
                if ( loop_mode == LOOP_OFF || show_looping == 0 ) return;

        lock_cur_object();

        /* check if we must advance to the next zone */

        if (    grid_fisst -> loop_zone == MULTI_ZONE &&
                grid_fisst -> loop_new_zone != 0                )
        {
            cur_zone = ACCESS2(main_acts.zone_typein_group, get_ivalue);

            if      (grid_fisst -> loop_new_zone ==  1) ++cur_zone;
            else if (grid_fisst -> loop_new_zone == -1) --cur_zone;

            ACCESS4(main_acts.zone_typein_group, set_ivalue, cur_zone, 1);
            grid_fisst -> loop_new_zone = 0;
```

```
        }

        /* update the data info typeout */

        update_data_info();

        d = grid_fisst -> direction;
        ranges = grid_fisst -> ranges[d];

#ifdef DEBUG
printf("grid_fisst -> loop_mode     = %d\n", grid_fisst -> loop_mode);
printf("grid_fisst -> loop_dir      = %d\n", grid_fisst -> loop_dir);
printf("grid_fisst -> loop_zone     = %d\n", grid_fisst -> loop_zone);
printf("grid_fisst -> loop_new_zone = %d\n", grid_fisst -> loop_new_zone);
#endif

        unlock_cur_object();

        /* redraw the current slider to indicate the new position */

        ACCESS5(main_acts.slider_group, set_ivalues, d, ranges, 0);
}

/*------------------------- END OF update_looping --------------------------*/









/*++ void update_minmax( void )
 *
 * PURPOSE:
 *
 *              Updates the minmax panel to reflect the possibley altered
 *              grid surface minmax values Note: the minmax mode must be
 *              a surface type and the update slider butto must be on for
 *              this to occur.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              3/90
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Minmax_Acts      minmax_acts     defined in this file
 *      extern Grid_Surface*    grid_fisst;             declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      void            update_minmax_sliders()         defined in this file
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
 --*/

/*---------------------------- update_minmax -------------------------------*/

void update_minmax( void )
{
        if (    minmax_acts.update_minmax_sliders_button -> val == 0.0 ||
                minmax_acts.mode_buttons[MULTI_ZONE_MINMAX] -> val == 1.0 ||
                minmax_acts.mode_buttons[SINGLE_ZONE_MINMAX] -> val == 1.0  )
        {
                return;
        }

        /* update and redraw the sliders */

        lock_cur_object();
        update_minmax_sliders(grid_fisst -> minmax, LEGEND);
        update_minmax_sliders(grid_fisst -> minmax, CLIP);
        update_minmax_sliders(grid_fisst -> minmax, NORM);
        unlock_cur_object();

        call_update_minmax_after_drawing = 1;

}

/*------------------------- END OF update_minmax ---------------------------*/









/*++ void init_panels( int panel_x, int panel_y )
```



```
 *
 * PURPOSE:
 *
 *              Creates the panels.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              6/89
 *
 * INPUT PARAMETERS:
 *
 *              int     panel_x         screen x position of main panel
 *              int     panel_y         screen y position of main panel
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_fisst              defined in this file
 *      extern Panel*           panels[]                defined in this file
 *      extern Main_Acts        main_acts               defined in this file
 *      extern ScaleGroup*      scale_group             defined in this file
 *      extern char*            color_labels[NUM_GS_COLORS]     defined in this file
 *      extern int              color_indices[NUM_GS_COLORS]    defined in this file
 *      extern float            color_rgbs[NUM_GS_COLORS][3]    ""
 *      extern tGraphicObject   cur_object              defined in this file
 *      extern char     cur_object_name[OBJECT_NAME_LENGTH]     defined in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      extern int      init_fast_cmap()                libcmap
 *      extern Panel*   make_panel()                    libpanu
 *      ScaleGroup*     make_scale_group()              libpanu
 *      extern Panel*   color_panel()                   libpanu
 *      extern Panel*   fld_data_panel()                libfldpan
 *      extern void     update_fld_data_panel()         libfldpan
 *      extern void     exit_module()                   file init.c
 *      Panel*          main_panel()                    defined in this file
 *      Panel*          minmax_panel()                  defined in this file
 *      Panel*          contour_panel()                 defined in this file
 *      void            data_select()                   defined in this file
 *      void            reset_state()                   defined in this file
 *      void            set_default_colors()            defined in this file
 *      void            update_object_typeout()         defined in this file
 *      void            copy_color()                    defined in this file
 *      tGraphicObject  make_new_object()               defined in this file
 *      tGraphicObject  attach_object()                 defined in this file
 *      int             get_object_id()                 defined in this file
 *      void            first_object_name()             defined in this file
 *      int             lock_cur_object()               defined in this file
 *      int             unlock_cur_object()             defined in this file
 *
 --*/

/*---------------------------- init_panels ---------------------------------*/

void init_panels( int panel_x, int panel_y )
{
        int             must_init_object = 0;   /* init object if new   */

        /* load up the tokens */

        command_init(tokens);

        /* find or create the initial graphic object */

        first_object_name(cur_object_name);

        if (cur_object_name[0] != '\0')
        {
                /* get the id of the graphic object and attach to it */

                cur_object = attach_object(get_object_id(cur_object_name));

                if (cur_object == NULL)
                {
                    Error("Cannot attach to first graphic object! Exiting...");
                    exit_module();
                }
        }
        else /* create the initial graphic object */
        {
                strcpy(cur_object_name, OBJECT_NAME);

                cur_object = make_new_object(cur_object_name);

                if (cur_object == NULL)
                {
                    Error("Cannot make initial graphic object! Exiting...");
                    exit_module();
                }

                must_init_object = 1;
        }

        /* lock it until we are done initializing */

        lock_cur_object();

        /*
         * Redefine the panel library colors
         * Note: EVERY module in the FAST environment that uses panels or
         * the colormap must call this function before invoking its panels!
         */
```

```
        init_fast_cmap(0);


        /* Create the panels: */

        /* create main panel */

        panels[MAIN_PANEL] = main_panel( MAIN_TITLE, panel_x, panel_y );

        if (panels[MAIN_PANEL] == NULL)
        {
                Error("Cannot create a panel! Exiting...\n");
                exit_module();
        }

        /*---------------------- create fld data panel ----------------------*/

        panels[DATA_PANEL] = fld_data_panel(    DATA_TITLE,
                                                DATA_WIN_X,
                                                DATA_WIN_Y,
                                                GRID_SOLUTION_TYPEOUT,
                                                MATCH_GRID_BUTTON,
                                                data_formats,
                                                data_select     );

        if (panels[DATA_PANEL] == NULL)
        {
                Error("Cannot create a panel! Exiting...\n");
                exit_module();
        }

        /*---------------------- create minmax panel ------------------------*/

        panels[MINMAX_PANEL] = minmax_panel(    MINMAX_TITLE,
                                                MINMAX_WIN_X,
                                                MINMAX_WIN_Y    );

        if (panels[MINMAX_PANEL] == NULL)
        {
                Error("Cannot create a panel! Exiting...\n");
                exit_module();
        }

        /*------------------- create vector scale panel ---------------------*/

        panels[VECTOR_PANEL] = make_panel(      VECTOR_TITLE,
                                                VECTOR_WIN_X,
                                                VECTOR_WIN_Y, 0, 1      );

        if (panels[VECTOR_PANEL] == NULL)
        {
                Error("Cannot create a panel! Exiting...\n");
                exit_module();
        }

        /* add a scale group to this panel */

        scale_group = make_scale_group(         panels[VECTOR_PANEL],
                                                SCALE_GROUP_X,
                                                SCALE_GROUP_Y,
                                                SCALE_GROUP_FRAMED,
                                                SCALE_GROUP_NUM_VALUES,
                                                scale_group_labels,
                                                scale_group_values,
                                                scale_group_slider_rates,
                                                scale_group_button_rates,
                                                vector_scale);

        if (scale_group == NULL)
        {
                Error("Cannot create actuators! Exiting...\n");
                exit_module();
        }

        /*---------------------- create color panel -------------------------*/

        /* initilize to the default colors */

        set_default_colors();

        panels[COLOR_PANEL] = color_panel(      COLOR_TITLE,
                                                COLOR_WIN_X,
                                                COLOR_WIN_Y,
                                                NUM_GS_COLORS,
                                                color_labels,
                                                color_indices,
                                                &color_rgbs[0][0],
                                                copy_color );

        if (panels[COLOR_PANEL] == NULL)
        {
                Error("Cannot create a panel! Exiting...\n");
                exit_module();
        }

        /*---------------------- create contour panel -----------------------*/

        panels[CONTOUR_PANEL] =contour_panel(   FISST_CONTOUR_LABEL,
                                                FISST_CONTOUR_WIN_X,
                                                FISST_CONTOUR_WIN_Y     );


        if (panels[CONTOUR_PANEL] == NULL)
        {
                Error("Cannot create a panel! Exiting...\n");
                exit_module();
        }

        if ( must_init_object )
        {
                /* init grid surface data structure */

                reset_state(0);
        }
        else
        {
                /* update the actuators to reflect the grid surface */

                update_actuators();
        }

        /* update the data panel */

        update_fld_data_panel();

        /* unlock the current object so it may be drawn */

        unlock_cur_object();
```



```
        /* this causes communication to viewer so unlock first */

        update_object_typeout();
}

/*------------------------- END OF init_panels ----------------------------*/









/*****************************************************************************/
/************************** PRIVATE FUNCTIONS ********************************/
/*****************************************************************************/









/*++ static tGraphicObject make_new_object( char* name )
 *
 * PURPOSE:
 *
 *              Allocates and returns a grid surface object
 *              based on the given name but extended with a unique number.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              11/90
 *
 * INPUT PARAMETERS:
 *
 *              char*           name            name of object
 *
 * OUTPUT PARAMETERS:
 *
 *              char*           name            name of object and number
 *
 * FUNCTION RETURN:
 *
 *              extern  tGraphicObject  grid surface object
 *
 * GLOBAL VARIABLES USED:
 *
 *      None
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 *              void    init_panels()                   defined in this file
 *              void    new_object()                    defined in this file
 *
 * FUNCTIONS CALLED :
 *
 *              extern tGraphicObject   view_single_buffer()    libviewI
 *
 --*/

/*------------------------------ make_new_object ----------------------------*/

static tGraphicObject make_new_object( char* name )
{
        tGraphicObject  object;         /* pointer to a new object      */

        DPRINT("make_new_object(name = %s)...\n", name);
        object = view_single_buffer(GRID_SURFACE, name, sizeof(Grid_Surface));
        DPRINT("   generates object = %d\n", object);
        return object;
}

/*------------------------- END OF make_new_object --------------------------*/









/*++ static Grid_Surface* lock_object( tGraphicObject object )
 *
 * PURPOSE:
 *
 *              Locks the given object and returns a pointer to its
 *              grid surface structure.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              11/90
 *
 * INPUT PARAMETERS:
 *
 *              tGraphicObject  object  object to lock
 *
```

```
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              Grid_Surface*   gs      pointer to a grid surface structure
 *
 * GLOBAL VARIABLES USED:
 *
 *              None
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              extern char*    view_start_write_lock() libviewI
 *
 --*/

/*-------------------------------- lock_object --------------------------------*/

static Grid_Surface* lock_object( tGraphicObject object )
{
        Grid_Surface*   gs = NULL;

        DPRINT("lock_object(object = %d)...\n", object);

        if ((gs = (Grid_Surface *) view_start_write_lock(object)) == NULL)
        {
                Error("Cannot lock to object!");
        }

        DPRINT("   generates gs = %d\n", gs);

        return gs;
}

/*--------------------------- END OF lock_object ------------------------------*/









/*++ static int lock_cur_object( void )
 *
 * PURPOSE:
 *
 *              Locks the current object (unless it has already been locked).
 *              In either case it increments and returns the locked count.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              7/89
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              int             locked count
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern  tGraphicObject  cur_object;             declared in this file
 *      extern Grid_Surface*    grid_fisst;             declared in this file
 *      extern int              locked                  declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              extern char*    view_start_write_lock() libviewI
 *              extern void     exit_module()           file init.c
 *
 --*/

/*------------------------------ lock_cur_object ------------------------------*/

int lock_cur_object( void )
{
        DPRINT("lock_cur_object(): before locked = %d\n", locked);

        if ( locked < 0 )
        {
                printf("%s: locked = %d\n", MODULE_NAME, locked);
                Error( "Too many unlocks - Exiting..." );

#ifdef DEBUG
{
int produce_core_file = 0; produce_core_file /= produce_core_file;
}
#endif

                exit_module();
```



```
        }
        else if ( locked++ == 0 )
        {
                DPRINT("after if ( locked++ == 0 ) locked = %d so LOCKING\n",locked);

                if ((grid_fisst = (Grid_Surface *)
                        view_start_write_lock(cur_object)) == NULL)
                {
                        Error( "Cannot lock to object! Exiting..." );
                        exit_module();
                }
        }

        DPRINT("lock_cur_object(): after  locked = %d\n", locked);

        return locked;
}

/*-------------------------- END OF lock_cur_object --------------------------*/









/*++ static int unlock_cur_object( void )
 *
 * PURPOSE:
 *
 *              Unlocks the current object and NULLS grid_fisst
 *              (unless it has already been unlocked).
 *              In either case it decrements and returns the locked count.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              7/89
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              int     locked count
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern  tGraphicObject  cur_object;             declared in this file
 *      extern Grid_Surface*    grid_fisst;             declared in this file
 *      extern int              locked                  declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              extern ????     view_end_write_unlock()         libviewI
 *
 --*/

/*---------------------------- unlock_cur_object ----------------------------*/

int unlock_cur_object( void )
{
        DPRINT("unlock_cur_object(): before locked = %d\n", locked);

        if ( locked <= 0 )
        {
                printf("%s: locked = %d\n", MODULE_NAME, locked);
                Error( "Too many unlocks - Exiting..." );

#ifdef DEBUG
{
int produce_core_file = 0; produce_core_file /= produce_core_file;
}
#endif

                exit_module();
        }
        else if ( locked-- == 1 )
        {
                DPRINT("after if ( locked-- == 1 ) locked = %d so UNLOCKING\n", locked);
                if ( redraw_object_when_unlocked )
                        view_end_write_unlock(cur_object, ERASE_AND_DRAW);
                else    view_end_write_unlock(cur_object, NO_NEED_TO_DRAW);
                grid_fisst = NULL;
        }

        DPRINT("unlock_cur_object(): after  locked = %d\n", locked);

        return locked;
}

/*-------------------------- END OF unlock_cur_object ------------------------*/



void wait_until_object_is_redrawn( void )
{
    lock_cur_object();
    unlock_cur_object();
    view_data_was_drawn( cur_object, WAIT );
}
```

```
/*++ static int inc_lock_count( void )
 *
 * PURPOSE:
 *
 *              Increments the locked count and returns the new count.
 *              This is used to keep the locks and unlocks matched
 *              when a lock is made on a new object.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              7/89
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              int     locked count
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern int              locked                  declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 *              void    new_object()                    defined in this file
 *
 * FUNCTIONS CALLED :
 *
 *              None
 *
 --*/

/*----------------------------- inc_lock_count -----------------------------*/

static int inc_lock_count( void )
{
        DPRINT("inc_lock_count(): before locked = %d\n", locked);
        return ++locked;
}
```

```
/*------------------------- END OF inc_lock_count -------------------------*/



/*++ static int get_object_id( char* object_name )
 *
 * PURPOSE:
 *
 *              Writes to the Hub and gets the shared memory id of the
 *              named object.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              11/90
 *
 * INPUT PARAMETERS:
 *
 *              char*   object_name     name of the object to find
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              int                     shared memory id of the named object
 *
 * GLOBAL VARIABLES USED:
 *
 *              None
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 *              void    select_object()         defined in this file
 *
 * FUNCTIONS CALLED :
 *
 --*/

/*----------------------------- get_object_id -----------------------------*/
```



```
static int get_object_id( char* object_name )
{
        int             id;                     /* shared memory id     */
        char            command[32];


        DPRINT("get_object_id(object_name = %s)...\n", object_name);

        sprintf(command, "GET_OBJECT %s", object_name);

        send_hub_command(command);
        module_read_sock ((char*)&id, sizeof(id));

        DPRINT("   got id = %d\n", id);

        return id;
}

/*------------------------- END OF get_object_id -------------------------*/



/*++ static tGraphicObject attach_object( int object_id )
 *
 * PURPOSE:
 *
 *              Attaches to the given object shared memory id and
 *              returns a graphic object.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              11/90
 *
 * INPUT PARAMETERS:
 *
 *              int     object_id               object id to attach to
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              tGraphicObject  object;         graphic object
 *
 * GLOBAL VARIABLES USED:
 *
 *              None
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 *              void    select_object()         defined in this file
 *
 * FUNCTIONS CALLED :
 *
 *              None
 *
 --*/

/*----------------------------- attach_object -----------------------------*/

static tGraphicObject attach_object( int object_id )
{
        tGraphicObject  object;         /* graphic object       */


        DPRINT("attach_object(object_id = %d)...\n", object_id);

        if (object_id == -1)
        {
                Error("object_id = -1 in attach_object()");
                return NULL;
        }

        object = shm_get_local_address(object_id);

        DPRINT("   generates object %d\n", object);

        return object;
}

/*------------------------- END OF attach_object -------------------------*/



/*++ static void detach_cur_object( void )
 *
 * PURPOSE:
 *
 *              Detaches from the current object and NULLS cur_object and
 *              grid_first.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
```

```
 * REVISION HISTORY:
 *
 *              11/90
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern tGraphicObject   cur_object              declared in this file
 *      extern Grid_Surface*    grid_fisst              declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 *              void    select_object()                 defined in this file
 *              void    new_object()                    defined in this file
 *
 * FUNCTIONS CALLED :
 *
 *              None
 *
 --*/

/*---------------------------- detach_cur_object ----------------------------*/

static void detach_cur_object( void )
{
        DPRINT("detach_cur_object(): before cur_object = %d\n", cur_object);

        shm_destroy_local_address(cur_object);
        cur_object = NULL;
        grid_fisst = NULL;
}

/*------------------------- END OF detach_cur_object -------------------------*/









/*++ static void delete_an_object( char* script_command )
 *
 * PURPOSE:
 *
 *              Deletes the contents of the object named in the script command.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              11/91
 *
 * INPUT PARAMETERS:
 *
 *              char*   script_command
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *              None
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 *              external Hub/Viewer routines
 *
 * FUNCTIONS CALLED :
 *
 *              None
 *
 --*/

/*---------------------------- delete_an_object ----------------------------*/

static void delete_an_object( char* script_command )
{
        int             object_id;
        char            object_name[OBJECT_NAME_LENGTH];
        tGraphicObject  object = NULL;

        parse_command( script_command, "%s", object_name );

        /* Get the object id from the object name */

        object_id = get_object_id( object_name );

        /* Attach to the object */
```



```
        if ( ( object = attach_object( object_id ) ) == NULL )
        {
                Error( "Cannot attach to object" );
                return;
        }

        /* Deallocate any object-specific data */

        deallocate_object_data( object );

        /* Detach from the object */

        shm_destroy_local_address( object ); object = NULL;

        /* Send the command to remove the object from the object list */

        module_command( "Viewer", script_command, NULL );

        /* Generate an updated listing of available objects and select one */

        update_object_typeout();
}

/*------------------------- END OF delete_an_object -------------------------*/









/*++ static void select_object( char* script_command )
 *
 * PURPOSE:
 *
 *              Sets the current object
 *
 * AUTHORS:
 *
 *              Fergus J. Merritt
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              3/90
 *              5/90    Todd Plessel
 *                      modified to work around locking & handle glyph etc.
 *              11/90   Todd Plessel
 *                      simplified and removed glyph and outline handling etc.
 *              4/91    Paul Kelaita
 *                      added command stuff
 *
 * INPUT PARAMETERS:
 *
 *              char*   script_command          actuator/command
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_fisst              defined in this file
 *      extern char             cur_object_name[OBJECT_NAME_LENGTH] " " " "
 *
 * FILES USED:
 *
 *      void            update_actuators()              defined in this file
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      extern  void    exit_module()           init.c
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *      void            detach_cur_object()     defined in this file
 *      tGraphicObject  attach_object()         defined in this file
 *      int             get_object_id()         defined in this file
 *      void            update_object_typeout() define in this file
 *
 --*/

/*---------------------------- select_object ----------------------------*/

static void select_object( char* script_command )
{
        tGraphicObject  new_obj;                        /* new object   */
        char            new_obj_name[OBJECT_NAME_LENGTH];
        int             new_obj_id;                     /* new obj id   */
        Actuator*       a = main_acts.object_typeout;


        if ( is_act( script_command ) )
        {
                /* if there are no object names in the list then return */

                if ( ! get_selection_name( a, new_obj_name,
                        OBJECT_NAME_LENGTH ) )
                {
                        return;
                }

                /* if the same object was selected then just return */

                if ( strcasecmp( new_obj_name, cur_object_name ) == 0 ) return;

                load_command( "SELECT_OBJECT %s", new_obj_name );
                return;
        }


        parse_command( script_command, "%s", new_obj_name );
        set_selection_name( a, new_obj_name );
```

```
        /* if the same object was selected then just return */

        if ( strcasecmp( new_obj_name, cur_object_name ) == 0 ) return;

        hourglass_cursor( ON );

        /* write to the hub for the shared memory id of the new selection */

        new_obj_id = get_object_id( new_obj_name );

        /* check the shared memory id for validity */

        if (new_obj_id == -1)
        {
                Warning("Selected object is no longer available!");
                update_object_typeout();
                hourglass_cursor( OFF );
                return;
        }

        /* attach to the new object's shared memory id */

        if ((new_obj = attach_object(new_obj_id)) == NULL)
        {
                Error("Cannot attach to new grid surface object!");
                hourglass_cursor( OFF );
                return;
        }

        /* detach from the current object's shared memory id */

        detach_cur_object();

        /* update the current object to the new one (and copy name) */

        cur_object = new_obj;
        strcpy(cur_object_name, new_obj_name);

        /* update all actuators */

        update_actuators();

        hourglass_cursor( OFF );

}

/*--------------------------- END OF select_object ---------------------------*/




/*++ static void new_object( char* script_command )
 *
 * PURPOSE:
 *
 *              Creates a new object (a default one)
 *              and makes it the current object to modify.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              5/91
 *
 * INPUT PARAMETERS:
 *
 *              char*   script_command          actuator/command
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_fisst              defined in this file
 *      extern tGraphicObject   cur_object              defined in this file
 *      extern char             cur_object_name[OBJECT_NAME_LENGTH] " " " "
 *      extern char             cur_object_name[OBJECT_NAME_LENGTH] " " " "
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      void            update_object_typeout()         defined in this file
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *      Grid_Surface*   lock_object()           defined in this file
 *      tGraphicObject  make_new_object()       defined in this file
 *      int             inc_lock_count()        defined in this file
 *      void            detach_cur_object()     defined in this file
 *
 --*/

/*-------------------------------- new_object --------------------------------*/

static void new_object( char* script_command )
{
        Grid_Surface*   new_grid_fisst;         /* -> new surface obj   */
        tGraphicObject  new_obj;                /* temp new object      */
        char            new_obj_name[OBJECT_NAME_LENGTH];       /* name */
        int             dir;                    /* I,J,K direction      */
        int             type;                   /* START,END,MID,ZONE   */
```



```
        if ( is_act( script_command ) )
        {
                load_command( "NEW_OBJECT" );
                return;
        }

        /* allocate a new object */

        strcpy(new_obj_name, OBJECT_NAME);

        new_obj = make_new_object(new_obj_name);

        if (new_obj == NULL)
        {
                Error("Cannot allocate new object!");
                return;
        }

        /* lock the new object and point to a new grid surface structure */

        if ((new_grid_fisst = lock_object(new_obj)) == (Grid_Surface *)NULL)
        {
                Error("Cannot lock new object!");
                return;
        }

        /* lock the current object while copying (grid_fisst is current one) */

        lock_cur_object();

        /* copy the old grid surface object to the new one */

        memcpy( new_grid_fisst, grid_fisst, sizeof(Grid_Surface) );

        /* Clear the shared memory ids of the normals in the new surface */

        for ( dir = 0; dir < 3; ++dir )
          for ( type = 0; type < 4; ++type )
            new_grid_fisst -> field_ids[NORM_ID_INDEX( dir, type )] = -1;

        /* Clear the shared memory id of the contours data */

        new_grid_fisst -> field_ids[CONTOURS_ID] = -1;

        /* unlock and detach from the current grid surface object */

        unlock_cur_object();
        detach_cur_object();

        /* update the current object to the new one (and copy name) */

        cur_object = new_obj;
        grid_fisst  = new_grid_fisst;
        strcpy(cur_object_name, new_obj_name);

        /*
         * must increment locked count here (to account for locked new object)
         */

        inc_lock_count();

        /* Reset everything to the default state */

        reset_state( 1 );

        unlock_cur_object();

        /*
         * update typeout (after lock since this does communication)
         * and select the new object
         */

        update_object_typeout();

}

/*--------------------------- END OF new_object ----------------------------*/




/*++ static void copy_object( char* script_command )
 *
 * PURPOSE:
 *
 *              Copies the current object and makes the copy the current
 *              object to modify.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              6/89
 *
 * INPUT PARAMETERS:
 *
 *              char*   script_command          actuator/command
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_fisst              defined in this file
 *      extern tGraphicObject   cur_object              defined in this file
 *      extern char             cur_object_name[OBJECT_NAME_LENGTH] " " " "
 *
 * FILES USED:
 *
```

```
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      int             copy_normals()                  defined in this file
 *      void            delete_normals()                defined in this file
 *      void            update_object_typeout()         defined in this file
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *      Grid_Surface*   lock_object()           defined in this file
 *      tGraphicObject  make_new_object()       defined in this file
 *      int             inc_lock_count()        defined in this file
 *      void            detach_cur_object()     defined in this file
 *              macro USES_NORMALS() is made available by grid_surface.h
 *
 --*/

/*------------------------------ copy_object ---------------------------------*/

static void copy_object( char* script_command )
{
        Grid_Surface*   new_grid_fisst;         /* -> new surface obj   */
        tGraphicObject  new_obj;                /* temp new object      */
        char    new_obj_name[OBJECT_NAME_LENGTH];/* temp object name    */


        if ( is_act( script_command ) )
        {
                load_command( "COPY_OBJECT" );
                return;
        }

        /* allocate a new object */

        strcpy(new_obj_name, OBJECT_NAME);

        new_obj = make_new_object(new_obj_name);

        if (new_obj == NULL)
        {
                Error("Cannot allocate new object!");
                return;
        }

        /* lock the new object and point to a new grid surface structure */

        if ((new_grid_fisst = lock_object(new_obj)) == (Grid_Surface *)NULL)
        {
                Error("Cannot lock new object!");
                return;
        }

        /* lock the current object while copying (grid_fisst is current one) */

        lock_cur_object();

        /* copy the old grid surface object to the new one */

        memcpy( new_grid_fisst, grid_fisst, sizeof(Grid_Surface) );

        /* unlock and detach from the current grid surface object */

        unlock_cur_object();

        detach_cur_object();

        /* update the current object to the new one (and copy name) */

        cur_object = new_obj;
        grid_fisst  = new_grid_fisst;
        strcpy(cur_object_name, new_obj_name);

        /*
         * must increment locked count here (to account for locked new object)
         */

        inc_lock_count();

        /* copy any existing normals data if it is needed */

        if (USES_NORMALS(grid_fisst)) if (!copy_normals()) delete_normals();

        /* also copy any existing contours data if it is needed */

        if ( grid_fisst -> render_mode == CONTOUR_LINES )
          if (!copy_contours()) delete_contours();

        unlock_cur_object();

        /*
         * update typeout (after lock since this does communication)
         * and select the new object
         */

        update_object_typeout();

}

/*--------------------------- END OF copy_object -----------------------------*/




/*++ static void update_objects( Actuator* a )
 *
 * PURPOSE:
 *
 *              Toggles the mode that makes all objects updated whenever
 *              the current object changes.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
```



```
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              12/90
 *
 * INPUT PARAMETERS:
 *
 *              Actuator*       a       update objects button
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern int      update_all_objects_mode         defined in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              None
 *
 --*/

/*------------------------------ update_objects ------------------------------*/

static void update_objects( Actuator* a )
{
        update_all_objects_mode = !update_all_objects_mode;
}

/*------------------------- END OF update_objects ----------------------------*/









/*++ static void toggle_draw_func( char* script_command )
 *
 * PURPOSE:
 *
 *              Toggling the draw flag.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/90
 *
 * INPUT PARAMETERS:
 *
 *              char*   script_command  script command or actuator
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *              extern Grid_Surface*    grid_fisst      defined in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
 --*/

/*------------------------------ toggle_draw_func ----------------------------*/

static void toggle_draw_func( char* script_command )
{
        Actuator*       a = main_acts.draw_object_button;
        char            on_off_str[16];


        if ( is_act( script_command ) )
        {
                interactive = 1;
                load_command( "DRAW %s", ON_OR_OFF( a -> val ) );
                return;
        }

        parse_command( script_command, "%s", on_off_str );

        if ( ! interactive )
        {
                a -> val = (float) ON_OR_OFF_VAL( on_off_str );
                pnl_fixact( a );
```

```
        }

        interactive = 0;

        lock_cur_object();
        grid_first -> draw = (int) a -> val;
        unlock_cur_object();
}

/*----------------------- END OF toggle_draw_func --------------------------*/









/*++ static void deallocate_object_data( tGraphicObject object )
 *
 * PURPOSE:
 *
 *              Deallocates the data associated with the given object.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              11/91
 *
 * INPUT PARAMETERS:
 *
 *              tGraphicObject  object
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *              None
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 *              void    delete_an_object()              defined in this file
 *
 * FUNCTIONS CALLED :
 *
 *              None
 *
 --*/

/*------------------------- deallocate_object_data -------------------------*/

static void deallocate_object_data( tGraphicObject object )
{
        Grid_Surface*   gs = NULL;
        int             i;

        /* Lock this object */

        if ( ( gs = lock_object( object ) ) == NULL )
        {
                Error( "Cannot lock to object" );
                return;
        }

        /* Delete field ids for object-specific data */

        for ( i = START_I_NORM_ID; i < NUM_FIELDS; ++i )
        {
                if ( gs -> field_ids[i] != -1 )
                {
                        SDEALLOCATE( gs -> field_ids[i] );
                        gs -> field_ids[i] = -1;
                }
        }

        /* Unlock this object and mark it so it WON'T be drawn */

        view_and_write_unlock( object, NO_DRAW );
}

/*------------------- END OF deallocate_object_data ------------------------*/









/*++ static void copy_colors( int attribute )
 *
 * PURPOSE:
 *
 *              Copy the current color into grid surface structure
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
```



```
 *              4/90
 *
 * INPUT PARAMETERS:
 *
 *              int     attribute       index of changed color
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern int      color_indices[NUM_GS_COLORS]; this file
 *      extern float    color_rgbs[NUM_GS_COLORS][3]; this file
 *      extern Grid_Surface*    grid_first;                 this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
 --*/

/*------------------------------ copy_color ------------------------------------*/

static void copy_color( int attribute )
{
  if ( attribute >= 0 && attribute < NUM_GS_COLORS )
  {
    lock_cur_object();
    grid_first -> color_indices[attribute] = color_indices[attribute];
    grid_first -> color_rgbs[attribute][R] = color_rgbs[attribute][R];
    grid_first -> color_rgbs[attribute][G] = color_rgbs[attribute][G];
    grid_first -> color_rgbs[attribute][B] = color_rgbs[attribute][B];
    unlock_cur_object();
  }
}

/*---------------------------- END OF copy_color ---------------------------*/









/*++ static void copy_colors( void )
 *
 * PURPOSE:
 *
 *              Copy the current colors into grid surface structure
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              4/90
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern int      color_indices[NUM_GS_COLORS]; this file
 *      extern float    color_rgbs[NUM_GS_COLORS][3]; this file
 *      extern Grid_Surface*    grid_first;                 this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
 --*/

/*------------------------------ copy_colors -----------------------------------*/

static void copy_colors( void )
{
  int i;
  for ( i = 0; i < NUM_GS_COLORS; ++i ) copy_color( i );
}

/*---------------------------- END OF copy_colors ---------------------------*/
```

```
/*++ static void update_colors( void )
 *
 * PURPOSE:
 *
 *              Copy the colors from the grid surface structure into
 *              the global colors array and then fixes the color edit
 *              panel to display these colors.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              4/90
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern int      color_indices[NUM_GS_COLORS]; this file
 *      extern float    color_rgbs[NUM_GS_COLORS][3]; this file
 *      extern Grid_Surface*    grid_first;             this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      extern void     fix_color_panel()               libpanu
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
 --*/

/*------------------------------ update_colors -------------------------------*/

static void update_colors( void )
{
        int             i;                      /* loop on colors       */

        lock_cur_object();

        for (i = 0; i < NUM_GS_COLORS; i++)
        {
                color_indices[i] = grid_first -> color_indices[i];

                color_rgbs[i][R] = grid_first -> color_rgbs[i][R];
                color_rgbs[i][G] = grid_first -> color_rgbs[i][G];
                color_rgbs[i][B] = grid_first -> color_rgbs[i][B];
        }

        fix_color_panel();

        unlock_cur_object();

}

/*--------------------------- END OF update_colors ---------------------------*/
```



```
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              None
 *
 --*/

/*---------------------------- set_default_colors ----------------------------*/

static void set_default_colors( void )
{
        memcpy( color_indices, default_color_indices, sizeof color_indices );
        memcpy( color_rgbs, default_color_rgbs, sizeof color_rgbs );
}

/*---------------------- END OF set_default_colors --------------------------*/

/*++ static Panel* main_panel( char* title, int win_x, int win_y )
 *
 * PURPOSE:
 *
 *              Creates and displays the main panel.
 *              The panel is titled if title is not NULL, and is positioned at the
 *              screen coordinates (win_x, win_y).
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              6/89
 *              12/90           rewrote to use groups
 *
 * INPUT PARAMETERS:
 *
 *              char*   title                   title for panel window
 *              int     win_x                   initial x position of window
 *              int     win_y                   initial y position of window
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              Panel*  p                       a pointer to the panel
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Main_Acts        main_acts;      declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              extern void     exit_module()           file init.c
 *              void            file_io_func()          defined in this file
 *              void            panels_func()           defined in this file
 *              void            attributes_func()       defined in this file
 *              void            type_func()             defined in this file
 *              void            render_func()           defined in this file
 *              void            options_func()          defined in this file
 *              void            update_objects()        defined in this file
 *              void            new_object()            defined in this file
 *              void            copy_object()           defined in this file
 *              void            dump_state()            defined in this file
 *              void            reset_state()           defined in this file
 *              void            loop_buttons_func()     defined in this file
 *              void            slider_buttons_func()   defined in this file
 *              void            surface_buttons_func()  defined in this file
 *              void            ijk_ranges_func()       defined in this file
 *              void            direction_func()        defined in this file
 *              void            boundary_surfaces_func()defined in this file
 *              void            select_object()         defined in this file
 *              void            toggle_draw_func()      defined in this file
 *              void            zone_func()             defined in this file
 *
 --*/

/*------------------------------ main_panel ----------------------------------*/

static Panel* main_panel( char* title, int win_x, int win_y )
{
        extern void fisstPanel(Panel*,float,float);     /* fisst panel */
        Panel*          p;                      /* tmp panel            */
        Actuator*       a;                      /* tmp actuator         */
        Actuator*       f;                      /* tmp frame actuator   */
        float           x = X_ORIGIN;           /* act x position       */
        float           y = Y_ORIGIN;           /* act y position       */


        /*--------------------------- Main Panel -----------------------------*/

        if ( ( p = make_panel(title, win_x, win_y, 1, 0 ) ) == NULL )
        {
                Error("Cannot create a panel! Exiting...\n");
```

```
        exit_module();
    }

    /*-------------------------- Module menu ------------------------*/

    panels[MAIN_PANEL] = p;

    if ( !module_menu( MODULE_NAME, panels, NUM_PANELS, file_io_func ) )
    {
        Error("Cannot create a module menu! Exiting...\n");
        exit_module();
    }

    x += MODULE_MENU_WIDTH;

    /*------------------------ Panels Menu ------------------------*/

    main_acts.panels_menu_group =
        make_menu_group(p,
                        x,
                        y,
                        PANELS_MENU_GROUPS,
                        panels_menu_items_per_group,
                        panels_menu_labels,
                        panels_menu_selections,
                        panels_menu_markable,
                        PANELS_MENU_JUSTIFY,
                        panels_func);

    if ( main_acts.panels_menu_group == NULL )
    {
        Error("Could not make a menu! Exiting...\n");
        exit_module();
    }

    x += menu_group_width( main_acts.panels_menu_group );

    /*------------------------ Options Menu ------------------------*/

    main_acts.options_menu_group =
        make_menu_group(p,
                        x,
                        y,
                        OPTIONS_MENU_GROUPS,
                        options_menu_items_per_group,
                        options_menu_labels,
                        options_menu_selections,
                        options_menu_markable,
                        OPTIONS_MENU_JUSTIFY,
                        options_func);

    if ( main_acts.options_menu_group == NULL )
    {
        Error("Could not make a menu! Exiting...\n");
        exit_module();
    }

    x += menu_group_width( main_acts.options_menu_group );

    /*------------------------ Type Menu ------------------------*/

    main_acts.type_menu_group =
        make_menu_group(p,
                        x,
                        y,
                        TYPE_MENU_GROUPS,
                        type_menu_items_per_group,
                        type_menu_labels,
                        type_menu_selections,
                        type_menu_markable,
                        TYPE_MENU_JUSTIFY,
                        type_func);

    if ( main_acts.type_menu_group == NULL )
    {
        Error("Could not make a menu! Exiting...\n");
        exit_module();
    }

    x += menu_group_width( main_acts.type_menu_group );

    /*------------------------ Render Menu ------------------------*/

    main_acts.render_menu_group =
        make_menu_group(p,
                        x,
                        y,
                        RENDER_MENU_GROUPS,
                        render_menu_items_per_group,
                        render_menu_labels,
                        render_menu_selections,
                        render_menu_markable,
                        RENDER_MENU_JUSTIFY,
                        render_func);

    if ( main_acts.render_menu_group == NULL )
    {
        Error("Could not make a menu! Exiting...\n");
        exit_module();
    }

    x += menu_group_width( main_acts.render_menu_group );

    /*------------------------ Attributes Menu ------------------------*/

    main_acts.attributes_menu_group =
        make_menu_group(p,
                        x,
                        y,
                        ATTRIBUTES_MENU_GROUPS,
                        attributes_menu_items_per_group,
                        attributes_menu_labels,
                        attributes_menu_selections,
                        attributes_menu_markable,
                        ATTRIBUTES_MENU_JUSTIFY,
                        attributes_func);

    if ( main_acts.attributes_menu_group == NULL )
    {
        Error("Could not make a menu! Exiting...\n");
        exit_module();
    }

    /*-------------------------- Object Frame ------------------------*/

    x = X_ORIGIN;
    y = Y_ORIGIN - 6.15;
```



```
    main_acts.object_frame =
    f = add_frame(p, x, y);
    if ( f == NULL )
    {
        Error("Could not make an actuator! Exiting...\n");
        exit_module();
    }

    /*-------------------------- Don't Draw Button ----------------------*/

    x = X_ORIGIN;
    y = Y_ORIGIN - SPACE;

    main_acts.draw_object_button =
    a = make_actuator( pnl_toggle_button, x, y, DRAW_BUTTON_LABEL );
    if ( a == NULL )
    {
        Error("Could not make actuators! Exiting...\n");
        exit_module();
    }
    a -> val = 1.0;
    a -> upfunc = (PNL_AFUNC) toggle_draw_func;
    pnl_addsubact(a, f);

    y -= Y_BUTTON_INC;

    /*------------------ Update All Objects Button -------------------*/

    main_acts.update_objects_button =
    a = make_actuator(pnl_toggle_button,x,y,UPDATE_OBJECTS_BUTTON_LABEL);
    if ( a == NULL )
    {
        Error("Could not make an actuator! Exiting...\n");
        exit_module();
    }
    a -> upfunc = (PNL_AFUNC) update_objects;
/* HACK: unimplemented in this release so hide it */ a -> visible = 0;
    pnl_addsubact(a, f);

    y -= Y_BUTTON_INC;

    /*---------------------- New Object Button ---------------------*/

    main_acts.new_object_button =
    a = make_actuator( pnl_wide_button, x, y, NEW_OBJECT_BUTTON_LABEL );
    if ( a == NULL )
    {
        Error("Could not make an actuator! Exiting...\n");
        exit_module();
    }
    a -> upfunc = (PNL_AFUNC) new_object;
    pnl_addsubact(a, f);

    y -= Y_BUTTON_INC;

    /*--------------------- Copy Object Button --------------------*/

    main_acts.copy_object_button =
    a = make_actuator( pnl_wide_button, x, y, COPY_OBJECT_BUTTON_LABEL );
    if ( a == NULL )
    {
        Error("Could not make an actuator! Exiting...\n");
        exit_module();
    }
    a -> upfunc = (PNL_AFUNC) copy_object;
    pnl_addsubact(a, f);

    /*------------------------ Object Typeout ------------------------*/

    y -= 0.5 * OBJECT_TYPEOUT_LINES + 0.25;

    main_acts.object_typeout =
    a = make_actuator( pnl_typeout, x, y, OBJECT_TYPEOUT_LABEL );
    if ( a == NULL )
    {
        Error("Could not make an actuator! Exiting...\n");
        exit_module();
    }
    a -> labeltype = PNL_LABEL_TOP_LEFT;
    a -> upfunc = (PNL_AFUNC) select_object;
    PNL_ACCESS(Typeout, a, size) = OBJECT_BUF_SIZE;
    PNL_ACCESS(Typeout, a, lin) = OBJECT_TYPEOUT_LINES;
    PNL_ACCESS(Typeout, a, col) = OBJECT_TYPEOUT_COLS;
    PNL_ACCESS(Typeout, a, delimstr) = "\n";
    PNL_ACCESS(Typeout, a, mode) |= PNL_TOM_NOCURSOR;
    pnl_addsubact(a, f);
    if ( PNL_ACCESS(Typeout, a, buf) == NULL ) return NULL;

    /*---------------------- Data Info Typeout ----------------------*/

    x += 6.5;
    y = Y_ORIGIN - 6.15;

    main_acts.data_info_typeout =
    a = make_actuator( pnl_typeout, x, y, DATA_INFO_LABEL );
    if ( a == NULL )
    {
        Error("Could not make an actuator! Exiting...\n");
        exit_module();
    }
    a -> labeltype = PNL_LABEL_TOP_LEFT;
    PNL_ACCESS(Typeout, a, size) = DATA_INFO_BUF_SIZE;
    PNL_ACCESS(Typeout, a, lin) = DATA_INFO_LINES;
    PNL_ACCESS(Typeout, a, col) = DATA_INFO_COLS;
    PNL_ACCESS(Typeout, a, delimstr) = "\n";
    PNL_ACCESS(Typeout, a, mode) |= PNL_TOM_NOCURSOR;
    PNL_ACCESS(Typeout, a, mode) |= PNL_TOM_NOREGION;
    pnl_addact(a, p);
    if ( PNL_ACCESS(Typeout, a, buf) == NULL ) return NULL;

    /*-------------------------- Looping Label ----------------------*/

    x = X_ORIGIN;
    y -= Y_BUTTON_INC;

    main_acts.looping_label =
    a = make_actuator( pnl_label, x, y, LOOPING_LABEL );
    if ( a == NULL )
    {
        Error("Could not make an actuator! Exiting...\n");
        exit_module();
    }
    pnl_addact(a, p);

    /*-------------------------- Loop Buttons ----------------------*/

    y -= 0.5 * (1 + LOOP_BUTTONS_ROWS) + SPACE;
```

```
        main_acts.loop_buttons_group =
                make_button_group(      p,
                                        x,
                                        y,
                                        LOOP_BUTTONS_FRAMED,
                                        LOOP_BUTTONS_RADIO_GROUPING,
                                        LOOP_BUTTONS_ROWS - 1 /* HACK */,
                                        loop_buttons_per_row,
                                        loop_buttons_types,
                                        loop_buttons_labels,
                                        loop_buttons_row_labels,
                                        NULL,
                                        loop_buttons_selections,
                                        loop_buttons_func       );

        if (main_acts.loop_buttons_group == NULL)
        {
                Error("Could not make actuators! Exiting...\n");
                exit_module();
        }


        /*------------------------- Slider Buttons ----------------------*/

        y -= 0.5 * (1 + SLIDER_BUTTONS_ROWS) + SPACE;

        main_acts.slider_buttons_group =
                make_button_group(      p,
                                        x,
                                        y,
                                        SLIDER_BUTTONS_FRAMED,
                                        SLIDER_BUTTONS_RADIO_GROUPING,
                                        SLIDER_BUTTONS_ROWS,
                                        slider_buttons_per_row,
                                        slider_buttons_types,
                                        slider_buttons_labels,
                                        slider_buttons_row_labels,
                                        NULL,
                                        slider_buttons_selections,
                                        slider_buttons_func     );

        if (main_acts.slider_buttons_group == NULL)
        {
                Error("Could not make actuators! Exiting...\n");
                exit_module();
        }


        /*------------------------- Surface Buttons ----------------------*/

        y -= 0.5 * (1 + SURFACE_BUTTONS_ROWS) + SPACE;

        main_acts.surface_buttons_group =
                make_button_group(      p,
                                        x,
                                        y,
                                        SURFACE_BUTTONS_FRAMED,
                                        SURFACE_BUTTONS_RADIO_GROUPING,
                                        SURFACE_BUTTONS_ROWS,
                                        surface_buttons_per_row,
                                        surface_buttons_types,
                                        surface_buttons_labels,
                                        surface_buttons_row_labels,
                                        NULL,
                                        surface_buttons_selections,
                                        surface_buttons_func    );

        if (main_acts.surface_buttons_group == NULL)
        {
                Error("Could not make actuators! Exiting...\n");
                exit_module();
        }


        /*------------------------ Zone Typein Group ---------------------*/

        x += 9.5;

        main_acts.zone_typein_group =
                make_typein_group(      p,
                                        NULL,
                                        x,
                                        y,
                                        INT_TYPE,
                                        ZONE_TYPEIN_FORMAT,
                                        ZONE_TYPEIN_WIDTH,
                                        zone_typein_values,
                                        ZONE_TYPEIN_LIMITED,
                                        ZONE_TYPEIN_LABEL,
                                        ZONE_TYPEIN_LABEL_TYPE,
                                        zone_func               );

        if (main_acts.zone_typein_group == NULL)
        {
                Error("Could not make actuators! Exiting...\n");
                exit_module();
        }


        /*------------------------ Slider Group -------------------------*/

        x = X_ORIGIN;
        y -= 8.25;

        main_acts.slider_group =
                make_slider_group(      p,
                                        x,
                                        y,
                                        SLIDER_GROUP_FRAMED,
                                        SLIDER_GROUP_SELECTION_BUTTONS,
                                        SLIDER_GROUP_TYPE,
                                        slider_group_values,
                                        slider_group_label_letters,
                                        slider_group_direction_label,
                                        ijk_ranges_func,
                                        direction_func,
                                        boundary_surfaces_func  );

        if (main_acts.slider_group == NULL)
        {
                Error("Could not make actuators! Exiting...\n");
                exit_module();
        }

        /* hide the selection buttons until we enter multi surface mode */
```



```
        ACCESS2(main_acts.slider_group, hide_select_buttons);

        /* highlight the middle slider */

        ACCESS3(main_acts.slider_group, highlight_slider, MID);

        /************ create first panel **********************************/

        firstPanel(p,X_ORIGIN,Y_ORIGIN);

        return p;
}


/*------------------------- END OF main_panel -------------------------------*/




/*++ static Panel* minmax_panel( char* title, int win_x, int win_y )
 *
 * PURPOSE:
 *
 *              Creates and displays the scalar minmax panel for first.
 *              The panel is titled if title is not NULL, and is positioned at the
 *              screen coordinates (win_x, win_y).
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              6/89
 *
 * INPUT PARAMETERS:
 *
 *              char*   title                   title for panel window
 *              int     win_x                   initial x position of window
 *              int     win_y                   initial y position of window
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              Panel*  p                       a pointer to the panel
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Minmax_Acts      minmax_acts;    declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      extern void     close_parent_panel()            libpanu
 *      extern Panel*   make_panel()                    libpanu
 *      extern Actuator*        make_actuator()         libpanu
 *
 *      void            set_minmax_func()               defined in this file
 *      void            reset_minmax()                  defined in this file
 *      void            adjust_minmax_func()            defined in this file
 *      void            set_minmax_modes()              defined in this file
 *
--*/

/*------------------------------ minmax_panel -------------------------------*/

static Panel* minmax_panel( char* title, int win_x, int win_y )
{
        Panel*          p;                      /* tmp panel            */
        Actuator*       a;                      /* tmp actuator         */
        Actuator*       ta;                     /* tmp actuator         */
        Actuator*       f;                      /* frame actuator       */
        float           x;                      /* act x position       */
        float           y;                      /* act y position       */
        float           h;                      /* act height           */
        float           w;                      /* act height           */
        int             i;                      /* looping index        */
        float           y_inc;                  /* y position increment */
        float           y_save;                 /* save y position      */
        static char     clip_ms_label[2][16];   /* clip multislider     */
        static char     norm_ms_label[2][16];   /* norm multislider     */
        static char     legend_labels[NUM_LEGEND_VALUES][16];
                                                /* legend values        */



        /*------------------------ Minmax Panel ------------------------*/

        p = pnl_mkpanel();
        p -> label = title;
        p -> x = (long) win_x;
        p -> y = (long) win_y;
        p -> ppu = 36.0;
        p -> visible = 0;

        x = X_ORIGIN;
        y = Y_ORIGIN;

        /*------------------- window close button ---------------*/

        a = pnl_mkact(pnl_wide_button);
        a -> label = "Close";
        a -> x = x;
        a -> y = y;
        a -> upfunc = (PNL_AFUNC) close_parent_panel;
        pnl_addact(a, p);
```

```
/*--------- Minmax Frame ------------*/

y -= SMM_FRAME_HEIGHT;

f = pnl_mkact(pnl_frame);
f -> x = x;
f -> y = y;
PNL_ACCESS(frame, f, mode) |= PNL_FM_FREE;
pnl_addact(f, p);

minmax_acts.minmax_frame = f;


/*------- Minmax Modes -------*/

x = X_FRAME_ACT;
y = Y_FRAME_ACT;

a = pnl_mkact(pnl_toggle_button);
a -> label = AUTO_MINMAX_UPDATE_LABEL;
a -> x = x;
a -> y = y;
a -> val = 1.0;
a -> u = AUTO_MINMAX_UPDATE_ID;
a -> upfunc = (PNL_AFUNC) set_minmax_modes;
pnl_addsubact(a, f);

minmax_acts.auto_minmax_update_button = a;

y -= 0.75;

a = pnl_mkact(pnl_toggle_button);
a -> label = UPDATE_MMSLIDERS_LABEL;
a -> x = x;
a -> y = y;
a -> u = UPDATE_MMSLIDERS_ID;
a -> val = 1.0;
a -> upfunc = (PNL_AFUNC) set_minmax_modes;
pnl_addsubact(a, f);

minmax_acts.update_minmax_sliders_button = a;

x += 6.0;
y += 0.75;

a = pnl_mkact(pnl_radio_button);
a -> label = MULTI_ZONE_MINMAX_LABEL;
a -> x = x;
a -> y = y;
a -> u = MULTI_ZONE_MINMAX_ID;
a -> val = 1.0;
a -> upfunc = (PNL_AFUNC) set_minmax_modes;
pnl_addsubact(a, f);

minmax_acts.mode_buttons[MULTI_ZONE_MINMAX] = a;

y -= 0.75;

a = pnl_mkact(pnl_radio_button);
a -> label = SINGLE_ZONE_MINMAX_LABEL;
a -> x = x;
a -> y = y;
a -> u = SINGLE_ZONE_MINMAX_ID;
a -> upfunc = (PNL_AFUNC) set_minmax_modes;
pnl_addsubact(a, f);

minmax_acts.mode_buttons[SINGLE_ZONE_MINMAX] = a;

y -= 0.75;

a = pnl_mkact(pnl_radio_button);
a -> label = SUBSET_MINMAX_LABEL;
a -> x = x;
a -> y = y;
a -> u = SUBSET_MINMAX_ID;
a -> upfunc = (PNL_AFUNC) set_minmax_modes;
pnl_addsubact(a, f);

minmax_acts.mode_buttons[SUBSET_MINMAX] = a;

y -= 0.75;

a = pnl_mkact(pnl_radio_button);
a -> label = SURFACE_MINMAX_LABEL;
a -> x = x;
a -> y = y;
a -> u = SURFACE_MINMAX_ID;
a -> upfunc = (PNL_AFUNC) set_minmax_modes;
pnl_addsubact(a, f);

minmax_acts.mode_buttons[SURFACE_MINMAX] = a;

y -= 0.75;

a = pnl_mkact(pnl_radio_button);
a -> label = SURFACE_SUBSET_MINMAX_LABEL;
a -> x = x;
a -> y = y;
a -> u = SURFACE_SUBSET_MINMAX_ID;
a -> upfunc = (PNL_AFUNC) set_minmax_modes;
pnl_addsubact(a, f);

minmax_acts.mode_buttons[SURFACE_SUBSET_MINMAX] = a;

pnl_endgroup(p);


x = X_FRAME_ACT;

/*------- Invert Clip Test Button ----------*/

a = pnl_mkact(pnl_toggle_button);
a -> label = INVERT_CLIP_TEST_LABEL;
a -> x = x;
a -> y = y;
a -> upfunc = (PNL_AFUNC) invert_clip_test;
pnl_addsubact(a, f);

minmax_acts.invert_clip_test_button  = a;

y -= 1.0;

/*------- Clip, Norm & Legend Labels -------*/

a = pnl_mkact(pnl_label);
a -> label = CLIP_LABEL;
a -> x = x;
```



```
a -> y = y;
pnl_addsubact(a, f);

minmax_acts.clip_label = a;

x += PALETTE_X_INC;

a = pnl_mkact(pnl_label);
a -> label = NORM_LABEL;
a -> x = x;
a -> y = y;
pnl_addsubact(a, f);

minmax_acts.norm_label = a;

x += PALETTE_X_INC;

a = pnl_mkact(pnl_label);
a -> label = LEGEND_LABEL;
a -> x = x;
a -> y = y;
pnl_addsubact(a, f);

minmax_acts.legend_label = a;

/*----- Clip, Norm & Legend Top/Max Typeins -----*/

x = X_FRAME_ACT;
y -= 1.0;

a = pnl_mkact(pnl_typein);
a -> labeltype = PNL_LABEL_LEFT;
a -> label = CLIP_TOP_LABEL;
a -> x = x;
a -> y = y;
a -> u = CLIP_TOP_ID;
a -> upfunc = (PNL_AFUNC) set_minmax_func;
PNL_ACCESS(Typein, a, len) = FLOAT_STRING_WIDTH;
PNL_ACCESS(Typein, a, str) = CLIP_TOP_NUMBER_STR;
pnl_addsubact(a, f);

minmax_acts.clip_top_typein = a;

x += PALETTE_X_INC;

a = pnl_mkact(pnl_typein);
a -> labeltype = PNL_LABEL_LEFT;
a -> label = NORM_TOP_LABEL;
a -> x = x;
a -> y = y;
a -> u = NORM_TOP_ID;
a -> upfunc = (PNL_AFUNC) set_minmax_func;
PNL_ACCESS(Typein, a, len) = FLOAT_STRING_WIDTH;
PNL_ACCESS(Typein, a, str) = NORM_TOP_NUMBER_STR;
pnl_addsubact(a, f);

minmax_acts.norm_top_typein = a;

x += PALETTE_X_INC;

a = pnl_mkact(pnl_typein);
a -> labeltype = PNL_LABEL_LEFT;
a -> label = LEGEND_MAX_LABEL;
a -> x = x;
a -> y = y;
a -> u = LEGEND_MAX_ID;
a -> upfunc = (PNL_AFUNC) set_minmax_func;
PNL_ACCESS(Typein, a, len) = FLOAT_STRING_WIDTH;
PNL_ACCESS(Typein, a, str) = LEGEND_MAX_NUMBER_STR;
pnl_addsubact(a, f);

minmax_acts.legend_max_typein = a;

/*------ Clip & Norm Multisliders ------*/

x = X_FRAME_ACT;
y -= MULTISLIDER_HEIGHT + 0.5;
w = MULTISLIDER_WIDTH;
h = MULTISLIDER_HEIGHT;

/* store the labels for the Clip multislider */

for (i = 0; i < 2; ++i)
{
        sprintf(clip_ms_label[i], INT_STRING_FORMAT, 0);
}

a = pnl_mkact(pnl_multislider);
a -> x = x;
a -> y = y;
a -> w = w;
a -> h = h;
a -> u = CLIP_MSLIDER_ID;
a -> activefunc = (PNL_AFUNC) adjust_minmax_func;
a -> minval = 0.0;
a -> maxval = 1.0;
PNL_ACCESS(Multislider, a, n) = 2;
PNL_ACCESS(Multislider, a, mode) = PNL_MSM_CONSTRAINED;
pnl_addsubact(a, f);

ta = a -> al;
ta -> extval = 1.0;
ta -> label = clip_ms_label[0];
ta = ta -> next;
ta -> extval = 0.0;
ta -> label = clip_ms_label[1];
pnl_fixact(ta);

minmax_acts.clip_multislider = a;

x += PALETTE_X_INC;

/* store the labels for the Norm multislider */

for (i = 0; i < 2; ++i)
{
        sprintf(norm_ms_label[i], INT_STRING_FORMAT, 0);
}

a = pnl_mkact(pnl_multislider);
a -> x = x;
a -> y = y;
a -> w = w;
a -> h = h;
a -> minval = 0.0;
a -> maxval = 1.0;
a -> u = NORM_MSLIDER_ID;
a -> activefunc = (PNL_AFUNC) adjust_minmax_func;
```

```
    PNL_ACCESS(Multislider, a, n) = 2;
    PNL_ACCESS(Multislider, a, mode) = PNL_MSM_CONSTRAINED;
    pnl_addsubact(a, f);

    ta = a -> al;
    ta -> extval = 1.0;
    ta -> label = norm_ms_label[0];
    ta = ta -> next;
    ta -> extval = 0.0;
    ta -> label = norm_ms_label[1];
    pnl_fixact(ta);

    minmax_acts.norm_multislider = a;


    /*------ Clip, Norm & Legend Palettes ------*/

    x = X_FRAME_ACT - PALETTE_WIDTH;

    /* Clip LOW */

    a = pnl_mkact(pnl_palette);
    a -> x = x;
    a -> y = y;
    a -> w = PALETTE_WIDTH;
    a -> h = 0.0;
    a -> minval = MIN_MAP;
    a -> maxval = MIN_MAP;
    pnl_addsubact(a, f);

    minmax_acts.palettes[CLIP][LOW] = a;

    /* Clip MED */

    a = pnl_mkact(pnl_palette);
    a -> x = x;
    a -> y = y;
    a -> w = PALETTE_WIDTH;
    a -> h = PALETTE_HEIGHT;
    a -> minval = MIN_MAP;
    a -> maxval = MAX_MAP;
    pnl_addsubact(a, f);

    minmax_acts.palettes[CLIP][MED] = a;

    /* Clip HI */

    a = pnl_mkact(pnl_palette);
    a -> x = x;
    a -> y = y + PALETTE_HEIGHT;
    a -> w = PALETTE_WIDTH;
    a -> h = 0.0;
    a -> minval = MAX_MAP;
    a -> maxval = MAX_MAP;
    pnl_addsubact(a, f);

    minmax_acts.palettes[CLIP][HI] = a;

    x += PALETTE_X_INC;

    /* Norm LOW does not exist */

    minmax_acts.palettes[NORM][LOW] = NULL;

    /* Norm MED */

    a = pnl_mkact(pnl_palette);
    a -> x = x;
    a -> y = y;
    a -> w = PALETTE_WIDTH;
    a -> h = PALETTE_HEIGHT;
    a -> minval = MIN_MAP;
    a -> maxval = MAX_MAP;
    pnl_addsubact(a, f);

    minmax_acts.palettes[NORM][MED] = a;

    /* Norm HI does not exist */

    minmax_acts.palettes[NORM][HI] = NULL;

    x += PALETTE_X_INC;

    /* Legend LOW */

    a = pnl_mkact(pnl_palette);
    a -> x = x;
    a -> y = y;
    a -> w = PALETTE_WIDTH;
    a -> h = 0.0;
    a -> minval = MIN_MAP;
    a -> maxval = MIN_MAP;
    pnl_addsubact(a, f);

    minmax_acts.palettes[LEGEND][LOW] = a;

    /* Legend MED */

    a = pnl_mkact(pnl_palette);
    a -> x = x;
    a -> y = y;
    a -> w = PALETTE_WIDTH;
    a -> h = PALETTE_HEIGHT;
    a -> minval = MIN_MAP;
    a -> maxval = MAX_MAP;
    pnl_addsubact(a, f);

    minmax_acts.palettes[LEGEND][MED] = a;

    /* Legend HI */

    a = pnl_mkact(pnl_palette);
    a -> x = x;
    a -> y = y + PALETTE_HEIGHT;
    a -> w = PALETTE_WIDTH;
    a -> h = 0.0;
    a -> minval = MAX_MAP;
    a -> maxval = MAX_MAP;
    pnl_addsubact(a, f);

    minmax_acts.palettes[LEGEND][HI] = a;


    /*----- Legend Number labels -----*/

    x += PALETTE_WIDTH + MULTISLIDER_WIDTH + 0.1;
    y_inc = h / (float) (NUM_LEGEND_VALUES - 1);
    y_save = y;
```



```
    y -= 0.15;

    for (i = 0; i < NUM_LEGEND_VALUES; ++i)
    {
            a = pnl_mkact(pnl_label);
            sprintf(legend_labels[i], FLOAT_STRING_FORMAT, 0.0);
            a -> label = legend_labels[i];
            a -> x = x;
            a -> y = y;
            pnl_addsubact(a, f);

            minmax_acts.legend_labels[i] = a;

            y += y_inc;
    }


    /*----- Clip, Norm & Legend Bot/Min Typeins -----*/

    x = X_FRAME_ACT;
    y = y_save - 1.0;

    a = pnl_mkact(pnl_typein);
    a -> labeltype = PNL_LABEL_LEFT;
    a -> label = CLIP_BOT_LABEL;
    a -> x = x;
    a -> y = y;
    a -> u = CLIP_BOT_ID;
    a -> upfunc = (PNL_AFUNC) set_minmax_func;
    PNL_ACCESS(Typein, a, len) = FLOAT_STRING_WIDTH;
    PNL_ACCESS(Typein, a, str) = CLIP_BOT_NUMBER_STR;
    pnl_addsubact(a, f);

    minmax_acts.clip_bot_typein = a;

    x += PALETTE_X_INC;

    a = pnl_mkact(pnl_typein);
    a -> labeltype = PNL_LABEL_LEFT;
    a -> label = NORM_BOT_LABEL;
    a -> x = x;
    a -> y = y;
    a -> u = NORM_BOT_ID;
    a -> upfunc = (PNL_AFUNC) set_minmax_func;
    PNL_ACCESS(Typein, a, len) = FLOAT_STRING_WIDTH;
    PNL_ACCESS(Typein, a, str) = NORM_BOT_NUMBER_STR;
    pnl_addsubact(a, f);

    minmax_acts.norm_bot_typein = a;

    x += PALETTE_X_INC;

    a = pnl_mkact(pnl_typein);
    a -> labeltype = PNL_LABEL_LEFT;
    a -> label = LEGEND_MIN_LABEL;
    a -> x = x;
    a -> y = y;
    a -> u = LEGEND_MIN_ID;
    a -> upfunc = (PNL_AFUNC) set_minmax_func;
    PNL_ACCESS(Typein, a, len) = FLOAT_STRING_WIDTH;
    PNL_ACCESS(Typein, a, str) = LEGEND_MIN_NUMBER_STR;
    pnl_addsubact(a, f);

    minmax_acts.legend_min_typein = a;

    /*-------- Clip, Norm & Legend Reset Buttons ------*/

    x = X_FRAME_ACT;
    y -= 1.0;

    a = pnl_mkact(pnl_wide_button);
    a -> label = RESET_CLIP_LABEL;
    a -> x = x;
    a -> y = y;
    a -> u = RESET_CLIP_ID;
    a -> upfunc = (PNL_AFUNC) reset_minmax;
    pnl_addsubact(a, f);

    minmax_acts.reset_clip_button = a;

    x += PALETTE_X_INC;

    a = pnl_mkact(pnl_wide_button);
    a -> label = RESET_NORM_LABEL;
    a -> x = x;
    a -> y = y;
    a -> u = RESET_NORM_ID;
    a -> upfunc = (PNL_AFUNC) reset_minmax;
    pnl_addsubact(a, f);

    minmax_acts.reset_norm_button = a;

    x += PALETTE_X_INC;

    a = pnl_mkact(pnl_wide_button);
    a -> label = RESET_LEGEND_LABEL;
    a -> x = x;
    a -> y = y;
    a -> u = RESET_LEGEND_ID;
    a -> upfunc = (PNL_AFUNC) reset_minmax;
    pnl_addsubact(a, f);

    minmax_acts.reset_legend_button = a;


    return p;
}

/*---------------------------- END OF minmax_panel ---------------------------*/



/*++ static Panel* contour_panel( char* title, int win_x, int win_y )
 *
 * PURPOSE :
 *
 *             The panel is titled if title is not NULL, and is positioned at
 *             the screen coordinates (win_x, win_y).
 *
```

```
* AUTHORS :
*
*             Paul Kelaita
*             NASA Ames Research Center
*             Sterling Software
*
* REVISION HISTORY :
*
*             12/90
*
* INPUT PARAMETERS :
*
*             char*   title                   title for panel window
*             int     win_x                   initial x position of window
*             int     win_y                   initial y position of window
*
* OUTPUT PARAMETERS :
*
*             None
*
*
* FUNCTION RETURN :
*
*             Panel*  p                       a pointer to the panel
*
* GLOBAL VARIABLES USED :
*
*
* FILES USED :
*
*             None
*
* NOTES :
*
* NON-STANDARD CODE :
*
* CALLED BY :
*
* FUNCTIONS CALLED :
*
*       extern void     close_parent_panel()            libpanu
*
*
--*/

/*------------------------------ contour_panel -------------------------------*/

static Panel* contour_panel( char* title, int win_x, int win_y )
{
        Panel*          p;                      /* tmp panel            */
        Actuator*       a;                      /* tmp actuator         */
        float           x;                      /* act x position       */
        float           y;                      /* act y position       */
        int             i;                      /* looping index        */
        float           y_inc;                  /* y position increment */
        float           y_save;                 /* save y position      */
        static char     contour_labels[NUM_LEGEND_VALUES][16];
                                                /* legend values        */



        /*----------------------- FISST Contour Panel -----------------------*/

        p = pnl_mkpanel();

        p -> label = title;
        p -> x = (long) win_x;
        p -> y = (long) win_y;
        p -> ppu = 36.0;
        p -> visible = 0;

        x = X_ORIGIN + 1.5;
        y = Y_ORIGIN + 5.0;

        /*------------------ window close button ---------------*/

        a = pnl_mkact(pnl_wide_button);
        a -> label = "Close";
        a -> x = x;
        a -> y = y;
        a -> upfunc = (PNL_AFUNC) close_parent_panel;
        pnl_addact(a, p);

        /*------------------ contour legend ---------------*/

        x = X_ORIGIN + 3.0;
        y = Y_ORIGIN - 6.0;

        a = pnl_mkact(pnl_palette);
        a -> x = x;
        a -> y = y;
        a -> w = PALETTE_WIDTH;
        a -> h = PALETTE_HEIGHT;
        a -> minval = MIN_MAP;
        a -> maxval = MAX_MAP;
        pnl_addact(a, p);

        contour_acts.palette = a;

        /*---------------- Legend Number labels ---------------*/

        x += PALETTE_WIDTH + MULTISLIDER_WIDTH + 0.1;
        y_inc = MULTISLIDER_HEIGHT / (float) (NUM_LEGEND_VALUES - 1);
        y_save = y;
        y -= 0.15;

        for (i = 0; i < NUM_LEGEND_VALUES; ++i)
        {
                a = pnl_mkact(pnl_label);
                sprintf(contour_labels[i], FLOAT_STRING_FORMAT, 0.0);
                a -> label = contour_labels[i];
                a -> x = x;
                a -> y = y;
                pnl_addact(a, p);

                contour_acts.contour_labels[i] = a;

                y += y_inc;
        }

        /*---------------- Contour Minimum/Maximum Typeins ----------------*/

        x -= 3.0;
        y = y_save - 1.0;

        a = pnl_mkact(pnl_typein);
        a -> labeltype = PNL_LABEL_BOTTOM;
        a -> label = "Contour Min";
        a -> x = x;
```



```
        a -> y = y;
        a -> u = CONTOUR_MIN_ID;
        a -> downfunc = (PNL_AFUNC) clear_typein;
        a -> upfunc = (PNL_AFUNC) set_contour_legend;
        PNL_ACCESS(Typein, a, len) = FLOAT_STRING_WIDTH;
        PNL_ACCESS(Typein, a, str) = LEGEND_MIN_NUMBER_STR;
        pnl_addact(a, p);

        contour_acts.contour_min_typein = a;

        y += MULTISLIDER_HEIGHT + 1.5;

        a = pnl_mkact(pnl_typein);
        a -> labeltype = PNL_LABEL_TOP;
        a -> label = "Contour Max";
        a -> x = x;
        a -> y = y;
        a -> u = CONTOUR_MAX_ID;
        a -> downfunc = (PNL_AFUNC) clear_typein;
        a -> upfunc = (PNL_AFUNC) set_contour_legend;
        PNL_ACCESS(Typein, a, len) = FLOAT_STRING_WIDTH;
        PNL_ACCESS(Typein, a, str) = LEGEND_MAX_NUMBER_STR;
        pnl_addact(a, p);

        contour_acts.contour_max_typein = a;

        /*--------------- Contours/Increment Typeins ---------------*/

        x = X_ORIGIN + 3.0;
        y = Y_ORIGIN + 3.5;

        a = pnl_mkact(pnl_typein);
        a -> labeltype = PNL_LABEL_TOP;
        a -> label = "Number of Contours";
        a -> x = x + 1.0;
        a -> y = y;
        a -> u = CONTOUR_NUM_ID;
        a -> downfunc = (PNL_AFUNC) clear_typein;
        a -> upfunc = (PNL_AFUNC) set_contour_legend;
        PNL_ACCESS(Typein, a, len) = INT_STRING_WIDTH;
        PNL_ACCESS(Typein, a, str) = "40";
        pnl_addact(a, p);

        contour_acts.contour_num_typein = a;

        y -= 1.5;

        a = pnl_mkact(pnl_typein);
        a -> labeltype = PNL_LABEL_TOP;
        a -> label = "Increment of Contours";
        a -> x = x;
        a -> y = y;
        a -> u = CONTOUR_INC_ID;
        a -> upfunc = (PNL_AFUNC) set_contour_legend;
        PNL_ACCESS(Typein, a, len) = FLOAT_STRING_WIDTH;
        PNL_ACCESS(Typein, a, str) = ZERO_STRING_16;
        pnl_addact(a, p);

        contour_acts.contour_inc_typein = a;

        return p;

}

/*----------------------- END OF contour_panel ------------------------*/



/*++ static void set_contour_legend( char* str )
*
* PURPOSE :
*
*             Sets the contours legend according to min, max, numcontours,
*             and increment
*
*
* AUTHORS :
*
*             Paul Kelaita
*             NASA Ames Research Center
*             Sterling Software
*
* REVISION HISTORY :
*
*             12/90
*             5/91            added scripting
*
* INPUT PARAMETERS :
*
*             char*   str             command/actuator
*
* OUTPUT PARAMETERS :
*
*             None
*
* FUNCTION RETURN :
*
*             None
*
* GLOBAL VARIABLES USED :
*
*             extern Minmax_Acts      minmax_acts     defined in this file
*             extern Grid_Surface*    grid_fisst      defined in this file
*
* FILES USED :
*
*             None
*
* NOTES :
*
*             Requires that math.h be included for atof()
*
* NON-STANDARD CODE :
*
* CALLED BY :
*
* FUNCTIONS CALLED :
```

```
 *
 *      void            update_legend()             defined in this file
 *      int             lock_cur_object()           defined in this file
 *      int             unlock_cur_object()         defined in this file
 *
 --*/

/*---------------------------- set_contour_legend ----------------------------*/

static void set_contour_legend( char* str )
{

#define IS_REMAINDER(max, min, inc)     \
        (((max - min)/inc) == (int)((max - min)/inc) ? 0 : 1)

        Actuator*       a;

        char*   min_str;
        char*   max_str;
        char*   numc_str;
        char*   inc_str;

        float   inc;
        float   t1, t2;
        int     numc;
        char    mode[16];
        int     inc_flag, minmax_flag;


        if (is_act(str))
        {
           a = (Actuator*) str;
           if (a == contour_acts.contour_min_typein ||
               a == contour_acts.contour_max_typein   )
           {
              min_str = PNL_ACCESS(Typein,contour_acts.contour_min_typein, str);
              max_str = PNL_ACCESS(Typein,contour_acts.contour_max_typein, str);
              load_command("CONTOUR %s %f %f", "MINMAX",
                                        atof(min_str), atof(max_str));
           }
           else if (a == contour_acts.contour_num_typein)
           {
              load_command("CONTOUR %s %d", "NUMBER",
                                atoi(PNL_ACCESS(Typein, a, str)));
           }
           else if (a == contour_acts.contour_inc_typein)
           {
              load_command("CONTOUR %s %f", "INC",
                                atof(PNL_ACCESS(Typein, a, str)));
           }
           return;
        }

        parse_command(str, "%s", mode);

        /* grab all the strings first */

        min_str = PNL_ACCESS(Typein, contour_acts.contour_min_typein, str);

        max_str = PNL_ACCESS(Typein, contour_acts.contour_max_typein, str);

        numc_str = PNL_ACCESS(Typein, contour_acts.contour_num_typein, str);

        inc_str = PNL_ACCESS(Typein, contour_acts.contour_inc_typein, str);

        inc_flag = minmax_flag = 0;

        if (strcasecmp(mode, "MINMAX") == 0)
        {
           minmax_flag  = 1;
           parse_command(str, "%s %f %f", mode, &t1, &t2);
           sprintf(min_str, FLOAT_STRING_FORMAT, t1);
           sprintf(max_str, FLOAT_STRING_FORMAT, t2);
           pnl_fixat(contour_acts.contour_min_typein);
           pnl_fixat(contour_acts.contour_max_typein);
        }
        else if (strcasecmp(mode, "NUMBER") == 0)
        {
           parse_command(str, "%s %d", mode, &numc);
           sprintf(numc_str, INT_STRING_FORMAT, numc);
           pnl_fixat(contour_acts.contour_num_typein);
        }
        else if (strcasecmp(mode, "INC") == 0)
        {
           inc_flag     = 1;
           parse_command(str, "%s %f", mode, &t1);
           sprintf(inc_str, FLOAT_STRING_FORMAT, t1);
           pnl_fixat(contour_acts.contour_inc_typein);
        }

        /*
           get validity of values
        */
        if ((int)atof(numc_str) < 2)
           sprintf(numc_str, INT_STRING_FORMAT, 2);

        if (atof(inc_str) <= 0.0)
           sprintf(inc_str, FLOAT_STRING_FORMAT, 0.1);

        /*
           begin checks
        */
        if (minmax_flag)
        {
           update_legend(atof(min_str), atof(max_str),
                         contour_acts.contour_labels);
        }

        else if (inc_flag)
        {
           numc = (int)((atof(max_str) - atof(min_str)) / atof(inc_str)) + 1 +
                  IS_REMAINDER(atof(max_str), atof(min_str), atof(inc_str));
           sprintf(numc_str, INT_STRING_FORMAT, numc);
           pnl_fixat(contour_acts.contour_num_typein);
        }

        /* update the increment no matter what */

        if (!inc_flag)
        {
           inc  = (atof(max_str) - atof(min_str)) / (atof(numc_str) - 1.0);
           sprintf(inc_str, FLOAT_STRING_FORMAT, inc);
           pnl_fixat(contour_acts.contour_inc_typein);
        }

        lock_cur_object();
```



```
        grid_first -> num_contours      = atoi(numc_str);
        grid_first -> contours_inc      = atof(inc_str);
        grid_first -> contours_min      = atof(min_str);
        grid_first -> contours_max      = atof(max_str);

        delete_contours();

        unlock_cur_object();

}

/*------------------------- END OF set_contour_legend ----------------------*/
```

```
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              None
 *
 --*/

/*------------------------------ file_io_func ------------------------------*/

static void file_io_func( char* file_name, int mode )
{
        Message("file_io_func: Unimplemented feature...");
}

/*------------------------ END OF file_io_func ----------------------------*/
```

```
/*++ static void file_io_func( char* file_name, int mode )
 *
 * PURPOSE:
 *
 *              Call back for save/restore file io.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              6/91
 *
 * INPUT PARAMETERS:
 *
 *              char*           file_name       pathed file name to read/write
 *              int             mode            0 = read, 1 = write
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *              None
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
```

```
/*++ static void panels_func( int group, int item )
 *
 * PURPOSE:
 *
 *              Handles the panels menu functions.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              12/90
 *
 * INPUT PARAMETERS:
 *
 *              int             group           menu group number
 *              int             item            item number within group
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
```

```
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *
 --*/

/*------------------------------ panels_func ---------------------------------*/

static void panels_func( int group, int item )
{
        load_command(   panels_menu_script_commands[0],
                        panels_menu_script_commands[item + 1]   );
}

/*---------------------------- END OF panels_func ----------------------------*/




/*++ static void attributes_func( int group, int item )
 *
 * PURPOSE:
 *
 *              Handles the attributes menu functions.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              6/89
 *              12/90   converted to a menu group call-back
 *              4/91    added for command stuff
 *              6/91    rewrote command stuff
 *
 * INPUT PARAMETERS:
 *
 *              int             group           menu group number
 *              int             item            item number within group
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *
 --*/

/*---------------------------- attributes_func -----------------------------*/

static void attributes_func( int group, int item )
{
        int     index = group_item_to_index( group, item,
                        attributes_menu_items_per_group,
                        ATTRIBUTES_MENU_GROUPS  );

        interactive = 1;
        load_command(   attributes_menu_script_commands[0],
                        attributes_menu_script_commands[index + 1]      );
}

/*-------------------------- END OF attributes_func --------------------------*/




/*++ static void set_attributes_func( char* script_command )
 *
 * PURPOSE:
 *
 *              Handles the attributes menu functions from a command
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              6/89
 *              12/90   converted to a menu group call-back
 *              4/91    added command stuff
 *              6/91    rewrote command stuff
 *
 * INPUT PARAMETERS:
 *
 *              char*   script_command
 *
 * OUTPUT PARAMETERS:
 *
 *              None
```



```
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_fisst;             declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              extern int      command_index()         libpanu
 *              void            delete_normals()        defined in this file
 *              int             lock_cur_object()       defined in this file
 *              int             unlock_cur_object()     defined in this file
 *
 --*/

/*---------------------------- set_attributes_func ----------------------------*/

static void set_attributes_func( char* script_command )
{
        char    param[32];
        int     index, item, group;


        parse_command( script_command, "%s", param );

        if ( ( index = command_index( param, attributes_menu_script_commands,
                        ATTRIBUTES_MENU_ITEMS + 1 ) ) == -1 )
        {
                return;
        }

        index_to_group_item( --index, &group, &item,
                attributes_menu_items_per_group, ATTRIBUTES_MENU_GROUPS );


        lock_cur_object();

        switch ( group )
        {
            case 0:
               grid_fisst -> contour_color_type = item;
               if ( ! interactive )
                   ACCESS6(main_acts.attributes_menu_group, set_selection,
                           0, item, 1, 0);
               break;
            case 1:
               grid_fisst -> vector_color_type  = item;
               if ( ! interactive )
                   ACCESS6(main_acts.attributes_menu_group, set_selection,
                           1, item, 1, 0);
               break;
            case 2:
               grid_fisst -> vector_tip_type    = item;
               if ( ! interactive )
                   ACCESS6(main_acts.attributes_menu_group, set_selection,
                           2, item, 1, 0);
               break;
            case 3:
               grid_fisst -> clip_mode          = item;
               if ( ! interactive )
                   ACCESS6(main_acts.attributes_menu_group, set_selection,
                           3, item, 1, 0);
               break;
            case 4:
               grid_fisst -> shaded             = item;
               if ( ! interactive )
                   ACCESS6(main_acts.attributes_menu_group, set_selection,
                           4, item, 1, 0);
               if ( IS_SCALAR_COLORED( grid_fisst ) && IS_SHADED( grid_fisst ) )
                  module_command( "Viewer", "SET_COLOR_MODE TRUE_COLOR", NULL );
               break;
            case 5:
               grid_fisst -> framed             = item;
               if ( ! interactive )
                   ACCESS6(main_acts.attributes_menu_group, set_selection,
                           5, item, 1, 0);
               break;
            case 6:
               if (grid_fisst -> rev_normals != item)
                  delete_normals();
               grid_fisst -> rev_normals                = item;
               if ( ! interactive )
                   ACCESS6(main_acts.attributes_menu_group, set_selection,
                           6, item, 1, 0);
               break;
            case 7:
               if (grid_fisst -> zone_normals != item)
                  delete_normals();
               grid_fisst -> zone_normals       = item;
               if ( ! interactive )
                   ACCESS6(main_acts.attributes_menu_group, set_selection,
                           7, item, 1, 0);
               break;
            default:
               break;
        }

        interactive = 0;
        unlock_cur_object();

}

/*------------------------ END OF set_attributes_func --------------------------*/




/*++ static void type_func( int group, int item )
 *
```

```
* PURPOSE:
*
*             Handles the type menu functions for commands
*
* AUTHORS:
*
*             Todd Plessel
*             NASA Ames Research Center
*             Sterling Software
*
* REVISION HISTORY:
*
*             9/89
*             12/90   converted to a menu group call-back
*             4/91    Paul Kelaita -- added command stuff
*             6/91    rewrote command stuff
*
* INPUT PARAMETERS:
*
*             int             group           menu group number
*             int             item            item number within group
*
* OUTPUT PARAMETERS:
*
*             None                            -
*
* FUNCTION RETURN:
*
*             None
*
* GLOBAL VARIABLES USED:
*
*
* FILES USED:
*
*             None
*
* NOTES:
*
* NON-STANDARD CODE :
*
* CALLED BY :
*
* FUNCTIONS CALLED :
*
*
--*/

/*------------------------------- type_func ---------------------------------*/

static void type_func( int group, int item )
{
        interactive = 1;
        load_command(   type_menu_script_commands[0],
                        type_menu_script_commands[item + 1]     );
}

/*---------------------------- END OF type_func -----------------------------*/
```

```
/*++ static void set_type_func( char* script_command )
*
* PURPOSE:
*
*             Handles the type menu functions from a command
*
* AUTHORS:
*
*             Todd Plessel
*             NASA Ames Research Center
*             Sterling Software
*
* REVISION HISTORY:
*
*             9/89
*             12/90   converted to a menu group call-back
*             4/91    Paul Kelaita -- added command stuff
*             6/91    rewrote command stuff
*
* INPUT PARAMETERS:
*
*             char*   script_command          command
*
* OUTPUT PARAMETERS:
*
*             None
*
* FUNCTION RETURN:
*
*             None
*
* GLOBAL VARIABLES USED:
*
*       extern Grid_Surface*    grid_fisst;             declared in this file
*       extern Main_Acts*       main_acts               declared in this file
*       extern Minmax_Acts      minmax_acts             declared in this file
*
* FILES USED:
*
*             None
*
* NOTES:
*
* NON-STANDARD CODE :
*
* CALLED BY :
*
* FUNCTIONS CALLED :
*
*             extern int      command_index()         libpanu
*             void            delete_normals()        defined in this file
*             int             lock_cur_object()       defined in this file
*             int             unlock_cur_object()     defined in this file
*
--*/

/*------------------------------- set_type_func -----------------------------*/

static void set_type_func( char* script_command )
{
```



```
        int             old_type;               /* previous type        */
        char            param[32];
        int             index;


        parse_command( script_command, "%s", param );

        if ( ( index = command_index( param, type_menu_script_commands,
                        TYPE_MENU_ITEMS + 1 ) ) == -1 )
        {
                return;
        }

        --index;


        lock_cur_object();

        /* save the old type */

        old_type = grid_fisst -> type;

        if ( old_type == index )
        {
                unlock_cur_object();
                return;
        }

        /* store new grid surface type and reset the panel if neccessary */

        grid_fisst -> type = index;

        if ( ! interactive )
                ACCESS6( main_acts.type_menu_group, set_selection, 0,
                        grid_fisst -> type, 1, 0 );
        interactive = 0;

        if (grid_fisst -> type != GRID_TYPE)
        {
            /* if in auto update mode then reset the scalar minmax */

            if (minmax_acts.auto_minmax_update_button -> val == 1.0)
            {
                reset_minmax( "RESET_MINMAX LEGEND" );
            }
        }

        /* if neccessary deallocate any normals data */

        if (USES_NORMALS(grid_fisst))
        {
            if (old_type == VECTOR_TYPE ||
                grid_fisst->type == VECTOR_TYPE)
            {
                delete_normals();
            }
        }

        /* if neccessary deallocate any contours data */

        if ( grid_fisst -> render_mode == CONTOUR_LINES )
        {
            delete_contours();
        }
```

```
        /*
         * if the render mode is inappropriate for the type then
         * then change the render mode to lines 1 and 2
         */

        if (((grid_fisst -> render_mode == PLAIN_VECTORS ||
              grid_fisst -> render_mode == NORM_VECTORS) &&
             grid_fisst -> type != VECTOR_TYPE) ||
            (grid_fisst -> render_mode == CONTOUR_LINES &&
            grid_fisst -> type != SCALAR_TYPE))
        {
          ACCESS6(main_acts.render_menu_group, set_selection, 0, LINES_1_2, 1, 0);
          set_render_func( "RENDER GRID_LINES_1_AND_2" );
        }

        unlock_cur_object();

}

/*---------------------------- END OF set_type_func -------------------------*/
```

```
/*++ static void render_func( int group, int item )
*
* PURPOSE:
*
*             Handles the render menu functions.
*
* AUTHORS:
*
*             Todd Plessel
*             NASA Ames Research Center
*             Sterling Software
*
* REVISION HISTORY:
*
*             6/89
*             12/90   converted to a menu group call-back
*             4/91    added command stuff
*             6/91    rewrote command stuff
*
* INPUT PARAMETERS:
*
*             int             group           menu group number
*             int             item            item number within group
*
* OUTPUT PARAMETERS:
*
*             None
*
* FUNCTION RETURN:
*
*             None
*
```

```
 * GLOBAL VARIABLES USED:
 *
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *
 --*/

/*------------------------------- render_func -------------------------------*/

static void render_func( int group, int item )
{
        interactive = 1;
        load_command(   render_menu_script_commands[0],
                        render_menu_script_commands[item + 1]   );
}

/*------------------------- END OF render_func -------------------------------*/




/*++ static void set_render_func( char* script_command )
 *
 * PURPOSE:
 *
 *              Handles the render menu functions from a command
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              6/89
 *              12/90   converted to a menu group call-back
 *              4/91    Paul Kelaita added command stuff
 *              6/91    rewrote command stuff
 *
 * INPUT PARAMETERS:
 *
 *              char*   script_command          command
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_fisst;             declared in this file
 *      extern Main_Acts*       main_acts               declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              extern int      command_index()         libpenu
 *              int             lock_cur_object()       defined in this file
 *              int             unlock_cur_object()     defined in this file
 *              void            delete_contours()       defined in this file
 *
 --*/

/*---------------------------- set_render_func -------------------------------*/

static void set_render_func( char* script_command )
{
        char            param[32];
        int             index;


        parse_command( script_command, "%s", param );

        if ( ( index = command_index( param, render_menu_script_commands,
                        RENDER_MENU_ITEMS + 1 ) ) == -1 )
        {
                return;
        }

        --index;


        lock_cur_object();

        if ( grid_fisst -> render_mode == index )
        {
                unlock_cur_object();
                return;
        }

        grid_fisst -> render_mode = index;

        if ( ! interactive )
                ACCESS6(main_acts.render_menu_group, set_selection, 0,
                        grid_fisst -> render_mode, 1, 0);
        interactive = 0;

        if (grid_fisst -> render_mode == CONTOUR_LINES)
        {
          delete_contours();
```



```
          ACCESS6(main_acts.type_menu_group,set_selection,0,SCALAR_TYPE,1, 0);
          set_type_func( "TYPE SCALAR" );
        }

        /* if a vector type was selected then select vector type */

        if (    grid_fisst -> render_mode == PLAIN_VECTORS ||
                grid_fisst -> render_mode == NORM_VECTORS       )
        {
          ACCESS6(main_acts.type_menu_group,set_selection,0,VECTOR_TYPE,1, 0);
          set_type_func( "TYPE VECTOR" );
        }

        unlock_cur_object();
}

/*------------------------- END OF set_render_func --------------------------*/




/*++ static void options_func( int group, int item )
 *
 * PURPOSE:
 *
 *              Handles the options menu functions.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              6/89
 *              12/90   converted to a menu group call-back
 *
 * INPUT PARAMETERS:
 *
 *              int             group           menu group number
 *              int             item            item number within group
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *
 --*/

/*------------------------------- options_func ------------------------------*/

static void options_func( int group, int item )
{
        int     val;

        interactive = 1;
        lock_cur_object();

        switch (group)
        {
          case 0:
                val = !grid_fisst->draw_outline;
                load_command("OPTIONS %s %s", "OUTLINE", ON_OR_OFF(val));
                break;
          case 1:
                val = !grid_fisst->draw_glyph;
                load_command("OPTIONS %s %s", "GLYPH", ON_OR_OFF(val));
                break;
          case 2:
                load_command("OPTIONS %s", "DELETE_NORMALS");
                break;
          case 3:
                load_command("OPTIONS %s", "RESET");
                break;
          case 4:
                load_command("OPTIONS %s", "DEBUG");
                break;
          default:
                interactive = 0;
                break;
        }

        unlock_cur_object();
}

/*------------------------- END OF options_func ------------------------------*/




/*++ static void set_options_func( char* script_command )
 *
 * PURPOSE:
 *
 *              Handles the options menu functions.
 *
 * AUTHORS:
 *
```

```
*               Todd Plessel
*               NASA Ames Research Center
*               Sterling Software
*
* REVISION HISTORY:
*
*               6/89
*               12/90   converted to a menu group call-back
*               5/91    converted to scripting
*
* INPUT PARAMETERS:
*
*               char*   script_command  command/act
*
* OUTPUT PARAMETERS:
*
*               None
*
* FUNCTION RETURN:
*
*               None
*
* GLOBAL VARIABLES USED:
*
*       extern Grid_Surface*    grid_fisst;             declared in this file
*       extern Main_Acts*       main_acts               declared in this file
*
* FILES USED:
*
*               None
*
* NOTES:
*
* NON-STANDARD CODE :
*
* CALLED BY :
*
* FUNCTIONS CALLED :
*
*       int             lock_cur_object()       defined in this file
*       int             unlock_cur_object()     defined in this file
*
--*/

/*------------------------------ set_options_func ----------------------------*/

static void set_options_func( char* script_command )
{
        char            param[32];
        char            val[8];


        parse_command(script_command, "%s", param);

        lock_cur_object();

        if (strcasecmp(param, "OUTLINE") == 0)
        {
           parse_command(script_command, "%s %s", param, val);
           grid_fisst->draw_outline = ON_OR_OFF_VAL(val);
           ACCESS6(main_acts.options_menu_group, set_selection, 0, 0, grid_fisst->draw_o
utline, 0);
        }
        else if (strcasecmp(param, "GLYPH") == 0)
        {
           parse_command(script_command, "%s %s", param, val);
           grid_fisst->draw_glyph = ON_OR_OFF_VAL(val);
           ACCESS6(main_acts.options_menu_group, set_selection, 1, 0, grid_fisst->draw_gl
yph, 0);
        }
        else if (strcasecmp(param, "DELETE_NORMALS") == 0)
        {
           delete_normals();
        }
        else if (strcasecmp(param, "RESET") == 0)
        {
           reset_state(1);
        }
        else if (strcasecmp(param, "DEBUG") == 0)
        {
           dump_state();
        }

        unlock_cur_object();

}

/*------------------------- END OF set_options_func -------------------------*/




/*++ static void dump_state( void )
*
* PURPOSE:
*
*               Prints the contents of the grid_fisst structure and globals.
*
* AUTHORS:
*
*               Todd Plessel
*               NASA Ames Research Center
*               Sterling Software
*
* REVISION HISTORY:
*
*               6/89
*
* INPUT PARAMETERS:
*
*               None
*
* OUTPUT PARAMETERS:
*
*               None
*
* FUNCTION RETURN:
*
*               None
*
* GLOBAL VARIABLES USED:
```



```
*
*       extern Grid_Surface*    grid_fisst      declared in this file
*
* FILES USED:
*
*               None
*
* NOTES:
*
* NON-STANDARD CODE :
*
* CALLED BY :
*
* FUNCTIONS CALLED :
*
*       int             lock_cur_object()       defined in this file
*       int             unlock_cur_object()     defined in this file
*
--*/

/*------------------------------ dump_state ---------------------------------*/

static void dump_state( void )
{
        lock_cur_object();

        printf("\n");
        printf("object_type                     = %d\n",
                grid_fisst -> object_type);
        printf("draw                            = %d\n",
                grid_fisst -> draw);
        printf("draw_outline                    = %d\n",
                grid_fisst -> draw_outline);
        printf("draw_glyph                      = %d\n",
                grid_fisst -> draw_glyph);
        printf("rev_normals                     = %d\n",
                grid_fisst -> rev_normals);
        printf("zone_normals                    = %d\n",
                grid_fisst -> zone_normals);
        printf("framed                          = %d\n",
                grid_fisst -> framed);
        printf("shaded                          = %d\n",
                grid_fisst -> shaded);
        printf("type                            = %d\n",
                grid_fisst -> type);
        printf("contour_color_type              = %d\n",
                grid_fisst -> contour_color_type);
        printf("vector_color_type               = %d\n",
                grid_fisst -> vector_color_type);
        printf("vector_tip_type                 = %d\n",
                grid_fisst -> vector_tip_type);
        printf("surface_mode                    = %d\n",
                grid_fisst -> surface_mode);
        printf("render_mode                     = %d\n",
                grid_fisst -> render_mode);
        printf("clip_mode                       = %d\n",
                grid_fisst -> clip_mode);
        printf("minmax_mode                     = %d\n",
                grid_fisst -> minmax_mode);
        printf("invert_clip_test                = %d\n",
                grid_fisst -> invert_clip_test);
        printf("auto_minmax_update              = %d\n",
                grid_fisst -> auto_minmax_update);
        printf("update_minmax_sliders           = %d\n",
                grid_fisst -> update_minmax_sliders);
        printf("direction                       = %d\n",
                grid_fisst -> direction);
        printf("loop_mode                       = %d\n",
                grid_fisst -> loop_mode);
        printf("loop_dir                        = %d\n",
                grid_fisst -> loop_dir);
        printf("loop_surface                    = %d\n",
                grid_fisst -> loop_surface);
        printf("loop_zone                       = %d\n",
                grid_fisst -> loop_zone);
        printf("loop_new_zone                   = %d\n",
                grid_fisst -> loop_new_zone);
        printf("prev                            = %d\n",
                grid_fisst -> prev);
        printf("reset_to_zone                   = %d\n",
                grid_fisst -> reset_to_zone);
        printf("show_looping                    = %d\n",
                grid_fisst -> show_looping);
        printf("boundary_flags[I J K][START END MID]   = ");
        printf("[%2d %2d %2d] [%2d %2d %2d] [%2d %2d %2d]\n",
                grid_fisst -> boundary_flags[I][START],
                grid_fisst -> boundary_flags[I][END],
                grid_fisst -> boundary_flags[I][MID],
                grid_fisst -> boundary_flags[J][START],
                grid_fisst -> boundary_flags[J][END],
                grid_fisst -> boundary_flags[J][MID],
                grid_fisst -> boundary_flags[K][START],
                grid_fisst -> boundary_flags[K][END],
                grid_fisst -> boundary_flags[K][MID]);
        printf("scale_factors[VECTOR, FRAME]    = [%f %f]\n",
                grid_fisst -> scale_factors[VECTOR_SCALE],
                grid_fisst -> scale_factors[FRAME_SCALE]);
        printf("color_indices[LINE_COLOR]       = %d\n",
                grid_fisst -> color_indices[LINE_COLOR]);
        printf("color_rgbs[LINE_COLOR]          = (%f %f %f)\n",
                grid_fisst->color_rgbs[LINE_COLOR][R],
                grid_fisst->color_rgbs[LINE_COLOR][G],
                grid_fisst->color_rgbs[LINE_COLOR][B] );
        printf("color_indices[POINT_COLOR]      = %d\n",
                grid_fisst -> color_indices[POINT_COLOR]);
        printf("color_rgbs[POINT_COLOR]         = (%f %f %f)\n",
                grid_fisst->color_rgbs[POINT_COLOR][R],
                grid_fisst->color_rgbs[POINT_COLOR][G],
                grid_fisst->color_rgbs[POINT_COLOR][B] );
        printf("color_indices[CONTOUR_COLOR]    = %d\n",
                grid_fisst -> color_indices[CONTOUR_COLOR]);
        printf("color_rgbs[CONTOUR_COLOR]       = (%f %f %f)\n",
                grid_fisst->color_rgbs[CONTOUR_COLOR][R],
                grid_fisst->color_rgbs[CONTOUR_COLOR][G],
                grid_fisst->color_rgbs[CONTOUR_COLOR][B] );
        printf("color_indices[VECTOR_COLOR]     = %d\n",
                grid_fisst -> color_indices[VECTOR_COLOR]);
        printf("color_rgbs[VECTOR_COLOR]        = (%f %f %f)\n",
                grid_fisst->color_rgbs[VECTOR_COLOR][R],
                grid_fisst->color_rgbs[VECTOR_COLOR][G],
                grid_fisst->color_rgbs[VECTOR_COLOR][B] );
        printf("color_indices[POLYGON_COLOR]    = %d\n",
                grid_fisst -> color_indices[POLYGON_COLOR]);
        printf("color_rgbs[POLYGON_COLOR]       = (%f %f %f)\n",
                grid_fisst->color_rgbs[POLYGON_COLOR][R],
                grid_fisst->color_rgbs[POLYGON_COLOR][G],
                grid_fisst->color_rgbs[POLYGON_COLOR][B] );
        printf("color_indices[OUTLINE_COLOR]    = %d\n",
```

```
        grid_fisst -> color_indices[OUTLINE_COLOR]);
    printf("color_rgbs[OUTLINE_COLOR]          = (%f %f %f)\n",
        grid_fisst->color_rgbs[OUTLINE_COLOR][R],
        grid_fisst->color_rgbs[OUTLINE_COLOR][G],
        grid_fisst->color_rgbs[OUTLINE_COLOR][B] );
    printf("color_indices[GLYPH_COLOR]         = %d\n",
        grid_fisst -> color_indices[GLYPH_COLOR]);
    printf("color_rgbs[GLYPH_COLOR]            = (%f %f %f)\n",
        grid_fisst->color_rgbs[GLYPH_COLOR][R],
        grid_fisst->color_rgbs[GLYPH_COLOR][G],
        grid_fisst->color_rgbs[GLYPH_COLOR][B] );
    printf("zones[GRID_TYPE][REG FLD]          = %d %d\n",
        grid_fisst -> zones[GRID_TYPE][REG],
        grid_fisst -> zones[GRID_TYPE][FLD]);
    printf("zones[SCALAR_TYPE][REG FLD]        = %d %d\n",
        grid_fisst -> zones[SCALAR_TYPE][REG],
        grid_fisst -> zones[SCALAR_TYPE][FLD]);
    printf("zones[VECTOR_TYPE][REG FLD]        = %d %d\n",
        grid_fisst -> zones[VECTOR_TYPE][REG],
        grid_fisst -> zones[VECTOR_TYPE][FLD]);
    printf("dims[GRID_TYPE][I J K]             = [%d %d %d]\n",
        grid_fisst -> dims[GRID_TYPE][I],
        grid_fisst -> dims[GRID_TYPE][J],
        grid_fisst -> dims[GRID_TYPE][K]);
    printf("dims[SCALAR_TYPE][I J K]           = [%d %d %d]\n",
        grid_fisst -> dims[SCALAR_TYPE][I],
        grid_fisst -> dims[SCALAR_TYPE][J],
        grid_fisst -> dims[SCALAR_TYPE][K]);
    printf("dims[VECTOR_TYPE][I J K]           = [%d %d %d]\n",
        grid_fisst -> dims[VECTOR_TYPE][I],
        grid_fisst -> dims[VECTOR_TYPE][J],
        grid_fisst -> dims[VECTOR_TYPE][K]);
    printf("ranges[I][START END INC CUR DIM]   = [%d %d %d %d %d]\n",
        grid_fisst -> ranges[I][START],
        grid_fisst -> ranges[I][END],
        grid_fisst -> ranges[I][INC],
        grid_fisst -> ranges[I][CUR],
        grid_fisst -> ranges[I][DIM]);
    printf("ranges[J][START END INC CUR DIM]   = [%d %d %d %d %d]\n",
        grid_fisst -> ranges[J][START],
        grid_fisst -> ranges[J][END],
        grid_fisst -> ranges[J][INC],
        grid_fisst -> ranges[J][CUR],
        grid_fisst -> ranges[J][DIM]);
    printf("ranges[K][START END INC CUR DIM]   = [%d %d %d %d %d]\n",
        grid_fisst -> ranges[K][START],
        grid_fisst -> ranges[K][END],
        grid_fisst -> ranges[K][INC],
        grid_fisst -> ranges[K][CUR],
        grid_fisst -> ranges[K][DIM]);
    printf("minmax[CLIP][MINI MAXI BOTTOM TOP] = [%f %f %f %f]\n",
        grid_fisst -> minmax[CLIP][MINI],
        grid_fisst -> minmax[CLIP][MAXI],
        grid_fisst -> minmax[CLIP][BOTTOM],
        grid_fisst -> minmax[CLIP][TOP]);
    printf("minmax[NORM][MINI MAXI BOTTOM TOP] = [%f %f %f %f]\n",
        grid_fisst -> minmax[NORM][MINI],
        grid_fisst -> minmax[NORM][MAXI],
        grid_fisst -> minmax[NORM][BOTTOM],
        grid_fisst -> minmax[NORM][TOP]);
    printf("field_ids[GRID IBLANK SCALAR VECTOR] = [%d %d %d %d]\n",
        grid_fisst -> field_ids[GRID_ID],
        grid_fisst -> field_ids[IBLANK_ID],
        grid_fisst -> field_ids[SCALAR_ID],
        grid_fisst -> field_ids[VECTOR_ID]);
    printf("field_ids[I: START END MID ZONE]   = [%d %d %d %d]\n",
        grid_fisst -> field_ids[START_I_NORM_ID],
        grid_fisst -> field_ids[END_I_NORM_ID],
        grid_fisst -> field_ids[MID_I_NORM_ID],
        grid_fisst -> field_ids[ZONE_I_NORM_ID]);
    printf("field_ids[J: START END MID ZONE]   = [%d %d %d %d]\n",
        grid_fisst -> field_ids[START_J_NORM_ID],
        grid_fisst -> field_ids[END_J_NORM_ID],
        grid_fisst -> field_ids[MID_J_NORM_ID],
        grid_fisst -> field_ids[ZONE_J_NORM_ID]);
    printf("field_ids[K: START END MID ZONE]   = [%d %d %d %d]\n",
        grid_fisst -> field_ids[START_K_NORM_ID],
        grid_fisst -> field_ids[END_K_NORM_ID],
        grid_fisst -> field_ids[MID_K_NORM_ID],
        grid_fisst -> field_ids[ZONE_K_NORM_ID]);
    printf("field_ids[CONTOURS]                = [%d]\n",
        grid_fisst -> field_ids[CONTOURS_ID]);
    printf("num_contours                       = %d\n",
        grid_fisst -> num_contours);
    printf("contours_inc                       = %f\n",
        grid_fisst -> contours_inc);
    printf("contours_min                       = %f\n",
        grid_fisst -> contours_min);
    printf("contours_max                       = %f\n",
        grid_fisst -> contours_max);
    printf("num_con_points                     = %d\n",
        grid_fisst -> num_con_points);

    printf("\n");

    unlock_cur_object();

}

/*---------------------------- END OF dump_state ----------------------------*/



/*++ static void reset_state( int reset_actuators )
 *
 * PURPOSE:
 *
 *              Resets the contents of the grid_fisst structure to the default
 *              state (empty) and halts looping. If reset_actuators is 1 then
 *              the actuators will be reset to their default states.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              6/89
```



```
 *
 * INPUT PARAMETERS:
 *
 *              int             reset_actuators         reset actuators too?
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_fisst              declared in this file
 *      extern Main_Acts        main_acts               defined in this file
 *      extern Minmax_Acts      minmax_acts             defined in this file
 *      extern ScaleGroup*      scale_group             defined in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      extern void     clear_typeout()                 libpanu
 *      extern void     fix_color_panel()               libpanu
 *      extern void     set_typein_fval()               libpanu
 *
 *      int             lock_cur_object()               defined in this file
 *      int             unlock_cur_object()             defined in this file
 *      void            update_legend()                 defined in this file
 *      void            delete_normals()                defined in this file
 *      void            set_default_colors()            defined in this file
 *      void            copy_colors()                   defined in this file
 *
 --*/

/*------------------------------ reset_state --------------------------------*/

static void reset_state( int reset_actuators )
{
    Actuator*       ta;                     /* temp act pointer     */
    int             i;                      /* 1st loop index       */
    int             j;                      /* 2nd loop index       */


    /* if called from the button, fix actuators to reflect state */

    if (reset_actuators == 1)
    {
        /* reset type menu */

        ACCESS3(main_acts.type_menu_group, reset_selections, 0);

        /* reset render menu */

        ACCESS3(main_acts.render_menu_group, reset_selections, 0);

        /* reset attributes menu */

        ACCESS3(main_acts.attributes_menu_group, reset_selections, 0);

        /* reset options menu */

        ACCESS3(main_acts.options_menu_group, reset_selections, 0);

        /* delete normals */

        delete_normals();

        /* delete contours */

        delete_contours();

        /* reset loop buttons group */

        ACCESS3(main_acts.loop_buttons_group, reset_selections, 0);

        /* reset slider buttons group */

        ACCESS3(main_acts.slider_buttons_group, reset_selections, 0);

        /* reset surface buttons group */

        ACCESS3(main_acts.surface_buttons_group, reset_selections, 0);

        /* reset zone typein group */

        ACCESS4(main_acts.zone_typein_group, set_ivalue, 0, 0);

        /* reset slider group */

        ACCESS5(main_acts.slider_group, set_fvalues, I, slider_group_values[I], 0
);
        ACCESS5(main_acts.slider_group, set_fvalues, J, slider_group_values[J], 0
);
        ACCESS5(main_acts.slider_group, set_fvalues, K, slider_group_values[K], 0
);
        ACCESS4(main_acts.slider_group, set_selections, slider_group_selections,
0);
        ACCESS2(main_acts.slider_group, hide_select_buttons);
        ACCESS4(main_acts.slider_group, set_direction, K, 0);
        ACCESS3(main_acts.slider_group, highlight_slider, MID);

        /* data info typeout */

        clear_typeout(main_acts.data_info_typeout);


        /* vector panel's scale group */

        ACCESS3(scale_group, reset_values, 0);


        /* draw button */

        main_acts.draw_object_button -> val = 1.0;
        pnl_fixact(main_acts.draw_object_button);
```

```
        /* reset scalar minmax: */

        /* clip, norm, and legend minmax typeins */

        set_typein_fval(minmax_acts.clip_top_typein, 0.0,
                FLOAT_STRING_FORMAT);
        set_typein_fval(minmax_acts.clip_bot_typein, 0.0,
                FLOAT_STRING_FORMAT);
        set_typein_fval(minmax_acts.norm_top_typein, 0.0,
                FLOAT_STRING_FORMAT);
        set_typein_fval(minmax_acts.norm_bot_typein, 0.0,
                FLOAT_STRING_FORMAT);
        set_typein_fval(minmax_acts.legend_max_typein, 0.0,
                FLOAT_STRING_FORMAT);
        set_typein_fval(minmax_acts.legend_min_typein, 0.0,
                FLOAT_STRING_FORMAT);

        /* clip and norm multisliders */

        ta = minmax_acts.clip_multislider;
        ta = ta -> al;
        ta -> extval = 1.0;
        sprintf(ta -> label, FLOAT_STRING_FORMAT, 0.0);
        ta = ta -> next;
        ta -> extval = 0.0;
        sprintf(ta -> label, FLOAT_STRING_FORMAT, 0.0);
        pnl_fixact(minmax_acts.clip_multislider);

        ta = minmax_acts.norm_multislider;
        ta = ta -> al;
        ta -> extval = 1.0;
        sprintf(ta -> label, FLOAT_STRING_FORMAT, 0.0);
        ta = ta -> next;
        ta -> extval = 0.0;
        sprintf(ta -> label, FLOAT_STRING_FORMAT, 0.0);
        pnl_fixact(minmax_acts.norm_multislider);

        /* restore default minmax modes */

        minmax_acts.auto_minmax_update_button -> val = 1.0;
        pnl_fixact(minmax_acts.auto_minmax_update_button);

        minmax_acts.update_minmax_sliders_button -> val = 1.0;
        pnl_fixact(minmax_acts.update_minmax_sliders_button);

        minmax_acts.invert_clip_test_button -> val = 0.0;
        pnl_fixact(minmax_acts.invert_clip_test_button);

        fix_radio_buttons( minmax_acts.mode_buttons,
                           NUM_SCALAR_MINMAX_MODES,
                           MULTI_ZONE_MINMAX );
    }


    lock_cur_object();

    /* reinitialize the grid surface data structure */

    memset( grid_fisst, 0, sizeof (Grid_Surface) );

    grid_fisst -> object_type               = GRID_SURFACE;
    grid_fisst -> surface_mode              = SINGLE_SURFACE;
    grid_fisst -> type                      = GRID_TYPE;

    grid_fisst -> render_mode               = LINES_1_2;
    grid_fisst -> scale_factors[VECTOR_SCALE] = scale_group_values[0];
    grid_fisst -> scale_factors[FRAME_SCALE]  = scale_group_values[1];
    grid_fisst -> contour_color_type        = SCALAR_COLOR;
    grid_fisst -> vector_color_type         = CONST_COLOR;
    grid_fisst -> vector_tip_type           = NO_TIP;
    grid_fisst -> direction                 = K;
    grid_fisst -> ranges[I][INC]            = 1;
    grid_fisst -> ranges[J][INC]            = 1;
    grid_fisst -> ranges[K][INC]            = 1;
    grid_fisst -> minmax_mode               = MULTI_ZONE_MINMAX;

    for (i = 0; i < NUM_GS_FIELD_IDS; ++i)
        grid_fisst -> field_ids[i]      = -1;

    for (i = 0; i < 3; ++i)
      for (j = 0; j < 3; ++j)
        grid_fisst -> boundary_flags[i][j]  = 1;

    grid_fisst -> draw                      = 1;
    grid_fisst -> shaded                    = DEFAULT_SHADING;
    grid_fisst -> clip_mode                 = STRAIGHT_CLIP;
    grid_fisst -> loop_mode = loop_mode     = LOOP_OFF;
    grid_fisst -> loop_dir                  = LOOP_FORWARD;
    grid_fisst -> loop_surface              = MID;
    grid_fisst -> loop_zone                 = SINGLE_ZONE;
    grid_fisst -> num_contours              = 40;

    grid_fisst -> reset_to_zone             = 1;
    grid_fisst -> show_looping              = 1;

    grid_fisst -> auto_minmax_update        = 1;
    grid_fisst -> update_minmax_sliders     = 1;


    unlock_cur_object();


    /* reset colors */

    set_default_colors();
    fix_color_panel();
    copy_colors();

    /* clear legend values */

    update_legend(0.0, 0.0, minmax_acts.legend_labels);

    /* Update data */

    update_fld_data_panel();
}

/*--------------------------- END OF reset_state ----------------------------*/
```



```
/*++ static void surface_buttons_func( int row, int col, int state )
 *
 * PURPOSE:
 *
 *              Handles setting the current surface button selections for
 *              commands
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/90
 *              12/90           converted to button group call-back
 *              4/91            added command stuff
 *
 * INPUT PARAMETERS:
 *
 *              int     row     selected button's row
 *              int     col     selected button's column
 *              int     state   selected button's state (0 or 1)
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *
 --*/

/*-------------------------- surface_buttons_func --------------------------*/

static void surface_buttons_func( int row, int col, int state )
{
        interactive = 1;
        load_command(   surface_buttons_script_commands[0],
                        surface_buttons_script_commands[1 + col] );
}

/*----------------------- END OF surface_buttons_func ----------------------*/
```

```
/*++ static void set_surface_buttons_func( char* script_command )
 *
 * PURPOSE:
 *
 *              Handles setting the current surface button selections from
 *              a command
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/90
 *              12/90           converted to button group call-back
 *              4/91            added command stuff
 *
 * INPUT PARAMETERS:
 *
 *               char* script_command   script command or actuator
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_fisst;             declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              extern int      command_index()         libpanu
 *              int             lock_cur_object()       defined in this file
 *              int             unlock_cur_object()     defined in this file
 --*/

/*-------------------------- set_surface_buttons_func ----------------------*/

static void set_surface_buttons_func( char* script_command )
```

```
{
        char            param[32];
        int             selections[NUM_SURFACE_BUTTONS];
        int             index;

        parse_command( script_command, "%s", param );

        if ( ( index = command_index( param, surface_buttons_script_commands,
                        NUM_SURFACE_BUTTONS + 1 ) ) == -1 )
        {
                return;
        }

        --index;

        lock_cur_object();

        grid_first -> surface_mode = index;

        if ( !interactive )
        {
                memset( selections, 0, sizeof selections );
                selections[grid_first -> surface_mode] = 1;
                ACCESS4( main_acts.surface_buttons_group,
                        set_selections, selections, 0 );
        }

        delete_contours();

        unlock_cur_object();

        /* show the select buttons if boundary surfaces mode */

        if ( index == BOUNDARY_SURFACES )
        {
                ACCESS2( main_acts.slider_group, show_select_buttons );
        }
        else /* hide them */
        {
                ACCESS2( main_acts.slider_group, hide_select_buttons );
        }
}

/*-------------------- END OF set_surface_buttons_func ---------------------*/
```

```
/*++ static void slider_buttons_func( int row, int col, int state )
 *
 * PURPOSE:
 *
 *              Handles setting the current slider button selections.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/90
 *              12/90           converted to button group call-back
 *
 * INPUT PARAMETERS:
 *
 *              int     row     selected button's row
 *              int     col     selected button's column
 *              int     state   selected button's state (0 or 1)
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *
 --*/

/*-------------------------- slider_buttons_func ---------------------------*/

static void slider_buttons_func( int row, int col, int state )
{
        interactive = 1;

        switch ( col )
        {
            case 0: load_command( "SLIDER_ACTION %s", "RESET" ); break;
            case 1: load_command( "SLIDER_ACTION %s %s", "RESET_ZONE",
                                                    ON_OR_OFF(state) );
                    break;
            case 2: load_command( "SLIDER_ACTION %s %s", "SHOW_LOOPING",
                                                    ON_OR_OFF(state) );
                    break;
            default: interactive = 0; break;
        }
}

/*----------------------- END OF slider_buttons_func ------------------------*/
```



```
/*++ static void set_slider_buttons_func( char* script_command )
 *
 * PURPOSE:
 *
 *              Handles setting the current slider button selections.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/90
 *              12/90           converted to button group call-back
 *              5/91            converted to scripting
 *
 * INPUT PARAMETERS:
 *
 *              char*   script_command  script command or actuator
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_first;             declared in this file
 *      extern int              show_looping            declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              int             lock_cur_object()       defined in this file
 *              int             unlock_cur_object()     defined in this file
 *              void            reset_ijk_ranges()      defined in this file
 *
 --*/

/*-------------------------- set_slider_buttons_func ------------------------*/

static void set_slider_buttons_func( char* script_command )
{
        int             col;
        char            param[16];
        char            cval[8];
        int             selections[3];


        parse_command( script_command, "%s", param );

        if      ( strcasecmp( param, "RESET"        ) == 0 )  col = 0;
        else if ( strcasecmp( param, "RESET_ZONE"   ) == 0 )  col = 1;
        else if ( strcasecmp( param, "SHOW_LOOPING" ) == 0 )  col = 2;
        else { interactive = 0; return; }

        lock_cur_object();

        switch ( col )
        {
                case 0:
                        reset_ijk_ranges();
                        break;
                case 1:
                        parse_command( script_command, "%s %s", param, cval );
                        grid_first -> reset_to_zone = ON_OR_OFF_VAL(cval);
                        break;
                case 2:
                        parse_command( script_command, "%s %s", param, cval );
                        show_looping = ON_OR_OFF_VAL( cval );
                        grid_first -> show_looping = show_looping;
                        break;
                default:
                        break;
        }

        if ( ! interactive )
        {
                selections[0] = 0;
                selections[1] = grid_first -> reset_to_zone;
                selections[2] = show_looping;
                ACCESS4( main_acts.slider_buttons_group,
                        set_selections, selections, 0 );
        }

        interactive = 0;

        unlock_cur_object();
}

/*----------------------- END OF set_slider_buttons_func --------------------*/
```

```
/*++ static void loop_buttons_func( int row, int col, int state )
 *
 * PURPOSE:
 *
 *              Handles setting the current loop button selections.
 *
```

```
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/90
 *              12/90           converted to button group call-back
 *
 * INPUT PARAMETERS:
 *
 *              int     row     selected button's row
 *              int     col     selected button's column
 *              int     state   selected button's state (0 or 1)
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_first;             declared in this file
 *      extern Main_Acts*       main_acts               declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              None
 *
 --*/

/*---------------------------- loop_buttons_func ----------------------------*/

static void loop_buttons_func( int row, int col, int state )
{
        interactive = 1;
        load_command(   loop_buttons_script_load_commands[row],
                        loop_buttons_script_param_commands[row][col]    );
}

/*------------------------ END OF loop_buttons_func ------------------------*/
```

```
/*++ static void set_loop_buttons_func( char* script_command )
 *
 * PURPOSE:
 *
 *              Handles setting the current loop button selections.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/90
 *              12/90           converted to button group call-back
 *
 * INPUT PARAMETERS:
 *
 *              char*           script command
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_first;             declared in this file
 *      extern Main_Acts*       main_acts               declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              extern int      view_set_draw_mode()    libviewI
 *              void            delete_contours()       defined in this file
 *              int             lock_cur_object()       defined in this file
 *              int             unlock_cur_object()     defined in this file
 *
 --*/

/*------------------------- set_loop_buttons_func -------------------------*/

static void set_loop_buttons_func( char* script_command )
{
        int             d;                      /* I,J,K direction      */
        int             new_loop_surface;       /* START or END plane   */
        int             old_loop_surface;       /* START or END plane   */
        int*            ranges;                 /* grid ranges[d]       */
        int             i, row, col;
        int             selections[NUM_LOOP_BUTTONS];
        char            command[32];
```



```
        char            param[32];


        sscanf( script_command, "%s", command );
        parse_command( script_command, "%s", param );

        if ( ( row = command_index( command,
                loop_buttons_script_load_commands, LOOP_BUTTONS_ROWS)) == -1 )
        {
                interactive = 0;
                return;
        }

        if ( ( col = command_index( param,
                loop_buttons_script_param_commands[row], 4 ) ) == -1 )
        {
                interactive = 0;
                return;
        }


        lock_cur_object();

        delete_contours();

        new_loop_surface = old_loop_surface = grid_first -> loop_surface;

        switch ( row )
        {
            case 0: loop_mode = grid_first -> loop_mode = col; break;
            case 1: old_loop_surface = grid_first -> loop_surface;
                    new_loop_surface = grid_first -> loop_surface = col;
                    ACCESS3( main_acts.slider_group, highlight_slider, col );
                    break;
            case 2: grid_first -> loop_zone = col; break;
            default: interactive = 0; return;
        }

        /* If necessary, delete the old surface normals and contours */

        if (    grid_first -> surface_mode == SINGLE_SURFACE &&
                grid_first -> zone_normals == 0 &&
                new_loop_surface != old_loop_surface    )
        {
            i = NORM_ID_INDEX( grid_first->direction, grid_first->loop_surface );

            if ( grid_first -> field_ids[i] != -1 )
            {

#ifdef DEBUG
printf("delete surface normals for loop_surface = %d\n", grid_first -> loop_surface);
#endif

                SDEALLOCATE( grid_first -> field_ids[i] );
                grid_first -> field_ids[i] = -1;
            }
        }

        /* if not interactive then fix the loop buttons */

        if ( ! interactive )
        {
                memset( selections, 0, sizeof selections );

                selections[grid_first -> loop_mode] = 1;
                i = loop_buttons_per_row[0];
                selections[i + grid_first -> loop_surface] = 1;
                i += loop_buttons_per_row[1];
                selections[i + grid_first -> loop_zone] = 1;

                ACCESS4( main_acts.loop_buttons_group,
                        set_selections, selections, 0 );
        }

        interactive = 0;


        /* set draw mode based on current looping status */

        if ( grid_first -> loop_mode == LOOP_OFF )
        {
            grid_first -> prev = 0;
            view_set_draw_mode( cur_object, NO_NEED_TO_DRAW );
        }
        else
        {
            d           = grid_first -> direction;
            ranges      = grid_first -> ranges[d];

            if ( old_loop_surface != MID && grid_first -> prev != 0 )
                ranges[old_loop_surface] = grid_first -> prev;

            if ( new_loop_surface != MID )
                grid_first -> prev = ranges[new_loop_surface];

            /* redraw the sliders to indicate the new position */

            ACCESS5( main_acts.slider_group, set_ivalues, d, ranges, 0 );

            view_set_draw_mode(cur_object, ALWAYS_DRAW);
        }

        update_data_info();

        update_minmax();

        unlock_cur_object();
}

/*---------------------- END OF set_loop_buttons_func ----------------------*/
```

```
/*++ static void zone_func( int new_zone )
 *
 * PURPOSE:
 *
 *              Handles setting the current zone.
 *
```

```
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/90
 *              12/90   converted to typein group call-back
 *
 * INPUT PARAMETERS:
 *
 *              int     new_zone;       new zone value to set
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *              None
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *
 --*/

/*------------------------------ zone_func ---------------------------------*/

static void zone_func( int new_zone )
{
        interactive = 1;
        load_command( "ZONE %d", new_zone );
}

/*-------------------------- END OF zone_func ---------------------------------*/









/*++ static void set_zone_func( char* script_command )
 *
 * PURPOSE:
 *
 *              Handles setting the current zone.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/90
 *              12/90   converted to typein group call-back
 *              5/91    converted to scripting
 *
 * INPUT PARAMETERS:
 *
 *              char*   script_command  script command or actuator
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *              None
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              extern int      set_fld_data_selection()        libfldpan
 *              extern void     update_fld_data_panel()         libfldpan
 *
 --*/

/*------------------------------ set_zone_func -----------------------------------*/

static void set_zone_func( char* script_command )
{
        int             new_zone;

        parse_command( script_command, "%d", &new_zone );

        /* select the new grid and update data */

        set_fld_data_selection( GRID, 0, new_zone );

        /* this will invoke the call-back function: data_select() */
```



```
        update_fld_data_panel();

        if ( ! interactive )
        {
                lock_cur_object();
                ACCESS4( main_acts.zone_typein_group, set_ivalue,
                         grid_first -> zones[GRID_TYPE][FLD], 0 );
                unlock_cur_object();
        }

        interactive = 0;
}

/*----------------------- END OF set_zone_func -------------------------------*/









/*++ static void ijk_ranges_func( int dir, float ranges[5] )
 *
 * PURPOSE:
 *
 *              Adjusts the IJK ranges based on the slider
 *              specifications.
 *
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              4/89
 *              12/90   converted to slider group call-back
 *
 * INPUT PARAMETERS:
 *
 *              int             dir;            I/J/K
 *              float           ranges[5]       START/END/INC/CUR/DIM
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *
 --*/

/*------------------------------ ijk_ranges_func ----------------------------*/

static void ijk_ranges_func( int dir, float ranges[5] )
{
        static char*    dir_str[3] = { "I", "J", "K" };
        static char*    slider_str[5] = { "START", "END", "INC", "CUR", "DIM" };
        int             s, plane;

        lock_cur_object();
        interactive = 1;
        for ( s = 0; s < 5; ++s )
        {
            plane = ROUND( ranges[s] );
            if ( plane != grid_first -> ranges[dir][s] )
                load_command( "SLIDER %s %s %d",
                              dir_str[dir], slider_str[s], plane );
        }
        unlock_cur_object();
}

/*-------------------------- END OF ijk_ranges_func ---------------------------*/









/*++ static void set_ijk_ranges_func( char* script_command )
 *
 * PURPOSE:
 *
 *              Adjusts the IJK ranges based on the slider
 *              specifications from a script command
 *
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              4/89
 *              12/90   converted to slider group call-back
 *              4/91    P. Kelaita -- added command handling
 *
 * INPUT PARAMETERS:
 *
 *              char*           str
```

```
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *              extern Grid_Surface*    grid_fisst        declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              void            delete_contours()       defined in this file
 *              void            update_data_info()      defined in this file
 *              int             lock_cur_object()       defined in this file
 *              int             unlock_cur_object()     defined in this file
 *              void            check_delete_normals()  defined in this file
 *
 --*/

/*------------------------- set_ijk_ranges_func ----------------------------*/

static void set_ijk_ranges_func( char* script_command )
{
        static char*    dir_str[3] = { "I", "J", "K" };
        static char*    slider_str[5] = { "START", "END", "INC", "CUR", "DIM" };
        int             d;                      /* IJK direction        */
        int             s;                      /* START/END/INC/CUR/DIM*/
        int             dir;
        int             plane;
        int             ranges[3][5];
        char            dir_str_param[8];
        char            slider_str_param[8];


        parse_command( script_command, "%s %s %d",
                       dir_str_param, slider_str_param, &plane );

        for ( dir = 0; dir < 3; ++dir )
          if ( strcasecmp( dir_str_param, dir_str[dir] ) == 0 ) break;
        if ( dir == 3 ) { interactive = 0; return; }

        for ( s = 0; s < 5; ++s )
          if ( strcasecmp( slider_str_param, slider_str[s] ) == 0 ) break;
        if ( s == 5 ) { interactive = 0; return; }

        if ( ! interactive ) /* Update slider group */
        {
          ACCESS6( main_acts.slider_group, set_ivalue, dir, s, plane, 0 );
        }

        interactive = 0;

        /* Get the new (possibly adjusted) ranges for this direction */

        ACCESS4( main_acts.slider_group, get_ivalues, dir, ranges[dir] );

        lock_cur_object();

        /* get all of the ranges */

        for ( d = 0; d < 3; ++d )
          for ( s = 0; s < 5; ++s )
            if ( d != dir ) ranges[d][s] = grid_fisst -> ranges[d][s];

        /* If necessary, delete the normals data */

        check_delete_normals( grid_fisst -> direction, ranges );

        /* Delete the contours data */

        delete_contours();

        /* update to these new ranges */

        for (s = 0; s < 5; ++s) grid_fisst->ranges[dir][s] = ranges[dir][s];

        if ( grid_fisst -> render_mode == CONTOUR_LINES ) delete_contours();

        unlock_cur_object();

        /* update the data info typeout */

        update_data_info();

        /* Update the scalar minmax panel */

        update_minmax();
}

/*---------------------- END OF set_ijk_ranges_func -------------------------*/




/*++ static void direction_func( int dir )
 *
 * PURPOSE:
 *
 *              Handles setting the current IJK direction, set up command
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
```



```
 *
 *              6/89
 *              12/90   converted to a slider group call-back
 *              4/91    changed to handle commands
 *
 * INPUT PARAMETERS:
 *
 *              int     dir     current direction
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *
 --*/

/*---------------------------- direction_func -------------------------------*/

static void direction_func( int dir )
{
        static char*    dir_str[3] = { "I", "J", "K" };

        interactive = 1;
        load_command( "DIRECTION %s", dir_str[dir] );
}

/*-------------------- END OF direction_func ------------------------------*/




/*++ static void set_direction_func( char* script_command )
 * -
 * PURPOSE:
 *
 *              Handles setting the current IJK direction from a command
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              4/91 added command capability
 *
 * INPUT PARAMETERS:
 *
 *              char*   script_command  script command or actuator
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_fisst;             declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              void            delete_contours()       defined in this file
 *              int             lock_cur_object()       defined in this file
 *              int             unlock_cur_object()     defined in this file
 *
 --*/

/*--------------------------- set_direction_func ---------------------------*/

static void set_direction_func( char* script_command )
{
        int             old_dir;                /* I,J,K direction      */
        int             loop_surface;           /* START or END plane   */
        int*            ranges;                 /* grid ranges[d]       */
        int             dir;
        char            dir_str[8];


        parse_command( script_command, "%s", dir_str );

        if      ( strcasecmp( dir_str, "I" ) == 0 ) dir = I;
        else if ( strcasecmp( dir_str, "J" ) == 0 ) dir = J;
        else if ( strcasecmp( dir_str, "K" ) == 0 ) dir = K;
        else { interactive = 0; return; }

        lock_cur_object();

        old_dir = grid_fisst -> direction;
        ranges = grid_fisst -> ranges[old_dir];
```

```
        /* If necessary, delete the normals data */

        check_delete_normals( dir, grid_first -> ranges );

        /* Delete the contours data */

        delete_contours();

        grid_first -> direction = dir;

        if ( ! interactive )
            ACCESS4( main_acts.slider_group, set_direction, dir, 0 );
        interactive = 0;

        if ( old_dir != dir && grid_first -> prev != 0 )
        {
            loop_surface = grid_first -> loop_surface;

            if ( loop_surface != MID && grid_first -> prev != 0 )
            {
                ranges[loop_surface] = grid_first -> prev;
                grid_first -> prev = grid_first -> ranges[dir][loop_surface];

                /* redraw the sliders to indicate the new position */

                ACCESS5( main_acts.slider_group,
                        set_ivalues,old_dir, grid_first->ranges[old_dir], 0 );
            }
        }

        if ( grid_first -> render_mode == CONTOUR_LINES ) delete_contours();

        update_minmax();

        unlock_cur_object();
}

/*---------------------- END OF set_direction_func ---------------------------*/
```

```
/*++ static void boundary_surfaces_func( int selections[3][3] )
 *
 * PURPOSE:
 *
 *              Handles setting the current boundary surface selections for
 *              commands.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/90
 *              12/90           converted to button group call-back
 *              4/91            P. Kelaita -- added command stuff
 *
 * INPUT PARAMETERS:
 *
 *              int     selections[3][3]        new surface button selections
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 --*/

/*------------------------- boundary_surfaces_func --------------------------*/

static void boundary_surfaces_func( int selections[3][3] )
{
        static char*    off_on_str[2] = { "OFF", "ON" };
        static char*    dir_str[3] = { "I", "J", "K" };
        static char*    slider_str[3] = { "START", "END", "MID" };
        int             dir, s, off_on;

        lock_cur_object();
        interactive = 1;
        for ( dir = 0; dir < 3; ++dir )
        {
          for ( s = 0; s < 3; ++s )
          {
            off_on = selections[dir][s];
            if ( off_on != grid_first -> boundary_flags[dir][s] )
                load_command( "BOUNDARY %s %s %s",
                              dir_str[dir], slider_str[s], off_on_str[off_on] );
          }
        }
        unlock_cur_object();
}

/*---------------------- END OF boundary_surfaces_func ----------------------*/
```



```
/*++ static void set_boundary_surfaces_func( char* script_command )
 *
 * PURPOSE:
 *
 *              Handles setting the current boundary surface selections for
 *              commands
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/90
 *              12/90           converted to button group call-back
 *              4/91            added command stuff
 *
 * INPUT PARAMETERS:
 *
 *              char*   script_command  script command or actuator
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_first;             declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              int             lock_cur_object()       defined in this file
 *              int             unlock_cur_object()     defined in this file
 *              void            check_delete_normals()  defined in this file
 *
 --*/

/*------------------------- set_boundary_surfaces_func ----------------------*/

static void set_boundary_surfaces_func( char* script_command )
{
        static char*    dir_str[3] = { "I", "J", "K" };
        static char*    slider_str[3] = { "START", "END", "MID" };
        int             dir, s, off_on, searching;
        char            param_dir_str[8];
        char            param_slider_str[8];
        char            param_off_on_str[8];

        parse_command( script_command, "%s %s %s",
                       param_dir_str, param_slider_str, param_off_on_str );

        if      ( strcasecmp( param_off_on_str, "OFF" ) == 0 ) off_on = OFF;
        else if ( strcasecmp( param_off_on_str, "ON"  ) == 0 ) off_on = ON;
        else { interactive = 0; return; }

        for ( dir = 0, searching = TRUE; searching && dir < 3; ++dir )
        {
          if ( strcasecmp( param_dir_str, dir_str[dir] ) == 0 )
          {
            for ( s = 0; searching && s < 3; ++s )
            {
              if ( strcasecmp( param_slider_str, slider_str[s] ) == 0 )
              {
                searching = FALSE;
              }
            }
          }
        }

        --dir; --s;

        if ( searching ) return;

        interactive = 0;

        lock_cur_object();

        grid_first -> boundary_flags[dir][s] = off_on;

        if ( ! interactive )
                ACCESS4( main_acts.slider_group, set_selections,
                         grid_first -> boundary_flags, 0 );

        check_delete_normals( grid_first -> direction, grid_first -> ranges );
        delete_contours();

        unlock_cur_object();
}

/*---------------------- END OF set_boundary_surfaces_func ------------------*/
```

```
/*++ static void reset_ijk_ranges( void )
 *
 * PURPOSE:
 *
 *              Resets the IJK ranges either to full dims or partially.
```

```
*
*
* AUTHORS:
*
*               Todd Plessel
*               NASA Ames Research Center
*               Sterling Software
*
* REVISION HISTORY:
*
*               1/90
*
* INPUT PARAMETERS:
*
*               None
*
* OUTPUT PARAMETERS:
*
*               None
*
* FUNCTION RETURN:
*
*               None
*
* GLOBAL VARIABLES USED:
*
*               extern Grid_Surface*    grid_fisst      defined in this file
*               extern Main_Acts        main_acts;      declared in this file
*
* FILES USED:
*
*               None
*
* NOTES:
*
* NON-STANDARD CODE :
*
* CALLED BY :
*
* FUNCTIONS CALLED :
*
*       int             lock_cur_object()       defined in this file
*       int             unlock_cur_object()     defined in this file
*       void            update_data_info()      defined in this file
*
--*/

/*---------------------------- reset_ijk_ranges ----------------------------*/

static void reset_ijk_ranges( void )
{
        int             full_reset;             /* reset to full dims?  */
        int             dir;                    /* direction: I, J, K   */
        int             start;                  /* starting I, J, K     */
        int             end;                    /* ending I, J, K       */
        int             dim;                    /* dimension: I, J, K   */
        int             min;                    /* 1 or 0 if dim is 0   */
        int             temp[20];               /* to check buttons     */


        /* check if this is a full or partial reset */

        ACCESS3(main_acts.slider_buttons_group, get_selections, temp);

        full_reset = temp[1]; /* reset when zone changes? */

        lock_cur_object();

        /* loop on direction and reset each component */

        for ( dir = 0; dir < 3; ++dir )
        {
                /* set the dimension of the range */

                dim = grid_fisst -> dims[GRID_TYPE][dir];
                grid_fisst -> ranges[dir][DIM] = dim;

                /* if the dimension is zero then zero out other components */

                if ( dim == 0 )
                {
                        grid_fisst -> ranges[dir][START]        = start = 0;
                        grid_fisst -> ranges[dir][END]  = end   = 0;
                        grid_fisst -> ranges[dir][INC]  = 1;
                        grid_fisst -> ranges[dir][CUR]  = 0;
                }

                /* else if this is a full reset then do it */

                else if ( full_reset == 1 )
                {
                        grid_fisst -> ranges[dir][START]        = start = 1;
                        grid_fisst -> ranges[dir][END]  = end   = dim;
                        grid_fisst -> ranges[dir][INC]  = 1;
                        grid_fisst -> ranges[dir][CUR]  = ROUND( dim / 2.0 );

                        if ( grid_fisst -> ranges[dir][CUR] == 0 )
                                grid_fisst -> ranges[dir][CUR] = 1;
                }

                /* else only change if out-of-bounds */

                else
                {
                        /*
                         * only make changes required to have
                         * 1 <= start <= cur <= end <= dim and
                         * 1 <= inc <= dim
                         * note: LIMIT_TO(a, min, max) insures that
                         * min <= a <= max
                         */

                        min = MIN(1, dim);

                        LIMIT_TO( grid_fisst -> ranges[dir][START], min, dim );
                        start =   grid_fisst -> ranges[dir][START];
                        LIMIT_TO( grid_fisst -> ranges[dir][END], start, dim );
                        end =     grid_fisst -> ranges[dir][END];
                        LIMIT_TO( grid_fisst -> ranges[dir][CUR], start, end );
                        LIMIT_TO_MAX( grid_fisst -> ranges[dir][INC], dim );
                        LIMIT_TO_MIN( grid_fisst -> ranges[dir][INC], 1 );
                }
        }

        /* update the sliders */

        ACCESS5( main_acts.slider_group,
                set_ivalues, I, grid_fisst -> ranges[I], 0 );
```



```
        ACCESS5( main_acts.slider_group,
                set_ivalues, J, grid_fisst -> ranges[J], 0 );
        ACCESS5( main_acts.slider_group,
                set_ivalues, K, grid_fisst -> ranges[K], 0 );

        delete_contours();

        unlock_cur_object();

        /* update data info typeout */

        update_data_info();
}

/*--------------------------- END OF reset_ijk_ranges ---------------------------*/









/*++ static void data_select( int type, int reg_num, int fld_num,
 *              FLDDataPtr fld_data_ptr )
 *
 * PURPOSE:
 *
 *              This routine is called from Surfer's fld data panel.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/89
 *
 * INPUT PARAMETERS:
 *
 *              int             type            GRID,SOLUTION,SCALAR,VECTOR
 *              int             reg_num         number of register selected
 *              int             fld_num         number of field selected
 *              FLDDataPtr      fld_data_ptr    -> fld data structure
 *                                              containing info about the
 *                                              selected data. Or this could
 *                                              be NULL indicating a reset
 *                                              affecting the given type
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Minmax_Acts      minmax_acts     this file
 *      extern Grid_Surface*    grid_fisst      defined in this file
 *      extern int              update_actuators_mode;  declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      extern void     set_typein_ival()       libpanu
 *      void            update_dims()           defined in this file
 *      void            reset_data()            defined in this file
 *      void            reset_minmax()          defined in this file
 *      void            delete_normals()        defined in this file
 *      void            update_data_info()      defined in this file
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
 --*/
/*------------------------------ data_select ------------------------------*/

static void data_select( int type, int reg_num, int fld_num,
                FLDDataPtr fld_data_ptr )
{
        int     must_delete_normals = 0; /* only if data changed */
        int     new_scalar_field    = 0; /* only if data changed */


        /*
         * check if the data is NULL, if so this indicates that a
         * selection was on a field that is no longer available (and
         * the fld data panel automatically updated its typeouts) but we must
         * now call Surfer's own reset data function and pass type
         * to indicate that the reset pertains only to this type
         */

        if ( fld_data_ptr == NULL )
        {
                reset_data( type );
                return;
        }

        lock_cur_object();

        /* otherwise a field was selected so switch on the type */

        switch ( type )
        {
            case GRID:

                /* update grid zone info */

                grid_fisst -> zones[GRID_TYPE][FLD] = fld_num;
                ACCESS4( main_acts.zone_typein_group, set_ivalue, fld_num, 0 );

                /* copy grid data id into grid surface structure */
```

```
            if (    grid_fisst -> field_ids[GRID_ID] !=
                    fld_data_ptr -> field_ids[VF]   )
            {
                    must_delete_normals = 1;
                    grid_fisst -> field_ids[GRID_ID] =
                            fld_data_ptr -> field_ids[VF];
            }

            /* copy grid iblanking data id */

            if ( fld_data_ptr -> attributes[IS_IBLANKED] )
            {
                    grid_fisst -> field_ids[IBLANK_ID] =
                            fld_data_ptr -> field_ids[ISF];
            }
            else
            {
                    grid_fisst -> field_ids[IBLANK_ID] = -1;
            }

            if (!MATCH_DIMS(grid_fisst->dims[GRID_TYPE],fld_data_ptr->dims))
            {
                    must_delete_normals = 1;
                    update_dims(GRID_TYPE, fld_data_ptr -> dims);
            }

            break;

        case SCALAR:

            /* update zone info in grid surface struct */

            grid_fisst -> zones[SCALAR_TYPE][REG] = reg_num;
            grid_fisst -> zones[SCALAR_TYPE][FLD] = fld_num;

            /* copy scalar data id into grid surface structure */

            if (    grid_fisst -> field_ids[SCALAR_ID] !=
                    fld_data_ptr -> field_ids[SF]   )
            {
                    new_scalar_field = 1;
                    grid_fisst -> field_ids[SCALAR_ID] =
                            fld_data_ptr->field_ids[SF];
            }

            /* update the dims */

            if (!MATCH_DIMS(grid_fisst->dims[SCALAR_TYPE],fld_data_ptr->dims))
            {
                    update_dims(SCALAR_TYPE, fld_data_ptr -> dims);
            }

            /* if in auto update mode then reset the scalar minmax */

            if ( minmax_acts.auto_minmax_update_button -> val == 1.0 &&
                 ( ! update_actuators_mode || new_scalar_field ) )
            {
                    reset_minmax( "RESET_MINMAX LEGEND" );
            }

            break;

        case SOLUTION:

            /* update zone info in grid surface struct */

            grid_fisst -> zones[VECTOR_TYPE][REG] = reg_num;
            grid_fisst -> zones[VECTOR_TYPE][FLD] = fld_num;

            /* copy vector data id into grid surface structure */

            if (    grid_fisst -> field_ids[VECTOR_ID] !=
                    fld_data_ptr -> field_ids[VF]   )
            {
                    if (    grid_fisst -> type == VECTOR_TYPE &&
                            USES_NORMALS(grid_fisst)        )
                    {
                            must_delete_normals = 1;
                    }

                    grid_fisst -> field_ids[VECTOR_ID] =
                            fld_data_ptr->field_ids[VF];
            }

            /* update the dims */

            if (!MATCH_DIMS(grid_fisst->dims[VECTOR_TYPE],fld_data_ptr->dims))
            {
                    if (    grid_fisst -> type == VECTOR_TYPE &&
                            USES_NORMALS(grid_fisst)        )
                    {
                            must_delete_normals = 1;
                    }

                    update_dims(VECTOR_TYPE, fld_data_ptr -> dims);
            }

            /* if in auto update mode then reset the scalar minmax */

            if ( grid_fisst -> type == VECTOR_TYPE )
            {
                if ( minmax_acts.auto_minmax_update_button -> val == 1.0 )
                {
                    reset_minmax( "RESET_MINMAX LEGEND" );
                }
            }

            break;

        default:

            return;
    }


    /* if needed, delete any old normals and contours data if it exists */

    if ( must_delete_normals )
    {
            /* reset this mode to allow normals to be deleted */

            update_actuators_mode = 0;
            delete_normals();
            delete_contours();
    }
```



```
    /* update the contour data if need be */

    if ( grid_fisst -> render_mode == CONTOUR_LINES ) delete_contours();

    /* update the data info typeout */

    update_data_info();

    unlock_cur_object();

}

/*-------------------------- END OF data_select ---------------------------*/









/*++ static void reset_data( int type )
 *
 * PURPOSE:
 *
 *              Resets the zone data of the indicated type
 *              in the grid_fisst structure to the default state (empty).
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              6/89
 *
 * INPUT PARAMETERS:
 *
 *              int             type    GRID, SCALAR, VECTOR
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_fisst              declared in this file
 *      extern Main_Acts        main_acts               this file
 *      extern Minmax_Acts      minmax_acts     this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      extern void     clear_typeout()                 libpanu
 *      extern void     set_typeout_fval()              libpanu
 *      extern void     update_fld_data_panel()         libfldpan
 *      void            delete_normals()                defined in this file
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
--*/

/*------------------------------ reset_data ---------------------------------*/

static void reset_data( int type )
{
        Actuator*       ta;                     /* temp act pointer     */


        /* reset zone data: clear appropriate registers and fields: */

        if ( type == GRID )
        {
                /* delete normals */

                delete_normals();

                /* delete contours */

                delete_contours();

                /* reset zone typein group */

                ACCESS4( main_acts.zone_typein_group, set_ivalue, 0, 0 );

                /* reset slider group */

                ACCESS5(main_acts.slider_group, set_fvalues, I, slider_group_values[I], 0
);
                ACCESS5(main_acts.slider_group, set_fvalues, J, slider_group_values[J], 0
);
                ACCESS5(main_acts.slider_group, set_fvalues, K, slider_group_values[K], 0
);
                ACCESS4(main_acts.slider_group, set_selections, slider_group_selections,
0);

                /* data info typeout */

                clear_typeout(main_acts.data_info_typeout);

                /* reinitialize part of the grid surface data structure */

                lock_cur_object();

                grid_fisst -> dims[GRID_TYPE][I]                = 0;
                grid_fisst -> dims[GRID_TYPE][J]                = 0;
                grid_fisst -> dims[GRID_TYPE][K]                = 0;
                grid_fisst -> ranges[I][START]          = 0;
                grid_fisst -> ranges[I][END]            = 0;
```

```
        grid_fisst -> ranges[I][INC]            = 1;
        grid_fisst -> ranges[I][CUR]            = 0;
        grid_fisst -> ranges[I][DIM]            = 0;
        grid_fisst -> ranges[J][START]          = 0;
        grid_fisst -> ranges[J][END]            = 0;
        grid_fisst -> ranges[J][INC]            = 1;
        grid_fisst -> ranges[J][CUR]            = 0;
        grid_fisst -> ranges[J][DIM]            = 0;
        grid_fisst -> ranges[K][START]          = 0;
        grid_fisst -> ranges[K][END]            = 0;
        grid_fisst -> ranges[K][INC]            = 1;
        grid_fisst -> ranges[K][CUR]            = 0;
        grid_fisst -> ranges[K][DIM]            = 0;
        grid_fisst -> field_ids[GRID_ID]                = -1;
        grid_fisst -> field_ids[IBLANK_ID]      = -1;

        unlock_cur_object();
    }
    else if ( type == SCALAR )
    {
        /* delete contours */

        delete_contours();

        /* reinitialize part of the grid surface data structure */

        lock_cur_object();

        grid_fisst -> zones[SCALAR_TYPE][REG]   = 0;
        grid_fisst -> zones[SCALAR_TYPE][FLD]   = 0;
        grid_fisst -> dims[SCALAR_TYPE][I]      = 0;
        grid_fisst -> dims[SCALAR_TYPE][J]      = 0;
        grid_fisst -> dims[SCALAR_TYPE][K]      = 0;
        grid_fisst -> minmax[CLIP][MINI]                = 0.0;
        grid_fisst -> minmax[CLIP][MAXI]                = 0.0;
        grid_fisst -> minmax[CLIP][BOTTOM]      = 0.0;
        grid_fisst -> minmax[CLIP][TOP]         = 0.0;
        grid_fisst -> minmax[NORM][MINI]                = 0.0;
        grid_fisst -> minmax[NORM][MAXI]                = 0.0;
        grid_fisst -> minmax[NORM][BOTTOM]      = 0.0;
        grid_fisst -> minmax[NORM][TOP]         = 0.0;
        grid_fisst -> field_ids[SCALAR_ID]      = -1;

        /* update and redraw the sliders */

        update_minmax_sliders(grid_fisst -> minmax, LEGEND);
        update_minmax_sliders(grid_fisst -> minmax, CLIP);
        update_minmax_sliders(grid_fisst -> minmax, NORM);

        unlock_cur_object();
    }
    else if ( type == VECTOR )
    {
        lock_cur_object();

        /* if vector shaded surface deallocate any normals data */

        if ( grid_fisst->type == VECTOR && USES_NORMALS( grid_fisst ) )
        {
            delete_normals();
        }

        /* reinitialize part of the grid surface data structure */

        grid_fisst -> zones[VECTOR_TYPE][REG]   = 0;
        grid_fisst -> zones[VECTOR_TYPE][FLD]   = 0;
        grid_fisst -> dims[VECTOR_TYPE][I]      = 0;
        grid_fisst -> dims[VECTOR_TYPE][J]      = 0;
        grid_fisst -> dims[VECTOR_TYPE][K]      = 0;
        grid_fisst -> field_ids[VECTOR_ID]      = -1;

        unlock_cur_object();
    }
}

/*---------------------------- END OF reset_data -----------------------------*/
```



```
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *
 --*/

/*---------------------------- vector_scale ----------------------------------*/

static void vector_scale( int index, float new_value )
{
    interactive = 1;
    load_command( scale_group_script_commands[index], new_value );
}

/*------------------------ END OF vector_scale -------------------------------*/
```

```
/*++ static void set_vector_scale( char* script_command )
 *
 * PURPOSE:
 *
 *              Sets the vector or frame scale value to this new value.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              11/90
 *              12/90   converted to scale group call-back
 *              5/91    converted to scripting
 *
 * INPUT PARAMETERS:
 *
 *              char*   script_command  script command or actuator
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *              extern Grid_Surface*    grid_fisst      defined in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
 --*/

/*---------------------------- set_vector_scale ----------------------------*/

static void set_vector_scale( char* script_command )
{
    float           new_value;
    int             index;
    char            command[32];

    sscanf( script_command, "%s", command );
    parse_command( script_command, "%f", &new_value );

    if ( ( index = command_index( command,
            scale_group_script_commands, SCALE_GROUP_NUM_VALUES ) ) == -1 )
    {
        interactive = 0;
        return;
    }

    lock_cur_object();

    grid_fisst -> scale_factors[index] = new_value;

    if ( ! interactive )
        ACCESS4( scale_group, set_values, grid_fisst -> scale_factors, 0);
    interactive = 0;

    unlock_cur_object();
}

/*------------------------ END OF set_vector_scale ----------------------------*/
```

```
/*++ static void set_minmax_func( char* script_command )
 *
 * PURPOSE:
 *
 *              Sets the clip and norm top and bottom values to the
 *              values entered in the typeins.
 *
 *
 * AUTHORS:
```

```
*
*               Todd Plessel
*               NASA Ames Research Center
*               Sterling Software
*
* REVISION HISTORY:
*
*               4/89
*
* INPUT PARAMETERS:
*
*               char*   script_command          actuator / command
*
* OUTPUT PARAMETERS:
*
*               None
*
* FUNCTION RETURN:
*
*               None
*
* GLOBAL VARIABLES USED:
*
*               extern Minmax_Acts      minmax_acts this file
*               extern Grid_Surface*    grid_fisst      defined in this file
*
* FILES USED:
*
*               None
*
* NOTES:
*
*               Requires that math.h be included for atof()
*
* NON-STANDARD CODE :
*
* CALLED BY :
*
* FUNCTIONS CALLED :
*
*       void            update_minmax_sliders()         defined in this file
*       int             lock_cur_object()               defined in this file
*       int             unlock_cur_object()             defined in this file
*
--*/

/*----------------------------- set_minmax_func -------------------------------*/

static void set_minmax_func( char* script_command )
{
        char*           num_str;                /* -> typein string     */
        float           minmax[2];              /* entered minmax value */
        int             clip_or_norm;           /* CLIP or NORM typein  */
        int             legend = 0;             /* 1 if act is legend   */
        Actuator*       a;
        Actuator*       a1;
        Actuator*       a2;
        float           min, max;
        char*           minval;
        char*           maxval;
        char            mode[16];

        if (is_act(script_command))
        {
           a = (Actuator*) script_command;

           if (strcmp(a -> u, CLIP_BOT_ID) == 0 ||
               strcmp(a -> u, CLIP_TOP_ID) == 0     )
           {
                minval = PNL_ACCESS(Typein, minmax_acts.clip_bot_typein, str);
                maxval = PNL_ACCESS(Typein, minmax_acts.clip_top_typein, str);
                load_command("MINMAX %s %f %f", "CLIP", atof(minval),
                                                        atof(maxval));
           }
           else if (strcmp(a->u, NORM_BOT_ID) == 0 ||
                    strcmp(a->u, NORM_TOP_ID) == 0     )
           {
                minval = PNL_ACCESS(Typein, minmax_acts.norm_bot_typein, str);
                maxval = PNL_ACCESS(Typein, minmax_acts.norm_top_typein, str);
                load_command("MINMAX %s %f %f", "NORM", atof(minval),
                                                        atof(maxval));
           }
           else if (strcmp(a -> u, LEGEND_MIN_ID) == 0 ||
                    strcmp(a -> u, LEGEND_MAX_ID) == 0     )
           {
                minval = PNL_ACCESS(Typein, minmax_acts.legend_min_typein, str);
                maxval = PNL_ACCESS(Typein, minmax_acts.legend_max_typein, str);
                load_command("MINMAX %s %f %f", "LEGEND", atof(minval),
                                                          atof(maxval));
           }
           else
           {
                return;
           }
           return;
        }

        parse_command(script_command, "%s %f %f", mode, &min, &max);

        if (strcasecmp(mode, "CLIP") == 0)
        {
                clip_or_norm    = CLIP;
                a1 = minmax_acts.clip_bot_typein;
                a2 = minmax_acts.clip_top_typein;
        }
        else if (strcasecmp(mode, "NORM") == 0)
        {
                clip_or_norm    = NORM;
                a1 = minmax_acts.norm_bot_typein;
                a2 = minmax_acts.norm_top_typein;
        }
        else if (strcasecmp(mode, "LEGEND") == 0)
        {
                legend          = 1;
                a1 = minmax_acts.legend_min_typein;
                a2 = minmax_acts.legend_max_typein;
        }
        else
        {
                return;
        }

        /* store the float equivalent in minmax */

        minmax[0] = min;
        minmax[1] = max;
```



```
        /* make sure the typein always displays a valid float number */

        if (minmax[0] == 0.0)
        {
                num_str = PNL_ACCESS(Typein, a1, str);
                sprintf(num_str, FLOAT_STRING_FORMAT, 0.0);
                pnl_fixact(a1);
        }
        if (minmax[1] == 0.0)
        {
                num_str = PNL_ACCESS(Typein, a2, str);
                sprintf(num_str, FLOAT_STRING_FORMAT, 0.0);
                pnl_fixact(a2);
        }

        /* make sure that max >= min */

        if (minmax[1] < minmax[0])
        {
                minmax[0] = minmax[1];

                num_str = PNL_ACCESS(Typein, a1, str);
                sprintf(num_str, FLOAT_STRING_FORMAT, minmax[0]);
                num_str = PNL_ACCESS(Typein, a2, str);
                sprintf(num_str, FLOAT_STRING_FORMAT, minmax[1]);
        }

        /* if the legend minmax was changed, limit the clip & norm minmax */

        lock_cur_object();

        if (legend)
        {
                /* set the clip & norm minmax to these values */

                grid_fisst -> minmax[CLIP][MAXI]                = minmax[1];
                grid_fisst -> minmax[CLIP][TOP]         = minmax[1];
                grid_fisst -> minmax[CLIP][BOTTOM]      = minmax[0];
                grid_fisst -> minmax[CLIP][MINI]                = minmax[0];
                grid_fisst -> minmax[NORM][MAXI]                = minmax[1];
                grid_fisst -> minmax[NORM][TOP]         = minmax[1];
                grid_fisst -> minmax[NORM][BOTTOM]      = minmax[0];
                grid_fisst -> minmax[NORM][MINI]                = minmax[0];

                /* update all of these actuators */

                update_minmax_sliders(grid_fisst -> minmax, LEGEND);
                update_minmax_sliders(grid_fisst -> minmax, CLIP);
                update_minmax_sliders(grid_fisst -> minmax, NORM);
        }

        /* else just adjusted a top or bot value so fix only that group */

        else
        {
                /* make sure values entered are within the legend minmax */

                if (minmax[0] < grid_fisst -> minmax[clip_or_norm][MINI])
                {
                        minmax[0] = grid_fisst -> minmax[clip_or_norm][MINI];
                }

                if (minmax[1] > grid_fisst -> minmax[clip_or_norm][MAXI])
                {
                        minmax[1] = grid_fisst -> minmax[clip_or_norm][MAXI];
                }

                /* reset the active minmax to this value */

                grid_fisst -> minmax[clip_or_norm][BOTTOM] = minmax[0];
                grid_fisst -> minmax[clip_or_norm][TOP] = minmax[1];

                /* update and redraw the sliders */

                update_minmax_sliders(grid_fisst -> minmax, clip_or_norm);
        }

        unlock_cur_object();

}

/*------------------------- END OF set_minmax_func ----------------------------*/



/*++ static void reset_minmax( char* script_command )
 *
 * PURPOSE:
 *
 *              Resets the clip and norm minmax values to the actual
 *              data minmax.
 *
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              4/89
 *              5/91            Added scripting
 *
 * INPUT PARAMETERS:
 *
 *              char*           script_command  command/actuator
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *              extern Minmax_Acts      minmax_acts;    defined in this file
 *              extern Grid_Surface*    grid_fisst      defined in this file
```

```
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      extern int      get_data_minmax()               libflddata
 *      extern int      get_data_list_minmax()          libflddata
 *      void            update_minmax_sliders()         defined in this file
 *      int             lock_cur_object()               defined in this file
 *      int             unlock_cur_object()             defined in this file
 *
 --*/

/*------------------------------ reset_minmax ------------------------------*/

static void reset_minmax( char* script_command )
{
        float           minmax[NUM_VARS][2];    /* zone minmax values   */
        int             type;                   /* CLIP, NORM or LEGEND */
        Actuator*       a;                      /* Actuator             */
        char            mode[16];



        if (is_act(script_command))
        {
            a = (Actuator*) script_command;

            if (strcmp(a -> u, RESET_CLIP_ID) == 0)
            {
                    load_command("RESET_MINMAX %s", "CLIP");
            }
            else if (strcmp(a -> u, RESET_NORM_ID) == 0)
            {
                    load_command("RESET_MINMAX %s", "NORM");
            }
            else if (strcmp(a -> u, RESET_LEGEND_ID) == 0)
            {
                    load_command("RESET_MINMAX %s", "LEGEND");
            }
            else
            {
                    return;
            }
            return;
        }

        parse_command(script_command, "%s", mode);

        if (strcasecmp(mode, "CLIP") == 0)
        {
                type = CLIP;
        }
        else if (strcasecmp(mode, "NORM") == 0)
        {
                type = NORM;
        }
        else if (strcasecmp(mode, "LEGEND") == 0)
        {
                type = LEGEND;
        }
        else
        {
                return;
        }

        lock_cur_object();

        if (type == LEGEND)
        {
                /* if we are not attached to any data then just return */

                if (grid_fiest -> zones[SCALAR_TYPE][FLD] == 0)
                {
                        unlock_cur_object();
                        return;
                }

                /* if multi zone then get the minmax of all fields in reg */

                if (minmax_acts.mode_buttons[MULTI_ZONE_MINMAX] -> val == 1.0)
                {
                    if (!get_data_list_minmax(  SCALAR +
                                grid_fiest -> zones[SCALAR_TYPE][REG],
                                minmax))
                    {
                        Error("No minmax data available for register!");
                        unlock_cur_object();
                        return;
                    }
                }

                /* else if we are in single zone mode, get this zone minmax */

                else if (minmax_acts.mode_buttons[SINGLE_ZONE_MINMAX] -> val == 1.0)
                {
                    if (!get_data_minmax(       SCALAR +
                                grid_fiest -> zones[SCALAR_TYPE][REG],
                                grid_fiest -> zones[SCALAR_TYPE][FLD],
                                minmax))
                    {
                        Error("No minmax for scalar field!\n");
                        unlock_cur_object();
                        return;
                    }
                }

                /* reset the active minmax to these data values */

                grid_fiest -> minmax[CLIP][MINI]                = minmax[0][0];
                grid_fiest -> minmax[CLIP][MAXI]                = minmax[0][1];
                grid_fiest -> minmax[CLIP][BOTTOM]      = minmax[0][0];
                grid_fiest -> minmax[CLIP][TOP]         = minmax[0][1];

                grid_fiest -> minmax[NORM][MINI]                = minmax[0][0];
                grid_fiest -> minmax[NORM][MAXI]                = minmax[0][1];
                grid_fiest -> minmax[NORM][BOTTOM]      = minmax[0][0];
                grid_fiest -> minmax[NORM][TOP]         = minmax[0][1];
```



```
                /* update and redraw the sliders */

                update_minmax_sliders(grid_fiest -> minmax, LEGEND);
                update_minmax_sliders(grid_fiest -> minmax, CLIP);
                update_minmax_sliders(grid_fiest -> minmax, NORM);
        }

        else
        {
                /* reset the top & bottom sliders to their minmax */

                grid_fiest -> minmax[type][BOTTOM] =
                                grid_fiest -> minmax[type][MINI];
                grid_fiest -> minmax[type][TOP] =
                                grid_fiest -> minmax[type][MAXI];

                /* update and redraw the sliders */

                update_minmax_sliders(grid_fiest -> minmax, type);

        }

        unlock_cur_object();

}

/*---------------------------- END OF reset_minmax ----------------------------*/




/*++ static void adjust_minmax_func( char* script_command )
 *
 * PURPOSE:
 *
 *              Adjusts the clip and norm minmax values based on the slider
 *              specifications.
 *
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              4/89
 *              5/91    Added scripting
 *
 * INPUT PARAMETERS:
 *
 *               char* script_command           Actuator/command
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *              extern Minmax_Acts      minmax_acts this file
 *              extern Grid_Surface*    grid_fiest      defined in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      extern float    Denorm()                Fgl
 *      void            update_palettes()       defined in this file
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
 --*/

/*---------------------------- adjust_minmax_func ----------------------------*/

static void adjust_minmax_func( char* script_command )
{
        Actuator*       a;                      /* points to slider act */
        float           minval, maxval;         /* typein act floats    */


        if (is_act(script_command))
        {
                a = (Actuator*) script_command;

                lock_cur_object();

                if (strcmp(a -> u, CLIP_MSLIDER_ID) == 0)
                {
                    a = minmax_acts.clip_multislider;
                    minval = Denorm(((a -> al) -> next) -> extval,
                                grid_fiest -> minmax[CLIP][MINI],
                                grid_fiest -> minmax[CLIP][MAXI]);
                    maxval = Denorm((a -> al) -> extval,
                                grid_fiest -> minmax[CLIP][MINI],
                                grid_fiest -> minmax[CLIP][MAXI]);
                    load_command("MINMAX %s %f %f", "CLIP", minval, maxval);
                }
                else if (strcmp(a -> u, NORM_MSLIDER_ID) == 0)
                {
                    a = minmax_acts.norm_multislider;
                    minval = Denorm(a -> al -> next -> extval,
                                grid_fiest -> minmax[NORM][MINI],
                                grid_fiest -> minmax[NORM][MAXI]);
                    maxval = Denorm(a -> al -> extval,
                                grid_fiest -> minmax[NORM][MINI],
                                grid_fiest -> minmax[NORM][MAXI]);
                    load_command("MINMAX %s %f %f", "NORM", minval, maxval);
                }
```

```
            unlock_cur_object();
            return;
        }
}

/*---------------------- END OF adjust_minmax_func ---------------------------*/




/*++ static void invert_clip_test( char* script_command )
 *
 * PURPOSE:
 *
 *              Sets the clip test mode based on the button selections.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/91
 *
 * INPUT PARAMETERS:
 *
 *              char* script_command            command/actuator
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Minmax_Acts      minmax_acts     defined in this file
 *      extern Grid_Surface*    grid_first              declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              int     lock_cur_object()       defined in this file
 *              int     unlock_cur_object()     defined in this file
 *
 *
 --*/

/*------------------------------ invert_clip_test ----------------------------*/

static void invert_clip_test( char* script_command )
{
        Actuator*       a = minmax_acts.invert_clip_test_button;
        char            mode[16];


        if ( is_act( script_command ) )
        {
            interactive = 1;
            load_command( "CLIP %s", a -> val == 1.0 ? "INSIDE" : "OUTSIDE" );
            return;
        }

        parse_command( script_command, "%s", mode );

        if ( ! interactive )
        {
            a -> val = (float) strcasecmp( mode, "OUTSIDE" );
            pnl_fixact( a );
        }

        interactive = 0;

        lock_cur_object();
        grid_first -> invert_clip_test = (int) a -> val;
        unlock_cur_object();
}

/*------------------------ END OF invert_clip_test --------------------------*/




/*++ static void set_minmax_modes( char* script_command )
 *
 * PURPOSE:
 *
 *              Sets the minmax mode based on the button selections.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              7/89    original version
 *              9/90    added multi zone minmax mode
 *              5/91    added scripting
```



```
 *
 * INPUT PARAMETERS:
 *
 *              char* script_command            command/actuator
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Minmax_Acts      minmax_acts     defined in this file
 *      extern Grid_Surface*    grid_first              declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              void    reset_minmax()          defined in this file
 *              int     lock_cur_object()       defined in this file
 *              int     unlock_cur_object()     defined in this file
 *
 --*/

/*------------------------------ set_minmax_modes ----------------------------*/

static void set_minmax_modes( char* script_command )
{
        Actuator*       a;
        char            mode[32], val[8];
        int             fix_minmax_buttons = 0;

        if (is_act(script_command))
        {
            interactive = 1;

            a = (Actuator*) script_command;
            if (strcmp(a -> u, AUTO_MINMAX_UPDATE_ID) == 0)
            {
                load_command("AUTO_MINMAX %s", ON_OR_OFF(a -> val));
            }
            else if (strcmp(a -> u, UPDATE_MMSLIDERS_ID) == 0)
            {
                load_command("UPDATE_MINMAX %s", ON_OR_OFF(a -> val));
            }
            else if (strcmp(a -> u, MULTI_ZONE_MINMAX_ID) == 0)
            {
                load_command("MINMAX_MODE %s", "MULTI_ZONE");
            }
            else if (strcmp(a -> u, SINGLE_ZONE_MINMAX_ID) == 0)
            {
                load_command("MINMAX_MODE %s", "SINGLE_ZONE");
            }
            else if (strcmp(a -> u, SUBSET_MINMAX_ID) == 0)
            {
                load_command("MINMAX_MODE %s", "ZONE_SUBSET");
            }
            else if (strcmp(a -> u, SURFACE_MINMAX_ID) == 0)
            {
                load_command("MINMAX_MODE %s", "SURFACE");
            }
            else if (strcmp(a -> u, SURFACE_SUBSET_MINMAX_ID) == 0)
            {
                load_command("MINMAX_MODE %s", "SURFACE_SUBSET");
            }
            else
            {
                interactive = 0;
            }
            return;
        }

        parse_command(script_command, "%s", mode);

        lock_cur_object();

        if (strncasecmp( script_command, "AUTO_MINMAX", 11 ) == 0)
        {
          grid_first -> auto_minmax_update = (int) ON_OR_OFF_VAL( mode );
          if ( grid_first -> auto_minmax_update == 1 )
          {
            if ( grid_first -> field_ids[SCALAR_ID] != -1 )
              reset_minmax( "RESET_MINMAX LEGEND" );
          }
          if ( ! interactive )
          {
            minmax_acts.auto_minmax_update_button -> val = ON_OR_OFF_VAL(val);
            pnl_fixact(minmax_acts.auto_minmax_update_button);
          }
        }
        else if (strncasecmp( script_command, "UPDATE_MINMAX", 13 ) == 0)
        {
          grid_first -> update_minmax_sliders = (int) ON_OR_OFF_VAL( mode );
          if ( ! interactive )
          {
            minmax_acts.update_minmax_sliders_button -> val = ON_OR_OFF_VAL(val);
            pnl_fixact(minmax_acts.update_minmax_sliders_button);
          }
        }
        else if (strcasecmp(mode, "MULTI_ZONE") == 0)
        {
          grid_first -> minmax_mode = MULTI_ZONE_MINMAX;
          reset_minmax( "RESET_MINMAX LEGEND" );
          clip_and_norm( TRUE );
          if ( ! interactive ) fix_minmax_buttons = 1;
        }
        else if (strcasecmp(mode, "SINGLE_ZONE") == 0)
        {
          grid_first -> minmax_mode = SINGLE_ZONE_MINMAX;
          reset_minmax( "RESET_MINMAX LEGEND" );
          clip_and_norm( TRUE );
          if ( ! interactive ) fix_minmax_buttons = 1;
        }
        else if (strcasecmp(mode, "ZONE_SUBSET") == 0)
        {
          grid_first -> minmax_mode = SUBSET_MINMAX;
```

```
        clip_and_norm( FALSE );
        if ( ! interactive ) fix_minmax_buttons = 1;
      }
      else if (strcasecmp(mode, "SURFACE") == 0)
      {
        grid_fisst -> minmax_mode = SURFACE_MINMAX;
        clip_and_norm( FALSE );
        if ( ! interactive ) fix_minmax_buttons = 1;
      }
      else if (strcasecmp(mode, "SURFACE_SUBSET") == 0)
      {
        grid_fisst -> minmax_mode = SURFACE_SUBSET_MINMAX;
        clip_and_norm( FALSE );
        if ( ! interactive ) fix_minmax_buttons = 1;
      }

      if ( fix_minmax_buttons )
      {
        fix_radio_buttons( minmax_acts.mode_buttons,
                           NUM_SCALAR_MINMAX_MODES,
                           grid_fisst -> minmax_mode );
      }

      interactive = 0;

      unlock_cur_object();

      /*
       * Must wait here until the object has been redrawn
       * since redrawing can change the minmax values.
       */

      wait_until_object_is_redrawn();

      /*
       * After drawing (which updates the minmax values),
       * update the scalar minmax panel (legend etc.).
       */

      update_minmax();
}

/*---------------------- END OF set_minmax_modes ---------------------------*/
```

```
/*++ static void clip_and_norm( int state )
 *
 * PURPOSE:
 *
 *              Enables or disables use of clipping and normalization sliders.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/91
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_fisst;             declared in this file
 *      extern Minmax_Acts      minmax_acts;            declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              int     lock_cur_object()       defined in this file
 *              int     unlock_cur_object()     defined in this file
 *
 --*/

/*------------------------------ clip_and_norm -----------------------------*/

static void clip_and_norm( int state )
{
        int     i;

        lock_cur_object();

        if ( state == FALSE ) /* FALSE so disable and hide the actuators */
        {
                /* Reset & hide clip test button (to prevent 100% clipping) */

                grid_fisst -> invert_clip_test = FALSE;
                minmax_acts.invert_clip_test_button ->  val = 0.0;
                minmax_acts.invert_clip_test_button ->  selectable = 0;

                /* Hide the clipping and normalization actuators */

                minmax_acts.clip_label ->               selectable = 0;
                minmax_acts.norm_label ->               selectable = 0;
                minmax_acts.clip_top_typein ->          selectable = 0;
                minmax_acts.norm_top_typein ->          selectable = 0;
                minmax_acts.legend_max_typein ->        selectable = 0;
```



```
                minmax_acts.clip_multislider ->         selectable = 0;
                minmax_acts.norm_multislider ->         selectable = 0;
                minmax_acts.clip_bot_typein ->          selectable = 0;
                minmax_acts.norm_bot_typein ->          selectable = 0;
                minmax_acts.legend_min_typein ->        selectable = 0;
                minmax_acts.reset_clip_button ->        selectable = 0;
                minmax_acts.reset_norm_button ->        selectable = 0;
                minmax_acts.reset_legend_button ->      selectable = 0;

                for ( i = 0; i < NUM_PALETTE_TYPES; ++i )
                {
                    if ( minmax_acts.palettes[NORM][i] )
                        minmax_acts.palettes[NORM][i]->  selectable = 0;
                  minmax_acts.palettes[CLIP][i] ->      selectable = 0;
                }
        }
        else /* TRUE so enable and show the actuators */
        {
                /* Show clip test button */

                minmax_acts.invert_clip_test_button ->  selectable = 1;

                /* Show the clipping and normalization actuators */

                minmax_acts.clip_label ->               selectable = 1;
                minmax_acts.norm_label ->               selectable = 1;
                minmax_acts.clip_top_typein ->          selectable = 1;
                minmax_acts.norm_top_typein ->          selectable = 1;
                minmax_acts.legend_max_typein ->        selectable = 1;
                minmax_acts.clip_multislider ->         selectable = 1;
                minmax_acts.norm_multislider ->         selectable = 1;
                minmax_acts.clip_bot_typein ->          selectable = 1;
                minmax_acts.norm_bot_typein ->          selectable = 1;
                minmax_acts.legend_min_typein ->        selectable = 1;
                minmax_acts.reset_clip_button ->        selectable = 1;
                minmax_acts.reset_norm_button ->        selectable = 1;
                minmax_acts.reset_legend_button ->      selectable = 1;

                for ( i = 0; i < NUM_PALETTE_TYPES; ++i )
                {
                    if ( minmax_acts.palettes[NORM][i] )
                        minmax_acts.palettes[NORM][i]->  selectable = 1;
                  minmax_acts.palettes[CLIP][i] ->      selectable = 1;
                }
        }

        pnl_fixact( minmax_acts.invert_clip_test_button );
        pnl_fixact( minmax_acts.clip_label );
        pnl_fixact( minmax_acts.norm_label );
        pnl_fixact( minmax_acts.clip_top_typein );
        pnl_fixact( minmax_acts.norm_top_typein );
        pnl_fixact( minmax_acts.legend_max_typein );
        pnl_fixact( minmax_acts.clip_multislider );
        pnl_fixact( minmax_acts.norm_multislider );
        pnl_fixact( minmax_acts.clip_bot_typein );
        pnl_fixact( minmax_acts.norm_bot_typein );
        pnl_fixact( minmax_acts.legend_min_typein );
        pnl_fixact( minmax_acts.reset_clip_button );
        pnl_fixact( minmax_acts.reset_norm_button );
        pnl_fixact( minmax_acts.reset_legend_button );

        for ( i = 0; i < NUM_PALETTE_TYPES; ++i )
        {
          if ( minmax_acts.palettes[NORM][i] )
                pnl_fixact( minmax_acts.palettes[NORM][i] );
            pnl_fixact( minmax_acts.palettes[CLIP][i] );
        }

        unlock_cur_object();
}

/*---------------------- END OF clip_and_norm ---------------------------*/
```

```
/*++ static int copy_normals( void )
 *
 * PURPOSE:
 *
 *              Allocates a new set of polygon normals data
 *              if needed, and copies the existing data into it.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              11/89
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              int             1 if successful else 0
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_fisst;             declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
```

```
 *
 *              SALLOCATE() macro
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
 --*/

/*------------------------------ copy_normals ------------------------------*/

static int copy_normals( void )
{
        int             old_id;                 /* old normals id       */
        float*          old_data;               /* -> old normals data  */
        int             new_id;                 /* new normals id       */
        float*          new_data;               /* -> new normals data  */
        int             dir;                    /* I/J/K                */
        int             type;                   /* START/END/MID/ZONE   */
        int             i;                      /* index of id          */
        int*            dims;                   /* -> IJK dimensions    */
        int             size;                   /* norm size (in bytes) */


#ifdef DEBUG
printf("debug: copying normals\n");
#endif

        lock_cur_object();

        dims = &(grid_first -> dims[GRID_TYPE][0]);

        for ( dir = 0; dir < 3; ++dir )
        {
            for ( type = 0; type < 4; ++type )
            {
                i = NORM_ID_INDEX( dir, type );
                size = 3 * sizeof (float);

                if ( type == ZONE ) size *= dims[I] * dims[J] * dims[K];
                else
                {
                    switch ( dir )
                    {
                        case I: size *= dims[K] * dims[J]; break;
                        case J: size *= dims[K] * dims[I]; break;
                        case K: size *= dims[J] * dims[I]; break;
                        default: break;
                    }
                }

                if ( ( old_id = grid_first -> field_ids[i] ) != -1 )
                {
                    if ( ( old_data = (float*) ATTACH( old_id ) ) == NULL )
                    {
                        Error("Could not attach to existing normals data!");
                        unlock_cur_object();
                        return 0;
                    }

                    if ( ( new_id = SALLOCATE( size ) ) == -1 )
                    {
                        Error("Could not allocate new normals data!");
                        DETACH( old_data );
                        unlock_cur_object();
                        return 0;
                    }

                    if ( ( new_data = (float*) ATTACH( new_id ) ) == NULL )
                    {
                        Error("Could not attach to new normals data!");
                        DETACH( old_data );
                        SDEALLOCATE( new_id );
                        unlock_cur_object();
                        return 0;
                    }

                    memcpy( new_data, old_data, size );
                    DETACH( old_data );
                    DETACH( new_data );

                    grid_first -> field_ids[i] = new_id;
                }
            }
        }

        unlock_cur_object();

        return 1;
}

/*---------------------------- END OF copy_normals ----------------------------*/




/*++ static void delete_normals( void )
 *
 * PURPOSE:
 *
 *              Deallocates the polygon normals data.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              11/89
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
```



```
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_first;             declared in this file
 *      extern int              update_actuators_mode;  declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              SDEALLOCATE() macro
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
 --*/

/*---------------------------- delete_normals ----------------------------*/

static void delete_normals( void )
{
        int             dir;                    /* I/J/K                */
        int             type;                   /* START/END/MID/ZONE   */
        int             i;                      /* index of id          */


        if ( update_actuators_mode == 1 ) return;

#ifdef DEBUG
printf("debug: deleting normals\n");
#endif
        lock_cur_object();

        /* deallocate existing normals data and set ids to -1 */

        for ( dir = 0; dir < 3; ++dir )
        {
            for ( type = 0; type < 4; ++type )
            {
                i = NORM_ID_INDEX( dir, type );

                if ( grid_first -> field_ids[i] != -1 )
                {
                    SDEALLOCATE(grid_first->field_ids[i]);
                    grid_first -> field_ids[i] = -1;
                }
            }
        }

        unlock_cur_object();

        /* (new normals data will be generated by the drawing routines) */

}

/*--------------------------- END OF delete_normals ---------------------------*/




/*++ static void check_delete_normals( int new_dir, int new_ranges[3][5] )
 *
 * PURPOSE:
 *
 *              Checks and then deallocates the normals data if necessary.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              6/91
 *
 * INPUT PARAMETERS:
 *
 *              int             new_dir         I, J, or K
 *              int             new_ranges[3][5] [I/J/K][START/END/INC/CUR/DIM]
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_first;             declared in this file
 *      extern int              update_actuators_mode;  declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              SDEALLOCATE() macro
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *      void            delete_normals()        defined in this file
 *
 --*/
```

```
/*------------------------- check_delete_normals ---------------------------*/

static void check_delete_normals( int new_dir, int new_ranges[3][5] )
{
        int             old_dir;                /* IJK direction        */
        int*            old_range;              /* START/END/MID/ZONE   */
        int             loop_surface;           /* START/END/MID        */
        int             zone_normals;           /* use zone normals?    */
        int             changed = 0;            /* delete them?         */
        int             cur;                    /* loop_surface index   */
        int             count;                  /* count surfaces drawn */
        int             i;                      /* index on normals id  */
        int             drawn;                  /* boundary surf drawn? */
        int             dir;                    /* IJK direction        */
        int             type;                   /* START/END/MID/ZONE   */


#ifdef DEBUG
printf("debug: check deleting normals\n");
#endif

        lock_cur_object();

        zone_normals    = grid_fisst -> zone_normals;
        loop_surface    = grid_fisst -> loop_surface;
        old_dir         = grid_fisst -> direction;
        old_range       = &(grid_fisst -> ranges[old_dir][0]);


        switch ( grid_fisst -> surface_mode )
        {
            case SINGLE_SURFACE:

                if ( new_dir != old_dir ) changed = 1;
                else if ( ! zone_normals )
                {
                    cur = loop_surface == MID ? CUR : loop_surface;
                    changed = old_range[cur] != new_ranges[old_dir][cur];

                    for ( dir = 0; dir < 3; ++dir )
                    {
                        i = NORM_ID_INDEX( dir, ZONE );

                        if ( grid_fisst -> field_ids[i] != -1 )
                        {
#ifdef DEBUG
printf("debug: deleting normals for surface dir = %d, ZONE\n", dir );
#endif

                            SDEALLOCATE( grid_fisst -> field_ids[i] );
                            grid_fisst -> field_ids[i] = -1;
                        }
                    }
                }

                break;

            case BOUNDARY_SURFACES:

                for ( dir = 0; dir < 3; ++dir )
                {
                    count = 0;
                    old_range = &(grid_fisst -> ranges[dir][0]);

                    for ( type = 0; type < 4; ++type )
                    {
                        i = NORM_ID_INDEX( dir, type );
                        drawn = 1;

                        if ( type != ZONE )
                          count += drawn = grid_fisst->boundary_flags[dir][type];

                        cur = type == MID ? CUR : type;
                        changed = !zone_normals &&
                                old_range[cur] != new_ranges[dir][cur];

                        if ( drawn == 0 || changed ||
                            ( zone_normals && type == ZONE && count == 0 ) )
                        {
                            if ( grid_fisst -> field_ids[i] != -1 )
                            {
#ifdef DEBUG
printf("debug: deleting normals for surface dir = %d, type = %d\n", dir, type);
#endif

                                SDEALLOCATE( grid_fisst -> field_ids[i] );
                                grid_fisst -> field_ids[i] = -1;
                            }
                        }
                    }
                }

                changed = 0;
                break;

            case SURFACE_RANGE: changed = old_dir != new_dir; break;

            default: break;
        }

        if ( changed ) delete_normals();

        unlock_cur_object();

}

/*---------------------- END OF check_delete_normals -------------------------*/
```

```
/*++ static int copy_contours( void )
 *
 * PURPOSE:
 *
 *              Allocates a new set of contours data
 *              if needed, and copies the existing data into it.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              11/89
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              int             1 if successful else 0
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_fisst;             declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *              SALLOCATE() macro
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
 --*/

/*------------------------------ copy_contours -----------------------------*/

static int copy_contours( void )
{
        int             old_id;                 /* old contours id      */
        float*          old_data;               /* -> old contours data */
        int             new_id;                 /* new contours id      */
        float*          new_data;               /* -> new contours data */
        int             size;                   /* size (in bytes)      */


#ifdef DEBUG
printf("debug: copying contours\n");
#endif

        lock_cur_object();

        size = grid_fisst -> num_con_points * 4 * sizeof (float);
```



```
        if ( ( old_id = grid_fisst -> field_ids[CONTOURS_ID] ) != -1 )
        {
          if ( ( old_data = (float*) ATTACH( old_id ) ) == NULL )
          {
            Error("Could not attach to existing contours data!");
            unlock_cur_object();
            return 0;
          }

          if ( ( new_id = SALLOCATE( size ) ) == -1 )
          {
            Error("Could not allocate new contours data!");
            DETACH( old_data );
            unlock_cur_object();
            return 0;
          }

          if ( ( new_data = (float*) ATTACH( new_id ) ) == NULL )
          {
            Error("Could not attach to new contours data!");
            DETACH( old_data );
            SDEALLOCATE( new_id );
            unlock_cur_object();
            return 0;
          }

          memcpy( new_data, old_data, size );
          DETACH( old_data );
          DETACH( new_data );

          grid_fisst -> field_ids[CONTOURS_ID] = new_id;
        }

        unlock_cur_object();

        return 1;
}

/*-------------------------- END OF copy_contours --------------------------*/
```

```
/*++ static void delete_contours( void )
 *
 * PURPOSE:
 *
 *              Deallocates the contours data.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
```

```
*
*               11/89
*
* INPUT PARAMETERS:
*
*               None
*
* OUTPUT PARAMETERS:
*
*               None
*
* FUNCTION RETURN:
*
*               None
*
* GLOBAL VARIABLES USED:
*
*       extern Grid_Surface*    grid_fisst;             declared in this file
*       extern int              update_actuators_mode;  declared in this file
*
* FILES USED:
*
*               None
*
* NOTES:
*
* NON-STANDARD CODE :
*
* CALLED BY :
*
* FUNCTIONS CALLED :
*
*               SDEALLOCATE() macro
*       int             lock_cur_object()       defined in this file
*       int             unlock_cur_object()     defined in this file
*
--*/

/*---------------------------- delete_contours ----------------------------*/

static void delete_contours( void )
{
        if ( update_actuators_mode == 1 ) return;

#ifdef DEBUG
printf("debug: deleting contours\n");
#endif
        lock_cur_object();

        /* deallocate existing contours data and set ids to -1 */

        if ( grid_fisst -> field_ids[CONTOURS_ID] != -1 )
        {
          SDEALLOCATE(grid_fisst->field_ids[CONTOURS_ID]);
          grid_fisst -> field_ids[CONTOURS_ID] = -1;
        }

        unlock_cur_object();

        /* (new contours data will be generated by the drawing routines) */
}

/*--------------------------- END OF delete_contours ----------------------*/
```

```
/*++ static void first_object_name( char* name )
*
* PURPOSE:
*
*               Passes back the name of the first available object or an empty
*               string if there are none available.
*
* AUTHORS:
*
*               Todd Plessel
*               NASA Ames Research Center
*               Sterling Software
*
* REVISION HISTORY:
*
*               1/91
*
* INPUT PARAMETERS:
*
*               None
*
* OUTPUT PARAMETERS:
*
*               None
*
* FUNCTION RETURN:
*
*               None
*
* GLOBAL VARIABLES USED:
*
*               None
*
* FILES USED:
*
*               None
* NOTES:
*
* NON-STANDARD CODE :
*
* CALLED BY :
*
*       void            init_panels()                   defined in this file
*
* FUNCTIONS CALLED :
*
--*/

/*---------------------------- first_object_name ----------------------------*/

static void first_object_name( char* name )
{
        static char     buf[OBJECT_BUF_SIZE];   /* object name listing  */
```



```
        char            command[32];
        int             i;                      /* index into buf/name  */


DPRINT("  inside first_object_name(%s):\n", name);

        memset( buf, 0, sizeof buf );

DPRINT("  buf = '%s'\n", buf);

        /* get an updated listing into the typeout buffer */

        sprintf (command, "LIST_OBJECTS %d", GRID_SURFACE);
        send_hub_command(command);
        module_read_sock(buf, sizeof(buf));

DPRINT("  back from module_read_sock( with buf = '%s')\n", buf);

        /* extract the name of the first object */

        i = 0;

        while (buf[i] != '\0' && buf[i] != '\n' && i < OBJECT_NAME_LENGTH - 1)
        {
                name[i] = buf[i];
                ++i;
        }

        name[i] = '\0';
}

/*------------------------ END OF first_object_name -------------------------*/
```

```
/*++ static void update_object_typeout( void )
*
* PURPOSE:
*
*               Updates the object typeout listing.
*
* AUTHORS:
*
*               Fergus J. Merritt
*               NASA Ames Research Center
*               Sterling Software
*
* REVISION HISTORY:
*
*               3/90
*               11/90   Todd Plessel
*                       added mutually recursive call and exit_module().
*
* INPUT PARAMETERS:
*
*               None
*
* OUTPUT PARAMETERS:
*
*               None
*
* FUNCTION RETURN:
*
*               None
*
* GLOBAL VARIABLES USED:
*
*       extern Main_Acts        main_acts;      declared in this file
*
* FILES USED:
*
*               None
* NOTES:
*
* NON-STANDARD CODE :
*
* CALLED BY :
*
*       void            init_panels()                   defined in this file
*       void            new_object()            defined in this file
*
* FUNCTIONS CALLED :
*
*       extern void     clear_typeout()                 libpanu
*       extern int      set_selection_num()             libpanu
*       extern void     select_object()                 defined in this file
*
--*/

/*--------------------------- update_object_typeout --------------------------*/

static void update_object_typeout( void )
{
        int             buf_size;
        char*           buf;            /* points into typeout buf      */
        char            command[32];


        clear_typeout(main_acts.object_typeout);

        buf = PNL_ACCESS(Typeout, main_acts.object_typeout, buf);
        buf_size = PNL_ACCESS(Typeout, main_acts.object_typeout, size);

        /* get an updated listing into the typeout buffer */

        sprintf (command, "LIST_OBJECTS %d", GRID_SURFACE);
        send_hub_command(command);
        module_read_sock(buf, buf_size);

        /* try to reselect the current object */

        if (set_selection_name(main_acts.object_typeout, cur_object_name) == 0)
        {
            /* if that failed try to select the first line */

            if (set_selection_num(main_acts.object_typeout, 1) == 0)
            {
                /* if that failed then there are no objects left so exit */
```

```
            Warning("There are no objects left so I'm exiting!\n");
            exit_module();
        }

        /* otherwise update grid surface object with this new selection */

        select_object( (char*) main_acts.object_typeout );
    }
}

/*-------------------- END OF update_object_typeout --------------------------*/
```

```
/*++ static void update_actuators( void )
 *
 * PURPOSE:
 *
 *              Resets the panel based on the contents of the grid_fisst
 *              structure.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              3/90
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Grid_Surface*    grid_fisst              declared in this file
 *      extern Main_Acts        main_acts               this file
 *      extern Minmax_Acts      minmax_acts     this file
 *      extern int              update_actuators_mode;  declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      extern void     set_typein_ival()               libpanu
 *      extern void     set_typein_fval()               libpanu
 *      extern int      set_fld_data_selection()        libfldpan
 *      extern void     update_fld_data_panel()         libfldpan
 *      void            update_colors()                 defined in this file
 *      void            update_minmax_sliders()         defined in this file
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
 --*/

/*--------------------------- update_actuators -------------------------------*/

static void update_actuators( void )
{
        int             i;
        int             temp[20];               /* for button updates   */


        /* set global flag to prevent unneeded deleting of normals data */

        update_actuators_mode = 1;

        lock_cur_object();


        /* fix the Draw the Object button */

        if ( ((int) main_acts.draw_object_button->val) != grid_fisst->draw )
        {
                main_acts.draw_object_button->val = (float) grid_fisst->draw;
                pnl_fixact( main_acts.draw_object_button );
        }

        /* fix type menu */

        ACCESS6(main_acts.type_menu_group, set_selection, 0, grid_fisst -> type, 1, 0);


        /* fix render menu */

        ACCESS6(main_acts.render_menu_group, set_selection, 0, grid_fisst -> render_mode,
1, 0);


        /* fix attributes menu */

        ACCESS6(main_acts.attributes_menu_group, set_selection, 0, grid_fisst -> contour_
color_type, 1, 0);
        ACCESS6(main_acts.attributes_menu_group, set_selection, 1, grid_fisst -> vector_c
olor_type, 1, 0);
        ACCESS6(main_acts.attributes_menu_group, set_selection, 2, grid_fisst -> vector_t
ip_type, 1, 0);
        ACCESS6(main_acts.attributes_menu_group, set_selection, 3, grid_fisst -> clip_mod
e, 1, 0);
        ACCESS6(main_acts.attributes_menu_group, set_selection, 4, grid_fisst -> shaded,
1, 0);
        ACCESS6(main_acts.attributes_menu_group, set_selection, 5, grid_fisst -> framed,
```



```
1, 0);
        ACCESS6(main_acts.attributes_menu_group, set_selection, 6, grid_fisst -> rev_nor
mals, 1, 0);
        ACCESS6(main_acts.attributes_menu_group, set_selection, 7, grid_fisst -> zone_no
rmals, 1, 0);


        /* fix the options menu */

        ACCESS6(main_acts.options_menu_group, set_selection, 0, 0, grid_fisst->draw_outl
ine, 0);
        ACCESS6(main_acts.options_menu_group, set_selection, 1, 0, grid_fisst->draw_glyp
h, 0);


        /* fix the loop buttons */

        memset( temp, 0, sizeof temp );

        temp[grid_fisst -> loop_mode] = 1;
        i = loop_buttons_per_row[0];
        temp[i + grid_fisst -> loop_surface] = 1;
        i += loop_buttons_per_row[1];
        temp[i + grid_fisst -> loop_zone] = 1;

        ACCESS4(main_acts.loop_buttons_group, set_selections, temp, 0);

        /* also set global */

        loop_mode = grid_fisst -> loop_mode;


        /* fix the slider buttons */

        memset( temp, 0, sizeof temp );
        temp[1] = grid_fisst -> reset_to_zone;
        temp[2] = grid_fisst -> show_looping;

        ACCESS4(main_acts.slider_buttons_group, set_selections, temp, 0);


        /* fix the surface mode buttons */

        memset( temp, 0, sizeof temp );
        temp[grid_fisst -> surface_mode] = 1;

        ACCESS4(main_acts.surface_buttons_group, set_selections, temp, 0);

        /* show the select buttons if boundary surfaces mode */

        if (grid_fisst -> surface_mode == BOUNDARY_SURFACES)
        {
                ACCESS2(main_acts.slider_group, show_select_buttons);
        }
        else /* hide them */
        {
                ACCESS2(main_acts.slider_group, hide_select_buttons);
        }

        /* fix the zone typein */

        ACCESS4(main_acts.zone_typein_group, set_ivalue, grid_fisst -> zones[GRID_TYPE][
FLD], 0);
```

```
        /* fix the sliders */

        ACCESS5(main_acts.slider_group, set_ivalues, I, grid_fisst -> ranges[I], 0);
        ACCESS5(main_acts.slider_group, set_ivalues, J, grid_fisst -> ranges[J], 0);
        ACCESS5(main_acts.slider_group, set_ivalues, K, grid_fisst -> ranges[K], 0);

        ACCESS4(main_acts.slider_group, set_selections, grid_fisst -> boundary_flags, 0);

        ACCESS4(main_acts.slider_group, set_direction, grid_fisst -> direction, 0);

        if (grid_fisst -> surface_mode != BOUNDARY_SURFACES)
                ACCESS2(main_acts.slider_group, hide_select_buttons);

        i = grid_fisst -> loop_surface;

        ACCESS3(main_acts.slider_group, highlight_slider, i);


        /* fix vector and frame scale factors */

        ACCESS4(scale_group, set_values, grid_fisst -> scale_factors, 0);


        /* update the scalar minmax panel */

        /* fix the invert clip test button */

        if ( ((int) minmax_acts.invert_clip_test_button->val) !=
                grid_fisst->invert_clip_test )
        {
                minmax_acts.invert_clip_test_button->val =
                        (float) grid_fisst->invert_clip_test;
                pnl_fixact( minmax_acts.invert_clip_test_button );
        }

        /* and the auto minmax update button */

        if ( ((int) minmax_acts.auto_minmax_update_button->val) !=
                grid_fisst->auto_minmax_update )
        {
                minmax_acts.auto_minmax_update_button->val =
                        (float) grid_fisst->auto_minmax_update;
                pnl_fixact( minmax_acts.auto_minmax_update_button );
        }

        /* and the update minmax sliders button */

        if ( ((int) minmax_acts.update_minmax_sliders_button->val) !=
                grid_fisst->update_minmax_sliders )
        {
                minmax_acts.update_minmax_sliders_button->val =
                        (float) grid_fisst->update_minmax_sliders;
                pnl_fixact( minmax_acts.update_minmax_sliders_button );
        }

        /* and the minmax mode buttons */

        fix_radio_buttons( minmax_acts.mode_buttons, NUM_SCALAR_MINMAX_MODES,
                        grid_fisst -> minmax_mode );

        update_minmax_sliders(grid_fisst -> minmax, CLIP);
        update_minmax_sliders(grid_fisst -> minmax, NORM);
        update_minmax_sliders(grid_fisst -> minmax, LEGEND);
```

```
        /* vector panel's scale group */

        ACCESS4( scale_group, set_values, grid_first -> scale_factors, 0 );

        /* update the color edit panel to display the new colors */

        update_colors();


        /* update the data panel by trying to select this object's zones */

        set_fld_data_selection( GRID,
                                grid_first -> zones[GRID_TYPE][REG],
                                grid_first -> zones[GRID_TYPE][FLD]);

        set_fld_data_selection( SCALAR,
                                grid_first -> zones[SCALAR_TYPE][REG],
                                grid_first -> zones[SCALAR_TYPE][FLD]);

        set_fld_data_selection( VECTOR,
                                grid_first -> zones[VECTOR_TYPE][REG],
                                grid_first -> zones[VECTOR_TYPE][FLD]);


        unlock_cur_object();

        /* this will invoke the call-back function: date_select() */

        update_fld_data_panel();

        /* reset global flag to normal */

        update_actuators_mode = 0;

}

/*------------------------- END OF update_actuators -------------------------*/
```

```
/*++ static void update_data_info( void )
 *
 * PURPOSE:
 *
 *              Updates the data info typeout.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              3/90
 *
 * INPUT PARAMETERS:
 *
 *              None
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Main_Acts        main_acts               defined in this file
 *      extern Grid_Surface*    grid_first;             declared in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      extern int      print_field_node_info()         libfldpan
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
 --*/

#define DIMS_MATCH(dims1, dims2)        (dims1[I] == dims2[I] &&        \
                                         (dims1[J] == dims2[J] &&        \
                                          dims1[K] == dims2[K]))

/*----------------------------- update_data_info ----------------------------*/

static void update_data_info( void )
{
        int             ind[NUM_DIMS];          /* holds current IJK    */
        int             dims[NUM_DIMS];         /* holds dimensions IJK */
        char*           buf;                    /* points into typeout  */
        int             temp_scalar_id;         /* pass to update data  */
        int             temp_vector_id;         /* pass to update data  */
        int             dir;                    /* IJK direction        */


        lock_cur_object();

        buf = PNL_ACCESS(Typeout, main_acts.data_info_typeout, buf);
        ind[I] = grid_first -> ranges[I][CUR];
        ind[J] = grid_first -> ranges[J][CUR];
        ind[K] = grid_first -> ranges[K][CUR];
        dims[I] = grid_first -> ranges[I][DIM];
        dims[J] = grid_first -> ranges[J][DIM];
        dims[K] = grid_first -> ranges[K][DIM];

        dir = grid_first -> direction;
```



```
        if ( grid_first -> loop_surface != MID )
            ind[dir] = grid_first -> ranges[dir][grid_first -> loop_surface];

        /* check that the dimensions are valid for the other data */

        temp_scalar_id = -1;
        temp_vector_id = -1;

        if (DIMS_MATCH(grid_first -> dims[GRID_TYPE], grid_first -> dims[SCALAR_TYPE]))
        {
                temp_scalar_id = grid_first -> field_ids[SCALAR_ID];
        }

        if (DIMS_MATCH(grid_first -> dims[GRID_TYPE], grid_first -> dims[VECTOR_TYPE]))
        {
                temp_vector_id = grid_first -> field_ids[VECTOR_ID];
        }

        print_field_node_info(ind, dims,
                grid_first -> field_ids[GRID_ID],
                grid_first -> field_ids[IBLANK_ID],
                temp_scalar_id,
                temp_vector_id,
                buf, DATA_INFO_BUF_SIZE);

        pnl_fixcat(main_acts.data_info_typeout);

        unlock_cur_object();
}

/*------------------------ END OF update_data_info --------------------------*/
```

```
/*++ static void update_dims( int type, int dims[3] )
 *
 * PURPOSE:
 *
 *              Updates the grid surface structure's dims and possible range.
 *
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              4/89
 *
 * INPUT PARAMETERS:
 *
 *              int     type            GRID_TYPE, SCALAR_TYPE, VECTOR_TYPE
 *              int     dims[NUM_DIMS]  IJK maxes
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *              extern Grid_Surface*    grid_first      defined in this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      void            delete_contours()               defined in this file
 *      void            reset_ijk_ranges()              defined in this file
 *      int             lock_cur_object()               defined in this file
 *      int             unlock_cur_object()             defined in this file
 *
 --*/

/*------------------------------ update_dims --------------------------------*/

static void update_dims( int type, int dims[3] )
{
        if (type != GRID_TYPE && type != SCALAR_TYPE && type != VECTOR_TYPE)
        {
                Error("Invalid type!\n");
                return;
        }

        /* update dims */

        lock_cur_object();

        grid_first -> dims[type][I] = dims[I];
        grid_first -> dims[type][J] = dims[J];
        grid_first -> dims[type][K] = dims[K];

        if (grid_first->render_mode == CONTOUR_LINES) delete_contours();

        unlock_cur_object();

        /* if grid type then reset the IJK ranges to within these dims */

        if (type == GRID_TYPE) reset_ijk_ranges();
}

/*---------------------------- END OF update_dims ---------------------------*/
```

```
/*++ static void update_minmax_sliders( float minmax[2][4], int type )
 *
 * PURPOSE:
 *
 *              Updates the scalar minmax sliders/typeins to
 *              the specified minmax values.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              6/89
 *
 * INPUT PARAMETERS:
 *
 *              float   minmax[2][4]    CLIP/NORM, MINI..TOP minmax values
 *              int     type;           CLIP or NORM
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Minmax_Acts      minmax_acts;    this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 * FUNCTIONS CALLED :
 *
 *      extern float    Norm()                  libfgl
 *      extern void     set_typein_fval()       libpanu
 *      void            update_legend()         defined in this file
 *      void            update_palettes()       defined in this file
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
 --*/

/*------------------------ update_minmax_sliders ----------------------------*/

static void update_minmax_sliders( float minmax[2][4], int type )
{
        Actuator*       ms;                     /* -> a multislider     */
        Actuator*       sa;                     /* -> a slider in ms    */
        Actuator*       mina;                   /* min typein           */
        Actuator*       maxa;                   /* max typein           */


        /* select appropriate multislider */

        if (type == CLIP)
        {
                ms      = minmax_acts.clip_multislider;
                mina    = minmax_acts.clip_bot_typein;
                maxa    = minmax_acts.clip_top_typein;
        }
        else if (type == NORM)
        {
                ms      = minmax_acts.norm_multislider;
                mina    = minmax_acts.norm_bot_typein;
                maxa    = minmax_acts.norm_top_typein;
        }
        else if (type == LEGEND)
        {
                mina    = minmax_acts.legend_min_typein;
                maxa    = minmax_acts.legend_max_typein;
        }
        else
        {
                return;
        }

        if (type != LEGEND)
        {
                /* update the top slider */

                sa = ms -> al;
                sprintf(sa -> label, FLOAT_STRING_FORMAT, minmax[type][TOP]);
                sa -> extval = Norm(    minmax[type][TOP],
                                        minmax[type][MINI], minmax[type][MAXI]  );

                /* update the bottom slider */

                sa = sa -> next;
                sprintf(sa -> label, FLOAT_STRING_FORMAT, minmax[type][BOTTOM]);
                sa -> extval = Norm(    minmax[type][BOTTOM],
                                        minmax[type][MINI], minmax[type][MAXI]  );

                /* redraw the appropriate multislider */

                pnl_fixact(ms);

                /* update the top and bottom typeins */

                set_typein_fval(maxa, minmax[type][TOP], FLOAT_STRING_FORMAT);
                set_typein_fval(mina, minmax[type][BOTTOM], FLOAT_STRING_FORMAT);

                /* update the palettes */

                update_palettes(type);
        }
        else /* legend */
        {
                /* update the min and max typeins */
```



```
                set_typein_fval(maxa, minmax[0][MAXI], FLOAT_STRING_FORMAT);
                set_typein_fval(mina, minmax[0][MINI], FLOAT_STRING_FORMAT);

                /* redraw legend numbers */

                update_legend(minmax[0][MINI], minmax[0][MAXI],
                                minmax_acts.legend_labels);


                /* now take care of the contour stuff */

                lock_cur_object();

                grid_fisst -> contours_min = minmax[0][MINI];
                grid_fisst -> contours_max = minmax[0][MAXI];
                grid_fisst -> contours_inc =
                        (minmax[0][MAXI] - minmax[0][MINI]) /
                                (float)grid_fisst -> num_contours;

                /* update the min/max typein */

                mina = contour_acts.contour_min_typein;
                maxa = contour_acts.contour_max_typein;

                set_typein_fval(maxa, minmax[0][MAXI], FLOAT_STRING_FORMAT);
                set_typein_fval(mina, minmax[0][MINI], FLOAT_STRING_FORMAT);

                /* redraw contour legend numbers */

                update_legend(minmax[0][MINI], minmax[0][MAXI],
                                contour_acts.contour_labels);

                set_typein_ival(contour_acts.contour_num_typein,
                        grid_fisst -> num_contours, INT_STRING_FORMAT);

                set_typein_fval(contour_acts.contour_inc_typein,
                        grid_fisst -> contours_inc, FLOAT_STRING_FORMAT);

                unlock_cur_object();
        }
}

/*--------------------- END OF update_minmax_sliders -------------------------*/









/*++ static void update_legend( float min_val, float max_val, Actuator** act )
 *
 * PURPOSE:
 *
 *              Updates the legend to the specified minmax values.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/89
 *
 * INPUT PARAMETERS:
 *
 *              float   min_val                 minumum legend value
 *              float   max_val                 maximum legend value
 *              Actuator*       *act
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Minmax_Acts      minmax_acts     this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 *              void update_minmax_sliders()    defined in this file
 *
 * FUNCTIONS CALLED :
 *
 *      int             lock_cur_object()       defined in this file
 *      int             unlock_cur_object()     defined in this file
 *
 --*/

/*----------------------------- update_legend --------------------------------*/

static void update_legend( float min_val, float max_val, Actuator** act )
{
        int             i;                      /* loop on labels       */
        float           val;                    /* value of label       */
        float           val_inc;                /* increment value      */
        Actuator*       ta;                     /* points at label acts */

        /* calc incremental value */

        val_inc = (max_val - min_val) / (float) (NUM_LEGEND_VALUES - 1);

        /* set initial value */

        val = min_val;

        /* fix each label (from the botton up) with incremental value */
```

```
        for (i = 0; i < NUM_LEGEND_VALUES; ++i)
        {
                ta = act[i];
                sprintf(ta -> label, FLOAT_STRING_FORMAT, val);
                pnl_fixact(ta);
                val += val_inc;
        }
}

/*------------------------ END OF update_legend -----------------------------*/
```

```
/*++ static void update_palettes( int clip_or_norm )
 *
 * PURPOSE:
 *
 *              Updates the palettes to reflect the new settings of
 *              the specified slider.
 *
 * AUTHORS:
 *
 *              Todd Plessel
 *              NASA Ames Research Center
 *              Sterling Software
 *
 * REVISION HISTORY:
 *
 *              9/89
 *
 * INPUT PARAMETERS:
 *
 *              int     clip_or_norm            CLIP or NORM multislider
 *
 * OUTPUT PARAMETERS:
 *
 *              None
 *
 * FUNCTION RETURN:
 *
 *              None
 *
 * GLOBAL VARIABLES USED:
 *
 *      extern Minmax_Acts      minmax_acts;    this file
 *
 * FILES USED:
 *
 *              None
 *
 * NOTES:
 *
 * NON-STANDARD CODE :
 *
 * CALLED BY :
 *
 *              void update_minmax_sliders()            defined in this file
 *              void adjust_minmax_func()               defined in this file
 *
 * FUNCTIONS CALLED :
 *
 *              extern void     draw_palettes()         libpanu
 *              int             lock_cur_object()       defined in this file
 *              int             unlock_cur_object()     defined in this file
 *
 --*/

/*------------------------------ update_palettes ------------------------------*/

static void update_palettes( int clip_or_norm )
{
        Actuator*       ms;                     /* clip or norm multislider */
        float           norm_bot_val;           /* norm bottom slider val */
        float           norm_top_val;           /* norm top slider val  */
        float           clip_bot_val;           /* clip bottom slider val */
        float           clip_top_val;           /* clip top slider val  */
        float           val;                    /* tmp normalized value */
        float           vals[NUM_PALETTE_TYPES][2]; /* bot/top pal vals */
        float           origin[2];              /* x/y coord of bot left */
        float           colors[NUM_PALETTE_TYPES][2]; /* bot/top color vals */



        /* get state of norm multislider */

        ms = minmax_acts.norm_multislider;
        norm_top_val = ms -> sl -> extval;
        norm_bot_val = ms -> sl -> next -> extval;

        origin[Y] = ms -> y;

        if (clip_or_norm == NORM)
        {
                /* fix norm palette */

                origin[X] = ms -> x - PALETTE_WIDTH;

                vals[MED][MAXIMUM] = norm_top_val;
                vals[MED][MINIMUM] = norm_bot_val;

                colors[MED][MAXIMUM] = MAX_MAP;
                colors[MED][MINIMUM] = MIN_MAP;


                draw_palettes(  NUM_PALETTE_TYPES,
                                minmax_acts.palettes[NORM],
                                vals,
                                colors,
                                PALETTE_HEIGHT,
                                origin);


                /* fix legend palettes */
```



```
                origin[X] += PALETTE_X_INC;

                vals[HI][MAXIMUM] = 1.0;
                vals[HI][MINIMUM] = norm_top_val;

                vals[LOW][MAXIMUM] = norm_bot_val;
                vals[LOW][MINIMUM] = 0.0;

                colors[HI][MAXIMUM] = MAX_MAP;
                colors[HI][MINIMUM] = MAX_MAP;

                colors[LOW][MAXIMUM] = MIN_MAP;
                colors[LOW][MINIMUM] = MIN_MAP;


                draw_palettes(  NUM_PALETTE_TYPES,
                                minmax_acts.palettes[LEGEND],
                                vals,
                                colors,
                                PALETTE_HEIGHT,
                                origin);

        }

        else
        {
                colors[HI][MAXIMUM] = MAX_MAP;
                colors[HI][MINIMUM] = MAX_MAP;
                colors[LOW][MAXIMUM] = MIN_MAP;
                colors[LOW][MINIMUM] = MIN_MAP;
        }


        /* fix clip palettes */

        /* get state of clip multislider */

        ms = minmax_acts.clip_multislider;
        clip_top_val = ms -> sl -> extval;
        clip_bot_val = ms -> sl -> next -> extval;

        origin[X] = ms -> x - PALETTE_WIDTH;

        /* calc positions and MED colors */

        if (clip_top_val >= norm_top_val)
        {
                vals[HI][MAXIMUM] = clip_top_val;
                vals[HI][MINIMUM] = MAX(clip_bot_val, norm_top_val);

                colors[MED][MAXIMUM] = MAX_MAP;
        }

        else
        {
                vals[HI][MAXIMUM] = clip_top_val;
                vals[HI][MINIMUM] = clip_top_val;

                val = Norm(clip_top_val, norm_bot_val, norm_top_val);
                colors[MED][MAXIMUM] = get_function_index(val);
        }

        if (clip_bot_val <= norm_bot_val)
        {
                vals[LOW][MAXIMUM] = MIN(clip_top_val, norm_bot_val);
                vals[LOW][MINIMUM] = clip_bot_val;

                colors[MED][MINIMUM] = MIN_MAP;
        }

        else
        {
                vals[LOW][MAXIMUM] = clip_bot_val;
                vals[LOW][MINIMUM] = clip_bot_val;

                val = Norm(clip_bot_val, norm_bot_val, norm_top_val);
                colors[MED][MINIMUM] = get_function_index(val);
        }

        /* MED palette is always between LOW and HI palettes */

        vals[MED][MAXIMUM] = vals[HI][MINIMUM];
        vals[MED][MINIMUM] = vals[LOW][MAXIMUM];


        draw_palettes(  NUM_PALETTE_TYPES,
                        minmax_acts.palettes[CLIP],
                        vals,
                        colors,
                        PALETTE_HEIGHT,
                        origin);

        pnl_fixact(minmax_acts.minmax_frame);

}

/*------------------------ END OF update_palettes ---------------------------*/



/*---------------------------- END OF FILE panels.c --------------------------*/
```