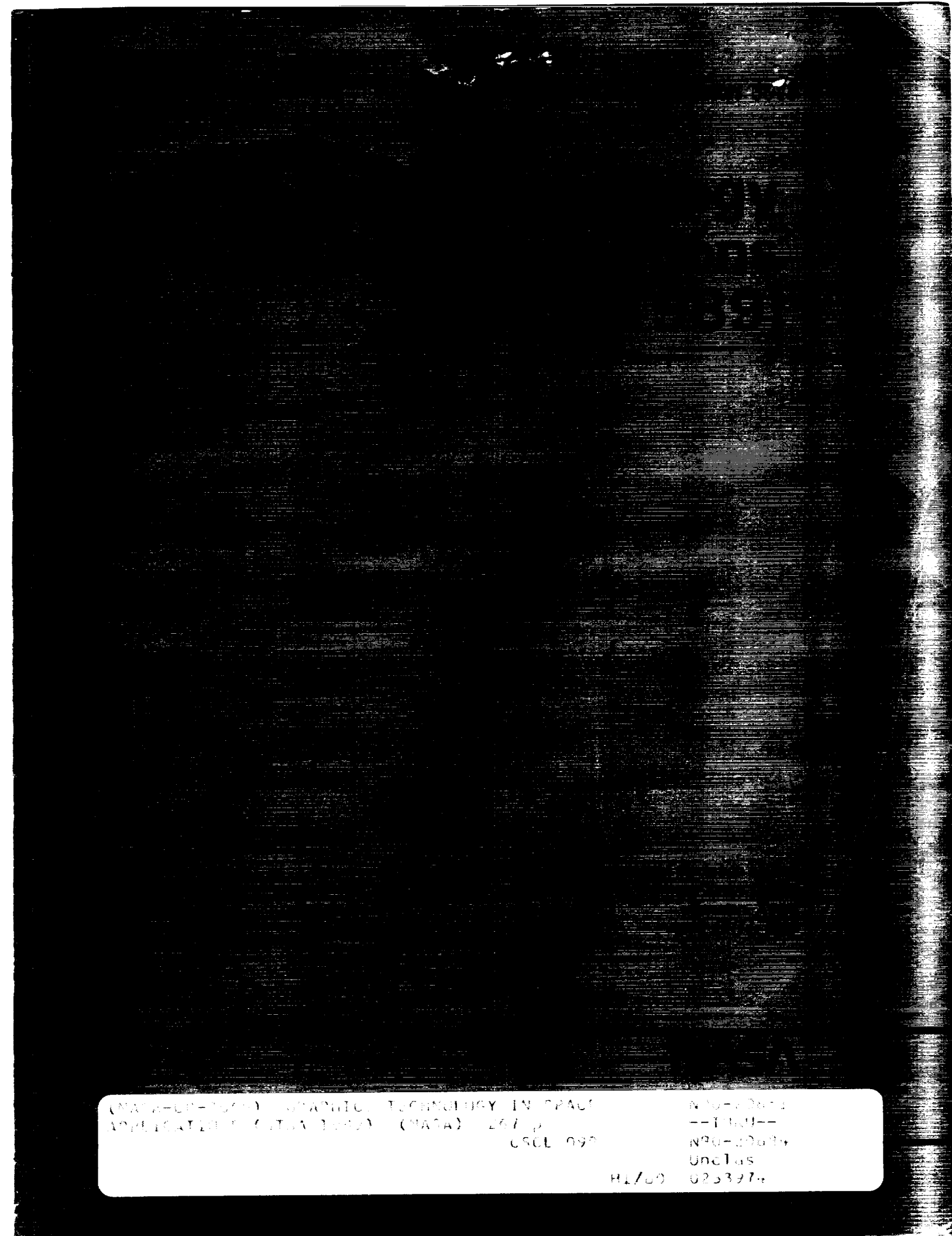
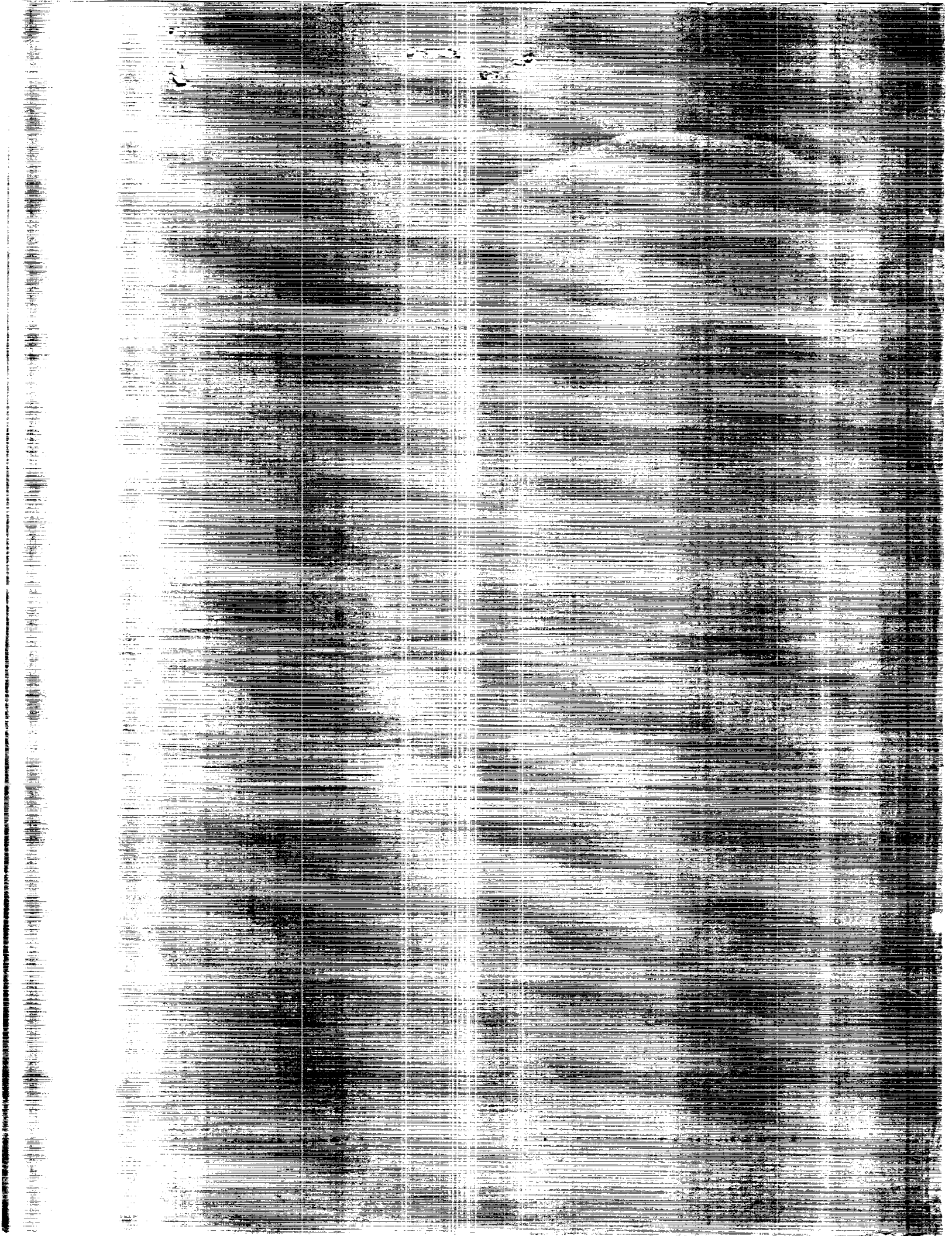


(NASA-CR-[illegible]) GRAPHITE TECHNOLOGY IN SPACE APPLICATIONS (TRW Inc.) (NASA) 26[illegible] p

CSCL 09D

N[illegible]-[illegible]
--Title--
N[illegible]-[illegible]
Unclas
H1/89 02[illegible]



NASA Conference Publication 3045

Graphics Technology in Space Applications (GTSA 1989)

Sandy Griffin, *Editor*
Lyndon B. Johnson Space Center
Houston, Texas

Proceedings of the first annual workshop sponsored by the National Aeronautics and Space Administration, Washington, D.C., and hosted by the University of Houston–Clear Lake, Houston, Texas, and held at NASA Lyndon B. Johnson Space Center
Houston, Texas
April 12–14, 1989

NASA
National Aeronautics and Space Administration
Office of Management
Scientific and Technical Information Division
1989

PREFACE

This document represents the proceedings of the Graphics Technology in Space Applications, which was held at NASA Lyndon B. Johnson Space Center on April 12 - 14, 1989 in Houston, Texas.

The papers included in these proceedings were published in general as received from the authors with minimum modification and editing. Information contained in the individual papers is not to be construed as being officially endorsed by NASA.

MESSAGE FROM THE GENERAL CHAIR

Purpose of Graphics Technology in Space Applications (GTSA)

This conference was created to facilitate the communication between industry and government in graphics technology in space applications. It is intended to provide a forum for information exchange by graphics researchers and practitioners for discussion of common problems, and for the basic education of non-practitioners relative to the potential of this technology.

Robert H. Brown

ACKNOWLEDGEMENT

Acknowledgements are due to all personnel who provided the logistic support necessary to the success of this workshop. Thanks are also due to CAE-Link Flight Simulation, the MITRE Corporation, Barrios Technology, McDonnell-Douglas Space Systems Corporation, and Omniplan Corporation in extending the support of their personnel.

GTSA ORGANIZING COMMITTEE

GRAPHICS TECHNOLOGY IN SPACE APPLICATIONS CONFERENCE

General Chair:
Robert H. Brown, Chief/Technology
Development and Applications Branch/Mission
Support Directorate NASA/Johnson Space Center

Technical Program Chair:
Frank S. Taylor IV, Manager
NASA Interactive Graphics Lab
McDonnell Douglas Space Systems Company

Executive Chair:
Sandra G. Griffin
NASA/Johnson Space Center

Administrative Co-Chairs:
Carla J. Armstrong
Barrios Technology

Glenn B. Freedman, Director
SEPEC
University of Houston-Clear Lake

Members:
Carla J. Armstrong, Barrios, Facilities
Wayne Boatman, Ford Aerospace Corp., Exhibits
Katherine Moser, UH-CL, Registration
Glenn B. Freeman, UH-CL, Finance
Eric A. Lloyd, UH-CL, Publication/Publicity
Glen Van Zandt, NASA, Registration/Publicity
Bruce Wood, NASA/JSC, Facilities

CONTENTS

N90-20652

TAE PLUS: Transportable Applications Environment Plus

Tools for Building Graphic-oriented Applications

Martha R. Szczur
NASA/Goddard Space Flight Center
Greenbelt, Maryland 20771
[MSZCZUR/GSFCMAIL] TELEMAIL/USA
Marti@DSTL86.span.nasa.gov
(301) 286-8609 FTS 888-8609

INTRODUCTION

The Transportable Applications Environment Plus (TAE Plus™), developed by NASA's Goddard Space Flight Center, is a portable User Interface Management System (UIMS), which provides (1) an intuitive WYSIWYG WorkBench for prototyping and designing an application's user interface, integrated with (2) tools for efficiently implementing the designed user interface and (3) effective management of the user interface during an application's active domain. During the development of TAE Plus, many design and implementation decisions were based on the state-of-the-art within graphics workstations, windowing system and object-oriented programming languages, and this paper shares some of the problems and issues experienced during implementation. The paper concludes with open issues and a description of the next development steps planned for TAE Plus.

TAE PLUS AS A UIMS

Before presenting TAE Plus as a UIMS it is first necessary to define what a UIMS is. The definition by Betts et al [1] which is defined in terms of activities and purposes best describes the objectives of TAE Plus:

"A User Interface Management System (UIMS) is a tool (or tool set) designed to encourage interdisciplinary cooperation in the rapid development, tailoring and management (control) of the interaction in an application domain across varying devices, interaction techniques and user interface styles. A UIMS tailors and manages (controls) user interaction in an application domain to allow for rapid and consistent development. A UIMS can be viewed as a tool for increasing programmer productivity."

TAE Plus is a tool for designing, building and tailoring an application's user interface (UI) and for controlling the designed UI throughout the application's execution. The main component of TAE Plus is a WYSIWYG user interface designers' "WorkBench" that allows an application developer to interactively construct the look and feel of an application screen by arranging and manipulating "interaction objects" (e.g., radio buttons, menus, icons, stretchers, rotators, etc.).

Once the application's screen has been designed, the WorkBench saves the user interface details in a resource file. TAE Plus includes runtime services, Window Programming Tools (WPTs), which are used by application programs to display and control the user interfaces designed with the WorkBench. Since the WPTs access the resource file during execution, the user interface details remain independent from the application code, allowing changes to be easily made to the look and feel of an application without recompiling or relinking the software. To change the user interface, the designer returns to the WorkBench, dynamically makes the modifications, and the resource files are automatically updated. The next time the application is run, the modifications will be in effect. Figure 1 illustrates the TAE Plus structure.

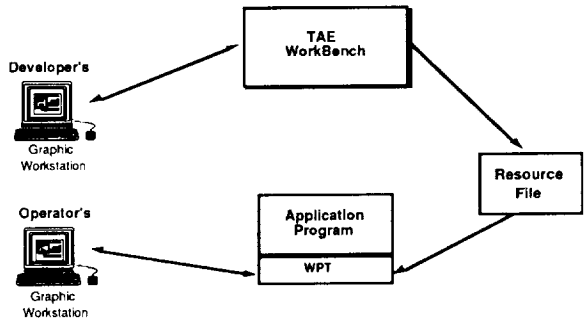


Figure 1. TAE Plus Plus Structure

In addition to providing the WPT runtime subroutines, TAE Plus also offers control of interaction objects from the interpreted TAE Command Language (TCL). This capability provides an extremely powerful means to quickly prototype an application's use of TAE Plus interaction objects and add programming logic without the requirement to compile or link.

INTERACTION OBJECTS AS BUILDING BLOCKS

The basic building blocks for developing an application's user interface are a set of interaction objects. All visually distinct elements of a display that are created and managed using TAE Plus are considered to be interaction objects. Within TAE Plus, interaction objects fall into three categories: user-entry objects, information objects and data-driven objects. User-entry objects are mechanisms by which an application can acquire information and directives from the end use, and include radio buttons, text entry fields, scrolling text lists, pulldown menus, and push buttons. Information objects are used by an application to instruct or notify the user, such as contextual online help information displayed in a scrollable static text object or brief status/error messages displayed in a bother box. Data-driven objects are vector-drawn graphic objects which have been "connected" to an application data variable, and elements of their view change as the data values change. Examples are dials, thermometers, and strip charts, The real-time data-driven objects are the most recent addition to the TAE Plus interaction object collection and currently, the types supported include rotators, stretchers, discretes, text and realtime graphs. Figure 2 illustrates the current set of TAE Plus interaction objects (which are referred to as *items* in the WorkBench). For advanced screen designs, these items can be grouped or composed into larger interaction objects, called *panels* by the WorkBench.

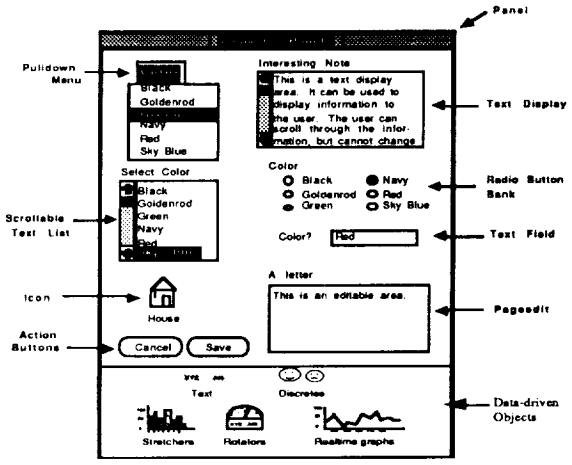


Figure 2. Current set of TAE Plus interaction objects

The use of interaction objects offers the application designer/programmer a number of benefits with the expected payoff of an increase in programmer productivity. [2]

• The interaction objects work together both visually and behaviorally to provide a consistent look and feel for the application's user interface. This consistency translates into reduced end-user training time, more attractive (from a graphic design point of view) screens, and an application which is easier to use.

• Interaction objects provide a common framework for diverse sets of application programs, and serve as a base set of well-documented standards for user interfaces in systems composed of many separate application programs.

• The set of user entry interaction objects covers most common data entry and manipulation needs, allowing the application programmer to spend more time on the content of the application program. The data driven interaction objects provide a standard means of displaying realtime data graphically. The object architecture also enables quick development and addition of new interaction objects into the TAE Plus object library.

• The interaction objects have been thoroughly tested and debugged, allowing the programmer to spend more time testing the application, and less time verifying that the user interface behaves correctly. This is particularly important considering the complexity of some of the objects, and the programming effort it would take to code them scratch.

WORKBENCH SCENARIO

The WorkBench provides an intuitive environment for defining, testing, and communicating the look and feel of an application system. As a designer tool, it provides the following key features:

- Customization and direct manipulation of user interaction objects
- Application code generator
- Capability to dynamically define "connections" between interaction objects
- Rehearsal capability to "try out" sequencing of the user interface design
- Icon editor and support for raster objects
- Undo capability
- Help icon/button on-line support
- Capability to dynamically draw and define data-driven graphic objects

Let's walk through a simple design scenario to get a feel for how the WorkBench operates. The application is a hardware monitoring task for a satellite data handling facility and the designer is going to layout the user interaction in which the

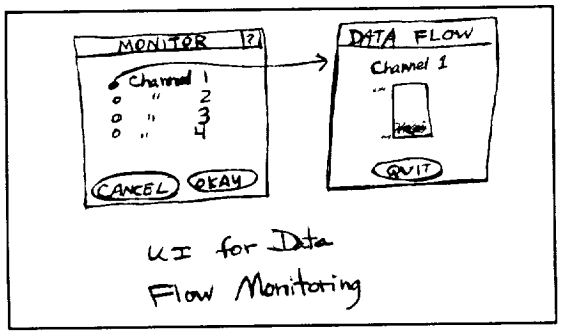


Figure 3. Hand-drawn sketch of application's user interface to be created with WorkBench

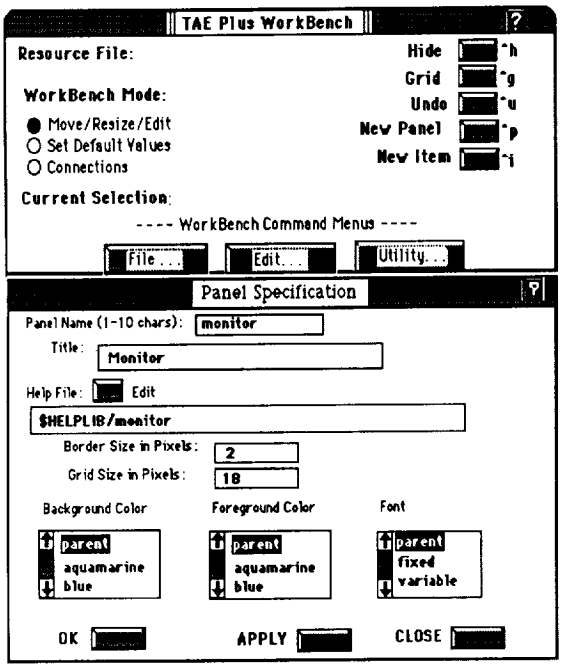


Figure 4. WorkBench's Main Menu and Panel Specification Panel

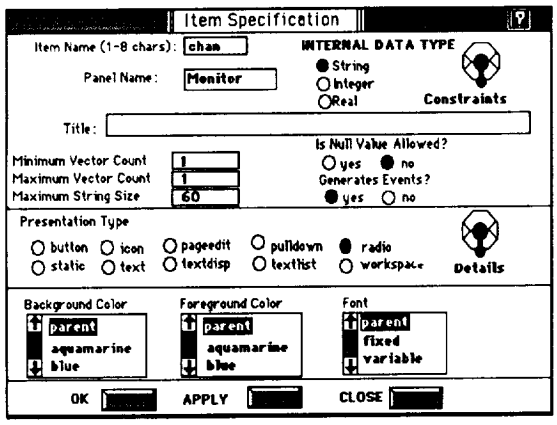


Figure 5. Defining the interaction objects to reside in a panel.

operator is prompted for a hardware channel number. Once the operator selects a channel, a new panel appears with a realtime sliding bar object displaying the amount of data flowing through the channel. Figure 3 shows a rough sketch of the two panels which are to be designed with the WorkBench in this scenario.

Functionally, the WorkBench allows an application designer to dynamically lay out an application screen, defining its static and dynamic areas. The tool provides the designer with a choice of predesigned interaction objects and allows for tailoring, combining and rearranging of the objects. To begin the session, the designer needs to create the base window into which interaction objects will be specified. He/she selects **New Panel** from the main WorkBench menu, which displays the panel specification panel (Refer to Figure 4) where the designer specifies presentation information, such as the title, scroll option, font, color, and optional on-line help for the panel *Monitor*.

The designer is now ready to define the interaction items to reside in the panel. He/she selects **New Item** from the WorkBench main menu and is presented with the item specification window. The designer defines both the presentation information and the context information. The item specification window has an associated *Constraints* (i.e., context) window within which the labels for each entry of a radio button bank object are specified (refer to Figure 5). For the scenario we are following here, the designer has created a radio button bank for the channel numbers, a *cancel* and *okay* button and a panel help icon. For icon support, the WorkBench has an Icon Editor, within which an icon can be drawn, edited and saved.

The designer also has the option of retrieving a "palette" of items (by selecting **File....Include** from the WorkBench menu). From this collection of previously created items, the designer can select and copy appropriate objects. The ability to reuse items saves programming time, facilitates trying out different combinations of items in the prototyping process, and contributes to standardization of the application's "look and feel". If an application system manager wanted to ensure consistency and uniformity across an entire application's UI, all developers could be informed to use only items from the application's palette of common items.

The designer goes through the same process to build the realtime display panel, *DataFlow*. This simple panel is made up of a data-driven stretcher item, selected from a pre-defined pallete of "output objects", and a *quit* button. The WorkBench provides a drawing tool [5] within which the static background and dynamic foreground of a data-driven object can be drawn, edited and saved. Once the object is created, the designer identifies presentation attributes for the object (i.e. the color

thresholds, maximum/minimum, delta).

Most often an application's UI will be made up of a number of related panels, sequenced in a meaningful fashion. Through the WorkBench, the designer defines the interface "connections". These links determine what happens when the user selects a button or a menu entry. The designer attaches "events" to interaction items and thereby designates what panel appears and what program executes when an event is triggered. Events are triggered by user-controlled I/O peripherals (e.g., point and click devices or keyboard input). In Figure 6, the designer has specified links causing the *Dataflow* panel to appear when the end user selects the option marked Channel 1 and the process *Flowcompute* to be executed. In turn, *Flowcompute* is the application process containing the data variable that drives the variations in appearance of the item *BarSlide* .

TAE Plus also offers an optional help feature which provides a consistent mechanism for supplying application specific information about a panel and any interaction items within the panel. In a typical session, the designer elects to edit a help file after all the panel items have been designed. Clicking on the edit help option brings up a text editor window in which the appropriate information can be entered. The designer can then define any button item or icon item to be " the" help item for the panel (in the scenario we are following, it would be the Help icon in the panel *Monitor*). During the application operation, when the end-user clicks on the question mark item, the cursor changes to a "?". The end-user then clicks on the panel itself or any item in the panel to bring up a help panel containing the associated help text.

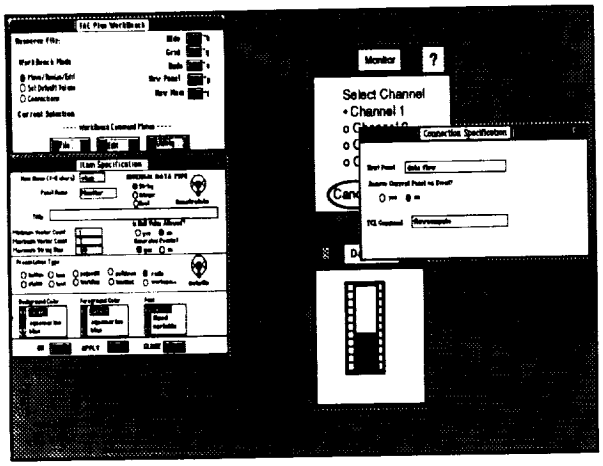


Figure 6.
Using the WorkBench to define "connections".

Having designed the layout of panels and their attendant items and having threaded the panel and items according to their interaction scenario, the designer must then be able to preview (i.e., to rehearse) the interface's operation. With this potential to "test drive" an interface, to make changes, and to dry-run again, iterative design becomes part of the prototyping process. When the designer selects the rehearse option (by selecting **Utility....Rehearse** from the WorkBench Menu), the screen is cleared and the WorkBench goes through the entire sequence as if the application were executing. With the rehearsal feature, the designer can evaluate and refine both the functionality and the aesthetics of a proposed interface. After the rehearsal, control is returned to wherever the designer left off in the WorkBench and he/she can either continue with the design process or save the defined UI in a resource file (by selecting **File....Save** from the WorkBench Menu).

Developing software with sophisticated user interfaces is a complex process, mandating the support of varied talents, including human factors experts and application program specialists. Once the UI designer (who may have limited experience with actual code development) has finished the UI, he/she can turn the saved UI resource file over to an experienced programmer. As a further aid to the application programmer, the WorkBench's "generate" feature (**Utility....Generate**) produces a fully annotated and operational body of code which will display and manage the entire WorkBench designed UI. Currently, source code generation of C, Ada and TCL are supported, with bindings for Fortran and C++ expected in later TAE Plus releases. The programmer can now add additional code to this template and make a fully functional application. Providing these code "stubs" helps in establishing uniform programming method and style across large applications or a family of interrelated software applications.

WINDOW PROGRAMMING TOOLS (WPTs)

The Window Programming Tools (WPTs) are a package of application program callable subroutines used to control an application's user interface. Using these routines, applications can define, display, receive information from, update and/or delete TAE Plus panels and interaction objects (refer to Figure 7 for a current list of WPT). WPTs support a modeless user interface, meaning a user can interact with one of a number of interaction objects within any one of a number of displayed panels. In contrast to sequential mode-oriented programming, modeless programming accepts, at any instance, a number of user inputs, o*revents*. Because these multiple events must be handled by the application program, event-driven programming can be more complex than traditional programming. TheWorkBench's auto-generation of the WPT event loop reduces the risk of programmer error within the UI portion of an applications' implementation.

Wpt_BeginWait	Display Busy Indicator
Wpt_CloseItems	Close Items on a Panel
Wpt_ConvertName	Get the X Window Id of a Names Window
Wpt_EndWait	Stop Displaying Busy Indicator
Wpt_Init	Initialize the Window System
Wpt_ItemWindow	Get WindowId of Window for an Item
Wpt_MissingVal	Determine if Missing Parameter Values
Wpt_NewPanel	Display an Interaction Panel
Wpt_NextEvent	Get Next Panel-Related Event from WPT
Wpt_PanelErase	Erase a panel from the screen
Wpt_PanelReset	Reset Panel to Initial Values
Wpt_PanelMessage	Display a Message for a WPT Panel
Wpt_PanelWindow	Get an X Window Id
Wpt_PanelXrId	Get the Xr Defined Panel Handle of a WPT Panel
Wpt_ParmReject	Reject the Current Value of a Parameter
Wpt_ParmUpdate	Update a Parameter on a Displayed Panel
Wpt_ViewUpdate	Update the view of a Parameter on a Displayed Panel

Figure 7. The Window Programming Tools (WPTs)

The WPTs utilize the X Window System™ [10] as its base windowing system. One of the strengths of X is the concept of providing a low-level abstraction of windowing support (Xlib), which becomes the base standard, and a high-level abstraction (X toolkits), which has a set of interaction objects (called "widgets" in the X world) that define elements of a UI's look and feel. The current version of TAE Plus (V3.2) is implemented with the latest release of X (X11.3) using the Xray toolkit, which was distributed with earlier versions of X. We are rewriting our WPTs to utilize the X Toolkit, which is becoming a de facto toolkit standard. The initial approach is to base our default set of interaction objects on the HP widget set delivered with the generic M.I.T. delivery of X (and which is in the public domain) while supporting an open architecture that allows adding to the widget set. A "cookbook" explaining the steps to be taken to replace/add widgets and update the WorkBench is in progress. This will enable TAE Plus to be used for designing and managing the user interface that adheres to whatever UI style is defined by an application group to be their preferred widget set.

The WPTs also provide a buffer between the application program and the X Window System services. For instance, to display a WorkBench-designed panel, an application makes a single call to Wpt_NextPanel. This single call translates into a function that consists of about 2800 lines of C code and makes about 50 calls to X Window System routines. For the majority of applications, the WPT services and objects supported by the WorkBench provide the necessary user interface tools and save the programmer from having to learn the complexities of programming directly with X. This can be a significant advantage, especially when considering that the full set of 17 Wpt routines consist of 5800 lines of C code and make a total of between 300-400 X calls.

PROTOTYPING IN TAE COMMAND LANGUAGE (TCL)

To provide an easy method for displaying and manipulating the newly designed user interface, we created a simple set of commands ("WPT" commands) within the TAE Command Language (TCL).

TCL offers a high-level set of commands used to invoke and manage application functions. Commands can be invoked dynamically during an interactive session or used to build command procedures.An advantage TAE Plus has over some other UIMS is that it does not just support the user interface component of an application, but has a full set of integrated tools to fully support an application, either a prototype or an operational version. These services include parameter manipulation, message logging, logon/logoff procedure, data file I/O, operating system services, scripting capability, session logging, procedure building capability, on-line help, and user-site tailoring of TAE Plus commands. Because user interface tools are integrated with general purpose application management services, the application need not be tightly tied to a particular operating system or computer.

Since TCL is an interpreted language, the commands can be used to prototype an application without having to recompile or relink every time a change is made. Just as with WPT routines used by application programs, the WPT commands can be used to directly define panels and items, or they can be used to access WorkBench-generated resource files that contain pre-defined panels and items. While the intended use of these commands is for prototyping, if the overhead performance of executing TCL commands is acceptable, then command procedures using WPT commands would be appropriate for operational systems.

TAE PLUS ARCHITECTURE

The TAE Plus architecture is based on a total separation of the user interaction management from the application-specific software. The current implementation is a result of having gone through several prototyped versions of a WorkBench and graphic support development during the 1986-87 period, as well as building on an exisiting application management system, the original TAE (affectionately referred to as "TAE Classic"). [9] TAE Classic architecture, which was designed in 1980, was based on a total separation of the user interaction in a much stricter sense than the TAE Plus implementation. All user dialogue was directed through a *terminal monitor*, including dialogues initiated from within an application. This central control of the UI easily facilitated the goal of providing a consistent look and feel across an applica-

tion, but was limited to an ASCII terminal.

The advent of the graphic workstation inspires more elaborate user interfaces and a closer interrelationship between the application program and the UI. The TAE architecture was enhanced to allow for an application to directly control the user interactions, while still maintaining presentation independence (i.e., an application doesn't need to know any of the details as to how a request for data is actually being presented to the user, only what the data is). Figure 8 illustrates how the TAE Plus structure maintains UI/application independence while providing run-time services to control and manipulate the user interactions from within an application.

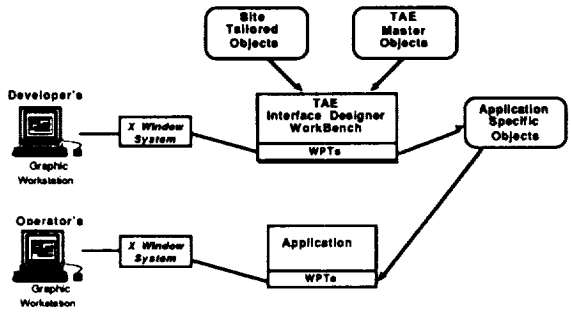


Figure 8.
TAE Plus architecture maintains separation of UI and application elements

SELECTION OF AN IMPLEMENTATION LANGUAGE

TAE Classic is implemented in the C programming language, which has proven to be an efficient and standard language across different hardware platforms, thus allowing for the porting of TAE source code with reasonable ease. However, we felt a "true" object-oriented language would provide us with the optimum environment for implementing the TAE Plus graphical user interface capabilities. (See Chapter 9 of Cox [3] for a discussion on the suitability of object-oriented languages for graphical user interfaces.)

In early 1987, before committing to an object-oriented language and as a means of demonstrating the utility of the X Window System in our UIMS concept, we built a rapid prototype of TAE Plus, using Smalltalk™ to implement the WorkBench. This proved to be a beneficial learning experience. The prototype demonstrated that object-oriented programming is a productive and effective method for building user interfaces. Although Smalltalk enabled us to generate a prototype in a timely manner, several concerns did surface during the implementation. For instance, at the time of the prototyping effort, Smalltalk was not based on the X Window System, which meant the WorkBench and the WPTs had different implementations of the interaction object functions. Another concern with the Smalltalk implementation was that the designer had to have some understanding of Smalltalk's interface conventions -- not a desirable feature since the user interface for applications operating in the TAE Plus environment would have a different set of conventions imposed by an X-based Window Manager. The issue of distribution of TAE Plus with a Smalltalk application was also a problem. With TAE, distribution only involves acquiring a license from COSMIC™ (NASA's distribution center), but for a site to run the WorkBench, they would also need a Smalltalk license. The limited use of Smalltalk in our user community made this undesirable. For these and other reasons [15] we looked at other languages for the operational implementation of the WorkBench .

Though the X Window System is written in C, we did not want to constrain ourselves to a procedure-based language, especially in light of the power of C++ and Objective C, and the fact that interfaces from these object-based languages exist to the X runtime library. For the past several years, we have closely followed the C++ versus Objective C debate. The Objective C argument is strong -- the language is a marriage of two powerful languages (Smalltalk and C), and provides much of the Smalltalk elegance without severe performance penalties. We selected C++, however, for several reasons [15]. For one, C++ seems to be a "cleaner" language (i.e., it is a conceptually strong expansion of C) and is becoming increasing popular within the object -oriented programming community. Another strong argument for using C++ is the growing availability of existing public domain X-based object class libraries. Utilizing an existing object library is not only a cost saver, but also serves as a learning tool, both for object-oriented programming and for C++. Delivered with the X Window System is the *InterViews* C++ class library and a drawing utility, *idraw*, both of which were developed at Stanford University. [4,5] The *InterViews* C++ class library has many attractive features. The class structure has gone through several major iterations and the current design is clean. The *idraw* utility is a sophisticated direct manipulation C++ application, which allows the WorkBench to create, edit and save the graphical data-driven interaction objects.

Many of the current implementations of C++ compilers are pre-processors generating standard C code, thus enabling the operational TAE Plus code to be delivered in C code and allowing for ease in porting. With this option and by utilizing sophisticated public domain software packages (X Window System, InterViews, and idraw) we avoid requiring our user community to purchase any additional software licenses or compilers.

Because of NASA's commitment to use Ada™ for all Space Station software development, the question arises "why not Ada"? We do not consider Ada a purely object-oriented language. [3,11,12,17] As mentioned earlier, we felt that the TAE Plus development would be better served by a "pure" object-oriented language -- one that supports data encapsulation, inheritance and polymorphism. These are the features associated with the type of object-oriented programming supported by Smalltalk and C++. Since TAE Plus software services can be accessed by Ada applications, we feel that implementing the TAE Plus environment in a pure object-oriented language is the most effective approach at this particular time.

PORTABILITY and MAINTAINABILITY

TAE is designed to be portable. At present, TAE Classic is successfully operating on 14 Unix-based computers, VAX/VMS and the IBM/VM environment. TAE Plus base development is being done on a Sun workstation under Unix. As of February 1989, it is also operational under Unix on the Apollo, VaxStation II (Ultrix), HP9000, Masscomp and the Macintosh II (A/UX). Ports are in progress for the IBM RT and IBM PS/2 under AIX and the VAX under VMS. TAE Classic has over 230 installations, of which 64 are NASA. The current beta version of TAE Plus is located at over 100 world-wide beta sites, including at least 30 NASA installations.

Every system is maintainable; *how easy it is to maintain* is the issue. When a UIMS is used as a tool to build and support an application's user interface, there is a legitimate concern about the application's dependency on a "black box". (Since an application program's UI control is isolated in the UIMS, it is frequently perceived by application programmers as a "black box".[6]) The UIMS architecture assure developers that corrections and upgrades to itself will have a minimal impact within the application domain. We knew when we began that TAE Plus was targeted for wide application utilization and for different machines, so ease of maintenance has always been important. By providing the application callable WPTs and WPT function commands, applications are isolated from the windowing system, and thus, if in a few years a newer, faster, fancier windowing standard shows up, only the WPTs require updating or rewriting; the application code is not affected. In effect, this is what we're doing with the rewrite of the WPTs to use the X11 Xtoolkit intrinsics. All applications, as well as the WorkBench, will get enhanced capability and performance without making any changes to themselves.

User support is another facet of maintainability. Since the first release of TAE Classic in 1981, we have provided user support through a fully staffed Support Office. This service has been one of the primary reasons for the success of TAE. Through the Support Office, users receive answers to technical questions, report problems, and make suggestions for improvements. In turn, the Support Office keeps users up-to-date on new releases, provides training sessions, and sponsors user workshops and conferences. This exchange of information enables the Project Office to keep the TAE software and documentation "in working order" and, perhaps most importantly, take advantage of user feedback to help direct our future development.

NEXT STEPS

The current TAE Plus provides a powerful and much needed tool for the continuum of software engineering -- from the initial design phases of a highly interactive prototype to the fully operational application package. However, there is still a long list of enhancements and new capabilities that we will be adding to TAE Plus in future releases. Features included on the "Wanted List" are extensions to the interaction objects, particularly in the data-driven object category; integration with the Open Software Foundation's (OSF) User Environment Component (UEC); direct manipulation support for application programs; ports to new workstation platforms; on-line tutorial and training tools; introduction of hypermedia technology; integration of expert system technology to aid in making user interface design decision; and implementation of additional user interface designer tools, such as a WYSIWYG graph builder.

CONCLUSION

Building large scale interactive systems has been a regular activity at NASA/Goddard Space Flight Center (GSFC) since the transition from card readers to interactive terminals. Although the applications vary from on-board flight instrument command and control to scientific data analysis, they have all required software to support the communication between the human user and the application tasks. In the early 1980's, GSFC sought to capitalize on common requirements in human-computer interaction by building TAE Plus Classic, a powerful tool for quickly and easily building consistent, portable user interfaces in an interactive alphanumeric terminal environment. With the emergence of sophisticated graphic workstations and the subsequent demands for highly interactive systems, the user interface becomes more complex and includes multiple window displays, the use of color, graphical objects and icons, and various selection techniques. Traditional UI paradigms give us only improvished models and

guidelines; they are inadequate for what can be accomplished with the new technology. Prototyping of different user interface designs, thus, becomes an increasingly important method for stabilizing concepts and requirements for an application. At GSFC, we had the requirement to provide a tool for prototyping a visual representation of a user interface, as well as establish an integrated development environment that allows prototyped user interfaces to evolve into operational applications. We feel TAE Plus is fulfilling this role by providing a usable, generalized, portable and maintainable package of development tools. TAE Plus is an evolving system and its development will continue to be guided by user-defined requirements. To date, each phase of TAE Plus's evolution has taken into account advances in virtual operating systems, human factors research, command language design, standardization efforts and software portability. With TAE Plus's flexibility and functionality, we believe it can contribute to both more advances and more standardization in user interface management system technology.

ACKNOWLEDGEMENTS

TAE Plus is a NASA software product being developed by the NASA/Goddard Space Flight Center and by Century Computing, Inc. The work is sponsored by the NASA Office of Space Science and Applications and the Office of Space Operations. Special thanks to Dr. Patricia Carlson for her quality editing and to the TAE Plus Support Office staff for their tireless service to the TAE Project.

TAE is a registered trademark of National Aeronautics and Space Administration (NASA). It is distributed through NASA's distribution center, COSMIC. For further information, contact the TAE Support Office at GSFC, (301) 286-6034.

REFERENCES

1. Betts, B., Burlingame, D., Fischer, G., Foley, J., Green, M., Kasik, D., Kerr, S., Olsen, D., Thomas, J., "Goals and Objectives for User Interface Software", COMPUTER GRAPHICS, 21:2, 1987.

2 .Bleser, Teresa, "TAE Plus Style Guide", NASA Contractor Document, February 1989.

3. Cox, Brad J., OBJECT ORIENTED PROGRAMMING, AN EVOLUTIONARY APPROACH, Addison-Wesley Publishing Company, Reading, Massachusetts, 1986.

4. Linton, Mark, Calder, Paul R., "The Design and Implementation of InterViews", Proceedings of the C++ Workshop, USENIX, November, 1987, pp.256-273.

5. Linton, Mark A., Vlissides, John M., Calder, Paul R., "Composing User Interfaces with Interviews", IEEE COMPUTER, February, 1989

6. Lowgren, Jonas, "History, State and Future of User Interface Management Systems", SIGCHI BULLETIN, 20:2, 1988.

7. Myers, B., "Gaining General Acceptance for UIMS", COMPUTER GRAPHICS 21:2, 1987.

8. Olsen, D., " Larger Issues in User Interface Man agement",COMPUTER GRAPHICS 21:2, 1987.

9. Perkins, D.C., Howell, D.R., Szczur, M.R., "The Transportable Applications Executive -- an interactive design-to-production development system", DIGITAL IMAGE PROCESSING IN REMOTE SENSING, edited by J-P Muller, Taylor & Francis Publishers, London, 1988.

10. Scheifler, Robert W., Gettys, Jim., "The X Window System", MIT Laboratory for Computer Science, Cambridge, MA., October 1986

11. Schmucker, Kurt J., OBJECT-ORIENTED PROGRAMMING FOR THE MACINTOSH, Hayden Book Company, Hasbrouck Heights, New Jersey 1986.

12. Seidewitz, Ed.,"Object-Oriented Programming in Smalltalk and Ada", Proceedings of the Object-Oriented Programming Systems, Languages and Applications (OOPSLA) Conference, October, 1987.

13. Space Station Program Office, "Space Station Information System User Support Environment Functional Requirements", Final Draft, JSC 30497, April, 1987

14. Stroustrup, Bjarne, THE C++ PROGRAMMING LANGUAGE, Addison-Wesley Publishing Company, Reading, Massachusetts, 1987.

15. Szczur, Martha R., Miller, Philip, "Transportable Applications Environment (TAE) Plus: Experiences in "Object"ively Modernizing a User Interface Environment", Proceedings of the Object-Oriented Programming Systems, Languages and Applications (OOPSLA) Conference, September 1988

16. TAE Plus V3.2 Documentation Set, Century Computing, Inc., NASA Contractor Documentation, January 1989

17. Wegner, Peter, "Dimensions of Object-Based Language Design", Proceedings of the Object-Oriented Programming Systems, Languages and Applications (OOPSLA) Conference, October, 1987.

5-60

RenderMan Design Principles

N90-20653

Tony Apodaca
Pixar
3240 Kerner Blvd.
San Rafael, CA 94901

Tom Porter
Pixar
3240 Kerner Blvd.
San Rafael, CA 94901

ABSTRACT

The two worlds of interactive graphics and realistic graphics have remained separate. Fast graphics hardware runs simple algorithms and generates simple-looking images. Photorealistic image synthesis software runs slowly on large expensive computers. The time has come for these two branches of computer graphics to merge.

The speed and expense of graphics hardware is no longer the barrier to the wide acceptance of photorealism. There is every reason to believe that high quality image synthesis will become a standard capability of every graphics machine, from superworkstation to personal computer. The significant barrier has been the lack of a common language, an agreed-upon set of terms and conditions, for 3-D modeling systems to talk to 3-D rendering systems for computing an accurate rendition of that scene.

Pixar has introduced RenderMan *to serve as that common language. This paper examines* RenderMan, *specifically the extensibility it offers in shading calculations.*

NASA has been at the forefront of developments in computer graphics. One area in particular has been the quest for realism in synthetic image generation. Voyager animations done at JPL a decade ago captivated many with the notion that the process of scientific discovery and the popular understanding of that process could both benefit from visually accurate computer generated imagery.

Computers have sped up since those animations were made. Tools for modeling and controlling the animation have also improved. Yet too often, the ability to produce complex and accurate renditions is relegated to specialized labs. The challenge that we face is in bringing this technology to the desktop, running it on every graphics platform, linked across the standard networks, fed from the common databases.

The goal is to unify the often divergent methodologies used in the computer-aided-design of a 3-D object, the analysis of that object during simulation, and the accurate representation of the object.

What is RenderMan?

RenderMan is an interface between 3-D modeling systems and photorealistic rendering systems. Modeling is the process of describing objects to a computer. We use modeling here to refer to all aspects of describing a scene, including its dynamics. Rendering is the process of generating an image of the scene from a given viewpoint. RenderMan is an interface proposal which will permit a large variety of geometric modelers to talk to a large variety of renderers with a straightforward, common format.

The central problem in making such a proposal is to accommodate the needs of advanced rendering in a clean way, while allowing standard CAD databases to feed the interface. Only then can photorealistic image synthesis be brought under the same wing, integrated into the same computing environment as other aspects of CAD and simulation.

Shape and Shading

An overriding principle in the design of RenderMan used to solve this problem is a recognition that an interface proposal must distinguish clearly between shape and shading, between the geometry of the scene and the visual characteristics of the geometry. The visual complexity of real world imagery is not found in the general shape of objects, but rather in the textures and materials and lighting and dynamics. In fact, the graphics community already *has* sufficient CAD tools to specify the shapes of things. We lack the tools to describe visual qualities, such as atmospheric conditions, reflectivity of materials, and characteristics of light sources.

A second principle is that shading computations need to be far more general than the Gouraud and Phong interpolation set forth in the textbooks. The world is not all plastic. We need rendering systems that can wrap an atmospheric texture around a spherical planet, that can compute a noise function to simulate the bumpiness of a surface, that can handle surface properties other than color, perhaps to compute renditions outside the visible spectrum.

The RenderMan interface is a specification for approximately 100 subroutines with which a modeler can completely describe all of the information that a renderer might need to generate an image of a scene. It provides entry points for geometric information, transformation hierarchies, color and material property information, camera parameters and output image characteristics.

The RenderMan interface supports a rich variety of geometric primitives. For example, convex polygons, concave polygons (with and without holes), polyhedral models, and a large number of quadric surfaces are supported. RenderMan includes a very comprehensive bicubic patch primitive, specified with an arbitrary basis matrix. RenderMan also supports non-uniform rational B-spline surfaces (NURBS).

Support for primitives such as these guarantees that most standard CAD packages can feed the RenderMan interface quite easily. There are two significant capabilities of the interface in extending the common notions about geometry:

First, RenderMan supports *procedural primitives*. One of the biggest problems in modeling natural phenomena (such as mountains, plants, fire, etc.) is that the geometric complexity is enormous. This problem is usually solved by writing programs which generate all of the tiny detail, rather than model it by hand. However, it can still be very expensive for the modeler to generate a huge complex model and then pass it to the renderer, particularly if the modeler doesn't know how much of it the renderer really needs. RenderMan's procedural primitives permit the user to give the renderer a pointer to a subroutine which will expand simple objects into more complicated ones, such as converting a triangle into a fractal mountain or a sphere into a particle system explosion. Using procedural primitives, the modeler can download a very complex model such as a fractal into the renderer in a carefully controlled way, so that only the required amount of detail is sent through the interface.

Secondly, RenderMan has a very general interface for specification of the arbitrary parameters on a surface. This permits the user to specify not simply the position and color, but also the surface normals and texture map coordinates on a per vertex basis. In addition, the vertex structure can actually be extended by the user at run-time, to include arbitrary information of his choosing such as temperature or stress or density or any other values that might be interesting to his particular application. These parameters can then be used to control the shading calculation.

Shading Language

Most software renderers have a subroutine which determines the color of the surface of an object. Typically, it will implement a single mathematical equation which uses a simple model of the reflection of light in order to calculate the contributions of the light sources and texture maps upon the surface color. The equation often has a lot of parameters (5 to 20, depending on the renderer) which the user tweaks to control the appearance of different kinds of materials (plastic, metal, chalk, etc.).

Very often, however, you want the surface to have some characteristic which you can't achieve with the fixed equation, such as the use of a texture map to modify some shading parameter. If you are fortunate enough to have the source code, you can add your function and recompile. If not, you are out of luck.

RenderMan changes this model, by providing the facility of the shading language, a C-like programming language which has new functions and data types that are specifically designed for the purpose of calculating colors based on geometric information. Programs which users write in the shading language are typically small (10 to 20 lines), and are loaded into the renderer at run-time when they are requested by some part of the scene geometry. These programs then replace the built-in shading equations. Users can use this language to customize the shading on a per-object basis. This new freedom gives the user the power to model the appearance of objects as carefully as he models their shape.

The shading language supports three basic data types, the `float`, the `point` and the `color`. `point` and `color` are abstract data types which are actually vectors of floating point values. The standard C arithmetic operators (`*`, `+`, `/`, etc.) work on these data types. In addition, there are some new operators for vector dot and cross product. The familiar C conditional and looping constructs are available (except `switch`), as are subroutine definitions and calls. There is a rich library of mathematical functions, as well as a library of functions which implement common shading operations such as normalizing vectors, transforming points between coordinate systems, calculating diffuse and specular lighting, interpolating colors, splining and calculating pseudorandom numbers.

RenderMan actually permits the user to define up to four separate shading language programs which provide different material characteristic information about each object: a *surface* shader, which determines what color we see when light reflects off the surface; a *displacement* shader, which can move the surface small amounts to add dents or fillets which are too small or too complex to model geometrically; a *light* shader, which describes how luminous objects emit light; and a *volume* shader, which describes how light is attenuated as it passes through the interior of a translucent object. This may seem a bit complicated, but it actually quite a straightforward way to think about the material properties of objects, particularly once you've seen them in action.

Shaders

The renderer calls the appropriate shading language program (*shader*) every time a light intensity, surface color, etc., is required. When a shader is called, it has available to it a large number of global variables which are provided by the renderer. These variables include all of the geometric information that the renderer knows about the surface being shaded, such as the position `P`, the surface normal `N`, the color `Cs` and opacity `Os` that the user specified, the texture coordinates `s,t` and others. The variables that the user applied to the vertices of his primitives are also available inside the shaders. Each type of shader accomplishes its specified task by calculating and modifying a specific part of this global state. For example, a surface shader is responsible for calculating and setting `Ci`, the color that the eye sees. A light shader is responsible for setting `Cl`, the light color.

Listing 1 shows an example of a simple surface shader. This shader calculates the reflectivity of a metallic object, using a simple equation. It makes use of the standard library functions `ambient`, `diffuse` and `specular` to determine the amount of light arriving on the surface from the light sources. These functions implement three customary equations based on the direction and strength of the incoming light. If those functions had not been appropriate, the surface shader has access to the lights and could have calculated whatever values it pleased from them. The shader then calculates a weighted average of the incoming light intensities and multiplies by the color of the object. Notice also that the shading language took care of the multiplication of float values by color vectors automatically, freeing the user from having to write the ugly loops which would have been present in most other languages.

The type of the shader (in this case `surface`) indicates its intended function. Parameters to the shader are specified using a syntax similar to ANSI C. This shader demonstrates another other unique feature of the shading language, the presence of default values in the parameter list. When a modeler requests this shader, it specifies the parameters it wishes to override by name. Any parameter not mentioned is left with the default value.

```
surface metallic (float Ka = .4,
                  Kd = .4, Ks = .6,
                  roughness = .25;)
{
   N = faceforward(normalize(N));
   Ci = Cs * (Ka * ambient() +
              Kd * diffuse(N) +
              Ks * specular(N,
              -normalize(I), roughness) );
}
```

Listing 1. A simple shader which simulates the reflection of light off of metallic objects.

Listing 2 demonstrates a displacement shader. The purpose of a displacement shader is to move the position of the surface around a little bit to simulate tiny fillets, dents and other minor surface perturbations. This greatly adds to the visual interest of an object, and makes it look much more realistic. This particular shader calculates a fractal dentedness using several iterations of *noise*, a function which produces a semirandom value which changes slowly over the surface of the object (using a purely random value would distort the surface beyond recognition, since adjacent points would have no relationship to each other). Getting the same effect by trying to model the intricate surface dents would be extremely difficult.

```
displacement dent (float scale = 1.0;)
{
    float size = 1.0, displace = 0.0;
    for (i=0; i<6; i+=1.0) {
      /* Calculate a simple
      fractal 1/f noise function */
      displace += abs(.5 - noise(P * size))
            / size;
      size *= 2.0;
    }
    /* Displace the surface and
      recalculate surface normals */
    P += N * pow(displace,3.0) * scale;
    N = calculatenormals(P);
}
```

Listing 2. Shader which simulates dents by moving the surface a small amount. This adds visual complexity which is very difficult to model convincingly using standard geometric modeling techniques.

Sensor Simulations

RenderMan can generate output much more general that the simple pinhole camera/RGB images provided by current systems. RenderMan can, for example, compute color in multichannel spectral spaces. Landsat data can be used as input texture maps to control muluple surface parameters mapped onto a planet surface. Shading language procedures can be written to use surface parameters such as temperature; in this way, multichannel sensor image acquisition can be simulated.

RenderMan allows the user to specify other parameters of the simulated camera, in order to provide information to renderers which support advanced rendering features. For example, the user can set the shutter time as well as the focal length, focal distance and f-stop of the camera, to simulate motion blur and depth-of-field. RenderMan allows the user to specify the positions, shapes and colors of the objects at multiple times during the shutter interval, so that sophisticated renderers that can simulate motion blur will know how the objects are moving.

High quality rendering requires a lot of attention to the sampling and filtering which is performed on the output pixels, in order to avoid *aliasing*. RenderMan gives the user independent control over the number of shading samples per pixel and the number of hidden surface samples per pixel, as well as the size and shape of the pixel filter function. In addition to the standard display parameters of output image name and device type and image resolution, RenderMan supports gamma correction and exposure control. These functions compensate for a monitor's phosphors' tendency to glow with exponentially increasing brightness as voltage increases linearly. It also contains the new concept of an *imager* shader, another shading language program which permits the user to implement various color manipulations on final pixels just before they are put into the framebuffer or file.

Conclusion

The RenderMan interface is a powerful interface between 3-D modeling systems and photorealistic rendering systems. It is designed to bring the highest quality in image synthesis into widespread use. Modern CAD modeling tools can feed RenderMan from their standard database of geometry. RenderMan provides simple built-in shading language procedures to provide for a range of standard material properties.

RenderMan provides a shading language for far-reaching extensibility in user specification of specific visual characteristics of the scene. The interface exposes a great deal of control over the shading process; modelers are encouraged to offer user-defined shading language procedures for renderers to execute. By partitioning the modeler/renderer interface in this way, high-quality rendering can be made accessible to a vast array of modeling systems and CAD databases.

RenderMan is the only graphics interface proposal to deal with issues in high-quality synthetic image generation such as antialiasing, texture mapping, motion-blur, shadows, spectral color models and programmable shading languages. These advanced features are not available on most of the rendering software and hardware that is currently available. As such, RenderMan represents a goal for sophisticated new graphics hardware and rendering software to shoot for.

Users of graphics workstations and personal computers will be the biggest winners, as photorealism becomes inexpensive, commonplace and compatible across a wide range of platforms.

Copies of *The* RenderMan *Interface, Version 3.0* are available from Pixar, 3240 Kerner Blvd., San Rafael, CA, 94901. Please enclose $15 to defer the cost of printing and mailing.

Further Reading

Foley, James D., and Andries VanDam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading, MA, 1982.

Hall, Roy, *Illumination and Color in Computer Generated Imagery*, Springer-Verlag, New York, 1988.

Joy, Kenneth I., et al, (ed.) *Image Synthesis*, Computer Society Press, Washington, DC, 1988.

Pixar, *The* RenderMan *Interface, Version 3.0*, May 1988.

Rogers, David F., *Procedural Elements for Computer Graphics*, McGraw-Hill, New York, 1985.

A NEW STANDARD BY IRIS GRAPHICS LIBRARY

Bob Petty
Silicon Graphics

(Paper not provided by publication date.)

NEWS

Arnold Kaber
Sun Micro Systems

(Paper not provided by publication date.)

The Real Time Interactive Display Environment (RTIDE), a Display Building Tool Developed by Space Shuttle Flight Controllers.

Thomas A. Kalvelage

Rockwell Shuttle Operations Company
NASA Johnson Space Center
Houston,TX

ABSTRACT

NASA's Mission Control Center, located at Johnson Space Center, is incrementally moving from a centralized architecture to a distributed architecture. Starting with STS-29, some host-driven console screens will be replaced with graphics terminals driven by workstations. These workstations will be supplied realtime data first by the Real Time Data System (RTDS), a system developed in-house, and then months later (in parallel with RTDS) by interim and subsequently operational versions of the Mission Control Center Upgrade (MCCU) software package. The Real Time Interactive Display Environment (RTIDE) was built by Space Shuttle flight controllers to support the rapid development of multiple new displays to support Shuttle flights. RTIDE is a display building tool that allows non-programmers to define object-oriented, event-driven, mouseable displays. Particular emphasis was placed on upward compatibility between RTIDE versions, ability to acquire data from different data sources, realtime performance, ability to modularly upgrade RTIDE, machine portability, and a clean, powerful user interface. The paper discusses the operational and organizational factors that drove RTIDE to its present form, the actual design itself, simulation and flight performance, and lessons learned in the process.

Key words: Space Shuttle, Mission Control Center, display building tool, RTDS.

INTRODUCTION

The U.S. Space Shuttle is monitored and controlled from the Mission Control Center (MCC) at NASAs Johnson Space Center (JSC) in Houston, Texas. The flight controllers involved in realtime interaction with the Shuttle work for the Systems Division of the Mission Operations Directorate (MOD).

In the MCC, the Shuttle telemetry is fed into a large minicomputer (the Telemetry Preprocessor Computer, or TPC). This machine decommutates the stream and passes it to a mainframe, the Mission Operations Computer (MOC). The MOC does simple limit checking and drives all the displays used by the flight controllers.

Workstations are used in the MCC to process offline programs. Flight controllers and support personnel have written many general and discipline-specific applications for these machines.

INCO Expert System Project

John Muratore, a NASA flight controller, began the INCO Expert System Project (IESP) in 1986 (INCO is the callsign for the Instrumentation and Communication Officer front room flight control position). This project's goal was to develop and test realtime rule-based expert system applications in an operational environment, i.e., during a Shuttle mission.

Because of safety considerations, the project could not use the MOC or TPC. To get realtime shuttle telemetry into a workstation, a Loral ADS-100 off-the-shelf telemetry processor was used. It decommutated the data stream and passed the data to the workstation, where it was moved to an applications interface with custom-built software. This entire system was called the Real Time Data System (RTDS), and it delivered realtime data to MCC workstations years earlier than previously planned.

RTDS and a set of hand-built application programs were used successfully on STS-26. These applications were certified for use in making critical flight calls during ascent.

Impact of Early Delivery of Realtime Data to Workstations.

To begin exploring the possibilities of improved displays, it was decided to remove a few MOC-driven CRTs from consoles and replace them with RTDS-driven graphics terminals.

The author, as a flight controller whose primary CRT was to be replaced, and as an IESP applications programmer, volunteered to write a few specific display applications. The original intent was to hand-code one or two narrowly focussed applications.

The idea of replacing CRTs with workstation terminals gained favor, and more CRTs were scheduled for replacement, including one of the INCOs CRTs. The INCO is a primary, front-room flight controller, and needs to monitor a large number of systems. It would be impractical to hand-code all the displays the INCO would need, so the author began building a tool (called the Real Time Interactive Display Environment, or RTIDE). Originally, this tool was to be a programmers toolkit, allowing rapid development of hardcoded displays. An internal survey was taken to determine requirements.

OPERATIONAL DESIGN CONSIDERATIONS

In general, the displays that RTIDE produced had to satisfy the users. To support this broad guideline, specific requirements were

drawn up.

The user interface had to be intuitive, consistant, and reliable.

To reduce console clutter, the mouse was chosen as the primary input device.

To reduce the chance of flight controller confusion, all mouse buttons had to be treated identically.

To reduce the possibility of selecting the wrong mouseable object, RTIDE had to inform the user when the mouse cursor was over a object (absolutely required, due to safety concerns).

RTIDE had to allow the user to interrogate the display for additional data.

RTIDE had to provide a consistent method of passing information to the user.

RTIDE had to show data in a variety of ways: as a digital value with highlighting when limits are exceeded; as a symbolic message when a value is zero or nonzero; in graphical plot form; and in bar graph form. All these had to make maximum use of color graphics.

RTIDE had to be able to support display of dynamic schematics, with lines and boxes driven by telemetry.

MAINTENANCE DESIGN CONSIDERATIONS

RTIDE was designed to provide a powerful user interface, but other considerations had higher priority. RTIDE would be maintained by flight controllers whose primary job was flight control, not software and data file maintenance. Maintenance phase costs had to be reduced to a minimum.

RTIDE displays had to be buildable by nonprogrammers. There were too many displays to be done by the limited number of flight controller programmers.

RTIDE had to be upwardly compatable with display definition files. Having to change display definition files because of changes to RTIDE is unacceptable.

RTIDE had to be easily expanded. Not only would this help the RTIDE manager incrementally improve the system, but it helps other disciplines who build graphical objects on their own.

RTIDE display definition files had to allow embedded comments. With this, the documentation of a particular display can be included in the display definition file. Then the file contains the entire description of the display and no costly parallel documentation need be maintained.

ORGANIZATIONAL DESIGN CONSIDERATIONS

Although RTIDE would be built and maintained by flight controllers, the hardware RTIDE ran on and the data sources RTIDE used generally were not. Consideration must be given to future changes to RTIDEs environment.

Multiple Data Sources

RTIDE had to be able to access different data sources. RTDS, an internal MOD development system, was the original data source. However, in 1990 the production Mission Control Center Upgrade (MCCU) realtime data interface will become available, and will have to be used.

In addition, a data retrieval system called Near Real Time (NRT) already operates in the MCC workstations, and RTIDE should run off of NRT data files. Besides providing a method of reviewing flight events, this will assist in training flight controllers.

Hardware Independance

RTIDE had to be hardware independant. Currently the MCC is transitioning from its five-year-old Masscomps to new models, requiring software changes to many offline programs.

Configuration Management

Configuration management was a key factor in basic systems design of RTIDE. Flight controllers do not have system manager authority over the machines they use. RTIDE was designed to be as simple and robust as possible, to increase reliability and to reduce the chance of misconfiguration.

Time Constraint

RTIDE was started in 6/88, and had to be ready for STS-29, in 2/89.

RTIDE DESIGN

The basic structure had to be powerful enough to support any reasonable improvement, and simple enough to be maintained by novice programmers unfamiliar with RTIDE.

Organization

The emphasis was on simplicity. RTIDE is a single process, its executable located in a single file, reducing the chance of having file permissions changed or files deleted. It also eliminates having to have the workstation configured a particular way for interprocess communication. RTIDE uses a single display definition file for each display. All the documentation for the display can be included in the same display definition file, eliminating the lag between documentation and implementation.

Event Driven

RTIDE is event-driven. The user (by pressing a key or mouse button, or by moving the mouse), the data sources (by supplying new data), or the operating system (by sending interrupts) may all trigger events that are detected by RTIDE. Event flags are either used directly by RTIDE or sent to the object currently selected by the user's mouse cursor. Event types can be added as desired.

Object Oriented

In the graphics sense, RTIDE is object-oriented. The dynamic symbols on the screen driven by data are objects. RTIDE keeps track of which object the mouse cursor is on, and sends event flags to the object when appropriate.

The hard code determining each objects behavior consists of five standard functions that are located in one source file (generally 500-1500 lines long). The behavior of an instance of an object is determined by a data structure maintained by RTIDE. Adding new objects can be done easily by building this file and adding a structure definition to the master include file.

User Interface

The user interface is designed to be highly interactive, using the mouse, and as simple as possible. Interaction is needed to request further data from the display (limit sets, telemetry status, value range, description, etc).

Display Definition File

The ASCII (for ease of maintenance) display definition file specifies the display's initial condition. Each entry is a series of arguments, each setting some variable (e.g., object colors, messages, data source, etc.).

Program Execution

RTIDE begins by opening the display definition file and reading in

the entries there one at a time. Some entrys (screen_size, data_source, etc) are used to configure RTIDE. The static graphics entries are stored in case the screen is refreshed later. Object entries are stored in the object list. Comments are not currently saved. RTIDE then initializes the graphics processor, displays all the objects, and falls into the main loop.

RTIDE moves constantly through a busy wait loop, first looking for events, then reacting to them. Every cycle, RTIDE pauses for less than a tenth of second, allowing the CPU to run other processes. Because pressing a mouse button interrupts this pause, a user can increase RTIDEs CPU usage by rapidly pressing the mouse buttons.

If new data appears at the interface (nominally, once a second), RTIDE begins to update the screen.

To minimize flicker, RTIDE divides its screen update into two seperate cycles, the process cycle and the display cycle. In the process cycle, RTIDE goes through the objects one at time (using each objects process function), using the new data to update the object. If the object needs to be updated, a draw_me flag is set. Then RTIDE goes through the display cycle, looking for draw_me flags. If one is set, RTIDE redraws it using the objects draw function.

Every cycle RTIDE polls the mouse to find its location. RTIDE compares the location with the information in its object list to see if it has entered or left an object. RTIDE, once the mouse cursor enters an object, only looks to see if it has left the object. This limitation prevents displays from using objects inside of objects.

The cycling continues until an object (usually the exit object) forces RTIDE to exit.

FUTURE DEVELOPMENTS

RTIDE continues to be developed, with new objects being developed for new applications. There is more emphasis being placed on telemetry-driven schematics to increase the efficiency of displays.

Right now we are placing all the documentation into the display definition file. That data is not retained by RTIDE. A later version of RTIDE will save that information, so a user can click on an object and see a complete description of what the measurement displayed means.

SIMULATION PERFORMANCE

RTIDE was installed in the MCC console on 2/16/89. After a few weeks to get the display definition files debugged, RTIDE provided high quality displays for flight controllers. Flight controllers particularly like to be able to get more information from the

display on request.

RTIDE will support STS-29 in March of 1989.

LESSONS LEARNED

Small Size

We found that running a display, its supporting algorithms, numerous fault detection algorithms and several other realtime applications does stress the machine. Keeping the display as efficient as possible is necessary to allow the entire workstation to keep up with the data. RTIDEs relative simplicity, originally specified for other reasons, has kept its executable size down to 237Kb (approximately 100Kb of which is the Masscomp graphics library).

Health and Status Messages

Experience has shown that it is vital to avoid misleading flight controllers, and a display should do its part by telling the controller when the data displayed is useable. The status should be more than a simple GO/NOGO; it should give the controller enough information to begin troubleshooting any data problem.

CONCLUSION

A user-friendly display-building tool has been developed. The object-oriented approach allows rapid display building in realtime command and control environments. The highly interactive user interface allows the user to easily access additional data describing the displays. This tool is being used in support of Space Shuttle missions.

ACKNOWLEDGEMENTS

The author would like to thank Rich Rodriguez of NASA, for his patience, encouragement, and support; John Muratore, for the opportunity to work on the INCO Expert System Project; and Mike Guzzo and Sarah Murray, for all the good work they've done on RTIDE.

REFERENCES

1. Muratore, October 1987, Trends in Space Shuttle Telemetry Applications, Proceedings of the International Telemetering Conference.

KNOWLEDGE REPRESENTATION IN SPACE FLIGHT OPERATIONS

Carl Busse
Jet Propulsion Laboratory
California Institute of Technology

In space flight operations rapid understanding of the state of the space vehicle is essential. Representation of knowledge depicting space vehicle status in a dynamic environment presents a difficult challenge. Space flight operations personnel must work rapidly integrating personal experience with knowledge representation provided by real-time data driven displays. Traditional methods have presented only an incomplete summation of limited system and subsystem information.

New avenues of graphics technology have provided a means for rapid display of complex knowledge and detailed parametric interrelationships. The use of large high resolution displays coupled with fast display update rates, on-screen command menu selection, as well as the inclusion of computer graphics and color oriented knowledge representation and intelligent screen formatting have allowed a rapid transition from information representation to human response.

The NASA Jet Propulsion Laboratory has pursued areas of technology associated with the advancement of spacecraft operations environment. This has led to the development of several advanced mission systems which incorporate enhanced graphics capabilities.
These systems include:

1) Spacecraft Health Automated Reasoning Prototype (SHARP),
2) Spacecraft Monitoring Environment (SME),
3) Electrical Power Data Monitor (EPDM),
4) Generic Payload Operations Control Center (GPOCC), and
5) Telemetry System Monitor Prototype (TSM).

Knowledge representation in these systems provides a direct representation of the intrinsic images associated with the instrument and satellite telemetry and telecommunications systems. The man-machine interface includes easily interpreted contextual graphic displays. These interactive video displays contain multiple display screens with pop-up windows and intelligent, high resolution graphics linked through context and mouse-sensitive icons and text.

INTRODUCTION

The Jet Propulsion Laboratory is a lead center for NASA's planetary exploration and earth science program. In support of this role JPL has pursued areas of technology associated with the advancement of the spacecraft operations environment.

The space flight operations environment presents large volumes of rapidly changing and complex information to flight control personnel. Rapid comprehension of spacecraft and ground support systems conditions are essential in flight operations. Representing complex knowledge is a difficult challenge because it requires operations personnel to integrate on-screen representation with past cognitive experience to often make demanding instantaneous decisions.

The Institutional Computing and Mission Operations Division (37) of the JPL provides the flight control and data management teams, which have supported NASA space mission from Explorer 1, to the Viking spacecraft landing on Mars

and the Voyager mission to the outer rim of the solar system and beyond.

The Division's concentration of on technology in the mission operations domain has centered on the use of enhanced graphics in support of spacecraft and related ground system command and control functions. Enhanced graphics capability in the mission environment is significant for three reasons. First, carefully presented high level knowledge representation reduces the workload of operations personnel. Second, computer-based graphics tools [Schneiderman 1987] improve accuracy of data processing and assist space flight control personnel in monitoring spacecraft and mission sensors where operating data rates may greatly exceed the ability of individuals to monitor successfully. And third, as the number of missions increases, the number of trained and experienced flight support personnel cannot keep up with the extreme demands caused by information overload. Graphic aids and on-screen operator assistance allow for productivity enhancement and maintaining the required level of flight support. [Hansen 1988]

KNOWLEDGE REPRESENTATION

Systems knowledge can be represented be via easily visualized contextual graphic displays [Park 1985]. These interactive video displays provide a direct representation of the intrinsic images associated with instrument and satellite telemetry and telecommunications systems. Multiple display screens with pop-up windows and high resolution graphics are linked throughgh context and mouse-sensitive icons and text.

In order to optimize JPL's mission operations environment, the Space Flight Operations Section (371) and the Project Test and Operations Section (374) have developed unique methods of knowledge

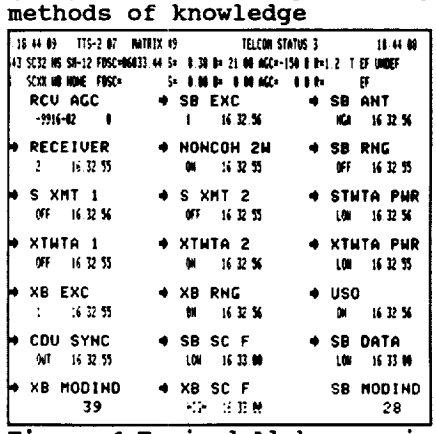


Figure 1 Typical Alphanumeric Spacecraft Telecommunications Display

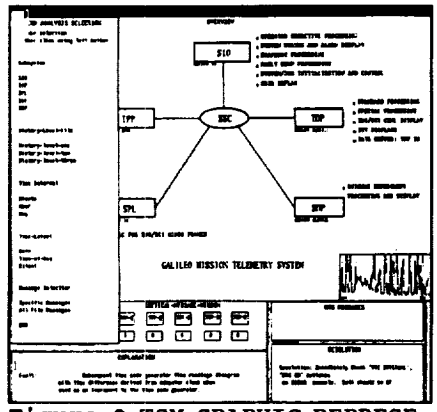


Figure 2 TSM GRAPHIC REPRESENTATION

representation. Mission operations needs have included on-screen display of data since the days of Explorer I. These original capabilities have slowly evolved as have the machines driving them. These capabilities, evolved in the present realm of graphics capability, include primarily alphanumerical paged displays, such as seen in Figure 1, which have been standardized since the Viking Mars landing and the launch of the Voyager spacecraft. Arrows in the display indicate the latest telemetry data channels updated. These traditional displays have met needs of space flight operationss because of relatively large mission operations staffs and relatively low spacecraft data rates. As the number of Flight Operations personnel are reduced and mission operations are streamlined due to budgetary and other considerations, existing personnel and support systems must function at peak capacity. The Institutional Computing and Mission Operations Division (37) is attempting to satisfy mission operations requirements of the future by employing the latest available graphics technology to provide knowledge representation as an aid to flight operations. A display reflecting increased use of visualization techniques can be seen in Figure 2. A key element in flight operations is the need to adequately represent knowledge concerning states and events. The graphics requirements needed to satisfy these needs consist of unique representational goals.

Specifically included as the primary knowledge representation goals in the design of graphics tools shown in table I.

Table I.

1.	Realtime display of large volumes of diverse information
2.	Rapid presentation of complex interrelated information
3.	Color categorization of interrelated and multiple related data fields
4.	Instantaneous display and detection of changes
5.	Promote visualization by decomposition of data into structures and control flows diagrams
6.	Enhanced interpretation of information
7.	Graphical representation of knowledge states

The Institutional Computing and Mission Operations Division has attempted to utilize advances in display technology to advance the spacecraft operations environment. This work has led to development of innovative mission systems which incorporate enhanced graphics capabilities to assist in flight operations visualization. This effort has led to development of graphics systems which provide improved representation of system knowledge which improves the JPL spacecraft and instrument command and control process. Because of the large screen space graphic representations require these systems mix graphics with textual representation. The flight

support system mentioned incorporate several factors in common which aid in knowledge representation and effective visualization of intelligence. Knowledge representation in flight control systems require, as a minimum, the capabilities shown in table II.

Table II.

1. Large high resolution displays
2. High level windowing capability
3. On-screen pull-down command menu selection
4. Color oriented knowledge representation
5. Rapid transition from informational representation to user response

Five flight support systems have been developed which provide examples of knowledge representation through data visualization made possible by graphics technology. These systems are:

1) Spacecraft Health Automated Reasoning Prototype (SHARP),
2) Spacecraft Monitoring Environment (SME),
3) Electrical Power Data Monitor (EPDM), and
4) Generic Payload Operations Control Center (GPOCC), and
5) Telemetry System Monitor (TSM).

Spacecraft Health Automated Reasoning Prototype (SHARP)

The Spacecraft Health Automated Reasoning Prototype incorporates experience of the lead Voyager spacecraft telecommunications engineer into a usable knowledge base. The Space Flight Operations Section (371) has provided an independent standalone graphics capability for the prototype. These representational capabilities will be used in support of the Voyager spacecraft-'s upcoming Neptune Encounter. System know-ledge is represented in terms of annotated space and ground systems context diagrams. Displayed objects are icons and selectable via mouse or keyboard.

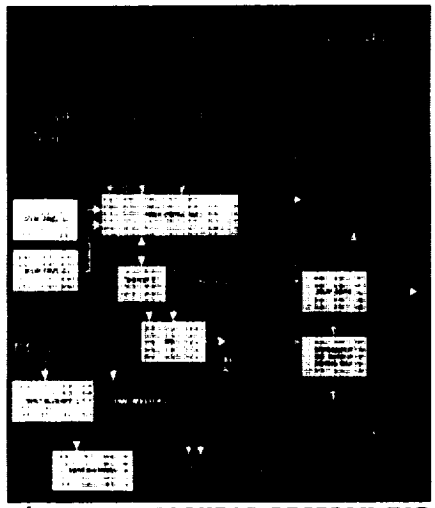


Figure 3 GRAPHICS DISPLAY FOR SHARP

Spacecraft Monitoring Environment (SME)

The Spacecraft Monitoring Environment has been developed by the Project Test and Operations Section (374) and Aura Systems to aid in the Galileo spacecraft integration and test process. The SME provides a real time autonomous spacecraft test sequencing and data monitoring of integration and test activity. The SME provides high-level contextual graphic displays and windowing capability. Command success knowledge is presented by windowing of command issued with telemetry responses .

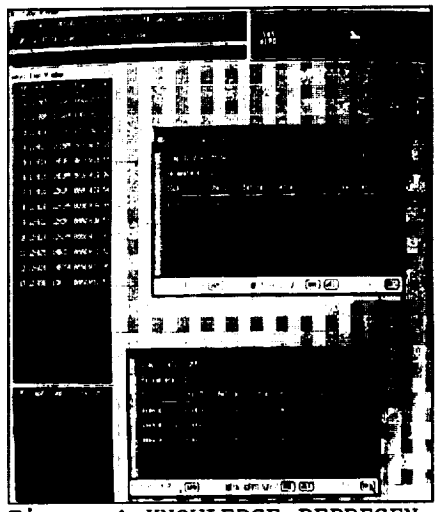


Figure 4 KNOWLEDGE REPRESENTATION IN AN SME TEXTUAL DISPLAY

Electrical Power Data Monitor (EPDM)

The Electrical Power Data Monitor is being developed by the Electrical Power Systems Section (342) and Aura Systems. The EPDM provides power system engineers with automated context diagrams representing power system knowledge [Bahrami 1987]. The EPDM will support the Voyager spacecraft during the up-coming Neptune Encounter. EPDM contextual representation power system knowledge is should in figure 5.

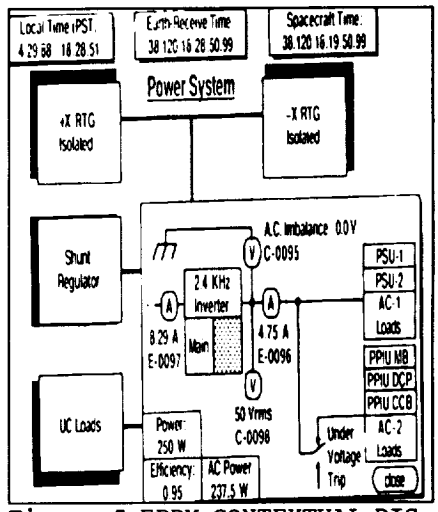


Figure 5 EPDM CONTEXTUAL DISPLAY

Generic Payload Operations Control Center (GPOCC)

The Generic Payload Operations Control Center is a conceptual prototype developed by the Project Test and Operations Section (374) and Aura Systems to apply automation and high level graphics capability to a mission operations environment. The GPOCC goal is to couple expert systems with high level contextual graphics to display to increase user comprehension [D-5435 1987]. A prototype was developed on an Apple MacIntosh II to demonstrate user interfaces and functionality of the GPOCC concept. An example of GPOCC contextual display representation is shown in figure 6.

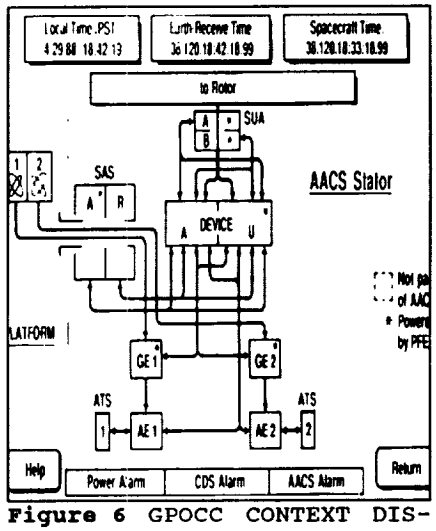


Figure 6 GPOCC CONTEXT DISPLAY

Telemetry System Monitor (TSM)

The Telemetry System Monitor (TMS) expert system, was developed by the Divisions Instrumentation Section (375). This system, using ART as expert system shell was developed on a Sun workstation. The TSM uses graphics capability to aid in rapid visualization of the health and display status of the Galileo Mission Ground processing system and display system fault cues [Mouneimne 1989-]. Knowledge relating the well-being of the telemetry system is contained in expert systems text messages. The graphic display pin-points faults area. An example of TSM representation is shown in figure 2.

Knowledge Representation in Future Mission Operations

As spacecraft data rates increase (the Earth Observing System project 300 Mbps), and are support by extremely large and distributed ground data system the need to provide interpretive knowledge of systems status and configuration increases. Graphics visualization will continue to provide a significant means to accomplish knowledge representnation. Future considerations in knowledge representation are shown in table III.

Table III.

1. Visual man-machine communication to replace alphanumeric communication.
2. Inclusion of hyper-media (including voice synthesis in knowledge representation.
3. Highly interactive display devices

SUMMARY

Knowledge representation in these systems provides a direct representation of the intrinsic images associated with satellite and ground support telecommunications systems. The man-machine interface includes easily interpreted contextual graphic displays. These interactive video displays contain multiple display screens with pop-up windows and intelligent, high resolution graphics linked through context and mouse-sensitive icons and text.

ACKNOWLEDGMENTS

The author gratefully acknowledges the talent, advice and suggestions from Mr. Harry Avant, Dr. Ted Bahrami, Mr. John Carnakis, Mr. David Hermsen, Mr. Warren Moore, Mr. Samih Mouneimne, and Dr. James Willett, of JPL; Mr. Mark Brown and Dr. Rogers Saxon, of Aura Systems; and Mr. Bruce Elgin of Telos Aerospace. I also wish to thank Mrs. Roberta Gray for editorial support.

The work described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

REFERENCES

1. Bahrami, K. A., Atkins, K.L., Saxon, R., and N. Kaufman, "Automated Workstation for the Operation of Spacecraft Engineering Subsystems," Information Systems Prototyping and Evaluation Quarterly Report No 3., NASA Jet Propulsion Laboratory, Pasadena, California, October, 1987.

2. Hansen, E., "Lowering the Cost of Satellite Operations", American Institute of Aeronautics and Astronautics, AIAA-88-0549, 1988.

3. D-5435, Generic Payload Operations Control Center Function Requirements Document, California Institute of Technology, Jet Propulsion Laboratory, Pasadena, (1988).

4. Schneiderman, Ben, Designing the User Interface, Addison-Wesley, Menlo Park, California, May 1987.

5. Park, Chan S., Interactive Microcomputer Graphics, Addison-Wesley, Menlo Park, California, January, 1985.

6. Mouneimne, Samih, "Mission Telemetry System Monitor; Demonstration Prototype", NASA Jet Propulsion Laboratory, Pasadena, California, January, 1989.

N90-20656

THE REAL TIME DISPLAY BUILDER (RTDB)

Erick D. Kindred and Samuel A. Bailey, Jr.
DUAL & Associates, Inc.
1300 Hercules, Suite 208
Houston, TX 77058
(713) 486-1984

ABSTRACT

The Real Time Display Builder (RTDB) is a prototype interactive graphics tool that builds logic-driven displays. These displays reflect current system status, implement fault detection algorithms in real time, and incorporate the operational knowledge of experienced flight controllers. RTDB utilizes an object-oriented approach that integrates the display symbols with the underlying operational logic. This approach allows the user to specify the screen layout and the driving logic as the display is being built. RTDB is being developed under UNIX in C utilizing the MASSCOMP graphics environment with appropriate functional separation to ease portability to other graphics environments. RTDB grew from the need to develop customized real-time data-driven Space Shuttle systems displays. One display, using initial functionality of the tool, was operational during the orbit phase of STS-26 Discovery. RTDB is being used to produce subsequent displays for the Real Time Data System project currently under development within the Mission Operations Directorate at NASA/JSC. This paper discusses the features of the tool, its current state of development, and its applications.

INTRODUCTION

The Real Time Display Builder (RTDB) is the result of the effort to provide timely display building support to the Real Time Data System (RTDS). RTDS is a prototype project to integrate commercial off-the-shelf telemetry equipment with minicomputer workstations to monitor shuttle systems telemetry data in real time. One of the initial goals of RTDS was to develop a display of the hydraulics system for the Mechanical, Maintenance, and Crew Systems (MMACS) flight controllers for operation during STS-26 Discovery. With three months to define the activity, choose the target MMACS system, layout the display, create the symbols, define and program an operating environment, build the databases, build the screen, and test the system, there was a need for time-saving tools.

To expedite development, the display operating environment and the drawing file format were developed concurrently. The drawing file format was initially built by hand, but was designed with a graphics oriented builder in mind. After the initial display was built and the MMACS flight controllers began their review, it was obvious that the most time-consuming effort would be the fine-tuning of symbol positions, colors, and sizes. At this point, initial functionality of what was to become RTDB was coded. The initial functions addressed the high priority items of position, color, size and rotation. These functions provided the capability to quickly prepare the display for flight monitoring.

As RTDS support of other systems

disciplines has expanded, functionality has been added to accommodate the required uses. RTDB has been used to build a graphical representation of the External Tank Ullage Pressure System for the Booster flight controllers. The display was used to test the ullage pressure fault detection algorithm. Also, RTDB has been used to build two other MMACS displays (Brakes/Tires and Elevons), modify the Hydraulics display, and build three new Integrated Communications Officer (INCO) displays operating with a new Real Time Interactive Display Environment (RTIDE) file format. The RTIDE file format was developed to provide specific functionality for the INCO discipline. All of these new developments were operational during STS-29 Discovery.

APPROACH

RTDB development was conceived as a phased introduction of features. Initial functionality was designed and implemented to satisfy initial project requirements, but were integrated into an environment that facilitated the introduction of new features and the enhancement of old. The development required a highly structured and modularized design with well-defined module interfaces, yet flexible data structures.

The data structures that described objects had to be flexible to accommodate the variety of attributes that determined object behavior. Presently, there are over 60 MMACS and RTIDE objects supported by RTDB. To maintain modularity and flexibility, the object attributes are processed internal to the object function. This relegates uniqueness to the object's code and independence from other parts of the system.

An overview of RTDB functional modularity is presented in Figure 1. The major levels are: the User Interface, the Menus, the Processes, the Object Libraries, the Translator, and the Graphics Libraries. The structure is top-down with higher levels deriving functionality from lower levels. The User Interface defines the user's operating environment. The primary interface to RTDB is through a mouse and minimal keyboard data entry. The mouse determines functionality through menu selection.

The Menus define command selection and execution. They act to interpret mouse manipulations and to provide the inputs to command processing.

The Processes direct command execution. The appropriate code is executed as defined by the command string or object record passed from the Menus. This level acts as the link between desired command behavior and the active elements that perform the command. The command behavior is stored as the object attributes; the active elements are executing processes of object instances; and object instances are the definitions of object behavior. This method of object selection, execution, and definition provides the separation necessary to maintain a high degree of modularity.

The Object Libraries consist of the file formats, object attributes, and process definitions. These represent the object-oriented nature of RTDB's operation. The libraries can be thought of as a pool of commands and objects upon which RTDB may call to perform various operations. This is where new functionality will be added. The higher levels will accommodate any additions.

The Translator is a proposed set of macros, functions, and makefile strategies that will facilitate porting RTDB to other

graphics environments (e.g. X-Windows, IBM PC, MacIntosh, MS-Windows, etc.). This level is still in the conceptual stage. It will consist of macro definitions and function calls that map the various graphics primitives into a consistent set of function names and argument lists. The makefile strategies will build an appropriate run-time module for the target system configuration. With a proposed conversion of RTDS to the X-Windows environment, this level will become a necessity.

The Graphics Libraries are a proposed set of graphics environments that will be supported. Currently the MASSCOMP Graphics is the only supported environment. Conversions to other environments will allow more people and machines to participate in the development process.

FUNCTIONS

RTDB is a mouse and keyboard operated, menu-driven, interactive, graphics application development tool that builds displays that operate under the RTDS environment. The best way to discuss its functionality is to trace through the menu hierarchy. Figure 2 depicts the menu hierarchy. It shows current functionality, current development (+), and proposed development (*).

The Main Menu provides the user access to subsequent functions. A menu item must be selected to transfer control to another functional mode.

FILE allows the user to retrieve and store drawing files. Drawing files contain the symbol records that define a display. A proposed feature is to retrieve and display scanned images. These images will act as backdrops for displays or become mouse sensitive with the placement of additional mouse sensitive regions.

COLOR allows the user to select a particular color from a color map, define a new color map, or select from a pre-defined set of color maps.

EDIT allows the user to modify any object attribute, display invisible objects or symbols, and to undo a previous change.
ADD allows the user to add a new object to a display. Available objects are displayed in a pop-up window. The object is selected and placed with the mouse. Multiple objects may be selected and placed while in this mode.

COPY allows the user to duplicate any visible object. After the initial object selection, subsequent left button clicks will place multiple copies. This mode must be exited and reentered to select a new object.

DELETE allows the user to remove an object from the display. Multiple objects may be deleted while in this mode.

MOVE allows the user to reposition any visible object. The object will be repositioned with subsequent left mouse button clicks.

EXIT allows the user to exit the RTDB environment. The user may elect to exit with a save or a no save of the current display buffer contents.

MOVE BEHIND allows the user to reposition an object in the drawing file. The source object's record is physically placed prior to the target object's record in the drawing file. This results in the source object being drawn prior to the target object and allows the target to be drawn on the top of the source.

MOVE IN FRONT OF allows the user to reposition an object in the drawing file. The source

object's record is physically placed after the target object's record in the drawing file. This results in the source object being drawn after the target object and allows the source to be drawn on top of the target.

LOGIC is a proposed function that allows the user to build dynamic logic-driven displays from within RTDB. The proposed method utilizes a combination of two existing tools: the Computational Development Environment (CODE), an RTDS Tool, and the C Language Integrated Production System (CLIPS), an expert system language. Displays have already been dynamically modified using CLIPS. The remaining development requires the integration of CODE and CLIPS through the graphical interface of RTDB.

OUTPUT is a proposed function that allows the user to print the display image to a laser printer.

CREATE OBJECT is a proposed function that allows the user to interactively build a new object. Primitive drawing objects can be combined with existing objects and stored as a new object. The object attributes will be created. The new object definition will be added to RTDB at the Object Library level.

APPLICATIONS

A tool that embodies the concepts of graphics, logic, and databases has a wide area of applicability. The most obvious being process monitoring in which a system is graphically modeled with sensing and control points highlighted. This tool would be applicable to the Space Shuttle, the Space Station, oil refineries, building environmental control, computer integrated manufacturing, etc.

Other uses include simulation, system testing, and training. Simulation follows from system modeling and leads to system testing. Verification of system components and/or data sets can be verified with the use of a simulation. An extension of simulations and models is their use as training tools. What better way to conduct training than to let the student build the system, component by component, specifying component linkages and operating parameters.

The evolution of RTDB as a graphics application builder is depicted in Figure 3. The RTDB will act as a conduit through which utilities will be integrated into a comprehensive application. Those utilities will be an applications developer providing expert system knowledge, RTIDE providing special object and symbol definitions, the RTDS Tool Set providing system utilities, and CLIPS providing an expert system environment. Graphical expert system applications developed in this manner can provide a consistent, controllable applications interface to RTDS, its data sources, and flight controllers.

CONCLUSION

RTDB is an evolving tool, growing to meet the graphical needs of a complex environment. As flight control techniques change and incorporate more graphical displays, the need to develop new displays, convert old displays, and preserve expert knowledge will require the development and use of new tools and techniques. RTDB represents a step in this new direction.

ACKNOWLEDGEMENTS

We would like to express our appreciation to John Muratore and the entire RTDS Team for providing a fertile environment in which new ideas can grow.

Also, we would like to thank the INCO, MMACS, BOOSTER, and EECOM flight controllers for their invaluable inputs.

REFERENCES

1. Aldus Corporation, "Tag Image File Format Specification Revision 5.0," Aldus Corp., Seattle, Washington, and Microsoft Corp., Redmond, Washington, August, 1988.

2. Cox, Brad, OBJECT-ORIENTED PROGRAMMING: AN EVOLUTIONARY APPROACH, Addison-Wesley, Reading, Massachusetts, 1986.

3. Hertzel, William, THE COMPLETE GUIDE TO DOFTWARE TESTING, QED Information Sciences, Inc., Wellesley, Massachusetts.

4. Hopgood, F. R. A., Duce, D. A., Gallop, J. R., Sutcliffe, D. C., INTRODUCTION TO THE GRAPHICAL KERNEL SYSTEM (GKS), 2nd ed., No. 28, Harcourt Brace Jovanovich, London, England, 1986.

5. Jaeschke, Rex, PORTABILITY AND THE C LANGUAGE, Hayden Books, Indianapolis, Indiana, 1989.

6. Johnson, Nelson, ADVANCED GRAPHICS IN C: PROGRAMMING AND TECHNIQUES, Osbourne McGraw-Hill, Berkeley, California, 1987.

7. Jones, Oliver, INTRODUCTION TO THE X WINDOW SYSTEM, Prentice Hall, Englewood Cliffs, New Jersey, 1989.

8. Kalvelage, Thomas, Murray, Sarah, and Guzzo, Michael, "Real Time Interactive Display Environment: Version 0.9 User's Guide," unpublished memorandum, NASA Johnson Space Center, Houston, Texas, October, 1988.

9. Meyer, Bertrand, OBJECT-ORIENTED SOFTWARE CONSTRUCTION, Prentice Hall, New York, New York, 1988.

10. "MCCU Display Builder/Manager Level B/C Requirements," No. JSC-12348, NASA Johnson Space Center - Mission Support Directorate, Houston, Texas, September, 1988.

11. Miller, Edward, Howden, William E., TUTORIAL: SOFTWARE TESTING & VALIDATION TECHNIQUES, 2nd ed., Computer Society Press.

12. Muratore, John, et. al., "Real Time Expert System Prototype for Shuttle Mission Control," Second Annual Workshop on Space Operations Automation and Robotics (SOAR '88), Dayton, Ohio, July, 1988.

13. Riley, Gary, Culbert, Chris, Savely Robert, and Lopez, Frank, "CLIPS: An Expert System Tool for Delivery and Training, "Third Conference on Artificial Intelligence for Space Applications, Huntsville, Alabama, November, 1987.

14. Robinson, Phillip, "Power to the Process," COMPUTER GRAPHICS WORLD, Vol. 12, No. 3, March, 1989, pp. 71-76.

15. Robinson, Phillip, "CIM's Missing Link: Object-Oriented Databases," COMPUTER GRAPHICS WORLD, Vol. 11, No. 10, October, 1988, pp. 53-58.

16. Salmon, Rod, and Slater, Mel, COMPUTER GRAPHICS SYSTEMS AND CONCEPTS, Addison-Wesley, Reading, Massachusetts, 1987.

17. Schmucker, Kurt, "Using Objects to Package User Interface Functionality," JOURNAL OF OBJECT-ORIENTED PROGRAMMING, Vol. 1, No. 1, April, 1988, pp. 40-45.

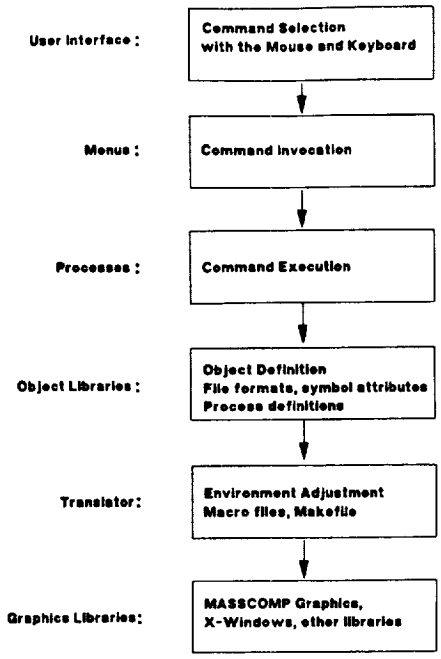


Figure 1 - Design Philosophy

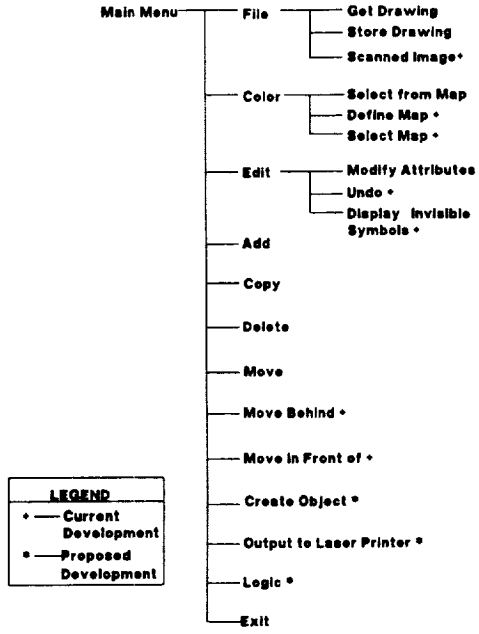


Figure 2 - Functions

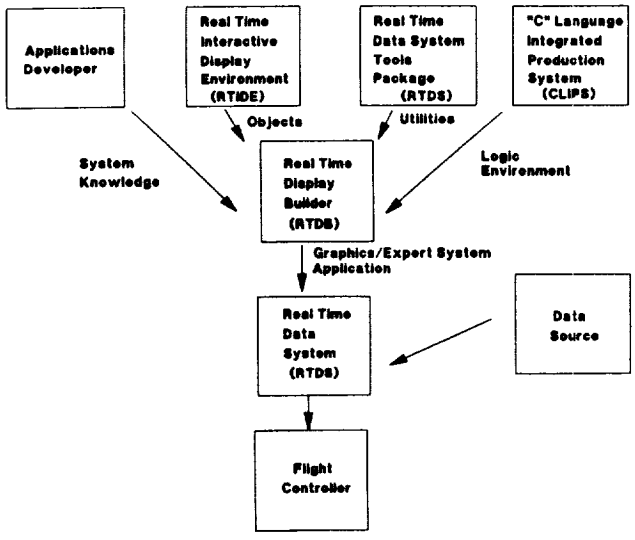


Figure 3 - RTDB Interface to RTDS

N90-20657

Binary Space Partitioning Trees and Their Uses

By: Bradley N. Bell
Barrios Technology Inc.
1331 Gemini
Houston, Texas 77058

ABSTRACT

Binary Space Partitioning (BSP) Trees have some qualities that make them useful in solving many graphics related problems. The purpose of this paper is to describe what a BSP tree is, and how it can be used to solve the problem of hidden surface removal, and constructive solid geometry. The BSP tree is based on the idea that a plane acting as a divider subdivides space into two parts with one being on the positive side and the other on the negative. A polygonal solid is then represented as the volume defined by the collective interior half spaces of the solid's bounding surfaces. The nature of how the tree is organized lends it self well for sorting polygons relative to an arbitrary point in 3 space. The speed at which the tree can be traversed for depth sorting is fast enough to provide hidden surface removal at interactive speeds. The fact that a BSP tree actually represents a polygonal solid as a bounded volume also makes it quite useful in performing the boolean operations used in constructive solid geometry. Do to the nature of the BSP tree polygons can be classified as they are subdivided. The ability to classify polygons as they are subdivided can enhance the simplicity of implementing constructive solid geometry.

INTRODUCTION

The goal of this paper is explain what a Binary Space Partitioning (BSP) tree is and how it can be used to depth sort polygons and perform boolean operations on polyhyedra. Depth sorting of polygons is a technique that has been widely used on personal computers to provide hidden surface removal. With the use of a BSP tree polygons can be sorted fast enough to support the interactive display of shaded polygons with hidden surfaces removed even on a personal computer. Also BSP trees can be employed to solve the problem of Constructive Solid Geometry (CSG). CSG, which is implemented in many model builders, provides the capability to describe complex objects as the intersection, union, and/or difference of simpler primitives. To understand how to use a BSP tree it is important that we have a clear idea of what one is.

BSP TREES

A Binary Space Partitioning (BSP) tree is a data structure that represents the partitioning of space where each branching node represents a plane that divides the space it occupies into two parts and each leaf represents either a polygon (for depth sorting) or a bounded volume (for boolean operations). Given any point in space polygons can be sorted far to near or near to far by using a simple but mathematically determined traversal of the tree. Boolean operations on polyhedron can be performed by cutting the polygonal representation of one operand by the BSP representation of the other.

BUILDING BSP TREE TO DEPTH SORT POLYGONS

In order to use a BSP tree to depth sort polygons the tree can be constructed by using the polygons themselves as planes that subdivide space. This can be accomplished by

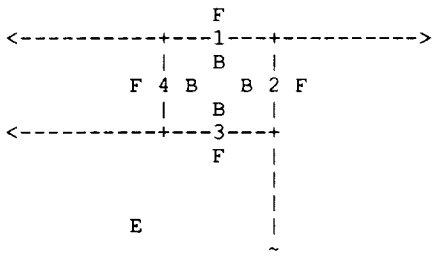
first determining the data structure needed to define a node in the tree. The following is used as an example.

```
typedef struct node
{
     float       A, B, C, D;
     POLYGON     *p_poly;
     struct node *p_front;
     struct node *p_back;
}
NODE;
```

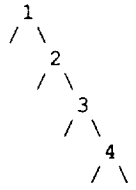
Note: It is convenient to use the sign of the value returned from the plane equation when determining if a point is in front or in back of the plane.

Here the node contains the polygon's plane equation coefficients, a pointer to the polygon representing the sub-dividing plane, and two pointers to nodes that represent the space in front and the space in back of this polygon. Well use a box to illustrate how the tree would be constructed (fig. 2).

BOX WITH
SUBDIVIDING PLANES SHOWN



BSP TREE OF BOX
FRONT<-- -->BACK



F = Front side of polygon
B = Back side of polygon
E = Example Eye point

(fig. 2)

Where the numbers are used to identify the polygons that make up the box. We'll start with a group of polygons at the root of the tree. Then selecting polygon number 1, we will divide the remaining polygons into two groups one representing the polygons in front and the other representing the polygons in back. In this example all of the polygons are on the back side of the first polygon. Next we will select a polygon from each group which will be used to subdivide its group in much the same way as we did the root of the tree, and when a group of polygons contains only one polygon then that branch of the tree is completed. Once the tree has been built a simple but mathematical traversal of the tree can be performed to determine which order to display the polygons in so that the nearest one gets drawn last.

SORTING FROM FAR TO NEAR

To begin sorting polygons from far to near start by entering the eye point into the plane equation of the root node to determine which side of the polygon the eye is on. In (fig. 2) the eye point is shown to be on the back side of polygon number 1. In order to sort the polygons from back to front the half of the tree representing the opposite side of the polygon from the eye must be traversed first then the polygon in this node then the side of the tree representing the side of the polygon the eye is on. So in this example we would traverse the Front side of the tree starting at the root before we would output polygon number 1 after which we would traverse the Back side of the tree. As we traverse the tree we perform the same eye plane test as was done before but using the plane equation at the node we are on in the tree to determine which branch of the node will be traversed first. The tree traversal proceedure can easily be implemented as a recurrsive function (fig. 3 as an example).

SORTING FROM NEAR TO FAR

To sort polygons from near to far the process is identical except instead of traversing the side of the polygon that is on the opposite side of the dividing plane first, you traverse the side of the polygon that the eye is on first.

```
FarToNear( n, x, y, z )
NODE *n;
float x, y, z;
{
  float p;

  if( n )
  {
    p = n->A * x + n->B * y
      + n->C * z + n->D;

    /* ASSUMING THE NORMAL OF
       THE PLANE IS POINTING
       TO FRONT HALF */

    if( p < 0.0 )
       FarToNear(n->p_front,x,y,z);
    else
       FarToNear(p->p_back,x,y,z);

    DrawPolygon(n->p_poly);

    if( p > 0.0 )
       FarToNear(n->p_front,x,y,z);
    else
       FarToNear(n->p_back,x,y,z);
  }
}
```

(fig. 3)

BOOLEAN OPERATIONS ON POLYGONAL MODELS

One way to perform boolean operations on polygonal models is to use a BSP tree. This can be accomplished by first constructing a BSP tree representation of each model then using the tree of one model to subdivide the polygons of the other model into inside and outside components. Then depending on the operation being performed the pieces needed are gathered together from both models to form the result. The BSP representation however differs slightly from the one used to depth sort polygons.

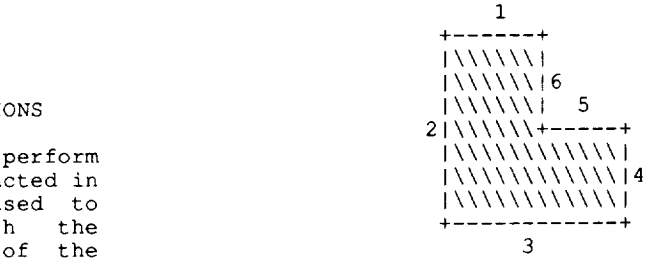
BUILDING BSP TREE TO PERFORM BOOLEAN OPERATIONS

The BSP tree used to perform boolean operations is constructed in a similar way as the one used to depth sort polygons with the exception that the branches of the tree represent the division of space into inside and outside components with the subdividing plane representing a polygon which is part of the model. In order to explain how the tree could be constructed we need to determine what type of data structure to use. The following is given as an example.

```
typedef struct node
{
     float       A, B, C, D;
     POLYGON     *p_poly;
     struct node *p_outside;
     struct node *p_inside;
}
NODE;
```

Here a node in the tree contains a plane equation, a reference to a polygon, a pointer to the branch of the tree representing the outside of the volume and a pointer to the branch representing the inside. To construct the BSP representation of the model we first select a polygon from the model that we will use as the dividing plane at the root of the tree. We then proceed to divide the remaining polygons by the dividing plane at the root of the tree into two groups, one to the outside of the plane and the other to the inside. Each group represents a branch from the root node of the tree. Next from each group a polygon is selected to become the dividing plane of its group and is placed into the appropriate node. Each group is then subdivided by its associated node and placed into two separate groups again representing the polygons to the inside and outside of the dividing polygonal plane. This proceedure is performed recurrsively until their is only one polygon left in the group which is then placed into its own node with both of its branch pointers set to 0. The following is given as an example.

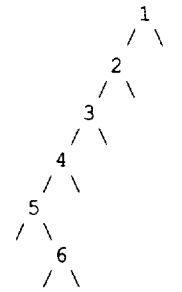
DIAGRAM OF SIMPLE MODEL



KEY:
1-6 Polygon identification
\ The interior of the model

TREE REPRESENTING THE ABOVE MODEL

INSIDE<--- --->OUTSIDE



Once the BSP tree has been constructed for both operands of the boolean operation the polygons of each model can be subdivided by the other model's BSP tree. To subdivide a polygon by the BSP tree we take the polygon and start at the root of the tree and test to see if the polygon is inside, outside, or on both sides of the subdividing plane. If the polygon is on both sides of the subdividing plane then it is split into two polygons with the one representing the inside part and the other representing the outside part. The polygon (parts) is (are) then tested against the plane in the node pointed to by the associated branch. This proceedure is performed recurrsively until a branch pointing to nothing is reached at which time the polygon (part) is given the classification of the branch. If the polygon being tested lies in the same plane as the subdividing plane then a more complex procedure is required. First we send the whole polygon down the inside branch of the tree. Next we make note of the classifications given to the resulting parts. Then we send each part down the outside branch of the tree. If the part comes back with the same classification as it did going down the inside branch of the tree then it is correctly classified. Should the part get subdivided while being passed down then the subparts that have the same classification are correctly classified. The the parts that come back with a different classification are considered as coplaner polygons and are assigned the classification of OPPOSITE if the polygon's normal points in the opposite direction as the normal of polygon it is coplaner to otherwise it is given the classification of SAME. Once all of the polygons in both operands have been classified in this manner the resultant model can be formed. The following table describes for each boolean operator which polygons are taken from each operand to form the resulting model.

OPERATION	AND		OR		-	
OPERANDS	A	B	A	B	A	B
INSIDE	X	X				F
OUTSIDE			X	X	X	
OPPOSITE					X	
SAME	X		X			

= Do not use to form result
X = Use directly to form result
F = Flip normal of polygon before using to form result

CONCLUSION

Even though Binary Space Partitioning (BSP) trees can be used to perform the tasks described in this paper they are not practical when working with models that are highly complex. The tree tends to grow exponentially as the model complexity grows linearly. However they do offer implementation simplicity and therefore have a useful place in software development.

ONBOARD SHUTTLE ON-LINE SOFTWARE REQUIREMENTS SYSTEM: PROTOTYPE

Barbara Kolkhorst (IBM)

Barry Ogletree (IBM)

INTRODUCTION

Late in 1987, the Spacecraft Software Division (SSD) of the Mission Operations Directorate of NASA's Johnson Space Center (JSC) in Houston asked IBM, as contractor for Onboard Shuttle Software (OBS), to investigate the problem of storing the existing Flight Software (FSW) requirements in an electronic form. These requirements define functions related to vehicle guidance, navigation and flight control and are thus critical to Shuttle missions. These documents, consisting of integrated text and engineering drawings, exist as many different documents residing at several NASA locations and were developed over approximately fifteen years as the Shuttle program evolved. The requirements should be accessible to the NASA community on-line; ultimately, automated requirements to code mapping should be available.

As a result, a small technical team worked in three phases to satisfy the NASA request. In the first phase, the team leader, several software requirements analyst's and a system engineer familiar with commercial product search techniques defined the problem to be attacked; this was documented as a request for information from NASA. In the second phase of the task, a solution for the problem was developed and an engineer experienced in electronic publishing systems was added to the team. Goals were developed to determine which solution would be proposed:

1. The requirements documents should be in electronic form under the central control of the Shuttle Avionics Software Control Board (SASCB) of NASA JSC.
2. Editing and publishing of the requirements should be under strict configuration control of the SASCB. On-line viewing is controlled by system security programs and the publishing system.
3. The solution should be a complete integrated solution which maximized the commercial software content to minimize development and maintenance costs of the system.
4. In addition, the eventual goal would be to provide a solution in which 'what is approved is published'. That is, what was approved by the SASCB had been submitted electronically and incorporated into the requirements data system automatically after proper approval; no rekeying of information would be necessary.

In the third phase of the project, a prototype was developed to prove that the proposed system could indeed be used on the Shuttle FSW requirements; several programmers were added to the team at this time.

This three-phase task was successful and provided a solution with very high commercial content which provided most of the function required. A prototype solution was demonstrated in November of 1989 to the Spacecraft Software Division (SSD) and to the NSTS Engineering Integration Office.

PROBLEM DESCRIPTION

The Shuttle FSW requirements documents consist of approximately 31,000 pages of integrated text and line drawings divided into roughly forty-five books averaging 650 pages each. The documents exist in several word processors and on paper at several NASA and contractor locations. Publication is disjointed across books and there is no consistent document architecture. Drawings are integrated into the documents using manual cut-and-paste methods. Modifications are proposed to these documents on a regular basis by many authors and must pass through an approval process controlled by the SASCB. Until the changes are approved, there is no hardcopy of the requirements documents. Only approved modifications can be added to the baseline document after a requirements writer has certified that all changes are correct. This results in a number of areas of concern.

First, due to the delay between submission and approval of changes and actual publication of a hardcopy version, the software developers are often working with changes plus outdated published requirements. Second, the requirements writer must also have access to the latest version of the baseline document for developing change requests. Since there is a time delay when modifications are being submitted and ultimately approved for publication, the requirements writer must work with outdated versions. Third, the changes are manually integrated into the baseline document for publication and here some transcription errors may occur.

Since requirements definition is critical in the process of maintaining space shuttle avionics software, the proposed system must address the areas of concern and provide ways to compensate for the evolutionary environment in which the software must operate. The needs are best satisfied by a host-based publishing system because these software requirements documents are organized in a book format, created by many authors, composed of information from numerous sources, published for many users, and require centralized configuration control.

Proposed Solution

The proposed solution includes initial document capture, storage, retrieval, hardcopy publishing, electronic distribution, security, change request disposition, and configuration management for the requirements documents.

The initial capture of the documents will be done by either scanning printed pages or through conversion of various electronic word processor formats to the system format. Scanned pages will be converted to text and image files by an intelligent recognition engine and custom software. Finally, the proposed system will provide the foundation for future interfaces to other systems for tracking Space Shuttle components.

As a further enhancement, application bridges to other NASA systems can be developed to connect the requirements document system to other Space Shuttle components and systems. It is also highly desirable that the system be integrated into the existing NASA computer software and hardware base.

Proposed Solution Rationale

It should be noted that alternative solutions were investigated. A solution was considered where the requirements documents were stored as scanned images with no modification capabilities. The existing process for document creation and modification could be used and a configuration control system could be built around this manual process. Neither of these two solutions would provide NASA with as much flexibility to manage and control the entire document process as would a publishing solution.

A host based solution was chosen over a work-station based solution because of the volume of documents to be managed, the security control required to protect access to and integrity of the documents, the greater variety of printers, terminals, and storage devices available for attachments, the ability to connect to the existing information network as a host system, and the capability of supporting a larger number of simultaneous end users. The proposed solution does take advantage of the power and flexibility of intelligent work-stations to download a section of the requirement documents, modify or print selected sections, and submit the modification as a Change Request. This proposed solution allows NASA to build a strategic electronic requirements document system now and for the future.

The hardware for the actual solution consists of a scanner capable of intelligent character recognition and the separation of images from text, all points addressable printers at both the workstation and host, an intelligent workstation processor, disk storage and an IBM compatible host on NASA's JSC Center Information Network (CIN). The software for the proposed solution consists of two parts: a host part (a publishing system with support for viewing and control of documents) and a workstation part (a desktop publishing product and some custom user interface software). In addition, there is software to allow documents on the workstation to be converted and transferred to the host, support for scanner operation, filters which convert documents created on other word processing systems to the host publishing system format, and software used to view the published documents. Security and configuration control are provided by either the publishing system or the operating system. See Figure 1 for a pictorial view of the system. Figure 2 describes the hardware and software defined for the solution which are included in the prototype system.

System Hardware

The publishing host (shown in Figure 3) consists of an IBM System 370 processor, magnetic disk storage, a tape unit, a disk controller, all points addressable (APA) printers, and terminal and communication controllers. (In the future, optical disk storage may be added to allow increased capacity.) It is proposed that NASA use or share an existing host hardware system (tapes, disk, terminal and communications controllers already in place) for this application.

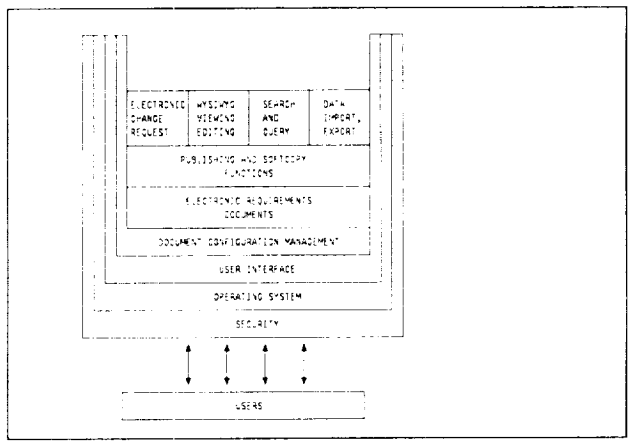


Figure 1. Modular view of the OBS On-line Software Requirements System

The magnetic disk storage will contain the active requirements documents and the application libraries. The application libraries will require approximately 1 megabyte of magnetic storage. The Onboard Shuttle Flight Software requirements documents will use 10 gigabytes of magnetic storage. The 10 gigabytes of storage will allow up to 200 active books (130,000 pages) to be maintained with on-line access. Frozen requirements documents will be archived on the optical storage jukebox. Sizing for the jukebox will be determined after initial implementation.

The page printer would be used to produce camera ready hardcopy documents. The IBM 3820 or 3800 printer is capable of printing complex pages consisting of text, graphics, and images. (However, any all points addressable printer capable of interfacing with the IBM Publishing System could be used to produce cameras ready documents.)

The recommended workstation for the Publishing Specialist, and the SASCB Administrator is an IBM Personal System/2 (PS/2) Model 80 (machine type 8580). The PS/2 has an 80386 microprocessor with MicroChannel architecture and 80 nanosecond memory. The workstation configuration consists of six megabytes of memory, a 115 megabyte fixed disk, a mouse, a 1.44 megabyte diskette drive, and a high resolution IBM 8514 display monitor.

	SOLUTION COMPONENT	PROTOTYPE
H	HOST (IBM COMPATIBLE ON CIN)	Y
A	HOST ALL-POINTS-ADDRESSABLE PRINTERS	Y
R	MAGNETIC STORAGE	Y
D	OPTICAL STORAGE	N
W	WORKSTATION (IBM PS/2 MOD 80 8514 DISPLAY)	Y
A	WORKSTATION (WS) ALL-POINTS-ADDRESSABLE PRINTERS	Y
R	SCANNER (CALERA CDP 9800)	Y
E	NECESSARY INTERFACES	N
S	IBM PUBLISHING SYSTEM	Y
O	DOCUMENT VIEWING SOFTWARE	Y
F	INTERLEAF PUBLISHING SYSTEM	Y
T	SCANNER SUPPORT SOFTWARE	Y
W	DATA EXPORT PROGRAM (TO HOST)	Y
A	WORDPROCESSOR TO HOST FORMAT FILTERS	PARTIAL
R	WS TO HOST IMAGE CONVERSION	Y
E	HOST TO WS PUBLISHING SYSTEM FORMAT CONVERSION	N

Figure 2. Further Comparison of Solution vs. Prototype

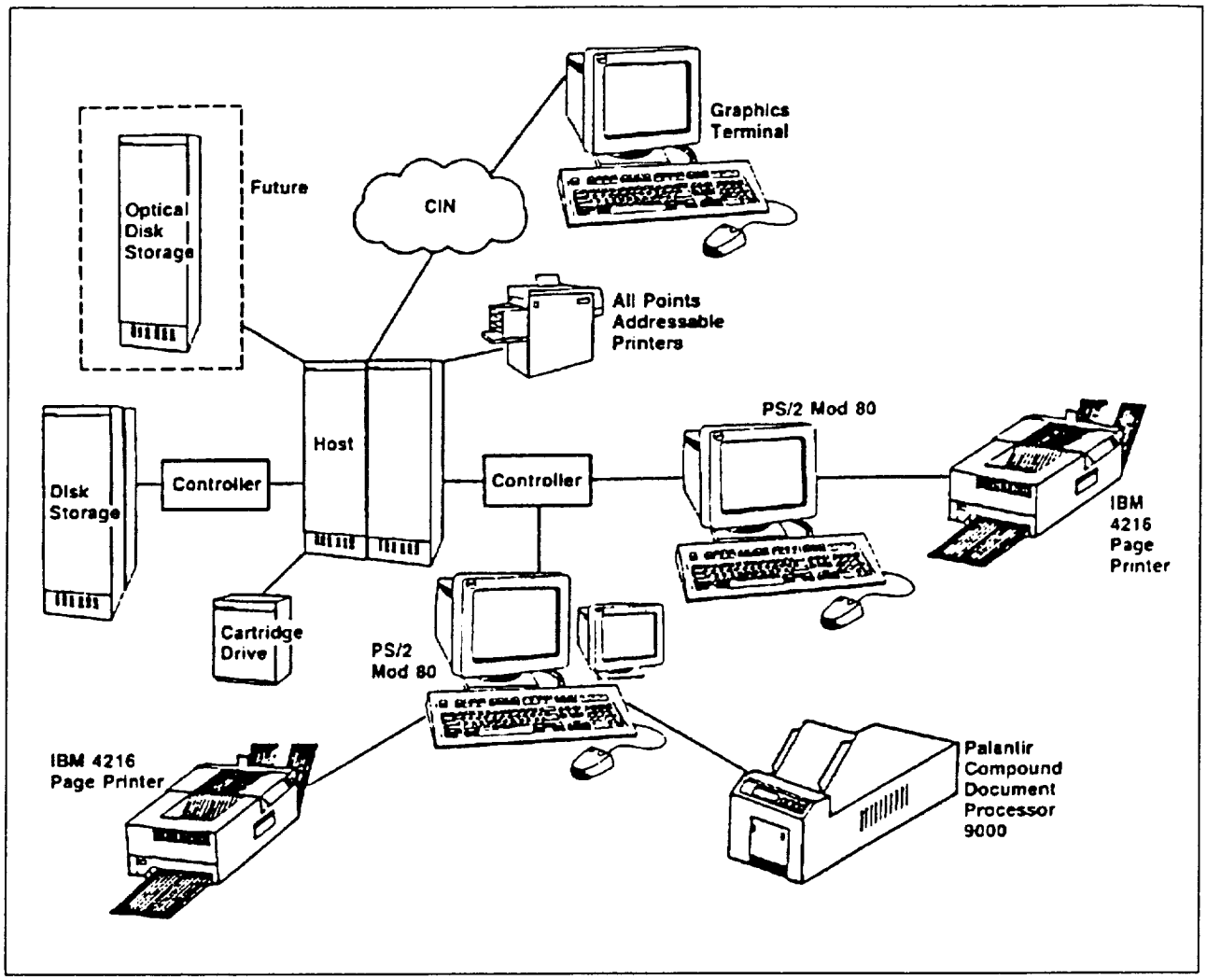


Figure 3. OBS Online Software Requirements Hardware Overview

Some workstations will have a scanner, such as the Palantir Compound Document Processor (CDP) 9000, and a workstation printer, such as the IBM 4216 Page Printer. The Palantir CDP 9000 will provide the capability to scan a document up to 8.5" X 14" in size at a resolution of 300 dots per inch; additional scanners using other resolution densities may also be attached to the Calera recognition engine. In addition, its recognition accuracy improves as more data is scanned. For workstations performing graphics modification, a second monochrome display may be required.

Graphics terminals on the CIN may be used to view documents and to write documentary change requests. These change requests must eventually be keyed into the source files on the host by a publishing specialist.

Scanners and Graphics Concepts

Scanners for graphics work can be categorized on the basis of several characteristics:

- Type of scanning mechanism (flatbed versus page feeder)
- Resolution (low of 75 dots per inch (dpi), high of 1500 dpi)
- Intelligent characteristics:
  - Text recognition (specific fonts versus any font)
  - "Tagging" (output of text to word processor formats)
  - Graphics handling (manual versus automatic)

For graphics work (especially the handling of integrated text and graphics), the ideal solution would be to let the scanner handle all aspects of the conversion process: feed the document into the scanner, separate out text and graphics, perform text recognition, and place the output into text and/or graphics files automatically. In practice, this is difficult to achieve:

- A major factor in this problem is the difficulty involved in identifying the start/end of graphics sections.

  One workaround is to let the user specify the location of graphics sections (this of course requires manual intervention).

- Inability for the computer to understand document "structure" (what figures go where).

  This might be due to the inability of most PC-based word processors to handle graphics (this is becoming less problematic with the advent of new word processors which support graphics manipulation).

Another area of difficulty is in text recognition. This is a "graphics-to-text" conversion where the scanner looks at the pattern of dots produced during scanning and makes a decision about the character represented by that pattern. Some scanners are unable to support this feature (requiring software to do the job); some scanners can only support a limited set of fonts. The most powerful machines

perform "intelligent character recognition", recognizing any font style or size; perform spelling correction, marking unrecognized words for later correction; and decipher page layout automatically, distinguishing between text and graphics sections.

The Calera scanner used in this prototype has the features required to support this project. It is a sheet-feed (50 pages maximum) scanner with adjustable resolution (maximum 300 dpi), spelling dictionaries, and intelligent character recognition. In addition, it is represented as a "compound document processor", being able to scan integrated text/graphics documents; however, in its stand-alone mode it requires user intervention to designate graphics areas which are later placed in separate files. There is a board available from the same company which purports to handle integrated text/graphics automatically, but it was not available at the time the prototype was developed.

During the development of our prototype, we encountered several problems relating to graphics/text work:

- Recognition of special characters.

  Special characters (underline, super and subscripts, etc.) are difficult to scan properly.

- Registration of pages in scanner

  Straight lines in the source document become "stair-step" lines in the printed version. The workaround is to use a flat-bed scanner (less problems, but the stair-step effect is still noticeable). This also means production work is more difficult due to the necessity of handling each page separately.

- Loss of image "content"

  The scanning process produces raster files; the original image may have been produced by a vector process. This means that the information about object structure has been lost. There are programs available which can re-vectorize a raster-based image, but the robustness of the conversion is unknown.

  Hardware and software to do image to vector conversions for engineering drawings will be studied later in this project.

- Lack of consistent support for "standard" image formats.

  Specifications are defined for various image formats (TIFF, PCX, etc.) but some programs support only a subset of the available options. If the programs being interfaced do not understand the same set of image data, problems occur. A workaround is to understand exactly what is required by available programs and select those with matching capabilities.

- Storage requirements may be prohibitive for raster format files.

  Scanning an 8 1/2 by 11 inch page at 300 dpi results in about 8.5 Mbits of data (uncompressed). Certain formats (e.g. TIFF) can support various compression schemes to reduce the requirement for storage space. The resulting file may still require about 1 Mbyte of storage; vectorized files require far less storage.

System Software

The proposed software will be distributed between the host and the PS/2 Model 80 workstations. The prototype host software consists of the VM Operating System, IBM Publishing System and the necessary support services and utilities. The workstation software consists of IBM's Disk Operating System (DOS), Interleaf Publisher, and scanner support and image editing software. Custom code in both the host and workstation will facilitate transfer and configuration management of data files.

The host Publishing System software executes in the IBM VM operating system environment and is designed for corporate in-house publishing. An MVS solution is planned for NASA JSC use; it integrates the in-house publishing process from start to finish, including typeset-quality output of documents containing text, graphics and image. IBM's electronic publishing solution uses the host computer, workstations and printers to create, display and print documents.

The host Publishing system is an integrated set of software products (shown in Figure 4) and consists of:

- Publishing Systems ProcessMaster: A set of menus that controls the overall operation of the publishing system and provides a document control library management facility.
- Publishing Systems BookMaster: A powerful document creation application based on IBM's Generalized Markup Language (GML) that provides the tools necessary to create complex document formats.
- Graphical Display and Query Facility (GDQF): A package for viewing and editing CAD/CAM and other graphics data files on the host.
- Publishing Systems BrowseMaster: A series of utilities (provided in GDQF) to:
  - View merged text, graphics and image
  - View and crop GDDM Graphics Data Format (GDF) files and convert them to page segments
  - Import drawings from non-IBM CAD/CAM systems
- Publishing Systems DrawMaster: A menu-driven line art drawing package for creating graphics for use in publications.
- Image Handling Facility: A program to manipulate images for inclusion in documents.
- BookManager: An application for electronically viewing documents stored at the publishing host (SmartBook, an IBM internal product, is used in the prototype).

The workstation publishing software will be the IBM Interleaf Publisher. This standalone product executes under DOS on an IBM Personal System/2 model 80. The IBM Interleaf Publisher is a full-function, integrated publishing program.

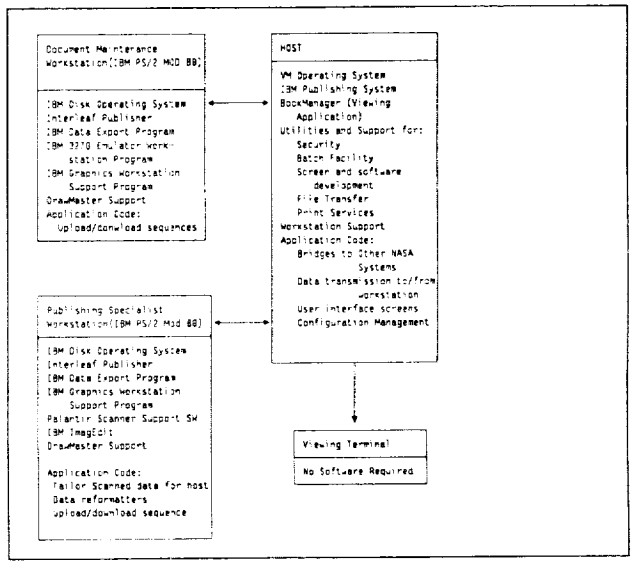


Figure 4. OBS On-line Software Requirements System Software Overview

Workstation-Based Functions

Scanner Support

Calera provides software with their scanner that assists the user in performing scanning chores. This software is divided into two types: applications (PAGEBLD, EDITPRO, TOPSCAN) and utilities (such as PDA2TIFF, DOCBUILD, and others provided by the scanner manufacturer to assist in custom software development).

PAGEBLD is the primary software used for scanning integrated documents. The scanner can be completely controlled from a full-screen (Windows-based) menu; functions to scan and read pages, save results, and work with the document are provided. The document is defined as having "zones" of information. Some zones contain text and are processed using text recognition techniques; other zones are graphics and are processed into CCITT-format files. To automate the process, zones may be predefined in "Zone Format Files"; this is useful when scanning must be automated, but it requires that pages adhere to a consistent format.

After the document has been processed by PAGEBLD, the next step is to use EDITPRO. This is a Windows-based application which helps the user find places where PAGEBLD and the scanner hardware had some difficulty recognizing text. Optional marks can be placed in the processed files; if present, these marks are used to drive the EDITPRO software. Functions are provided to move from mark to mark and the errors found may be changed while in EDITPRO (no need to use a separate word processor). After errors have been removed, files are created with the corrected information.

TOPSCAN is an application that provides scanning functions which understand most popular PC-based word processors and graphics formats. Text recognized by the system can be placed directly into a format understood by the user's word processor; graphics files are placed in TIFF format and can be used by any program understanding this file type.

Utilities

The scanner manufacturer supplies a set of utilities which assist the user in developing customized scanning applications. These utilities include standalone special-purpose programs that can build document files from text or image input, compress and decompress image files using CCITT Group 3 or Group 4 algorithms, modify text files to remove white space, and operate the scanner in a command-line driven (rather than graphics menu-driven) manner.

Graphics Manipulation

Manipulation of image files can be performed on the workstation or on the host system. For workstation-based image editing, IBM's ImageEdit is available. This program understands various file types (including TIFF) and provides editing to a pixel-level as well as the capability to draw lines, circles, and other basic shapes. It can produce TIFF files in both uncompressed and compressed forms.

Host-Based Functions

Because of requirements specified during the prototype definition phase, the major portion of the system resides on IBM mainframe computers. The environment (especially from a software and printer viewpoint) is considerably different from the personal computer environment; graphics formats are unique (GDF is used for vector files, IMG is used for image files). Image Handling Facility (IHF) and other programs in the IBM Publishing System are required to convert images to the format required for printing; this format (Page Segments or PSEG) is used because of the system which prints documents with imbedded images (Document Composition Facility or DCF).

Graphics Manipulation

The primary formats utilized on the host are:

- Vector-based

  GDF, CGM

- Raster-based

  IMG

Host software is available to manipulate both types of format. DrawMaster is a product which produces vector-based files (GDF and others); IHF is available to edit raster-based (IMG) files.

Viewing

Viewing of documentation is provided by two programs: BookManager (for text-based document reference with graphics support) and BrowseMaster (for publishing system specialists required to proof documents before printing). For the prototype, an IBM internal use tool called SmartBook was used to provide BookManager functions; it was the precursor to the BookManager software.

BookManager is the program of choice when users must refer to text and be able to browse figures which are present in the document. It operates by displaying the document in text mode (which means that users without a graphics terminal will be able to read the document) unless the user requests that a figure be displayed; the system then changes to a graphics mode and displays pictures specified by a user command. BrowseMaster is most useful to individuals requiring information about the layout of the document and who must provide error-free printing (as far as layout and appearance are concerned). It is used to provide a preview of the layout (a page image including margins and simulated text) so that those individuals responsible for printing the document can insure there are no major errors before submitting the job to the system printer. This method is similar to some PC-based word processors which allow the user to look at a page before printing it, resulting in savings of time and system resources such as paper.

Printing

Printers available on the host system range from line-based to laser-compatible (IBM's 3820 printer is the printer of choice). The 3820 used in the prototype is a host-connected printer capable of 240 dots per inch and a print speed of 20 pages per minute; since its resolution differs from the resolution available with the Calera scanner, a problem with image degradation occurs. This problem can be avoided in two ways: scan images and reduce them to the required size using the Publishing system, or use an alternate scanner (such as the IBM 3118) which is capable of scanning at the same resolution as the printer (240 dpi).

Summary

The prototype discussed in this paper was developed as proof of a concept for a system which could support high volumes of requirements documents with integrated text and graphics; the solution proposed here could be extended to other projects whose goal is to place paper documents in an electronic system for viewing and printing purposes. The technical problems (such as conversion of documentation between word processors, management of a variety of graphics file formats, and difficulties involved in scanning inte-

grated text and graphics) would be very similar for other systems of this type. Indeed, technological advances in areas such as scanning hardware and software and display terminals insure that some of the problems encountered here will be solved in the near-term (less than five years). Examples of these "solvable" problems include automated input of integrated text and graphics, errors in the recognition process, and the loss of image information which results from the digitization process.

The solution developed for the Online Software Requirements System is modular and allows hardware and software components to be upgraded or replaced as industry solutions mature. The extensive commercial software content allows the NASA customer to apply resources to solving the problem and maintaining documents, rather than spending a large portion of the maintenance resources on custom software.

The actual conversion of scanned text and drawing images to a form which can be stored in a publishing system provides NASA with the capability to transfer any paper documents to editable electronic form for maintenance and update. As the various filters are procured or developed, documents which exist in other word processor formats may be added to the central files. The central repository may consist of magnetic storage for active documents and optical storage for documents which have been frozen in final format. This system may be used for storing and maintaining any documents consisting of integrated text and drawings.

This electronic base of information is suitable for future applications such as hypertext, where specific reference points in the documents are electronically linked to other documents, other parts of the same documents or note information. Additional search and query capability will also provide the NASA community with the ability to obtain information more rapidly than was ever possible with paper-based documents.

Definition of Acronyms

CAD. Computer Aided Design

CAM. Computer Aided Manufacturing

CGM. Computer Graphics Metafile

CIN. Center Information Network

CDP. Compound Document Processor (from Calera)

DOS. Disk Operating System

DPI. Dots Per Inch

FSW. Flight SoftWare

GDDM. Graphical Data Display Manager

GDF. Graphics Data Format

GDQF. Graphical Display and Query Facility

GML. Generalized Markup Language

IHF. Image Handling Facility

IMG. IMaGe Format

JSC. Johnson Space Center

MVS. Multiple Virtual Storage

NSTS. National Space Transportation System

OBS. OnBoard Shuttle software

PC. Personal Computer

PCX. PC Paintbrush Graphics File Format

PPM. Pages Per Minute

SASCB. Software Avionics Software Control Board

SSD. Spacecraft Software Division

TIFF. Tagged Image File Format

VM. Virtual Machine

WS. WorkStation

DESIGN CONSIDERATIONS FOR A SPACE DATABASE

Lance M. Moss

Evans & Sutherland Computer Corporation
600 Komas Drive
Salt Lake City, Utah 84108

Part of the information used in a real-time simulator is stored in the visual database. This information is processed by an image generator and displayed as a real-time visual image. The database must be constructed in a specific format, and it should efficiently utilize the capacities of the image generator that it was created for. A visual simulation is crucially dependent upon the success with which the database provides visual cues and recognizable scenes. For this reason, more and more attention is being paid to the art and science of creating effective real-time visual databases. This paper investigates the database design considerations required for a space-oriented real-time simulator. Space applications often require unique designs that correspond closely to the particular image-generator hardware and visual-database-management software. Specific examples from the databases constructed for NASA and its Evans & Sutherland CT6 image generator illustrate the various design strategies used in a space-simulation environment. These database design considerations are essential for all who would create a space database.

1.0 INTRODUCTION

During 1987 and 1988, Evans & Sutherland designed and developed three databases for NASA's System Engineering Simulator. Much of the experience gained and many of the techniques used in the construction of a terrain database were transferred to the construction of the space databases, but many new challenges were encountered. This paper describes some of the challenges unique to the design of a space database and explores the techniques and strategies developed to meet these challenges.

2.0 VISUAL SYSTEM

A major part of a real-time vehicle simulator is the visual-scene-generation system. The visual system's role is to provide the visual cues that help make the simulation effective. In general, a visual system comprises a computer image generator, which is the processing hardware; displays on which the computer-generated imagery is viewed; a visual database, which is the information the image generator processes to produce images; and a set of managing software, known as the real-time system. The image generator includes a set of processors, among them viewpoint processors and channel processors. The visual database is a model of the real-world environment. The real-time system is the interactive software package that controls and manages the image generator's transfer and processing of the data in the visual database.

For a database to reach its utmost visual potential, each construct within it must be used efficiently. Therefore, no database should be designed without an understanding of its visual system's processing capabilities. Some of the necessary design information includes:

1. The configuration of the image generator (including the number of viewpoint processors and channel processors) and the update rate.
2. The maximum input and output of each processor in the image generator
3. The maximum visible output for each channel
4. The amount of time allotted for each processor to accomplish its task
5. The amount of on- and off-line storage available for the database

Database engineers must analyze all the processing requirements before a design is developed. With the resultant knowledge, a database can be constructed that will fully utilize the capabilities of the image generator. However, this information must be correlated with the specified database requirements. Sometimes database requirements tax the capabilities of the given visual system. In these cases, ingenuity in the design can satisfy the requirements and still not overload the image generator. Extreme cases may occur where specific database requirements have to be modified slightly to meet the image generator's capabilities and still provide an effective simulation.

3.0 SPACE DATABASES AND TERRAIN DATABASES

Terrain databases are usually designed at a 1:1 scale with the terrain they model and only represent a relatively small portion of the real world. A space database, on the other hand, because of the sheer size of the model environment, cannot possibly model all of its objects at a 1:1 scale. For example, one of the requirements of the NASA space databases was to create an accurate starfield. The star nearest Earth, Alpha Centauri, is approximately 4.4 light-years away. Hence, some models must be created so that they appear visually correct even though they may be mathematically incorrect.

A space database has the potential of depicting its modeled environment more realistically than a terrain database can because many of the models included in it — the Orbiter, the Space Station, the Shuttle Remote Manipulator System (SRMS), and payloads, for example — are manmade and tend to be geometrically regular. Computer graphics is an excellent medium for rendering such regularities. Natural objects such as Earth's surface and the moon can be easily and effectively modeled with photo-derived-texture patterns.

Space databases generally include less information than terrain databases. Most terrain databases cover many hundreds of square nautical miles of terrain and contain enough information to create scenes that include bushes, trees, rocks, rivers, roads, cities, and farms. But a space database must contain only a few hundred accurate stars, a brilliant sun, a realistic Earth, a life-like moon, and a few extremely high-fidelity models to create an uncanny likeness of the real thing. Figure 1 is a photograph taken of an actual display that illustrates this realism.

The space databases designed for NASA contained all these models and used only approximately one fiftieth of the file space required for a typical terrain database. Although the space databases were small with respect to their storage requirements, their visual effectiveness equalled that of any terrain database designed to date.

4.0 ORGANIZATION OF A SPACE DATABASE

A visual database is a composite data structure in the form of a tree that specifies the hierarchical relationships among the items in the database. A space database has the same format as any other type of visual database, but the design and organization of a space database are quite different from those of a terrain database.

4.1 Static and Dynamic Models

The models that compose a space database can be categorized as *static* (remaining fixed at a specified origin) or *dynamic* (having independent motion capabilities).

Figure 1. Photograph of a Displayed Space Database

Depending on the simulator's mission and other factors, models may switch categories. Since everything in space is dynamic, it might initially seem easiest and most logical to make all the models in a space database dynamic. However, the image generator accommodates only a finite number of dynamic coordinate systems and may not (depending on its load) be able to process all those systems at once. A static model requires very little processing time for its geometry to be displayed; a dynamic model requires additional processing time, depending upon the type of motion specified. Therefore, the number of dynamic models should generally be restricted as much as possible in order to limit the amount of time used to process them.

Depending on the particular scenario's requirements, any or all of the models of Earth, the sun, the moon, stars, or other items in the database could be static and at the same time serve in an effective simulation. For example, in a given scenario designed only to train specialists in the use of the SRMS, if the Orbiter never needed to move, it could remain static at the database's global origin. A static orbiter would have a fixed orientation with respect to the other database models, but this fact might not limit the effectiveness of the images produced for the scenario in any significant way.

The number of dynamic coordinate systems and the corresponding amount of processing time constrain the number of dynamic models that can appear in a given scenario. However, useful scenarios require

that many objects (the Orbiter, the SRMS, and payloads in the space databases, for instance) be categorized as dynamic. Therefore, the database engineers must decide which models have to be dynamic for each scenario. Determining a finite list of dynamic models for a given scenario is not difficult, but if the dynamic models required for one scenario are different from the ones required for others, the complexity of the task increases significantly. When the set of scenarios requires more dynamic models than the image generator can process, an alternative must be found. The approach taken in designing and developing the space databases for NASA entailed two strategies: design of each database as a *scenario-dependent database* and a technique for choosing *model selects*.

4.2 Scenario-dependent Database

A scenario-dependent datatabase is a database that is broken into smaller, simpler pieces. Evans & Sutherland CT6 linked databases contain between 1 and 60 *entry-point sets*. Each entry-point set is independent of all the other sets and consists of a specific collection of related database-tree structures. Each entry-point set can have its own unique tree structure, and therefore each can represent a given scenario with an independent database design. Separate linked databases could be used in the same manner as entry-point sets within a database. A scenario-dependent database provides great flexibility. Since each entry-point set (or separate database) has its own tree structure, each can have its own attendant dynamic and static models.

4.3 Model Selects

The databases were organized so that the viewpoint processor makes a series of hierarchically structured choices, or model selects, that determine the requirements for any given scenario. A CT6 image generator allows 128 different model selects for each dynamic coordinate system and has the unique capability of performing model selects of dynamic coordinate systems in real time. This feature is often used for animation. Because the top-level structure is designed so that there are several mutually exclusive model selects, when the viewpoint processor starts its trace of the database tree it also starts to determine the current scenario's requirements based on the model selects chosen for that field by the simulator-host-computer program. Each of the top-level model selects points to another structure that in turn is attached to a dynamic coordinate system with multiple model selects. The viewpoint processor continues this tracing of nested model selects until it incorporates all the specified scenario requirements.

5.0 DESIGN CONSIDERATIONS

Once the organization of a space database is determined, other aspects of the design can be considered. Some of the main issues and challenges encountered in the design of the space databases, the resolutions of the challenges, and the ramifications of the decisions made concern the database's coordinate system, database units, the construction and placement of models, the uses of texture, collision detection, the simulation of closed-circuit-television systems, and the overall modularity and flexibility of the databases.

5.1 Coordinate System

NASA's space databases were designed with an orthogonal Cartesian coordinate system. If the simulator-host-computer program were to use a different coordinate system than the image generator does (a celestial coordinate system, for example), all motions controlled by the simulator host computer would have to be transformed to map that coordinate system into the image generator's coordinate system.

All of the NASA models were constructed about or relative to the origin of the image generator's coordinate system. However, the placement of the origin, the database unit of measure, and the overall size of the database affected the design of the individual models.

The most likely location for the database's global origin is the center of geometry (or center of mass) of the main scenario model. The Orbiter's origin was chosen as the global origin of the database that included the Orbiter, SRMS, and Space Telescope models. Each of these models was easily defined relative to the origin of the Orbiter, and every scenario maintained the Orbiter as a static model at the global origin. However, this origin was not appropriate for the other two NASA databases, and therefore other origins were selected.

5.2 Database Units

The size or volume of a database is limited by the highest number the image generator allows and by the chosen unit of measure. A CT6 image generator does not use an inherent unit of measure as a database unit; it simply interprets all database values as numbers with a maximum and minimum range. Any unit of measure can be used to create a database as long as the relative scale between database models is maintained. The most common database unit for terrain databases is the foot. The database unit chosen for the space databases was the inch.

5.3 Models

The models used in a space simulator must be extremely accurate. They are the only visual cues in the database because black space, unlike terrain, does not provide much in the way of feedback. The models are also generally viewed at a much closer range than models in a terrain database, so they need to include

as much of the geometry of the real-world object they represent as the image generator can process. Most scenarios require extreme accuracy in motion, collision detection, and interaction, so the models must provide all of the cues needed to make a given scenario effective.

Each model in the space databases was constructed with information derived mostly from detailed engineering drawings, but because many of the actual objects had not yet been constructed, the models were continually subject to reevaluation. The Orbiter was the only item that had been constructed before development of the databases began. Because such high fidelity was required of the models, the Orbiter model geometry was created through a process that involved automatically scanning a scaled replica, digitizing hundreds of vertices, and defining the polygonal boundaries. Although the process was involved, it was very efficient and quickly accomplished, and the end result was extremely accurate.

Since the chosen database unit and the maximum value allowed by the image generator limited the overall size of the database, some of the database models could not be constructed at a 1:1 scale. Instead, they had to be created at other scales and then positioned to appear their correct sizes and to maintain the integrity of the simulation.

One of the requirements specified for the space databases was an accurate starfield. The inclusion of stars of various intensities and locations in a sea of black greatly enhances the illusion of space. Space without stars is about as useful as unembellished terrain is for a flight simulator. Digistar, a star-projection system developed at Evans & Sutherland for planetariums, provided the information used to create the starfield for the space databases.

The starfield model has more than 1600 unique stars that appear to be in their correct locations. The model is a sphere consisting of light points; the center of the sphere is at the global origin of the database. The sphere was scaled to fit within the allowable database size and each light point was assigned an intensity that would simulate the apparent magnitude of the star it modeled. Because many constellations can be recognized in this starfield model, it could easily be used for directional information.

Two static models and dynamic-texture motion were used to create the illusion of an orbiting satellite and a rotating Earth. The database unit coupled with the maximum size of the database determined that a full-sized Earth could not be modeled. Therefore, a scaled model of Earth was created and placed artificially close to the orbiting vehicles. This strategy effectively created the illusion of an accurate orbit and met the requirements of the application.

To complete the illusion of Earth as seen from an orbiting satellite, photo-derived dynamic texture was applied to the Earth model and given a directional velocity that approximates the speed at which a satellite such as the Orbiter passes over a given point on Earth's surface. So the simulated orbit is in essence the opposite of what happens in the real world: the orbital vehicle is static and Earth moves by below it. This effect was possible because the CT6 image generator has the ability to create real-time texture motion.

Because Earth, in this case, could be built as a static model, it was modeled as a disk rather than as a sphere. Modeling Earth as a disk makes it difficult to simulate its dark side, but this disadvantage is offset by the fact that valuable dynamic coordinate systems and viewpoint-processor time are preserved for other uses. However, the same illusionary techniques could be used to construct a spherical model of Earth.

Since there is no atmosphere to interfere with light in space, colors are more vibrant than on Earth and there is a crisp contrast between light and dark. Therefore, color assignments, usage, and tuning are important design factors. Since the visual-system displays cannot reproduce the contrasts found in space exactly, relative brightnesses must be assigned to certain models. No model should be colored as black as the blackness of space, for example.

5.4 Texture

2D texture can realistically enhance computer-generated imagery and has the great advantage of generally not affecting the processing load. Because texture in a CT6 image generator is processed in parallel with polygons, a scene consisting of the maximum number of processable polygons can be enhanced by the addition of texture and still not overload the image generator.

Since the fidelity of the models was an important criterion of the space-database requirements, texture was often used to effect visual cues without increasing the number of polygons. These applications of texture significantly enhanced the Earth and Orbiter models.

The entire surface of Earth is simulated by a photo-derived, self-repeating texture map applied to a group of polygons. The Earth model, because of its special texture application, maintains a slim polygon budget and is therefore a very efficient model.

The Orbiter was also modeled with texture. The insignia on the outside surfaces were created with texture. The American flag, all thirteen red and white stripes and all 50 stars on a blue background, was created with one texture map and four polygons. Photo-derived, self-repeating patterns adorn the inner bay with a thermal wrap pattern and the outer skin with tiles.

All of the additional visual cues created with texture allow the image generator's polygon capacity to be spent where texture cannot be used, as for the hundreds of truss polygons on the Space Station.

Simple 2D polygons with texture applied to them can simulate complex 3D structures. The most notable of these illusions is the texture application that simulates the inside volume of the End Effector (EE) with just one polygon. The EE is essentially a hollow cylinder whose inside portion is seen from many eyepoints. Since the use of polygons on an inside surface was considered wasteful, a special texture pattern was applied in a plane other than that of the polygon that capped the end of the EE. The resulting parallax created an illusion of depth that is almost indistinguishable from that of an actual modeled volume. Noncoplanar texture is another robust capability of the CT6 image generator without which this illusion could not have been created.

5.5 Collision Detection

One of the most important functions of a space simulator is the detection of collisions between various objects. The collision-detection function can be used for purposes other than detecting when two models collide, however. For example, the complex simulator task of grappling a payload with the SRMS would be virtually impossible without the aid of collision detection. An appropriate viewpoint-processor-timing budget must include enough time for all of the collision-detection requirements.

Collision detection is another reason for constructing accurate models. The database must reproduce the real world in order for the simulation to be effective. Otherwise, collision detection might not be reliable.

5.6 Closed-circuit Television

In an actual space mission, a significant portion of the scenes outside the Orbiter are viewed through closed-circuit-television (CCTV) systems. These sophisticated systems have a built-in process called automatic level control (ALC) that adjusts the CCTV according to the light sources it receives. Remotely similar to the light meter and *f*-stop combination on a camera, ALC is an automatic process used to improve and adjust image quality.

One of the requirements of the space databases was to simulate the effects of ALC. The simulation is done through detection of light from its various sources (sun, moon, Earth, and floodlights) and concomitant adjustment of the scene illumination. Unique structures had to be created on each of the light-source models. The image-generator function used these unique structures to detect when the light-source models came into view. The real-time system would notify the simulator-host-computer program when the models came into view, and the simulator-host-computer program would then adjust the illumination of that channel. Without the characteristics of the CCTV system and ALC, every scene would appear as though it were an out-the-window view and the simulation would not be accurate.

5.7 Modularity and Flexibility

The structure of a space database must be modular and flexible because the database will probably be used for many applications over a wide range of projects. Therefore, each model must be designed, organized, and documented so that another engineer can make modifications to it in the future. Furthermore, the top-level structure must be clean, concise, and simple, with adequate and descriptive documentation. This is especially important for scenario-dependent databases or fully compatible model-select databases. The criteria of modularity and flexibility are arguments for simple databases over extremely large, complex, all-encompassing databases.

6.0 CONCLUSION

The design and development of a space database is a very challenging and rewarding experience. Although all real-time visual databases are hierarchically-ordered groups of geometric data, the differences between familiar terrain databases and space databases are significant. The organization of a space database is substantially dependent upon the simulation and should meet the requirements of specified scenarios. Hence, the number of and the relationships among the static and dynamic models are crucial to a successful design. More dynamic models and better image-generator performance result from a well-organized database design. The scenario requirements and all of the database requirements must be taken into account when the visual system's resources are budgeted so that the most processing-efficient and visually-effective database possible is designed.

Furthermore, the potential for improved scene realism is generally greater for a space database because of the ability of computer-generated imagery as a medium to depict regular geometric items and because of the visual quality of photo-derived-texture applications. Highly efficient and accurate models are essential not only for visual cueing but for supporting other visual-system functions such as collision detection. Finally, well-engineered database source code and its accompanying documentation provide service and support for many future project applications.

7.0 ACKNOWLEDGEMENTS

A considerable amount of research was involved in creating the NASA space databases. The entire development team, consisting of NASA and Evans & Sutherland personnel, participated in these innovative designs and thereby contributed greatly to the content of this paper. Specifically, Mike Bartholomew, David C. Christianson (NASA), Mercedes DeLugo, Ralph Howes, Michael D. Jackson, Daniel Lunt, Janice Poulson, and James R. Smith (NASA) deserve special recognition for the information, guidance, and assistance they provided. Curtis G. Booth rendered invaluable technical and professional services in the preparation and organization of this document.

Tools for 3D Scientific Visualization in Computational Aerodynamics

Gordon Bancroft ,Todd Plessel, and Fergus Merritt (Sterling Software Inc.), Val Watson (NASA Ames)

NASA/Ames
Workstation Applications Office - Code RFW
Mail Stop 258-2, Moffett Field, California 94035
bancroft@amelia.nas.nasa.gov
(415) 694-4052

ABSTRACT

The purpose of this paper is to describe the tools and techniques in use at the NASA Ames Research Center for performing visualization of computational aerodynamics, for example visualization of flow fields from computer simulations of fluid dynamics about vehicles such as the Space Shuttle.

The hardware used for visualization is a high-performance graphics workstation connected to a super computer with a high speed channel. At present, the workstation is a Silicon Graphics IRIS 3130, the supercomputer is a CRAY2, and the high speed channel is a hyperchannel.

The three techniques used for visualization are post-processing, tracking, and steering. *Post-processing* analysis is done after the simulation. *Tracking* analysis is done during a simulation but is not interactive, whereas *steering* analysis involves modifying the simulation interactively during the simulation. Using *post-processing* methods, a flow simulation is executed on a supercomputer and, after the simulation is complete, the results of the simulation are processed for viewing. This is by far the most commonly used method for visualization of computational aerodynamics. The next two methods are much more desirable, yet much less common given the current state of supercomputer and workstation evolution and performance. Both of these are more sophisticated methods because they involve analysis of the flow codes as they evolve. *Tracking* refers to a flow code producing displays that give a scientist some indication how his experiment is progressing so he could, perhaps, change some parameters and then restart it. *Steering* refers to actually interacting with the flow codes during execution by changing flow code parameters. (Steering methods have been employed for grid generation pre-processing as well to substantially reduce the time it takes to construct a grid for input to a flow solver). When the results of the simulation are processed for viewing by distributing the process between the workstation and the supercomputer, it is called distributed processing.

This paper describes the software in use and under development at NASA Ames Research Center for performing these types of tasks in computational aerodynamics. Workstation performance issues, benchmarking, and high-performance networks for this purpose are also discussed as well as descriptions of other hardware for digital video and film recording.

A new software environment, FAST, is introduced that is currently being developed at NASA Ames for implementation on workstations that will be procured in the latter half of 1989. This modular software environment will take advantage of the multiple processor and large memory configurations and other features as specified in the NASA RFP for these workstations and is a natural evolution of the techniques described in this paper.

1. INTRODUCTION

Using computational aerodynamics, scientists are now able to model complex fluid mechanics problems using supercomputers and new numerical algorithms. To gain a better understanding of these complex flow fields, scientists use high-performance computer graphics workstations to view and in some cases animate these simulations. This paper describes this application, the hardware, the software, and the techniques used by the Fluid Dynamics Division of the NASA Ames Research Center.

2. DESCRIPTION OF THE APPLICATION

The simulations involve visualizing flow field solutions generated on supercomputers. The raw data from these simulations consists of density, momentum vector, and total energy per unit volume specified at each grid point in the computational domain. A typical computational domain may contain 1 million grid points. This raw data must be converted to a scene depicting the physics in a manner the scientist can easily interpret. Color and visual cues (shading, animation, etc.) are used to demonstrate the physics of the particular result. PLOT3D, GAS, SURF, RIP and a new software environment FAST (currently under development) are visualization tools described further in this paper.

3. VISUALIZATION REQUIREMENTS

The views of the simulation portrayed by the computer graphic workstations must be 3D because visualization of the inter-related flows of all three dimensions simultaneously is important. The displays must be dynamic in order for the time-variant features of the flow fields to be understood. Although the motion need not be real time, the motions must be rapid enough to gain a proper understanding of the dynamic features of the flow. The flow fields typically have a large range of scales; therefore, the scientist must be able to zoom into a region of small scale features and zoom back out to view the overall flow field. Furthermore, the displays should be high definition to contain adequate detail at all scales. The displays should simultaneously contain solid body objects, such as an aircraft (with hidden surfaces removed), and points or lines (such as lines representing the paths of tracer particles inserted into the flow field). As the displays evolve in time illustrating the flow dynamics (e.g., the movement of tracer particles) the viewing position must be simultaneously changeable in real time (as the flow is evolving) in order to maintain the best view or to get a different perspective. Dynamic change of the viewing position is one of the best cues for enhancing the 3D aspects of the display. In addition, new visualization effects such as ribbon traces, smoke, shading of function mapped parts, anti-aliasing, variable transparency, volume visualization and stereo are being requested by the scientists studying the flow fields.

4. OVERALL APPROACH

At the current time, no workstations costing less than approximately $100K have been available that can meet the requirements described above for dynamic viewing of complex solids embedded in flow fields. Therefore, the approach has been to obtain workstations with the highest performance available at the time of the procurement, and to augment these workstations with equipment for recording on video tape and 16mm film to permit dynamic viewing of complex scenes that could not be viewed dynamically on the workstations.

The next generation workstation is expected to be procured in approximately the third quarter of 1989. The performance of the Silicon Graphics 4D/240 GTX is given in the table below, and it is the approximate expected performance of the next generation workstation. These workstations are expected to meet most of the requirements for dynamic viewing listed above. A more complete description of the features expected in the next generation workstation is given in reference 2. Phong lighting, material maps, alpha blending, and a windowing system in a parallel programming environment are additional features of this next generation workstation.

Benchmark software has been developed at NASA/Ames to test, among other things, what kind of

Table 1: Features of Current and (typical) Next Generation Workstations

Feature	IRIS 3130	IRIS 4D/240 GTX
	CPU	
CPU performance MHz	1 MC 68020 0.1 MFLOPS/16MHz	4 x R3000 (RISC) 16 MFLOPS/25
FPU performance	1 MC 68881	4 x R3010 (RISC)
RAM	16MB	128MB
disk storage	474 MB	9.6 GB
Computations	0.1 MFLOPS	40 MFLOPS
	GRAPHICS	
Resolution	1024 x 768	1280 x 1024
Image memory	24 bitplanes	48 bitplanes (+overlay,alpha)
Z-buffer	12 bits	24 bits
Pixel rate	1,000,000 pixels/sec	8,000,000 pixels/sec
3D coordinate transformations	80 K/sec	400 K/sec
polygon transformation*	16 K/sec (flat, not z-buffered)	100 K/sec (Gouraud, lighted, and z-buffered)

(*polygons are 400 pixel quadrilaterals)

graphics performance can be expected from these next generation workstations (Note: the numbers quoted in the table above are NOT measured with these benchmarks, but are published numbers from Silicon Graphics Inc.) The graphics capabilities emphasized by the benchmark include color, simultaneous vector and polygon display, double buffering (ref. 7, p.84), hidden surface removal, smooth shaded polygons and coordinate transformation rates. The benchmarks also test display list operation (creating an object in a application program and displaying it) and frame buffer performance. This software can be obtained through the authors from NASA/Ames Research Center.

5. HARDWARE CONFIGURATION

The hardware configuration is shown in figures 1, 2, and 3. Figure 1 shows the hardware configuration for creating and viewing flow field solutions. Figure 2 specifies the hardware for creating video tapes, and figure 3 specifies the hardware for creating 16mm film.

The calculations to generate the flow field solutions are done on the supercomputer. The conversion from these solutions on 3D grid points to scenes depicting the physics (e.g. particle traces about the body) can be done in three ways. The first way is to transfer the whole solution file (containing the solution at each grid point) to the large disk on the workstation and generate the scene on the workstation. The second way is to produce graphics files on the supercomputer (and transfer these graphics files to the workstation). The software for creating and viewing scenes using these two methods is described below in the **software** section. The third way is to create the scene using the supercomputer interactively while viewing the scene on the workstation. The software for this method includes tasks that run simultaneously on both the supercomputer and the workstation. This concept involves separating the computationally intensive portion of the processing on the supercomputer from the graphics on the workstation and having the two processes communicate over a high speed network. One scenario involves sending pre-computed display list (ref. 7 p 348) information to the workstation using a remote graphics library developed for just such a purpose (this graphics library allows a graphics program to be implemented on a supercomputer). Other scenarios involve more standard networking schemes, where subroutine and/or interprocess communication are utilized. The bottleneck in those types of schemes can often be the large amount of data that has to be transferred from one computer to another. Existing software and techniques being utilized at NASA/Ames Research Center are described further in references 3, 5 and 6.

The key features of the workstation are its rapid 3D transformation speed (for changing the viewing position), its high definition display, and its rapid display creation speed. 3D coordinates can be transformed at a rate of 80,000 coordinates per second. The display has high spatial resolution (1024 pixels horizontally by 768 pixels vertically) and high color resolution (24 color planes giving more than 16 million simultaneous colors). (The color planes can be divided into two buffers with 12 color planes each to obtain the double buffering required for most dynamical displays. This reduces the number of simultaneous colors to 4096.) Displays with a very simple solid object and thousands of lines or points can be generated at a rate of more than 10 per second — a rate that provides satisfactory motion for understanding dynamics. The Space Shuttle illustrated in

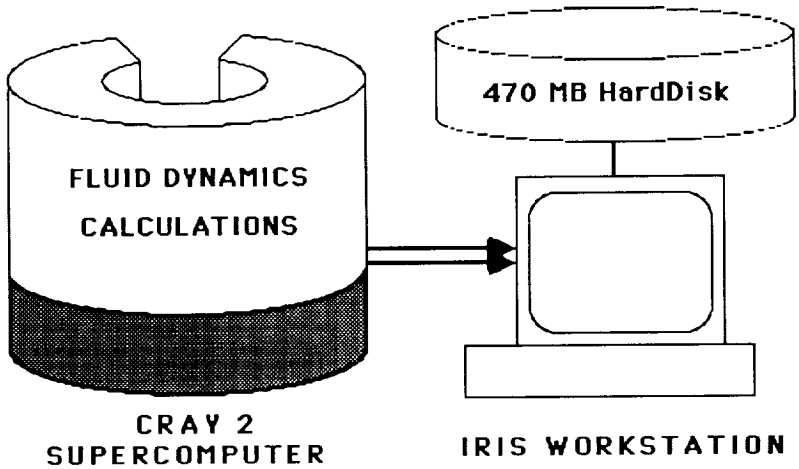


Figure 1. Hardware configuration for creating and viewing flow field solutions

the video tape represents a maximum display complexity for studying dynamics directly with the workstation, as the time to create each display (frame for film) is approximately 1/2 second — a rate that is marginal for viewing dynamic motion. For this display, the Space Shuttle is represented by approximately 8000 polygons (ref. 7 p. 87) and the painter's algorithm was used for hidden surface removal.

The workstation contains a Z buffer (ref. 7, p. 560) for hardware implementation of hidden surface removal. In addition, the Gouraud shading (ref. 7, p.498) calculations get an assist from the workstation hardware. However, many seconds are required to create displays of typical aerodynamic vehicles if the Z buffer and Gouraud shading are used. Therefore, these displays must be recorded on video tape or 16mm movie to view the dynamics satisfactorily.

The hardware used to record the displays on video disk is shown in figure 2. The high definition display is digitally sampled by a scan converter to a lower resolution RS170a format that can be encoded by the encoder into the standard single NTSC (National Television Standards Committee) signal used by standard video recorders and players. (The loss in spatial and color resolution during this conversion is described later in the section "**Discussion**".) A time base corrector must be inserted into this system prior to the 1" video recorder to generate the precision signal timing required for "broadcast" quality signals (necessary for broadcasting over the air). The Abekas A62 video disk recorder is controlled via a standard RS232 interface. As each frame is displayed, control information tells the Abekas to record. It then stores the frame as digital NTSC. This process occurs at standard video rates; that is the digital video system

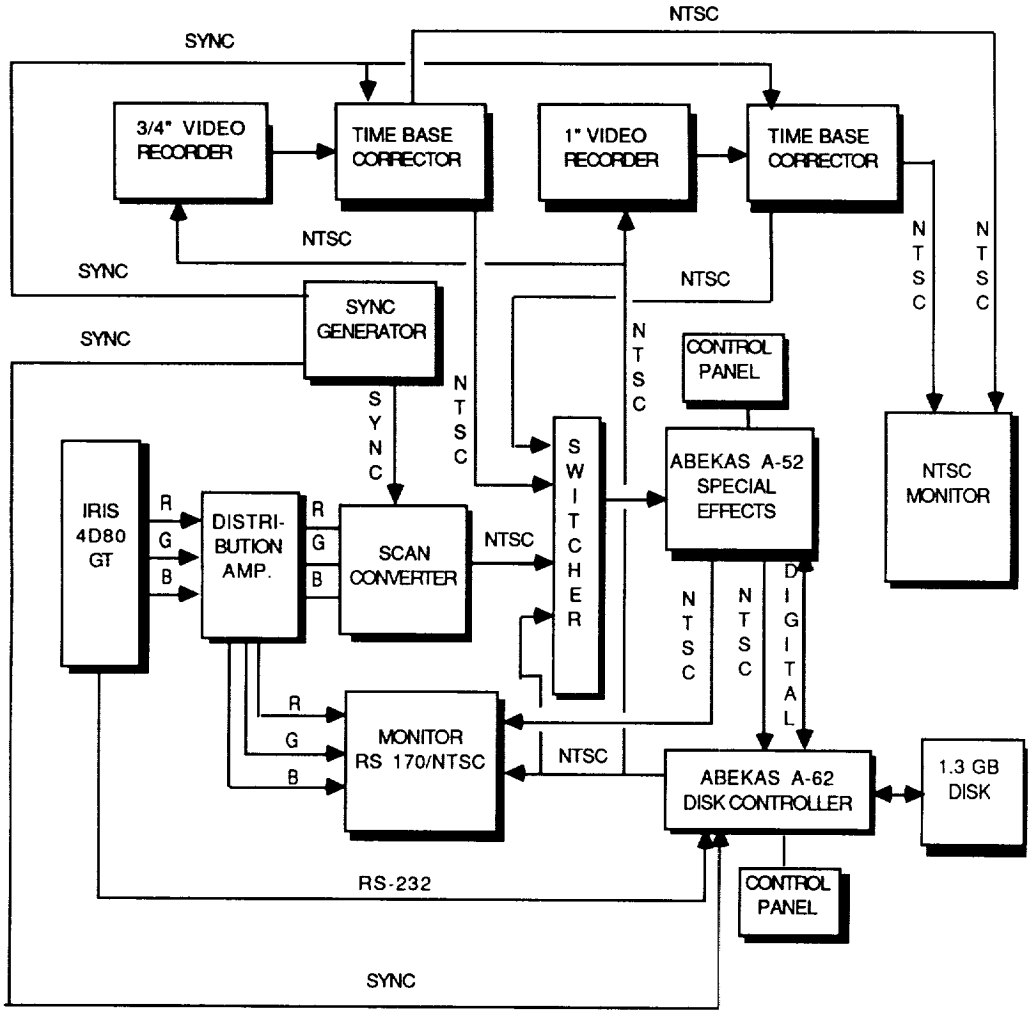


Figure 2. Hardware configuration for digital video disk recording

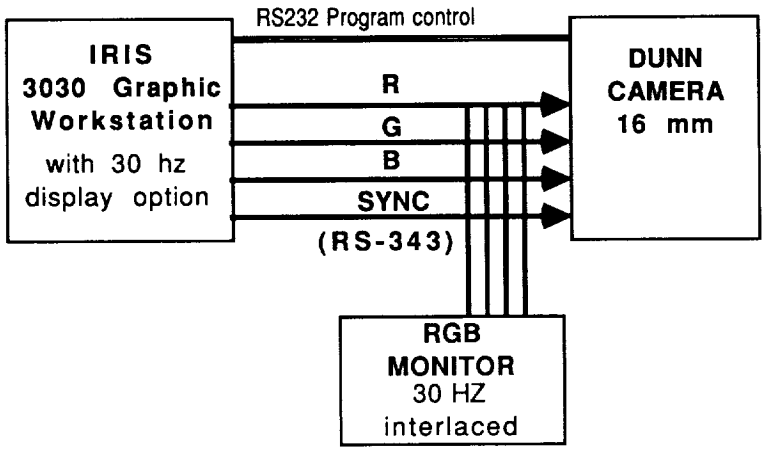


Figure 3. Hardware configuration for 16mm film recording

has the capability to record analog NTSC at real-time rates so the time required to record a computer workstation frame is limited by the time it takes to render it. The workstation then continues on with the next frame, and repeats the process until the animation is complete. The Abekas uses Winchester disk technology (1.3 gigabytes storing 100 seconds of video), allowing stored video to be edited (using the A52 special effects) or the disks to be re-recorded. There are no generation losses within the system due to the digital formatting.

The hardware for recording the displays on 16mm film is shown in figure 3. The Dunn Camera is controlled from the workstation using an RS232 hardware connection and the GAS software described later in this paper.

6. SOFTWARE

The three techniques used for visualization are *post-processing, tracking, and steering*. Using post-processing methods, a flow simulation is executed on a supercomputer and, after the simulation is complete, the results of the simulation are processed for viewing. This technique is by far the most common for visualization of computational aerodynamics, given existing computing resources. The following are examples of post-processing software packages in use at the NASA Ames Research Center:

PLOT3D accepts as input the flow field solutions from the supercomputer and creates as output a variety of displays that can be viewed dynamically with the workstations (or statically from other graphical display devices). The software makes extensive use of color and 3D cues (such as shading and perspective: ref. 7, p. 269). A very popular display is path lines of particles released at selected points inside the flow field. An example of particle paths in the flow field is shown in figure 4. A second example of displays from PLOT3D is color mapping on a vehicle surface representing the magnitude of some scalar property on the surface, such as pressure. A third example is a shock surface within the flow field (or some other surface of constant scalar value) represented as a partially transparent surface so the vehicle creating the shock can be seen through the shock. PLOT3D software can be run on the workstations, the Cray supercomputers, and on a VAX 11/780 minicomputer.

SURF(Surface Modeller) allows scientists to input grid and solution files and interactively build a 3D model consisting of wireframe, shaded, and function mapped parts. These parts can be interactively viewed, edited, and output to ARCGRAPH files which can then be loaded into GAS and then animated. SURF has a mouse driven interface (similar to GAS). Gouraud shaded parts can have their color and specular highlighting adjusted interactively. Shaded parts are created based on user specified lightsources (up to 20), a viewpoint, and an ambient light level. The function mapped parts can also have their color spectrum adjusted interactively. Legends can be created to show the correlation of color and normalized function values. Also, function mapped parts can be "clipped" so that they only show areas within a specified range of function values (e.g. normalized pressure between 1 and 2). SURF computes the following functions: pressure, density, temperature, Mach Number, and custom (user defined) functions.

GAS(Graphical Animation System) permits the scientist to interactively and dynamically view the 3D displays created by PLOT3D (or several other graphical packages) while simultaneously changing the viewing position within the 3D space. In addition,

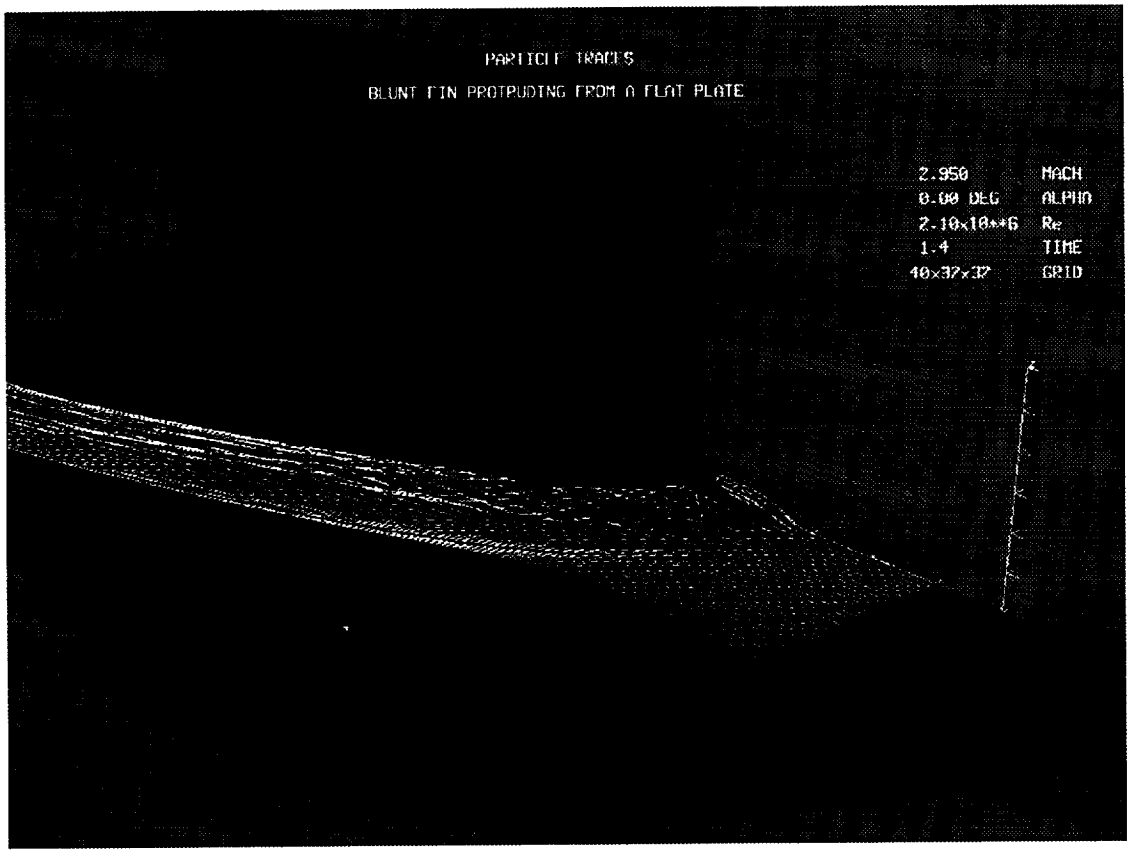


Figure 4. Example of figure created with PLOT3D.

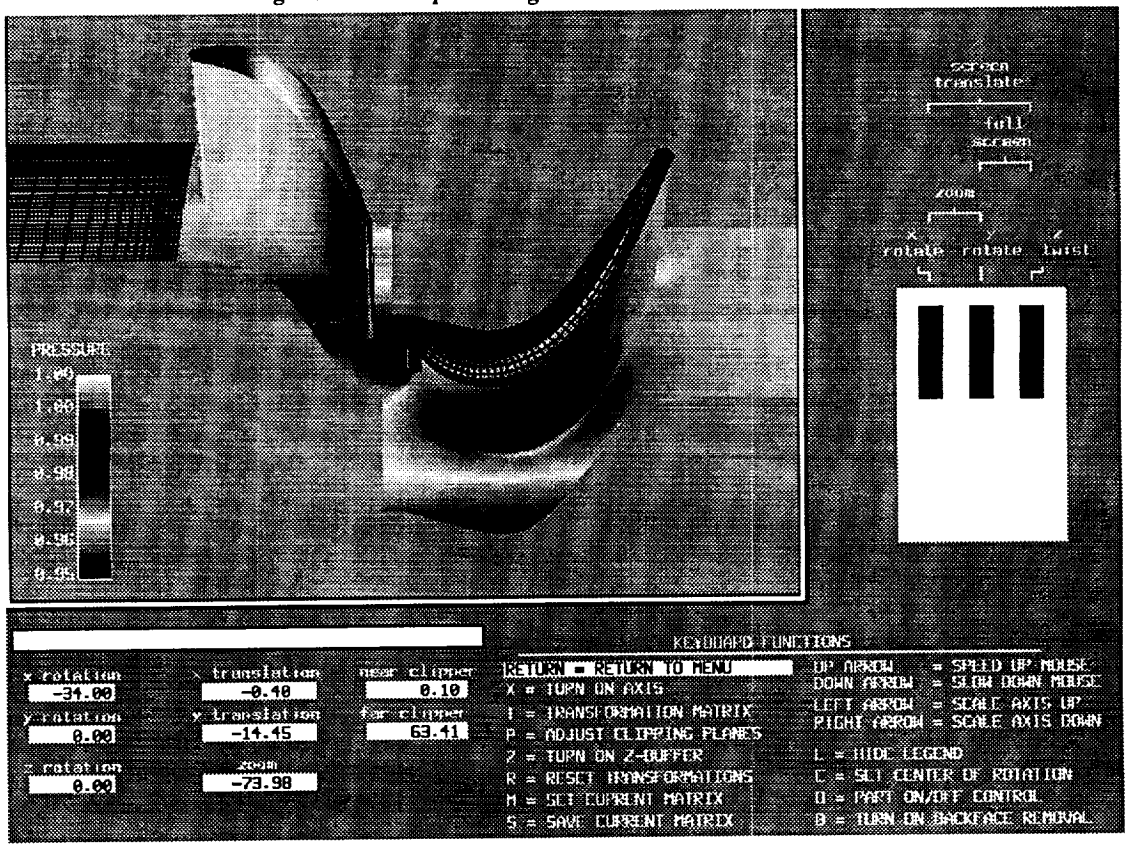


Figure 5. Example of figure created with SURF.

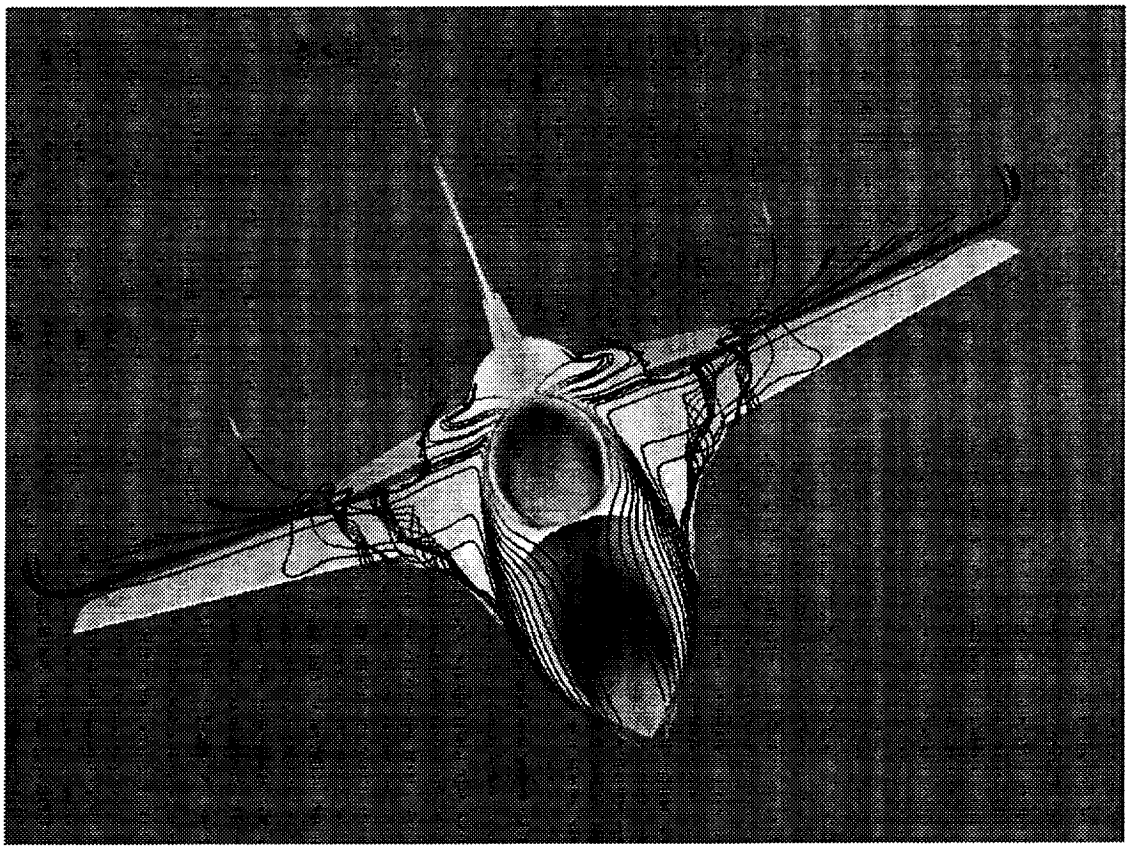


Figure 6. Example of figure created with GAS.

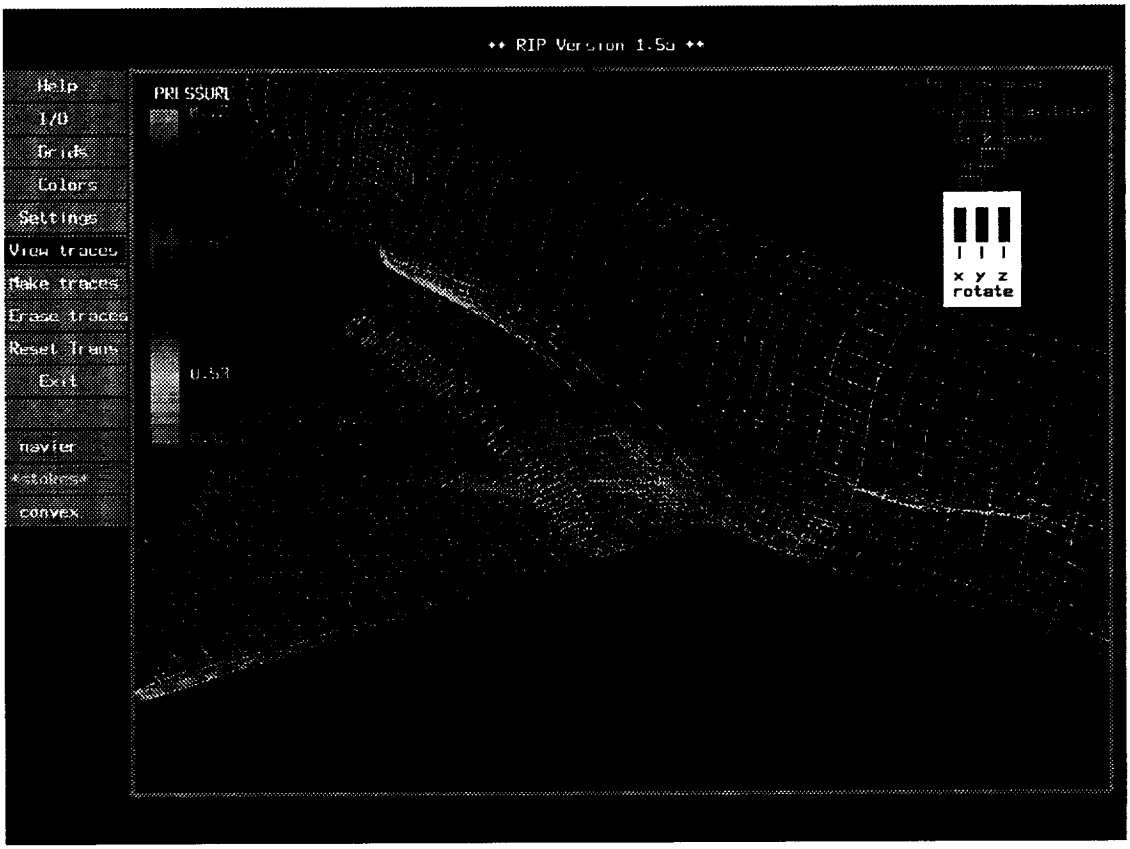


Figure 7. Example of figure created with RIP.

it permits the scientist to generate an animation sequence with smooth 3D transitions between a series of specified positions. Both the animation speed and the number of "tweening" steps (automatically added to give smooth transition between specified positions) are under user control. Titles can be inserted, and the resulting "movie" can automatically be recorded by the video equipment or the 16mm film recorder which are under control of the GAS software. This software is device specific and runs only on a Silicon Graphics IRIS Workstation. It was written in the C programming language under the UNIX operating system.

FAST(Flow Analysis Software Toolkit) is a new proposed standard fluid-dynamics graphics environment. The purpose of FAST is to provide the scientist with a single software environment for handling many graphics needs (some functionality exists in the programs described above) in a way that is quick, powerful, and easy to use. The programs above were designed and built for the current Silicon Graphics IRIS 3130 workstation, whereas the FAST environment is being designed for the capabilities of the next generation workstation (see Table 1.). The new capabilities of these machines warrant a new approach to building graphics tools. The goal is to allow a scientist to quickly and easily perform fluid dynamics scientific visualization from this environment. Initial software features include (1) a standardized user interface, (2) data sharing, communication and memory management between modules, (3) high quality rendering and advanced animating capabilities, (4) new ways for viewing and interpreting fluid dynamics. The five initial modules are (1) the main FAST module to load and unload the other modules and manage data structures, (2) The MODELLER module to read grid and solution data and create models, (3) The FLOW TRACER module for illustrating the flow field in a variety of ways (tracers, ribbons, smoke), (4) The TITLER module for titling and labelling, and (5) The ANIMATOR for advanced animation and recording.

RIP(Real-time Interactive Particle-Tracer) is an example of a distributed graphics tool and actually consists of two programs that communicate over a high-speed network. One program computes the flow traces from raw data on a supercomputer and the other program renders these traces for interactive viewing on a workstation. Particle tracing is then interactive, where a scientist selects a trace or rake of traces for display and the traces are computed and then drawn in most cases almost instantaneously, much like a smoke wand in a wind tunnel.

There are other codes in use at NASA Ames that employ tracking and steering methods, although these codes are typically in more prototype use than part of day-to-day simulation efforts. Versions of **ARC2D** (Ames Research Center 2-d flow solver), a code in use at NASA Ames, exist that "track' the progress of a simulation. Simple examples of 'steering' a flow code exist as part of the interactive grid generation program **IZ** (Interactive Zoner). In this example of 'steering', you can generate a grid and then run a flow solver on it using the distributed graphics techniques discussed earlier. This example is only 2-d because workstations have not had the resources (until recently) to allow 'steering' a 3-dimensional flow solver.

7. DISCUSSION

Of the three visualization methods discussed in this paper (post-processing, tracking, and steering), post-processing is by far the most common. Current supercomputer and workstation performance make this the most practical method for viewing solutions of computational aerodynamic solutions. Probably 90% of all simulation is performed in this manner, with the remaining 10% made up of scientists using tracking codes, and, to an even lesser extent, scientists using steering codes.

Post-processing techniques include (1) dynamic, interactive viewing on the workstation, (2) recording and playback on video disk and then to tape, and (3) recording and playback on 16-mm film. These techniques have greatly improved the ability of scientists at NASA Ames to conduct fluid dynamics research, although these techniques necessarily mean a loss of interactivity, take a long time to record, and, for video, mean a loss of spatial and color resolution.

With direct viewing on the workstation, the capability to interactively manipulate the viewing position and the animation sequence was found to be very effective in providing a quicker and more complete understanding of the flow field solutions. This capability is lost if the displays are so complex that they must be recorded for playback. A solution for this problem is to increase the performance of the workstation. As mentioned earlier, the display creation speeds of workstations are projected to increase an order of magnitude over the next year. This will permit many complex displays that now must be recorded for satisfactory motion analysis to be viewed directly on these newer workstations. Nevertheless, there will still be displays that are too complex to view with adequate rates of motion on the new workstations;

recording will still be required for these displays. (In addition, recordings are required for group presentations.) Therefore, it is important to improve the recording techniques also.

Recording on the Abekas video disk or Dunn film recorder with the hardware shown in figures 2 and 3 requires a much longer time than simple viewing on the workstation — a typical recording time is 1/2 to 1 hour for every one minute of playback time (based on 30 frames per second video playback and 24 frames per second film playback). The film medium takes longer due to the nature of the recording process. The Dunn film recording system requires cycling of red, green, and blue filters for each frame (or exposure). The Abekas system records each frame essentially instantaneously, so the length of recording time is determined by the time it takes to render each frame, which is determined by the rendering techniques being used for the simulation (1 sec to 2 minutes).

Recording on video disk also causes a loss in picture quality (a loss in picture definition and shifting in colors). The initial spatial resolution must be cut nearly in half (down to 512 x 512) for the conversion to RS170a RGB format, and the further encoding to the single composite video signal (NTSC) causes another substantial reduction in quality. Analog recorders that rewind and pre-roll also cause some loss in quality. The digital video system mentioned above provides a partial solution to the loss in picture quality. There is no loss of resolution or shifting of colors in the editing because the pictures are stored digitally. The loss of quality during recording is also reduced by using continuous recording rather than a frame at a time and by using the larger 1" tape format rather than the 3/4" tape format used in older analog recording systems. The capability to record individual "fields" of video is also an important feature of the digital process. Animation sequences can be separated by fields (instead of by full frames of video). The effect on playback is very smooth motion, as the eye cannot detect or distinguish between these fields. This technique is borrowed from commercial television computer graphics applications where it is used often.

Recording on video disk and tape could be substantially improved with the addition of real-time digital video output from a workstation frame buffer. Certain digital video component manufacturers are already standardizing on the D2 (Sony, Ampex) composite digital video format and, although many workstation manufacturers have discussed such an option, the authors are not aware of it being available at the time of this writing. Not only would this option eliminate the need for much of the outboard equipment necessary for video recording, it could potentially improve the video quality by eliminating numerical sampling error going from digital to analog and back to digital again.

Recording on film requires a long time primarily because film processing at NASA/Ames is done off-site. This processing time could be reduced from days to hours if a film processor were placed on-site.

The need for these recording techniques arises from the current capacity and performance limitations touched on earlier in this paper and summarized in table 1.

While there will always be a demand for presentation videos, partially reducing the dependence on these recording techniques would require ultimate performance in a computer graphics workstation. A spatial resolution of 1280 x 1024 requires 100,000 polygons/sec updated at 10-12 frames/sec for baseline performance (with hidden surfaces removed, antialiasing and interactive, advanced lighting models).Workstations that approach this level of performance are discussed in this paper. Other possible configurations NOT discussed in this paper include fast frame buffer configurations utilizing a very high speed network interface to a supercomputer (100 mbyte/sec) or an RGB digital video system (although this would NOT be interactive).

As further advances are made in supercomputing, parallel architectures, networks and workstation graphical performance, the authors predict development of more and more software environments where tracking and steering techniques are employed. At the time of this writing state-of-the-art resources allow for only minimal examples of these types of scientific visualization of computational aerodynamics (see Table 1).

8. CONCLUSIONS

The high resolution, high performance 3D graphical workstation combined with specially developed display and animation software has provided the scientists conducting fluid flow simulations with a good tool for analyzing flow field solutions obtained from supercomputers. A video tape recorder or 16mm film recorder, and the controlling animation software, are needed in addition to the workstation for very complex displays that cannot be created rapidly enough with at this point in time to yield satisfactory dynamics on the workstation alone.

REFERENCE

1. The computational and graphics facilities described are the joint efforts of the NASA Numerical Aerodynamics Simulation (NAS), and the Fluid Dynamics Division Workstation Applications Office, both at NASA/Ames Research Center, Moffett Field, California.

2. Lasinski, T., NASA Ames Research Center Request For Information RFI2-33166(RCB) April 9, 1987

3. Choi, D. and Levit, C., An Implementation of a Distributed Interactive Graphics System for a Supercomputer Environment, Second International Conference on Supercomputing (ICS), March 1987

4. Buning, P., and Steger, J., Graphics and Flow Visualization in Computational Fluid Dynamics, AIAA-85-1507-CP, AIAA 7th Computational Fluid Dynamics Conference, July 15-17, 1985

5. Rogers, S., Buning, P., Merritt, F. Distributed Interactive Graphics Applications in Computational Fluid Dynamics , submitted to the International Journal of Supercomputing Applications, July 1987

6. Buning, P., Bancroft, G., Lasinski, T., Choi, D., Rogers, S., Merritt, F. Flow Visualization of CFD Using Graphics Workstations AIAA 87-1180, 8th Computational Fluid Dynamics Conference, June 9-11, 1987

7. Foley, J., Van Dam A.Fundamentals of Interactive Computer Graphics Addison Wesley, 1982

APPLICATIONS OF GRAPHICS TO SUPPORT A TESTBED FOR AUTONOMOUS SPACE VEHICLE OPERATIONS

K. R. Schmeckpeper, J. P. Aldridge, S. Benson, S. Horner, A. Kullman, T. Mulder, W. Parrott, D. Roman, G. Watts
McDonnell Douglas Space Systems Company
Houston, Texas 77062

Daniel C. Bochsler
LinCom Corporation - Houston Operations
Houston, Texas 77058

ABSTRACT

We describe our experience using graphics tools and utilities while building an application, AUTOPS, that uses a graphical Macintosh (TM)-like interface for the input and display of data, and animation graphics to enhance the presentation of results of autonomous space vehicle operations simulations. AUTOPS is a test bed for evaluating decisions for intelligent control systems for autonomous vehicles. Decisions made by an intelligent control system, e.g., a revised mission plan, might be displayed to the user in textual format or he can witness the effects of those decisions via "out of the window" graphics animations. Although a textual description conveys essentials, a graphics animation conveys the replanning results in a more convincing way. Similarily, iconic and menu-driven screen interfaces provide the user with more meaningful options and displays. We present our experiences with the SunView and TAE Plus graphics tools that we used for interface design, and the Johnson Space Center Interactive Graphics Laboratory animation graphics tools that we used for generating our "out of the window" graphics.

INTRODUCTION

For several years, much effort has gone into the development and application of enabling and enhancing technologies for support of space operations. Many new technologies and methods, such as artificial intelligence and expert systems, have been applied to flight design software, user interface problems, ground and flight crew training, ground based mission control operations, robotic operations, flight systems management, etc. [1] The AUTOPS (autonomous operations) test bed integrates many of these technologies into a single framework to develop effective operations management, an element of mission success that is equal in importance to reliable hardware and software. [2]

AUTOPS is an evolving tool that has thus far been developed to the point of a feasibility demonstration that makes considerable use of animated graphics and screen interaction graphics. The animations are used for demonstrating proximity operations autonomy in operation planning, mission monitoring, and fault management. Screen graphics additionally assist in demonstrating vehicle monitoring and health maintenance expert systems and rendezvous planning activities. Because these items form uniquely informative means to convey system behavior to an analyst, they form an important feature of AUTOPS.

Although the importance of good graphics is unquestionable, their development has previously represented a significant commitment of time and effort. The availability of graphics tools has significantly changed this level of commitment. In this paper, we discuss our recent experience with using some of these tools.

AUTOPS CONCEPT

Figure 1 illustrates the architecture of AUTOPS. The test bed consists of a collection

of objects dedicated to specific activities: a test bed controller and vehicles that contain subobjects such as intelligent vehicle control systems, orbital and hardware simulations, and data management capabilities. The graphic capabilities are isolated from the computational capabilities in the graphics and operator interface objects controlled by the test bed controller. This architecture permits the reuse of code developed by others or the use of tools developed by others to produce the desired interfaces. Intelligent control is accomplished through cooperating expert systems that perform mission direction, mission monitoring, operations planning, and system health monitoring and fault recovery.

Currently, "vehicles" use software simulation as the means for providing orbital motion parameters and consistent sensor response to the orbital environment and vehicle subsystem operation. It is our intent to provide the capability to integrate hardware into the test bed to provide some of these data. For example, if it were desirable to test the ability of a vision sensor for use in close proximity operations, a television picture could be generated using the animated graphics and fed back to the vision hardware for the appropriate vehicle. A more immediate example is to use a fuzzy logic hardware chip to provide engine firings in place of the fuzzy logic controller software used in the feasibility demonstration.

Finally, other features of AUTOPS include the execution of the operation in real time and integration of currently available programs, especially simulation software. Real-time operation means here that the simulation computations will occur often enough to reflect actual behavior of an autonomous space vehicle and that time spent by expert systems in arriving at a decision for action will be taken into account.

SCREEN INTERFACES

Our feasibility demonstration required three screen interface designs: a main Operator Interface (OI), an interface to the electrical power system expert system (EPSYS), and an interface to the propulsion system expert system (PROPSYS). These interfaces were constructed over a period of time in which we were significantly increasing our graphics tool capability. The first to be built, the EPSYS interface, was created with SunView which is system software for our SUN network. The OI and PROPSYS interfaces were created with TAE Plus software obtained from Goddard Space Flight Center.

EPSYS is a prototype diagnostic expert system for monitoring the electrical power system of an autonomous shuttle-like space vehicle. Its function is to detect and explain anomalies and generate plans to recover from system faults. EPSYS supports a window- and menu-based user interface. The user-interface is composed of a base window that is subdivided among a group of graphic and text subwindows (Figure 2). Each graphic subwindow represents a control panel for a physical subsystem. The control panels are composed of parameter headings and a matrix of associated status lights and trend symbols. A command button and hierarchical menu system were designed to allow the user to easily communicate with the expert system. The final component of the interface is a scrollable text subwindow. The function of this window is to organize and display the textual representation of the high-level interactions and conclusions within the expert system.

Our choices for the development of the EPSYS interface were SunView and X. We chose SunView largely because we had access to the source code of a SunView-based interface which supported many of the same functional requirements that EPSYS possessed. Also, SunView is well-documented. At this time, our in-house version of X had several bugs and lacked complete and accurate documentation. In addition, the documentation we possessed supplied few examples. Also, our version of TAE Plus, an X code generator, was an early release and did not support many of the functions we needed to implement. The EPSYS interface was completed in three weeks by two programmers, including learning the SunView system.

The second interface we built was for the OI for inputting orbital parameters and showing calculational results. We elected to use TAE Plus for this task. TAE Plus allows the user to build a graphics interface with a Macintosh (TM)-like feel by using a graphics workbench tool with a mouse. It adds a layer of

programming over standard X code, such that the developer is required to have little, if any, X programming knowledge. Once the developer has the interface screen or panels designed, the workbench tool can generate code that implements it. Currently, the workbench will generate code in the C and Ada languages with Fortran and C++ generators under development.

Approximately one week was spent in learning how to use TAE Plus and how to integrate its generated code into an application. The original OI design was completed and implemented in four days by one programmer. An additional week was spent in editing the interface by "tweaking" the placement of items in a panel. Figure 3 presents the prototype OI master control panel and vehicle states output panels. The graphics workspace is the only panel that requires direct X programming.

Figure 4 shows an overlayed Initialization panel where the user can select one of ten rendezvous cases and either accept default data or modify any of the orbital elements. This panel required the most time to complete, as all work was performed on a SUN 3/50. TAE Plus was designed to run on a SUN 3/60. A twenty-four character limitation on display text length required that the titles on the rendezvous case selection buttons be created in halves and dragged to their locations on the panel.

The Propulsion Expert System (PROPSYS) is another prototype for a fault management system. PROPSYS will be a part of a distributed network of cooperating expert systems forming the System Monitor for an autonomous vehicle. It is a rule-based system written in CLIPS. Its user interface was developed using TAE Plus and X. The user interface is composed of a main control panel which is used to generate subsystem faults (Figure 5). The subsystem chosen brings up other panels with menus to enter parameters necessary for fault generation. After fault generation is complete, display panels that are appropriate for monitoring the subsystem during fault analysis and recovery appear (Figure 6). A standard X window displays text provided by the expert system during its operation. The text provides information on high-level interactions and conclusions made by the expert system.

The PROPSYS interface was completed in about four weeks by two programmers. This included learning TAE Plus and integrating its generated code with the application code. Access to existing TAE Plus code provided invaluable assistance and reduced our development time.

ANIMATION GRAPHICS EXPERIENCE

The integration of the AUTOPS Testbed Prototype with an existing graphics package was a simple, straight-forward procedure. In order to connect the prototype to the graphics, the AUTOPS Testbed Prototype software was loaded onto a Sun workstation located in the NASA Interactive Graphics Lab. This Sun contained Raster Technologies' graphics boards to provide a graphics engine and was connected to a high-resolution color monitor.

The modification of code in order that AUTOPS could be integrated with the graphics was also a minor procedure that consisted of customizing three routines and a data file. The three routines and the data file were copied from the graphics package into the AUTOPS simulation code. They were then modified to fit our requirements. This consisted of picking the vehicle models that we were using, in this case, models of the Shuttle Orbiter and the Orbital Maneuvering Vehicle, choosing information such as eye-point position, background models for the stars and the Earth, lighting, and size of the vehicle models. After the modifications were completed, the resulting code was compiled and linked into the simulation code. The Prototype was then executed in the same way that it was before the graphics was integrated into it. The procedure for integrating the AUTOPS prototype with the graphics required two programmers for two days.

Figure 7 shows one of the runs made with this system. The asterisks show positions of the orbiter at constant time intervals. Speed is thus indicated by the separation of successive indicators. This example illustrates the triggering of a replan by an expert system planner in response to an anomaly, in this case, a loss of general purpose computer redundancy. Flight rules specify that the vehicle shall back straight out to a 200 foot

range in this event. The graphics emphasize and record this behavior.

CONCLUSION

We have found that graphics tools provide a practical solution to quickly building excellent interfaces, that the tools are rapidly improving, and that the time for changes is growing sufficiently short that timely modifications of the interfaces to accomodate user preferences is now practical. Also, animated graphics can be easily adapted to enhance computational results without extensive modification of an application that does not support such capacity.

REFERENCES

1) Wang, Lui and Bochsler, Daniel, "Space Shuttle Onboard Navigation Console Expert/Trainer System," p. 11, NASA CONFERENCE PUBLICATION 2491, First Annual Workshop on Space Operations Automation and Robotics (SOAR 87), Houston, TX, August 5-7, 1987.

2) Beck, Harold,"Application of Technology to Space Flight Operations," Workshop on Space Launch Management and Operations,

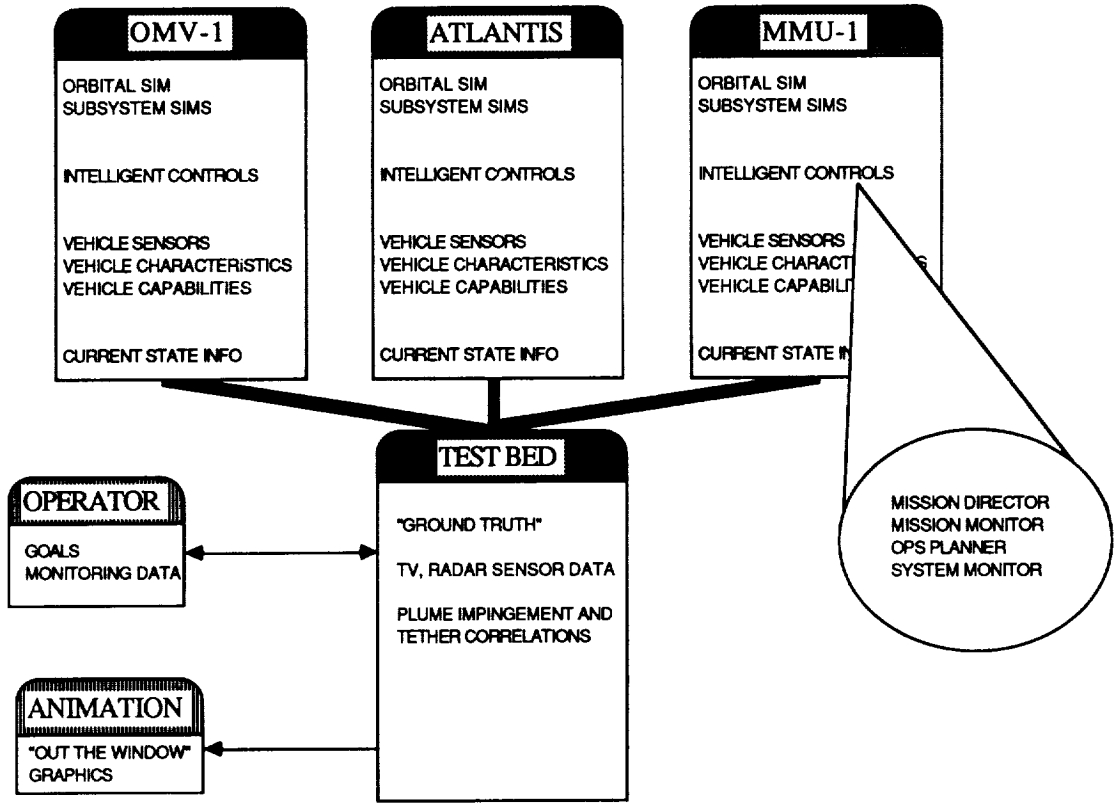


Figure 1. AUTOPS architecture

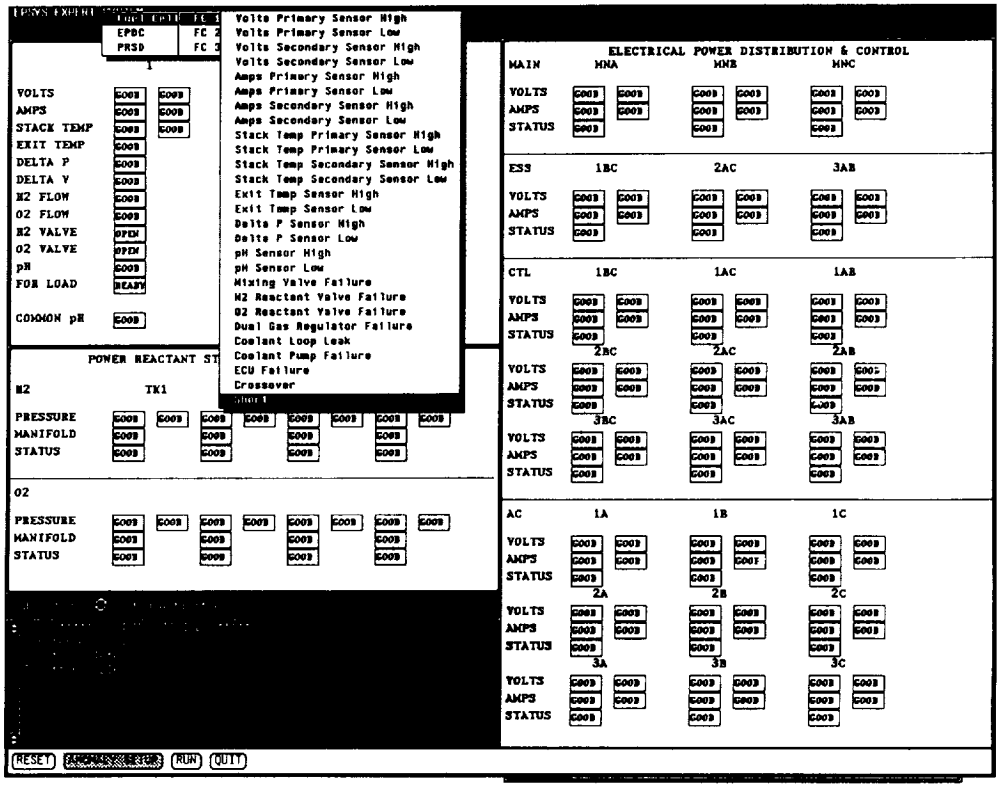


Figure 2 EPSYS User Interface

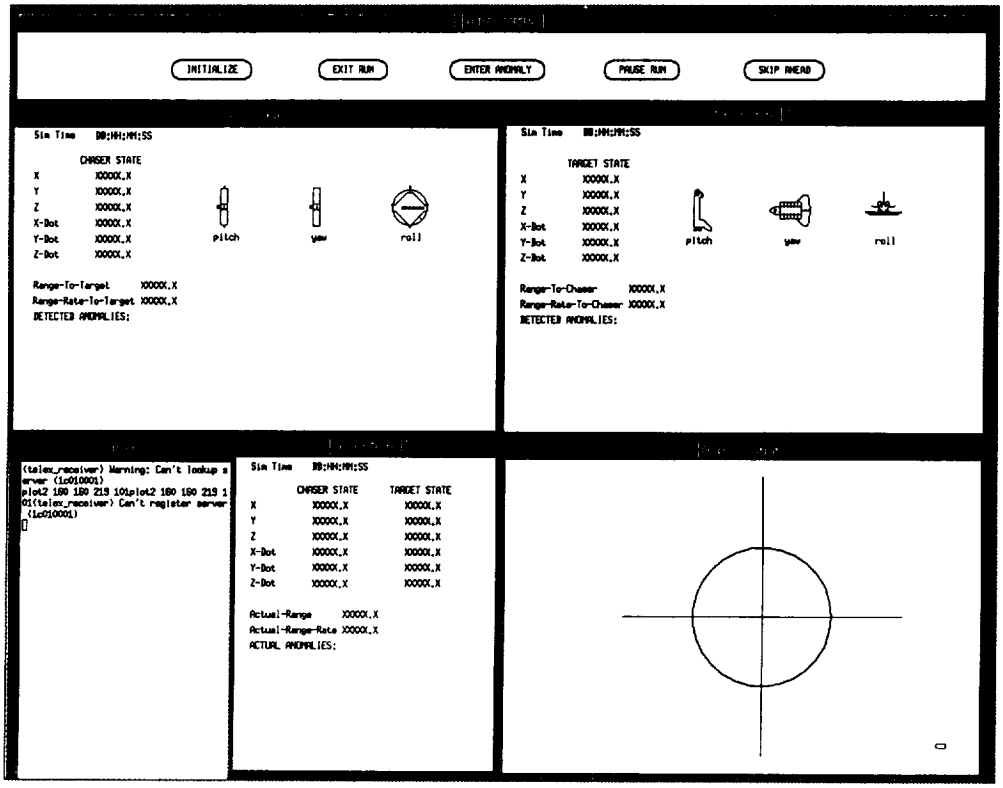


Figure 3 AUTOPS Operator Interface

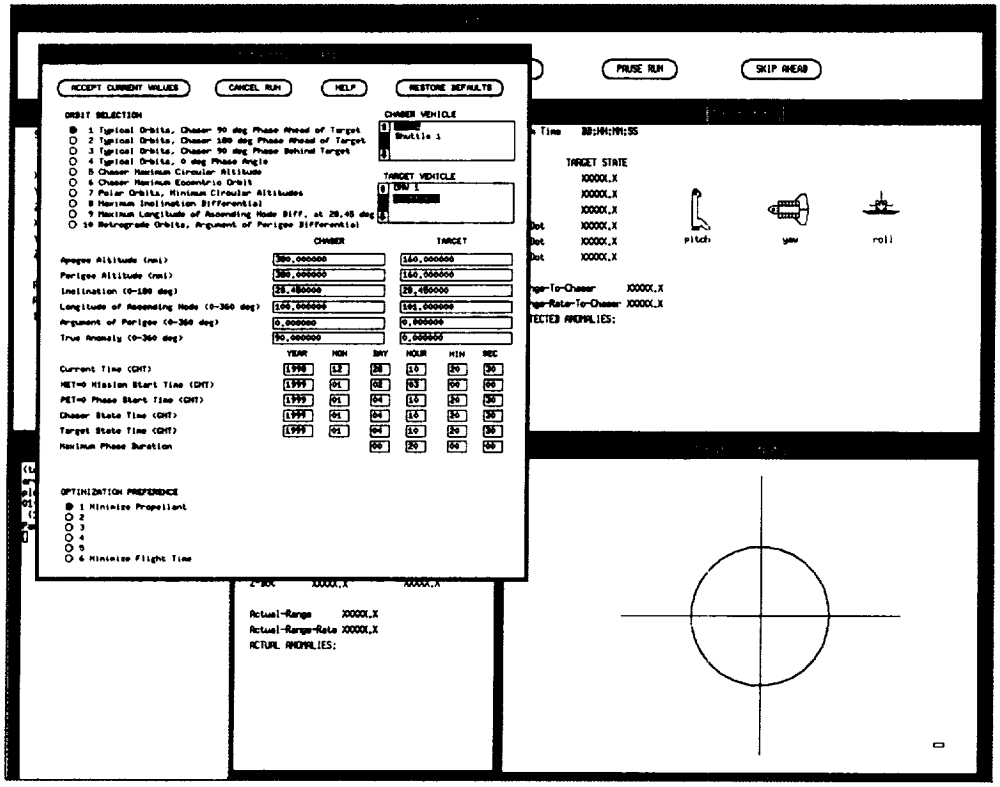


Figure 4 AUTOPS Initialization Interface

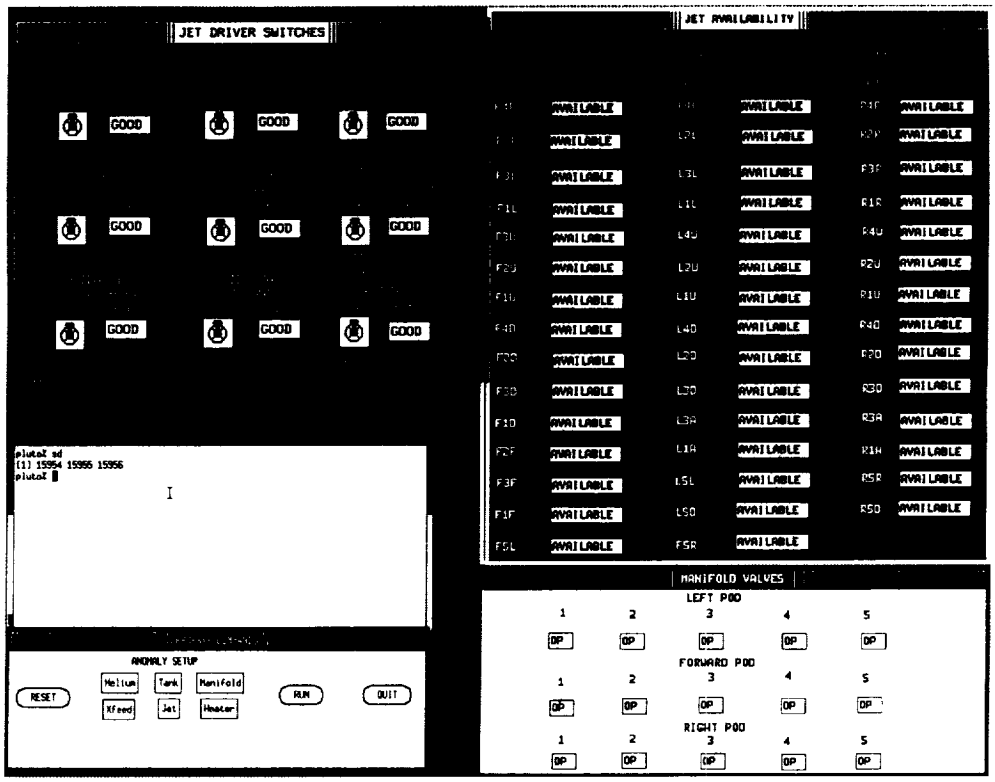


Figure 5 PROPSYS Main Control Panel

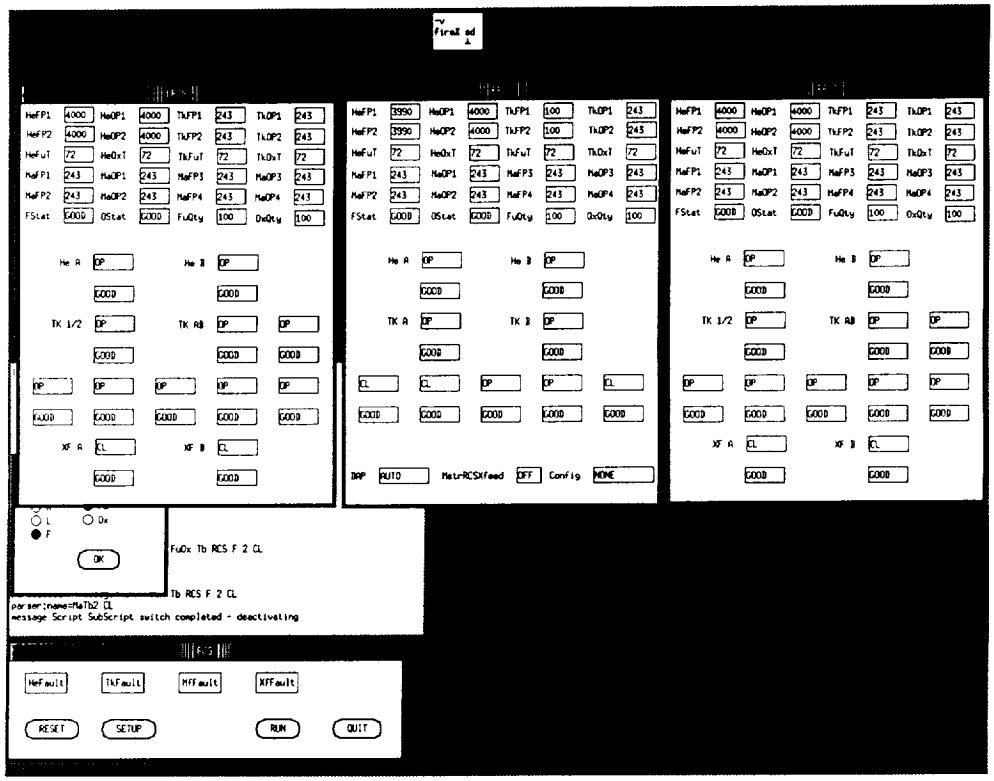


Figure 6 PROPSYS Fault Monitoring Panel

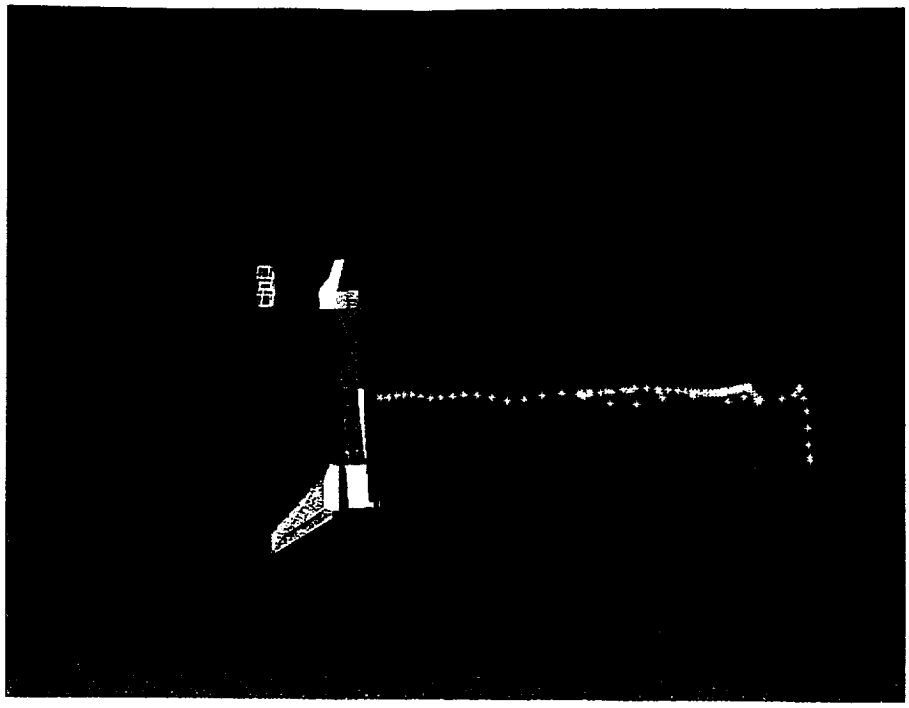


Figure 7 AUTOPS Animation Graphics Output

DESTINATION MARS

Mike Remus
Morton Thiokol, Inc.

(Paper not provided by publication date.)

LARGE SCREEN DISPLAY FOR THE MISSION CONTROL CENTER

Martin J. Skudlarek

Ford Aerospace Corporation, Space Information Systems Operation
MS B2E, P. O. Box 58487, Houston, Texas 77258-0487

INTRODUCTION

The Mission Control Center (MCC), located at the Johnson Space Center near Houston, Texas, is the primary point of control and monitoring for National Space Transportation System (NSTS) flight activities. NSTS flight managers monitor and command spacecraft from one of two Flight Control Rooms (FCR). Each FCR is equipped with five large screen displays for group dissemination of spacecraft system status and vehicle position relative to Earth geography. The primary *or center screen* display is ten feet in height and twenty feet in width. The secondary *or side screens* are seven and one-half feet high and ten feet wide. The center screen projection system is exhibiting high maintenance costs and is considered to be in wear-out phase.

The replacement of the large center screen displays at the MCC is complicated by the unique requirements of the Flight Controller user. These requirements demand a very high performance, multiple color projection system capable of the display of high resolution text, graphics and images produced in near real time. This paper describes the current system to be replaced, the replacement system requirements, the efforts necessary to procure the major element of this system (the projector) for the government, and how the new capabilities are to be integrated into the existing MCC operational configuration.

OVERVIEW OF EXISTING SYSTEM

The current center screen projection system represents the state-of-the-practice for electro-optical systems in the early half of the 1960's. This rear screen based projection system is comprised of three major subsystems; the projector, mirror and screen, and the driving electronics.

The projector

Manufactured by LTV Corporation, the projector subsystem utilizes seven individual xenon lamp/slide assemblies, mounted on a common structure, to form composite images on the screen. (See figure 1.) Each assembly performs a specific function and by way of color filtering, each can provide a specific color. Basic slides are constructed of glass, with a metal film coating, a few microns thick, to provide opaqueness, and are mounted to a metal frame.

Five of the seven projectors are equipped with carousels to provide a supply and repository for new and used slides respectively. Four of those five projectors, referred to as *plotters*, are equipped with diamond tipped scribes driven from X/Y servos, which scrape off the metal coating, allowing light to pass through the slide. The two projectors without carousels, *spotters*, utilize slides prepared with artwork, whose subject (i.e., shuttle outline) may be positioned within the frame of the image by X/Y servos similar to the scribing mechanism. The remaining carousel projector, *background*, displays a static background from prepared artwork (i.e., map). To prevent damage to the slides from the heat produced by the close proximity of the xenon lamps to the slides, compressed air must be forced onto the slide surface. When simultaneously illuminated, color modified through filters, and modulated by art work or scribed slides, the light from these individual projector assemblies is integrated on the screen to develop the familiar "world map" image.

The Mirror and Screen

Due to a floor space constraint (see figure 2), and long throw distance requirement of the projector, the optical path to the center screen must be folded. This is accomplished through the utilization of a large front surface reflection system. This system consists of two five by six and one half foot pieces of front surface aluminum coating number 749 glass. Glass mounting

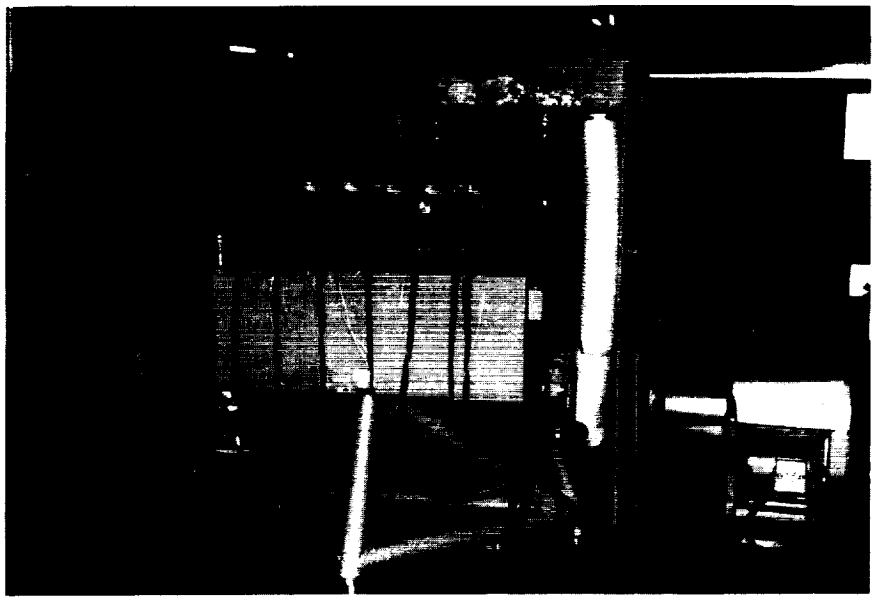


Figure 1. Existing LTV/Ford Aerospace center screen projector.

structures allow for tilt and azimuth adjustment. The support structure for the glass and glass mounts elevates the entire assembly to the required height of eleven feet, centerline.

The viewing screen is a single ten foot, one inch, by twenty foot, one inch, by 0.375-inch sheet of coated glass. The rear-projection coating faces the viewers.

Driving Electronics

Ford Aerospace specially designed and built the electronics to command the projector because of the unique nature of the projector interfacing requirement, for example; when to change slides or color. Nine standard equipment racks, six feet in height, house the interface electronics and xenon lamp power supplies. Commands are received from the mission operations mainframe computers and translated into analog voltages for the scribing pen or slide positioning servos. The logic is at a five inch by five inch card level. Each card performs a logic function, i.e., NAND gate. Physical fatigue is reducing the reliability and availability of this interface. Due to a sagging card cage support structure, the logic cards become unseated from their edge connectors and even with redundant channels, the subsystem has a high failure rate.

NEW SYSTEM REQUIREMENTS

Despite the complexities and short comings discussed to this point, the current system produces extremely sharp, bright, functional displays to which the user has become accustomed. Defining realistic and achievable requirements for a replacement system to match the current system's capabilities has been a difficult task. As will be discussed in a subsequent section, several iterations to generate requirements, release Request for Proposals (RFP), and evaluation of submitted proposals were necessary to finally achieve a successful procurement. The requirements delineated below correspond to the projector to be manufactured under the current subcontract. In most major procurements for the government, a committee is usually assembled to define system requirements. This project was not unique in that regard.

Display Characteristics

The display shall fill the entire ten by twenty foot screen surface from a maximum throw distance of not more than 35 feet. Provide an average illumination of 2500 lumens, flat modulated white light, 30 to 1 contrast ratio, uniform across the entire screen within ±25% *as measured from the rear of the screen with a standard photometer.*

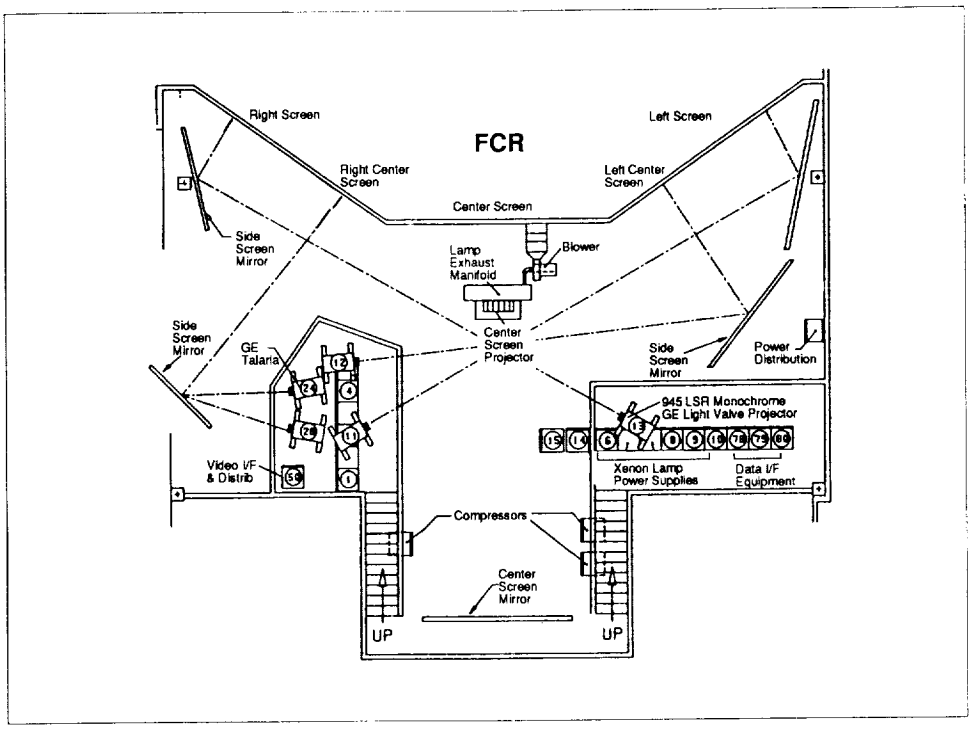


Figure 2. FCR projection room layout.

The projector shall be compatible with the analog RGB video output of several sources, such as; non-interlaced high resolution engineering workstations, the current 945 line scan rate repeat field monochrome video system and standard RS-170A color video. The projector shall have a "sense and select" automatic scan rate lock capability for a predetermined range of the possible video input scan rates designated for its use. The projector shall also be capable of accepting an extremely high resolution input (1800x900x60) *non-interlaced*, wide aspect signal for use as a center screen replacement projector.

The projector shall exhibit geometric distortion less than or equal to 0.5 percent of the screen height in a circle with origin at the center of the screen, with a diameter equal to the screen height, and less than 1 percent outside of that circle. Primary color registration shall be within 0.1 percent of screen height.

Commonality

To provide for economies of scale in initial procurement and in logistical support considerations, commonality is an important requirement. It is planned that, not only the projector, but the workstation/graphics processor driving the projector shall be common elements in many other manned-mission support disciplines. In the FCR two additional projectors are planned to satisfy the secondary or side screen replacement requirements. There are several new manned-spaceflight control facilities in the initial stages of development. These centers are expected to require large screen displays:

- Space Station Freedom Control Center
- Orbital Maneuvering Vehicle Control
- Crew Emergency Return Vehicle Control.

Additionally, space mission simulators require high light output, extremely high resolution projectors for simulation of ascent, on-orbit, and entry activities. The Shuttle Mission Training Facility is currently undergoing an upgrade and plans are being drawn up for the Space Station Freedom Training Facility. In all, over twenty projection systems may be required in the next two to five year period.

PROCUREMENT OVERVIEW

The requirements committee initially defined requirements for a projector to satisfy a center screen replacement in 1984. It was learned, after an RFP cycle and some fact finding, that such a projector was not available commercially. Subsequently, the Government determined that procurement of side screen replacement projectors could be supported by commercially available products. The requirements were modified to reflect the needs of the side screen replacement systems. In essence, these requirements are to display information similar to that available on a FCR nineteen inch engineering workstation CRT. The RFP was released in late 1985 and fact finding commenced in early 1986.

Fact finding consisted of visits to various supplier's manufacturing facilities, and demonstrations in the FCR itself. Six qualified bidders who met the specifications as written were selected. Ford Aerospace recommended, however, that none of the respondents offered a product satisfactory to the users requirements and that the procurement effort should be terminated.

Considerable experience had been amassed in the previous procurement attempts. An understanding of the large screen display state-of-the-practice had been acquired by the government and Ford Aerospace. Also, the MCC user community was educated to the fact that large screen display devices had significant display restrictions over CRTs and that the best quality display available would be expensive to acquire. Considering all lessons learned, a specification was developed that, if met, would satisfy a wide range of requirements (see above). Sufficient funding had been allocated to allow for the development of augmentations, modifications or upgrades to commercial products, if necessary, to satisfy the known requirements. In late 1986, an RFP was released defining such a projector. Six respondents submitted proposals of which two were determined to be in a competitive range. The six respondents were categorized as follows; two were direct laser projection products; two were Oil-film based projectors; and, two were based upon the polarizing light characteristics of crystals. The demonstration and fact finding procedures clearly identified the best proposal and candidate projector. The candidate selected was the Hughes Aircraft Company, Ground Systems Group/Fullerton, HDP-6000B projector, a liquid crystal based product.

Ford Aerospace is currently engaged with Hughes in negotiations to provide enhancements (see 6.0) to the original HDP-6000B projector. The two projectors currently subcontracted are scheduled to be delivered to Ford Aerospace in July of 1990. Production of the two HDP-6000B units will commence in February of 1989 at the Hughes Industrial Products Division in Carlsbad, CA.

PROOF-OF-CONCEPT

Given the level of funding required to purchase two large screen display projectors of this unique nature, NASA development managers desired a "check point" or Proof-of-Concept (POC) demonstration to ascertain if the projector could perform the assigned task adequately. Ford Aerospace included in the subcontract to Hughes the provisions for such a demonstration at the MCC FCR facility. Previous demonstrations, as part of procurement fact finding, yielded comments from the MCC user community expressing concern about insufficient illumination and display size. The subcontract included a clause that provides a POC demonstration of the specified brightness and display size prior to commencement of production.

Hughes modified a "brassboard" prototype projector to satisfy the POC requirements. The standard Arc lamp reflector was modified to produce twice the light output. The Xenon lamp size was increased to 2500 watts from 1000 watts. The scan circuitry was modified to accept video signals from the graphics processor selected for use by Ford Aerospace during the POC.

Ford Aerospace acquired a Sun 3/160 workstation to serve as the workstation host for the graphics processor and application software necessary to provide and control video signals to the POC projector. The application software had been under development at Ford Aerospace for approximately a year and consisted of nearly twenty thousand lines of Unix based C language code. The graphics processor, used for POC, was a Parallax 1280 series VME board set with RS-170A video overlay capability. The Parallax was chosen for its RGB video output timing flexibility and TV overlay capabilities. By using the flexible video output timing characteristics of the Parallax the development team was able to match the maximum possible scan

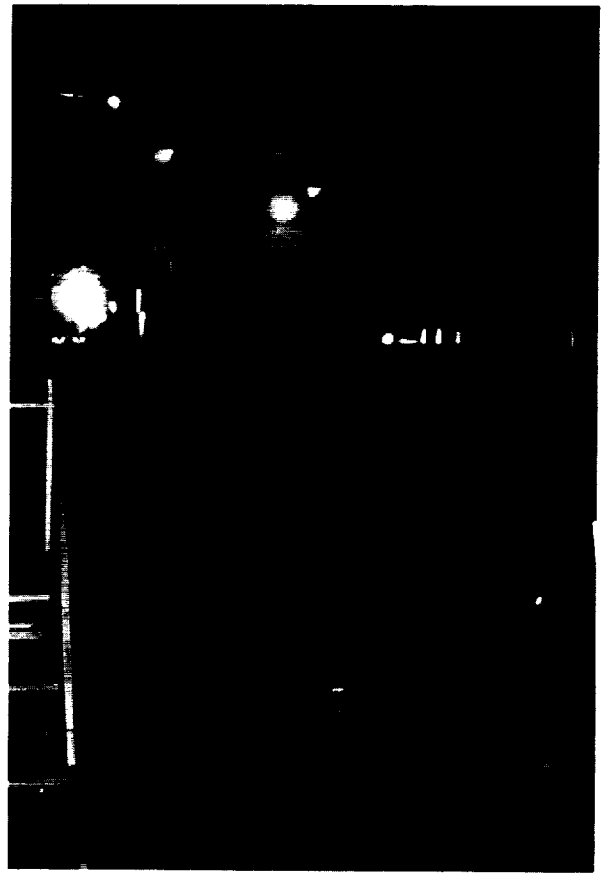


Figure 3. Proof-of-Concept Projector, lift table and Sun 3/160 workstation.

rate available from the POC projector, thereby, providing the optimum quality obtainable from the POC projector's electronics package reducing the overall cost of the subcontract to the government. It would have been prohibitively expensive and self defeating to produce a full production projector for POC. A series of four integration tests was preformed over a six month period at Hughes to define the POC projector's operational scan limitations and to integrate, test and align the POC system.

Hughes personnel installed the POC projector in the FCR projection room on a hydraulic lift table (see figue 3) provided by Ford Aerospace specifically for the POC. The lift table allowed placement of the POC projector's output optics at the center line of the screen. The workstation was configured so that the user interface hardware was on the user side of the screen and the workstation CPU/Graphics Processor combination was located adjacent to the projector. After an extensive checkout, alignment and test cycle, a performance verification test was preformed to contractually verify POC performance. The modulated light output, measured from the rear of the screen by a photometer, at nine locations adjacent and perpendicular to the screen surface, was over 2800 lumens. It was also determined that the projector, when located an additional two feet further from the screen, would provide a full ten by twenty screen display. (POC size was set at ten by eighteen feet due to the existing system's mirror support structure placement.)

The POC was considered to be successful by a majority of the MCC user community, NASA development managers, and the Ford Aerospace development team. (See figure 4.) A questionnaire was distributed during the user demonstration segment of the POC which asked for the user's opinion of the suitability of the demonstrated system as a replacement for the existing system. Over eighty percent of respondents to the questionnaire responded favorably. Consequently, NASA has directed Ford Aerospace to continue in our efforts to replace the center screen systems and investigate the commonality candidates.

ENHANCEMENTS

From the user comments during POC and NASA development manager's inputs, it was determined that further enhancements should be incorporated into the HDP-6000B production projectors. A brief description of the proposed enhancements follows.

Scan Circuits to 68 KHz

This enhancement provides for a higher level of commonality between various projector requirements, ease of manufacture, and logistical sparing. With the enhanced version of scan electronics, virtually any engineering class workstation, or Advanced Television source may be connected to the HDP-6000B inputs. This electronics package shall also be used in another Hughes program.

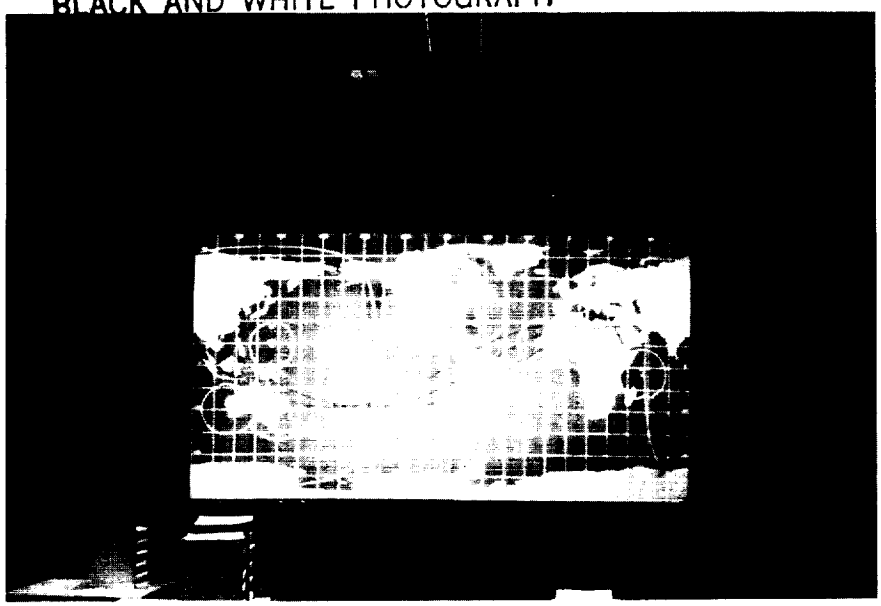


Figure 4. POC demonstration display of the "World Map."

Video Bandwidth to 110 MHz

Given a resolution requirement for the center screen of 1800x900x60 *non-interlaced,* it was necessary to increase the video bandwidth (sharpen pixel rise/fall times) in the production version of the HDP-6000B.

FUTURE PLANS

The existing subcontract shall be modified to include the enhancements discussed in the previous paragraphs. Hughes shall initiate production of the two HDP-6000B projectors shortly thereafter. As the actual fabrication of the two production HDP-6000B projectors is in progress, Ford Aerospace shall be monitoring that effort, in addition to developing the telemetry/trajectory data to video signal processing and generation software and equipment.

Workstation

The final host platform for the world map generation application has yet to be identified. Once the projector input requirements are set, a procurement cycle will be initiated for the graphics processor to satisfy the high resolution display requirements and match the video signal timing requirements of the projector. As the candidates for the graphics processor are identified, analysis to identify the workstation to serve as host to the graphics processor and application software will commence. The processing speed of the individual workstation's flavor of Unix, native software development environment, internal bus structure and local area network commonality are prime considerations in the analysis process.

Software

As was stated previously, the application software development task had been proceeding in parallel to the hardware identification, procurement and development activities. In addition to generating and updating a world tracking map from data delivered to the application from spacecraft telemetry, the application must generate near real time graphical displays that convey spacecraft parameters during ascent and entry phases of the flight. At POC, a stable prototype of the application was available for use in the demonstration of the POC projector. The application was initially developed on a Masscomp 5600 class workstation. The Parallax, which is a VME based product, (Masscomp 5600s are Multibus based) required a port of the application software to the VME based Sun 3/160. Approximately 95% of the code transported transparently, with only the graphics specific calls in need of a rewrite. The original application uses the native graphics calls specific to the Masscomp workstation graphics processor and, in the case of the Sun/Parallax, uses the Parallax native graphics instruction set.

POC provided a useful degree and quantity of feedback from the user community that simply could not be obtained from demonstrations of the application on a 19 inch CRT. Comments concerning character size were the most prevalent. Ford Aerospace intends to provide to NASA a human factors analysis on color and character usage for the large screen displays prior to the official final release of the application software.

Installation

Upon completion of fabrication, checkout and alignment of the projectors will be accomplished at the Hughes facilities prior to shipment. Upon receipt and receiving inspection by Ford Aerospace, the projectors will then be shipped to the Johnson Space Center, Building 30, where they will be installed in the two FCRs. Each installation will consist of the projector, new screen, projector lift table, workstation/graphics processor, application software, maintenance monitors, maintenance documentation and the removal and subsequent surplus of the existing system. When the installation is complete, a final Acceptance Test will be preformed verifying the performance of each projector to specifications.

A trial period of non critical support may be required to acquaint the flight control personnel with the new system's capabilities and idiosyncrasies prior to the removal of the current system. This concept is facilitated by the projector/lift table configuration which allows the HDP-6000B to be raised/lowered, in/out of the optical path of the existing system for use of either system as the situation would warrant.

Ford Aerospace has included training, for the maintenance and operations contract personnel who operate and maintain all the flight support equipment for the MCC, in the subcontract.

CONCLUSIONS

NASA and Ford Aerospace have selected what we believe is the optimum large screen display projector for our requirements. A methodology for the continuation of the system development and integration is in place and shall deliver the superior product NASA/JSC is accustomed to.

ACKNOWLEDGEMENTS

The author wishes to thank NASA, Ford Aerospace, Hughes Aircraft Company, and Lamar Flanagan of NASA/JSC/FS73, for their support of this effort. This project is funded through NASA contract NAS9-15014.

EFFICIENT UTILIZATION OF GRAPHICS TECHNOLOGY FOR SPACE ANIMATION

Gregory Peter Panos
Lead Engineer, REGIS Lab.
Rockwell International Corporation
Space Transportation Systems Division
Simulation and Systems Test Department
REGIS Computer Animation Laboratory
12214 Lakewood Boulevard MC DA-46
Downey, California 90241
(213) 922-0200

ABSTRACT

Efficient utilization of computer graphics technology has become a major investment in the work of aerospace engineers and mission designers. These new tools are having a significant impact in the development and analysis of complex tasks and procedures which must be prepared prior to actual space flight.

Design and implementation of useful methods in applying these tools has evolved into a complex interaction of hardware, software, network, video and various user interfaces. Because few people can understand every aspect of this broad mix of technology, many specialists are required to build, train, maintain and adapt these tools to changing user needs.

We have set out to create system where an engineering designer can easily work to achieve their goals with a minimum of technological distraction. We have accomplished this with high-performance flight simulation visual systems and supercomputer computational horsepower. Sophisticated but simple to use geometry generation, translation and modification systems input designer concepts to a motion design systems after which our visualization and scene rendering tools are invoked. Control throughout the creative process is judiciously applied while maintaining generality and ease of use to accomodate a wide variety of engineering needs.

INTRODUCTION

Planning of space missions has historically been a slow and tedious process. Drawings and exact measurements were drafted on paper for the various sequences that had to occur throughout a mission scenario. Launch preparation, mission operations, return to earth and post-flight ops are analyzed for time-line schedule conflicts, potential problems, and for detailed failure avoidance.

Within the last decade, CAD systems have become the predominant tools to assist with mission data design, operational determinations and detailed analysis. 3-D CAD systems have proved extremely valuable in the areas of 3D design data storage, distribution, retrieval and modification. Although most mission information can be processed by traditional CAD systems, there are major gaps in their ability to rapidly work out "what-if" changes and to easily create video based presentation materials.

Recent Developments

High-performance simulation graphics systems have ushered in a new age of productivity with tools that allow orders of magnitude increase in performance. These systems allow an analyst to rapidly prototype changes and evaluate operational procedures in real-time, while providing videotape recordings of their results.

The variety of new systems, software tools and fully interconnected networks, allow complex planning and analysis scenarios to be automated.They also tend to be virtually self-documenting. Distribution of video-taped presentations to high level decision makers has rapidly become standard operating procedure for critical projects that require rapid turn-around.

In our quest to satisfy advanced visualization needs for hardware design and operational simulation, we have identified several key areas of concern.

3D Object Data Configuration

Data Compatibility Interfaces

Multi-System Communication

Production System Integration

Production Compatibility

3D OBJECT DATA CONFIGURATION

When working with 3D geometric representations of objects whether they be parts of a large spacecraft or minute gears and wires, it is very important to know their position, scale, and orientation. A designer must define an object hierarchy relative to a global zero point and specify a rotational point for each moving component, otherwise, no useful motion can be performed. Part colors and shading type, levels of detail, transparency and texture choices must also be carried along with object data geometry and topology.

We have constructed a variety of tools which allow a designer to read, status, break up, combine, delete, add, modify, reorient or otherwise change object geometry and / or attributes with a series of simple commands.

For example: p2p -ro 90 -45 30 -su 100 -tr 1000 10 -50 < part > newpart

This "p2p" filter will scale file "part" by 100, rotate X by 90, Y by -45 and Z by 30 degrees, will translate the object to 1000, 10, -50 and create a new file "newpart". These tools allow a designer to avoid manually searching and editing 3D object data files to make changes. Many of these tools can be easily adapted to menu based windowing environments.

We have found that the more robust a 3D object database structure is, the easier it is for designers to define important object attributes early on. This reduces the need to manually add information later. Changes can be made easily, quickly and effectively when a complete database structure already exists.

DATA COMPATIBILITY INTERFACES

Once a 3D object database incorporates all pertinent information, that data must be made compatible with differing 3D graphics systems and software. Many of these systems require their own special format for object data and some reqire more than one file of information to process an image for a single object. Compatibility can become time consuming when migrating useful information from one system to another. To deal with this problem efficiently, a series of tools have evolved. They are:

Filters : Strip out, add or process numerical information or object attributes from one input object to another. Filters usually work on data used by one type of system or software.

Translators : Reformat object data, often in a major way, for use in different computer display system hardware and rendering software packages which require very specific input formats.

Compressors : Strip out unused information, truncate long numbers, optimize, encode and combine data in databases to to avoid redundancy. Archive utilities are specialized compressors. They are useful in reducing data storage requirements and in minimizing data transfer time.

Switchers : Substitute one section of data for another, often geomety, when a boundary condition is reached or some external flag has been set. These are useful for performing dynamic "Level Of Detail" changes on display systems. Limited throughput often degrades performance degrades as level of detail increases. As an object approaches the viewer, higher fidelity versions are switched into display system memory

Pixel Encoders : Allow data-rich pixel based images to be reduced into smaller more compressed files for better storage size.

Color Compressors : Allow images with many colors to be reduced and averaged down for systems with less capability to process and display that data. Also reduces file size.

Compositors : Allow a neutral user designated color to act as a window for another image to show through. A composite image file of both images can be saved independently.

Mappers : Allow a designer to define sections of an image to be used as surface texture maps for wrapping around or pasting bit-map images onto objects. Often images are scanned in with a video device to create the images used in texture mapping applications. This technique can be useful, giving the viewer the impression greater apparent detail exists on an object's surface then is contained in its geometry description alone.

Flexible data compatibility tools allow designers to free themselves of the limitations imposed by different display hardware and / or rendering software. Virtually any type data can be used anywhere it is needed (with a little help).

MULTI-SYSTEM COMMUNICATION

Data transfer and remote system control

Production environments are often complex and interconnected. This imposes many constraints upon a designer. Passing data between unlike systems can occupy valuable creative time. Human interaction in performing file transfers and conversions is unneccesary and inefficient. These tasks should be handled in an automated fashion. To this end, the following tools have been developed:

Transfer Utilities : Small, command driven tools which allow a designer to transfer single or multiple files from one place to another place with a minimum of headaches. These tools can be highly intelligent. They might check specific system directories to determine what files are there and which of them are current so as to retieve or send them.

For example: Getnet Regis DUA0 users.panos DAT 5 /usr/greg/data

This command will Get all version 5 .DAT files from the directory users.panos on disk DUA0 on the Regis system over the network link and it will place them in the directory /usr/greg/data on the system where the command was evoked.

Control Utilities : Allow a designer to send a series of control commands from a system port to an external peripheral device (such as a Videotape Recorder) to do something useful.

For example: Vtr -m 1 -l 6 -b 2000 -r 10 -S -f Regis::Renderer

This command will send a command out to a pre-designated port on the system where the command was executed. The port is wired to a Videotape Controller and the command is asking it to select VTR machine number 1 (-m 1) and to connect the incoming video line number 6 to the machine's input (-l 6). The -b 2000 option places the Vtr edit in-point to frame 2000 and allows a 10 frame edit (-r 10) at that point. The -S option asks the Vtr to go to "Standby" mode after it is done and the -f option will send a "done flag" to the "Renderer" program which is running back on the "Regis" system so it may begin another task.

Very often these utilities are highly system and software specific. Many of these tools contain security passwords, system identification numbers, codes, data-word sizes, and directory destinations and will allow privledged access that should be carefully protected.

Production environment developers must determine the safest and most efficient scheme for inter and intra-system communication and control. Frequent changes to information embedded within these tools should be avoided. Insured reliability of use for all users and the programs that serve them should be a top priority.

PRODUCTION SYSTEM INTEGRATION

Production system designers often overlook analog video signal routing problems. Digital computer display system details and their networks are often closely studied, while requirements for video signal distribution, propagation and interfacing are left to last and regarded as the least important aspects of the system.

Video signal needs must be attended to as a primary area of concern for the production system designer. It is a common fallacy to believe that many desired effects can be achieved digitally with rendering tricks and additional computer based techniques. Although this may be true for those who posess intimate knowledge of these tools, it is much simpler to create a desired effect with the use of video signal mixing and compositing with multi-track recording, time-base correction, and encoder function controls. This is the cross-over point where designers who have concentrated on becoming very proficient programmers become lost. Video engineers, with their knowledge of RGB, sync, key-channel, matting and analog calibration often take over at this point. Here are a few examples of video signal processing options:

Encoding : All Frame-buffer and display systems require signal conversion of their RGB output into a composite video form, usually NTSC. Encoders perform this conversion and are able to fine tune and juxtapose certain components of the output video. For special-effects, calibration, and interfacing to video switchers, color-keyers and VCRs, encoders are needed.

Keying : Enables one encoded video signal to be superimposed over a background video signal from a different source. A neutral color acts as a window on the foreground video signal. This is a very important feature for any advanced production system.

Synching : All input and output video signal sources should be locked to a synchronization clock signal to allow glitch-free effects. Source switching, dissolving, fading, keying, etc. all work much better when all systems are "genlocked" to house sync.

Switching : Multiple video sources can be switched electronically by computer or with keyboard based control functions in a good switcher. Different video lines can be re-channeled on the fly as production needs change and advanced effects like bordering, split-screen, quad-screen, title insertion, fade, dissolves, wipes, and highlights can be performed by good video switcher component which has been properly configured.

Recording : VTR Controls that are properly interfaced to the production system can be a major benefit to creative usage of all possible effects. It is a disadvantage for a designer to have to manually set up and button push a VTR's console to get it to perform the required actions. Today, most VTRs have controllers available to enable remote operation. Computers may also be allowed control of a VTR with multi-tasking.

These options enable a designer to create advanced visual material with the appropriate interconnection of video component and display system output. Multiple passes allow for increased scene complexity when using matting and keying techniques that a computer display alone would not be capable of. Simple effects like a wipe or page turn tend to be very time consuming for a digital computer to perform while a video effects switcher will function in real-time to accomplish such an effect.

With proper initial configuration of the above video components, a designer will be able to dynamically set up each scenario by using commands and script based execution sequences run from the host computer where they are already doing most of their work. This approach can remove the need to hand wire a video panel, perform manual VTR edits, or keep a video person around on a full time basis.

PRODUCTION COMPATIBILITY

Naming Conventions

In any production environment, there exist many different formats to contend with. Digital text and binary data files can be extremely varied in their formats. A program which requires a picture file for display on a computer frame-buffer may want it's data in ASCII, human readable text, while a CAD program needs a BINARY compressed geometry file representation as input.
Therefore a good plan for production compatibility becomes essential to assure continuity and efficiency throughout the creative process. Step one is to think out a good labeling scheme for the varity of information one needs to deal with.

Good use of terminology for different information types can be essential to properly understand what takes place in a production script. As a production evolves and is modified, it can be virtually impossible to keep track of files and taped sequences for replacement and editing if sensible names are not used. Disk and tape archiving is greatly simplified with this practice as well. Refer to Appendix A for details regarding suggested format naming conventions.

Commenting

When program tools modify files of data it is important to add comment lines describing what has been changed in the data and when changes were made.
For example, here is a geometry file that was processed through our "p2p" filter to reposition the 3D object data:

```
| Tue Feb 28 11:41:55 1989  polyp2p -ro 90 180 -45 -su 10 -tr 100 200 -300
surface Active FlatShaded   |   attributes Active FlatShaded
v  7    (7 Vertices)        |   p  5        (5 Polygons)
w 256                       |   0 1 2 3     (255 175 0)
100 -1610 980 256           |   4 5 6 0     (255 0 255)
100 -1610 -1580 256         |   0 6 5 1     (255 255 255)
1910 200 -1580 256          |   3 2 5 4     (0 0 255)
1910 200 980 256            |   3 4 6 0     (255 0 0)
100 2010 980 256
100 2010 -1580 256
-1710 200 -1580 256
```

In the above example, the "p2p" filter added the comment line to the top of the file saying what exactly has been done. In this case the file was rotated by X = 90, Y = 180, Z = -45, scaled up by 10, and translated by 100, 200 -300.

If necessary, all the values in the comment line could be used to convert the data back to the original unmodified form by negating the numbers and re-filtering the modified file. This feature can save hours for a designer who has made a mistake on or deleted by accident a critical file.

Listing

Another good practice is for a designer to build complex objects out of as many smaller object component files as possible. It is easy to combine them later with a "concatenate" command or in a script. This allows a great deal of flexibility when making slight changes, color-coding pieces or when showing only what is needed to aviod overloading a display generator or renderer.
Using a "Listing" file is the preferred way of specifying many different parts to be treated as one big part at runtime.

For example, in a Space Shuttle made of 7 major component files, the "Listing" file would be defined like this:

C.2

```
# ---------------------------- Space Shuttle components ------------------#
#
# Pathname/file              Description                    # of polygons
# -------------              -------------                  -------------
/shuttle/fuselage.P          # FUSELAGE                     p  170
/shuttle/blkhd.P             # BULKHEAD                     p  60
/shuttle/cnopyfwd.P          # CANOPY                       p  80
/shuttle/bay.P               # BAY                          p  50
/shuttle/vrtail.P            # VERT. TAIL                   p  30
/shuttle/wings.P             # WINGS                        p  80
/shuttle/rms.P               # rms pieces (4 of them)       p  70
```

When a display program is invoked we would specify the "shuttle.l" file as the file to display. This would treat all the pieces as one singular object. They would all move together but each would retain their own attributes such as color, shading type, transparency level, etc.

Display tools

The "examine" program allows a designer to work with 1 to 4 objects, each with up to 16 data files. This enables a designer to easily rotate and translate the view, change the background color and light direction, and to manipulate each object separately. The program also has a helpful arrow which points at the light source and a clock hand which rotates once every second to indicate the display system frame rate for use in overload assessment.

For example: examine -f0 shuttle.l -f1 earth.p -f2 gps1.p -f3 gps2.p

Here we control the shuttle, the earth and two gps satellite models all as separately moving objects. Each object has independently controllable offset, rotation, and translation, and is manipulated through separate data tracks.

"Examine" is one of our major workhorse animation programs that we use in our production environment. It runs on the Poly 2000e computer image generator, however, its functionality is extensible to virtually any real-time display. File flipping of 3D object data, where object data varies its geometry from one frame to the next, is also possible. This feature will allow animated display of incrementally deformed 3D objects and to rapidly flip through them while maintaining real-time control of their position and view orientation. However, one must always watch their polygon count when loading these large amounts of data into a real-time system. Otherwise, performance may degrade and other undesirable display artifacts may appear. Refer to Appendix B for the examine program's options and argument specification.

Rapid Prototype Generation

The last Real-time animation program that I will mention is the "RPG" program or Rapid Prototype Generator. This "CASE-like" tool was originally conceived by many different people at different facilities simultaneously (like all great ideas). RPG is another major tool which helps us to carry the banner for more efficient and creative production.

The concept is to allow a non-programmer (and hopefully good designer) to rapidly define, build and animate complex hierarchies of 3D object components. Without the need for a great deal of complex technical display-system-specific knowledge, a designer can use RPG to do this in record time.

First an inventory is made of the 3D objects that are to be used. Determination of what is going to move relative to what (defining the hierarchy) and finding offset distances and an axis of rotation for each moving part is next.

This is done by making a simple move to the center point (0,0,0) in any simple display program and then reading the numbers off the dial box or screen). A text script is then created by typing in the object hierarchy, offset values, track assignments and display diagnostic features. This step can take from 5 minutes to a half hour depending on how well a plan is mapped out and how fast one can type.

The script is then run through the RPG program "update"on the host system. Essentially a compiler, "update" generates C source code and compiles it into an an executable module. Next, a "builder" program is run on the image generator system where the hierarchy (as described in the RPG script) is assembled and the 3D object modules are loaded into real-time display list memory.

Last, the executable module created by "update" is run on the image generator. The designer can then use standard channel-based animation package features to define, save and preview keyframe files.

A detailed description of the statement format and syntax for RPG scripts is beyond the scope of this paper. We also feel it would be possible to make the implementation we have chosen even simpler to use. A menu driven script builder utility with a full graphical interface has been suggested and will be developed in the near future. Generalized structures in the program will allow differing real-time displays to utilize the same scripts and 3D object data through the use of special device driver modules that can be linked to the RPG program. Refer to Appendix C for an example of a simple RPG program script.

CONCLUSION

Only through years of experience have we been able to best determine what was needed to enable efficient production of computer animation and effects. After many hours of difficult 3D object geometry debugging, we were able to design flexible and easy to use tools to allow us to do in minutes what used to take hours. After many days and weeks of writing custom non-reusable C programs for every animation, we developed "RPG" with which we could produce our working programs in minutes. Once we created our animation package library we could generate and save reusable motion data files that took hours to develop by hand. Motions can be recalled in seconds and used in virtually any animation.

Once things begin to work as efficiently as possible, the greatest burden lies with the designer to dream up, build and produce the visuals that they desire. And yet, after countless creative sessions, more efficient ways to facilitate the creative process always seem to emerge.

ACKNOWLEDGEMENTS

I wish to thank Kenneth M. Stern, my associate in the REGIS Lab. for his fine contributions and corrections to this material. I would also like to thank Ben Thompson, STSD Advanced Engineering for additional comments and inspiration.

APPENDIX A

Description of suggested naming convention for different data file types.

Pixel Image data file formats:

```
Image.BIN   : An Image file in computer readable BINARY.
Image.RLE   : An Image file in "Run Length Encoded" BINARY.
Image.ASC   : An Image file in ASCII, human readable TEXT.
Image.HAM   : An Image file in Amiga "IFF HAM mode" BINARY.
Image.IFF   : An Image file in Amiga "IFF" Low, Med or HiRes BINARY.
Image.R8    : An Image file in BINARY RLE for "Cubicomp Picturemaker".
Image.PIX   : An Image file in BINARY RLE for Cubicomp Map Mode.
```

Text data file formats:

```
file.COM  : A Script file of run-time Commands in TEXT.
file.LOG  : An output file of program run results in TEXT.
file.POS  : A "TRACER" hierarchical Scene data / offset file in TEXT.
file.MAT  : A "TRACER" part Material coefficient data file in TEXT.
file.LGT  : A "TRACER" Light source description file in TEXT.
file.ENV  : An "TRACER" Environment data file in TEXT.
file.RPG  : An RPG hierarchy, offset and control data file in TEXT.
file.HDR  : A "Cubicomp Picturemaker" Header file in TEXT.
file.WFT  : A "Cubicomp Picturemaker" Wireframe test filein TEXT.
file.SCN  : A "Cubicomp Picturemaker" Scene test file in TEXT.
file.MMP  : A "Cubicomp Picturemaker" Color Map file in BINARY.
file.ENV  : A "Cubicomp Picturemaker" Environment file in BINARY.
file.CM   : A "Cubicomp Picturemaker" Command Macro file in TEXT.
file.L    : A List of Geometry files with various components in TEXT.
```

3D Object Geometry data file formats:

```
data.GEO  : "Aegis Videoscape-3D" Polygon floating-point TEXT.
data.MOV  : "MOVIE.BYU" Polygon floating-point TEXT.
data.OBJ  : "Symbolics S-GEOMetry" polygon floating-point TEXT.
data.TRI  : "TRACER" Polygon triangle floating-point  TEXT.
data.SPH  : "TRACER" Sphere data in floating-point TEXT.
data.DAT  : "RI-CDAS" quartic data in floating-point BINARY.
data.WS   : A "Cubicomp Picturemaker" WorkSpace file in BINARY.
data.P    : "Poly 2000" Polygon 16 bit integer BINARY.
data.p    : "Poly 2000" Polygon 16 bit integer TEXT.
```

Motion data file formats:

```
data.MOT   : "Aegis Videoscape-3D" Object motion floating-point TEXT.
data.CAM   : "Aegis Videoscape-3D" Camera motion floating-point TEXT.
data.PW   : "Cubicomp Picturemaker" keyframe Position Word TEXT file.
data.K    : "Animation Package" Multi-Channel 16 bit integer BINARY.
data.k    : "Animation Package" Multi-Channel 16 bit integer TEXT.
```

APPENDIX B

Examine program options and arguement specifications.

Flag	Meaning	Default
-a#m <f>	animate object # using	
	mode m	once
	changing every f frames	1
	modes: [a]dd - add files starting at current position	
	[o]nce - 0 up to n.	
	[b]ounce - 0 up to n, down to 0, ...	
	[c]ycle - 0 up to n, 0 up to n, ...	
-b <file>	filename of background objectn	one
-c#	# channels	# present
-d	debug mode	false
-f#	filename(s) of object # follows	
-k	keyframe (.K) filename follows	default.K
-m	light moves with view	light is motionless
-r #	new rotation sensivity follows	32
-s	silent mode (no beeps) when switching files	false
-tn #	new translate W value for object n follows	256
	if n not present, value used for view	
-T	do not clear or write to text screen	write help screen
-v # # #	view offset is # # # (x,y,z)	0 0 0
-w #	new scale W value follows	256

APPENDIX C

Simple RPG text script for an interlocking gear mechanism animation.

```
# -------------- Interlocking Gear RPG Script ----------------------------#

First we Define the Hierarchy

TREE

world :         Chan0
                100 101 102 103 104 105
                XYZ.1
box :           world
                0
                box.1
gear0 :         box
                10 11
                gear0.1
gear1 :         box
                20 21 22
                gear1.1

# Next we assign the Offset and Track assignments

MATRICES

trans(  100, val[0][0], val[0][1], val[0][2] )

rotx(   101, val[1][0]*32 )
roty(   102, val[1][1]*32 )
rotz(   103, val[1][2]*32 )
trans(  104, val[2][0], val[2][1], val[2][2] )

scaleu( 105, val[0][3] + val[1][3] + val[2][3] + 256 )

trans(    0, val[3][0], val[3][1], val[3][2])
trans(   10, 0, 160, -110 )
rotz(    11, val[5][0] *32 )
trans(   20, 0, 40, 0 )
roty(    21, val[5][0] * ( -32) + 4096 )
rotx(    22, 16384 )

# Last we Define our Display Diagnostics

leds(0,"0 K0:%6d K1:%6d K2:%6d K3:%6d",val[0][0],val[0][1],val[0][2],val[0][3])

# ---------------------------------- EOF ------------------------------------
```

ENGINEERING VISUALIZATION UTILIZING ADVANCED ANIMATION

Gunter R. Sabionski
NASA Johnson Space Center/FM7
Houston, Texas 77058

Thomas L. Robinson Jr.
Barrios Technology, Inc.
1331 Gemini Drive/BARR01
Houston, Texas 77058

INTRODUCTION

Engineering visualization is the use of computer graphics to depict engineering analysis and simulation in visual form from project planning through documentation. Graphics displays let engineers see data represented dynamically which permits the quick evaluation of results. The current state of graphics hardware and software generally allows the creation of two types of 3D graphics. One type features highly-detailed, realistic images that are displayed in non-real-time. The other type permits real-time display, but only for relatively crude graphics. (Real-time for our purposes is defined as the update of a display at a sufficient rate to make the change invisible to the human eye, typically 30 frames per second.) There are simulators capable of producing realistic 3D motion in real-time but their cost can be prohibitive. Animation provides an alternate route to generating realistic 3D graphics which are recorded on video for later playback in real-time. A fully produced video animation has the power to provide the organization, clarity, and attention to detail demanded for the communication of complex engineering concepts.

This paper presents the use of animated video as an engineering visualization tool. The engineering, animation, and videography aspects of animated video production are each discussed. Specific issues include the integration of staffing expertise, hardware, software, and the various production processes. A detailed explanation of the animation process reveals the capabilities of this unique engineering visualization method. Automation of animation and video production processes are covered and future directions are proposed.

EVOLUTION OF ANALYSIS AND SIMULATION VISUALIZATION

Before turning to a discussion of animation and video, it would be helpful to review the evolution of engineering visualization. Some ten to twenty years ago, analysis and simulation results were printed out as pure data. Mountainous stacks of computer paper were produced that required hours to sift through. The engineer could graph the data in two dimensions on graph paper or perhaps have a draftsman illustrate it. The development of plotters eliminated the need to produce these graphics by hand.

Later, as computers grew more sophisticated, it became possible to represent three-dimensional objects using complex data structures. The plotter was immediately employed to produce two-dimensional renderings of the objects. It also became possible to produce static 2D displays of the 3D objects on a CRT. The natural next step was to set the objects in motion. Doing this has proved to be a most challenging task.

A great deal of computing power and memory is required to render a 3D object quickly enough to provide the illusion of motion. The

complexity of the scene being rendered is a major factor in the amount of time required. The result is that a crude representation may move at a satisfactory rate while a realistically detailed image moves too slow in real-time to be of value. Obviously the goal is to produce realistic real-time graphics. Technology today is on the verge of providing such graphics but only at a relatively high cost. The speed and realism of the specialized high-performance simulators is astounding, but so is the price. It is this gap between affordable high-speed systems and realistic real-time display that animation fills so well. Depending on the level of complexity desired, animation may require a great deal of time to generate, but when completed it provides real-time speed via videotape playback.

DEVELOPING AN INTEGRATED APPROACH TO VISUALIZATION

The process of engineering visualization is a complex task. It is made even more difficult when the demands of engineering documentation are added to it. Although engineering simulation, graphics, video, and documentation have all been previously used as stand-alone entities, the integration of them as a full-fledged production tool has just begun. By identifying the staffing expertise required, coordinating that expertise, employing the appropriate hardware and software, and standardizing the processes involved, engineering visualization can simultaneously be integrated and simplified (Figure 1). This type of approach was used in the development of the Mars Rover Sample Return Mission video shown at the 1989 Graphics Technology in Space Applications conference video show.

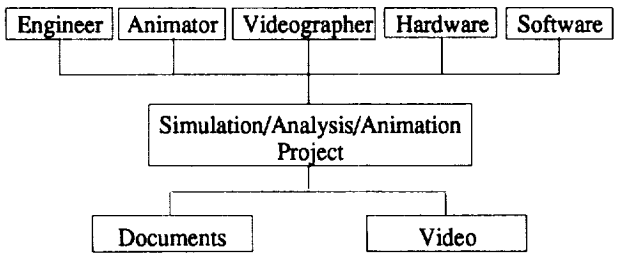


Figure 1 - Project Integration Overview

DIVERSITY OF EXPERTISE REQUIRED

The staffing expertise required for engineering visualization using animation consists of three general categories: engineering, animation, and videography (see Table I). Although the topic is "engineering" visualization, the animation and videography staff provide unique expertise and talent that cannot be overlooked.

Table I -Diversity of Expertise Required

Engineering:	Animation:	Videography:
Aerospace	Artistic Creativity	(Video)
Sciences	Visual Composition	Single-frame Record
Analysis	Storyboarding	Editing Operations
Simulation	Computers	Switching
Hardware Design	Model Building	Special Effects Units
Software Dev.	Motion Hierarchy	Character Generators
Communications	Motion Generation	LaserDiscs
User Interfaces	Kinematic Timing	
Graphics	Rendering	(Audio)
Fractal Geometry	Image Manipulation	Scriptwriting
	Paint Systems	Announcing
	Digitization	Recording
		Editing
		Music
		Sound Effects
		Sound Mixing
		Digital Processing

Engineering expertise can be broken down into two dissimilar disciplines. The first represents the "line organization" engineering staff that contributes technical direction to the effort. The areas of expertise of this group include those traditionally associated with aerospace engineering: sciences, research, design, analysis, and simulation. The second consists of software engineering personnel. They provide direct support in the form of system administration, graphics programming, and software development.

The animation staff, besides operating the animation software, are responsible for adding the realism and presentation value to the animation. They collaborate with both the line engi-

neer and the videography staff to produce a storyboard which summarizes the visual content of the animation. They then contribute a variety of skills and artistic talents including visual composition, image manipulation, kinematic timing, model building and other components vital to the creation of an animation.

Videography, like engineering, involves two separate areas of expertise. One area includes the creative skills and technical knowledge necessary to produce the audio soundtrack. The scriptwriter must know how to write for a speaker. The announcer must be able to annunciate the text and must be trained to avoid popping and hissing speech. Audio technicians must understand an array of recording, editing, and mixing equipment. Video is the other, and more obvious area, of videography expertise required. The hardware intensive nature of video production requires considerable technical knowledge. Technical savvy must be combined with artistic and creative talents to produce an effective video presentation. The videography staff also works closely with the animation staff providing both technical and creative direction.

ELEMENTS OF ANIMATED VIDEO PRODUCTION

The process of creating an animated video production occurs in four major steps. Planning is the first step. It includes the development of a script, a storyboard, and a shot list. The script is a written narrative for the presentation that is carefully timed to coincide with the corresponding images on the screen. The storyboard is a series of thumbnail sketches of the images themselves. The shot list is a description of the images with details about length and other technical issues. These are developed together by the engineering, animation, and videography staffs and provide a basic plan for the final production. All are used to guide the other steps of the process. The second step is the generation of the animation. The animation staff uses the storyboard and the shot list to generate the needed sequences. When the animation is completed it is recorded on videotape or other video media. The third step is the development of an audio track. The script is recorded and edited to the proper length. Music and sound effects are then added. When the audio track is completed, it is recorded onto the sound channel of a videotape or disc. The animation and audio track development steps may occur simultaneously if planning has been thorough. The final step is video post-production. The animation and audio (now both recorded on video media) must be edited together and title and credit sequences added.

The transmission of knowledge and information from the engineering workstation to the video presentation is made possible by animation. What distinguishes animation from a simple recording of images from the workstation? Single frame recording to video provides real-time playback of both, but animation adds valuable visual realism in a variety of ways. This is revealed through a more detailed examination of the animation process (Figure 2). This examination is based on the assumption

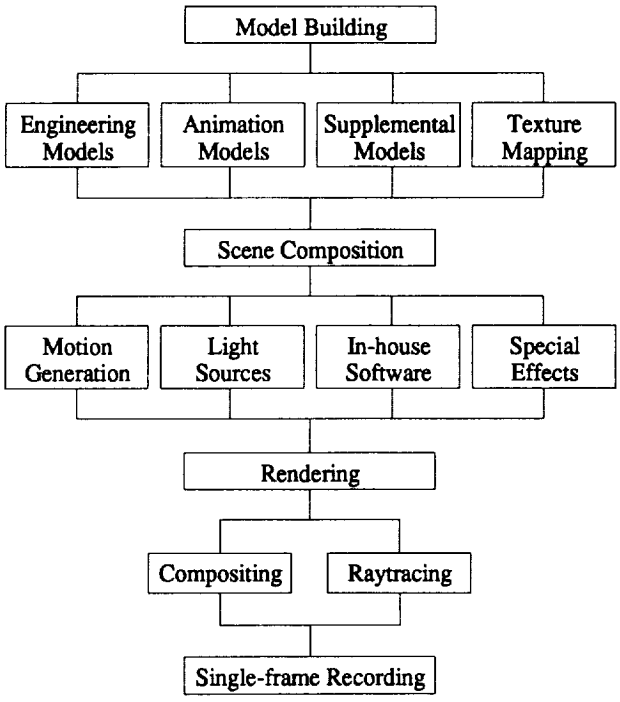


Figure 2 - Elements of Animation

that the planning phase has been completed and that a storyboard, shot list, and script have thus been developed.

Model Building

Model building is one of the first activities required in the animation process. There are many levels of 3D modeling. The highest level includes the models familiar to most engineers. The precise, technically complete, 3D models used in engineering are invaluable. The detail of these models is demanded by the rigorous requirements of design and analysis. At the other end of the modeling spectrum are crude, inaccurate representations of objects valuable for only the most cursory evaluation of size and motion. The advantage to these latter models is the speed with which they can be manipulated. Animation requires the complete redrawing of every model on the screen thirty times to generate just one second of motion. The more detail a model has, the more time is required to draw it. The need for detail must then be weighed against the amount of time available to produce an animation. Animation models are developed with an awareness of this fact. The optimal relationship between speed and detail is usually found by using models in which unseen mechanical details are omitted. Very close attention is paid to the external visual details but the unseen inner workings are ignored. If focus on a particular mechanical feature is desired, it can be modeled and added at the appropriate point in the sequence. Millions of unnecessary calculations can be eliminated by the use of these simpler models. At the same time, to the viewer, the model is physically complete. The animator may develop supplemental models to represent scenery and other items.

Texture Mapping

Once models have been developed, their realism can be enhanced by a technique called texture mapping. Texture mapping is the process of generating a 2D image which is then mapped onto the surface of a 3D model. There are several forms of texture mapping and each form excels in a particular situation.

The simplest type of texture mapping is where a digitizer or paint system image is mapped directly onto the surface of a model. This is highly effective for producing foregrounds and backgrounds, and for adding logos, flags, and unusual surfaces to models. In the case of a foreground or background, the image is simply mapped onto flat or deformed surfaces and placed in the scene. Alternately, an image can be wrapped around a model to simulate the appearance of metal or some other material. In the Mars Rover video, this type of mapping was used extensively in the Titan launch sequence and to create the rocky Mars surface.

A specialized type of texture mapping called bump mapping can be used to simulate a relief surface where none exists. The image is mapped onto an object but no color is used. Instead, bump mapping uses the luminance values of the 2D image to determine how to deflect light falling on the object. Luminance is a measure of the black, white, and grey values contained in the image.

Similar to bump mapping is transparency mapping. Like bump mapping it uses only the luminance values of the 2D image. The difference is that transparency mapping uses these values to determine a level of transparency for the surface onto which the image is mapped. White will cause the surface to be opaque, black will cause it to be transparent.

One other useful type of texture mapping is reflection mapping. Reflection mapping retains the 2D image's color and maps it onto the surface of an object. What makes reflection mapping special is how the map image is generated. The animation software is instructed to render a view of the scene from the location and orientation of the surface that is to show the reflection. This view is used to generate the map image which may then be sized and distorted as

necessary to cover the surface.

Motion Generation

The next step in creating the animation is the actual generation of movement. One of the most common methods for accomplishing this is called keyframe animation. Keyframe animation allows the animator to define frames depicting the starting and ending positions for objects in the scene. The number of frames that are to occur between the starting and ending frame (at a rate of 30 frames per second) are then specified. The computer interpolates the position of all objects for each of the intermediate frames. Transformations along the objects' X, Y, and Z axes can be used to create non-linear paths. Translation, rotation, scaling, and skewing are some of the many attributes that can be specified singly or in combination to cause transformations. If connecting parts of an object must move relative to one another, this may be depicted using a technique called hierarchical motion. Hierarchical motion involves the assignment of relationships between different parts of a model. The main section of a model may be designated as the "parent" and its appendages as "offspring." The action of the connecting joints may also be defined with regard to limiting angles and rotational axes. The result is that when the parent moves, the offspring follow in a way defined by these relationships. This can make the simulation of realistic movement much easier to accomplish. Ultimately, the animator must use these techniques to create motion which is realistic and compositionally logical to the viewer, but also technically accurate in the eyes of the engineer.

Lighting

Convincing motion of realistically designed models goes a long way in creating an authentic animation. There are other compositional elements that must also be considered. The animation software provides an environment somewhat analogous to a windowless room: if no lights are turned on, nothing can be seen. The placement of light sources then becomes an issue. In the simulation of aerospace operations, it might seem logical to have only one light source representing the sun. In truth however, other sources of light must be considered. Light reflected from the Earth, the Moon, or other bodies can brighten unlit areas. Small man-made sources of light may also need to be simulated. Several different types of light sources are usually available in the animation software. Flood, spot, unidirectional, and omnidirectional sources are common and may be used to create specular, diffuse, and ambient light. Color can often be added to light sources as well. The selection and placement of light sources generally follows the same principles used in photographic or motion picture lighting.

Special Effects

The presence of light would also imply the presence of shadows. Shadows are not an ordinary by-product of a lighted object in a computer however. The location and darkness of shadows must be computed based on the location and characteristics of the light sources and the shape and location of the objects in the scene. It is a complex problem with a computationally intense solution. Implementation of shadowing usually results in a significant increase in the amount of time required to render each frame but is justified by a dramatic increase in image realism. The advent of shadowing options is a fairly recent development in animation software. At the current stage of hardware and software development, shadowing is still considered a "special effect."

Other capabilities which fall under the category of special effects include transparency and distortion of objects. While a transparency map can be used to give an object a transparent quality, variations in degree of transparency are not a standard feature. Such an effect can be used to reveal hidden structure within an object. An example of this appears in the canister transfer sequence of the Mars Rover Sample

Return Mission video. There is no standard method of producing variable transparency among those animation packages that support it. It is achieved in one package by entering a transparency factor for each individual frame in which the object's surface appears. This can be a tedious process but the result provides a very effective way to present technical detail.

Distortion of objects is a special effect that is useful for depicting non-rigid objects. Plastic, rubber, and textile objects are likely candidates for the use of distortion effects. Distortion is still the subject of much research and, like variable transparency, is achieved by a number of methods. Metamorphic distortion is available in some animation packages. This method allows a model's shape to be altered from one key frame to the next. The animation software then interpolates between the shapes in the same way that it interpolates an object's motion. Another form of distortion permits an object to be stretched along one of its three axes while it is compressed along the other two. This is sometimes referred to as volume distortion. Skew is a diagonal distortion of a model along two of its axes. These techniques can sometimes be used in combination with one another. This can be very effective at adding naturalism to the motion of flexible objects.

In-House Software Development

Most of the object manipulation methods discussed thus far are made possible by the standard or advanced features of commercially available animation packages. Animation of certain types of objects or effects are not currently within the capabilities of these packages however. The use of in-house graphics programming expertise may be helpful in such cases.

Research was needed in several instances to determine how to simulate various effects for the Mars Rover video. The action of the lander vehicle's parachute filling with air required the development of specialized code. Smoke for the Titan IVs and the Mars ascent vehicles was initially generated with a complex mathematical algorithm employing Fourier transforms and digital filtering. Another method of producing smoke was subsequently developed using an innovative combination of polygon distortion and moving bump maps. Custom software may sometimes be required for the creation of specific types of motion as well. The movement of the rover vehicle across the Martian surface is one example. Each of the six rolling wheels has to maintain surface contact with the irregular terrain yet remain in the correct orientation to the rest of the vehicle. In a more recent project, research was required to convincingly depict dust being stirred and settling back to the ground in a low-gravity environment. Problems like these are of particular interest in aerospace where unusual conditions are so frequently encountered.

Rendering

When all of the image manipulation and motion problems of the animation sequence have been addressed, there remains one more step in the animation process. The animation sequence must be rendered in solid form. Rendering is the computing of the final animation frames using all of the options and effects that have been defined by the animator. Prior to rendering, the animation exists only as a wireframe representation in the animation software. Texture mapping, lighting, shadows, and other effects are not yet visible. It is in this process where time intensive operations take their toll. A great deal of computing speed, memory, and storage are needed to render even a few seconds of animation. When a high-speed rendering engine is used, a complex frame full of intricate objects, reflections, and shadows may take from four to forty minutes to create. Additional time may be required if certain specialized rendering methods are used. One of these methods is called ray-tracing.

Ray-tracing results in highly realistic animated images. It works by calculating the path of ev-

ery beam of light from each light source to each pixel of each object on the screen and then back to the viewer's eye. It is especially effective when objects have glass or other highly reflective surfaces. Multiple levels of ray-tracing allow an increasing amount of detail but at a great penalty in computing time for each additional level. Ray-traced images have an almost photographic realism.

Compositing is also part of the rendering process and allows layering of separately generated images. This may be used to add background elements to a scene that was generated without them. Compositing reduces the amount of time required to render an image because static background images are rendered only once. Only the moving objects are recalculated for each new frame. This would seem to make compositing highly desirable. Actually a time trade-off takes place because compositing increases the amount of time required to display the completed image. This slows down the recording process. Whether compositing is desirable is determined by evaluating how computing resources may best be allocated between rendering and recording operations. Another way of compositing is available that eliminates the recording time penalty. The use of digital video storage devices and special effects units allows real-time compositing but requires the purchase of more expensive hardware.

Single-frame Recording

As each frame of animation is rendered, it is stored on the computer's mass storage unit, usually a hard disk. It cannot be played back in real-time from an ordinary hard disk however because access time and reconstruction of the display is too slow. Instead, software is used to display each frame individually and to trigger a video recording device to record it via specialized animation-control hardware. Among the devices that may be used for single-frame recording are videotape recorders, optical laserdisc recorders, and digital frame storage devices. Generally only the high-end, professional-format videotape recorders are capable of recording in single-frame mode. Recording of the frames to video completes the animation step of the animated video production process.

Editing

Editing is the arrangement of raw audio and video into a complete and coherent presentation (Figure 3). Audio editing is the next step of the animated video production process. This step technically begins when the script is written. Scripts are written in a two column format with the spoken text in the right column and a description of the corresponding visual images in the left column. The right column also contains details about music and sound effects and instructions to the audio technician. This format allows each portion of text to be carefully timed to match the correct image. The language of the script must have a narrative flow that sounds natural when spoken aloud. The script is recorded onto multi-track audio tape using a professional announcer. (Untrained speakers find it difficult to eliminate certain extraneous sounds from their speech. These can be very distracting in a recorded soundtrack.)

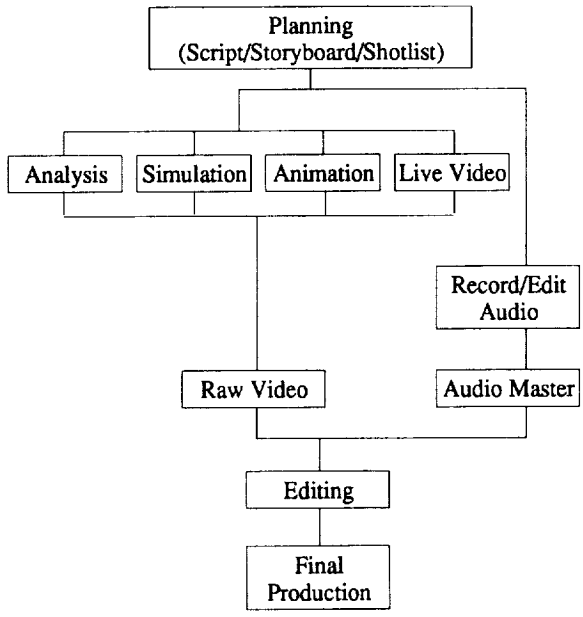


Figure 3 - Video Production Overview

After the script has been recorded, it must be edited to eliminate mistakes and to make it the proper length. Music and sound effects are then generated or selected from a library and added behind the voice on another track. A time code recorded on one track of the tape makes very precise editing possible. The completed soundtrack is referred to as the audio master or the soundtrack master. Because of the timing elements involved, implementing changes in the script after the edit session are very difficult. Usually, re-recording and re-editing of the entire soundtrack is necessary.

When the audio master has been completed, it is transferred to a blank videotape or other video media. The final step in the animated video production process is editing of the raw animation video onto the audio master tape. Live video footage may also be incorporated into the production at this point. Some animated (and live) sequences that have been generated must be slightly longer than the amount of time they are to appear on-screen. Additional footage at the beginning and end of each sequence gives the video editor a place to create transitions between scenes. Cuts, dissolves, and wipes are among the many types of transitions available through modern video editing equipment. The primary tools of a video production facility include character generators, edit controllers, video and audio switchers, special effects units, time base correctors, and video recording devices. These are used to assemble and refine the final video presentation.

Character generators allow the creation of titles and credits. A special effects unit may be used to overlay these onto the video footage or place them on a colored background. Special effects units can also provide the previously mentioned transitions and other manipulation of the video images. Switchers are used to generate dissolves, wipes, upstream and downstream chroma and luminance keying as well as soft and hard-edged transitions via special effects. A video edit controller allows precise editing of the video. A device called a time base corrector may be used to compensate for anomalies in the video signal. Typically, at least two video sources are connected through time base correctors to a video switcher and an edit controller for output to a recording device. Time code is usually recorded on the video media to provide frame-accurate control of the editing process. A standard for time code, developed by the Society of Motion Picture and Television Engineers (SMPTE), is used for most video and audio editing.

In editing the animation video, the soundtrack is the reference for assembling the scenes. This is why integral development of the script, the storyboard, and the shot list is so important in the planning phase. Sound effects must precisely coincide with the corresponding visual action. The narrative description must match exactly what is depicted on the screen. The viewer's interest must be maintained with an accurate, clear, and concise explanation of the concepts being presented. With careful planning and an integrated approach to the animated video process, this can be achieved very effectively.

AUTOMATION OF THE ANIMATED VIDEO PRODUCTION PROCESS

Engineering visualization is still a fairly new concept. Much of its potential remains unrealized because of a lack of automated processing and standard interfaces between and within its various disciplines (Figure 4). Some relationships between engineering, animation, and video exist already, but in each case the development of automation and interfaces has been an isolated process.

Engineering analysis and simulation have utilized 3D computer graphics since the late-1970s. Software and hardware interfaces are plentiful and standards are becoming fairly well established. An entire industry has arisen around the development of CAD/CAE workstations. Networking of computer systems has contributed an element of transparency to the

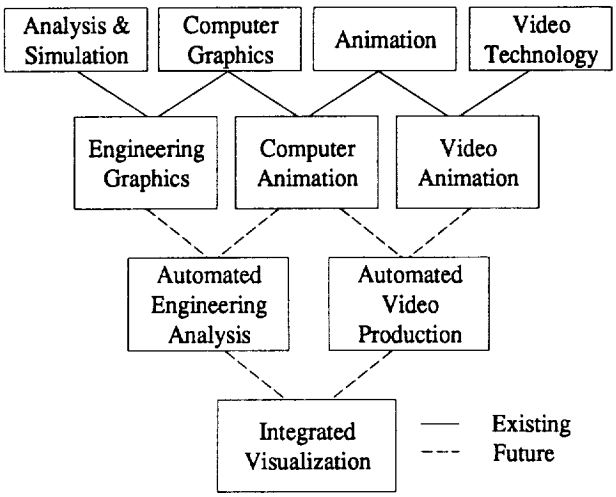


Figure 4 - State of Visualization Technology

interfaces. The Initial Graphics Exchange Specification (IGES) and Computer Graphics Metafile (CGM) are among the many graphics file standards developed for use within this field.

Television standards have been around for several decades and in no way reflect the current state of graphics visualization hardware. Television standards were designed to allow use of compatible analog signal by either black and white or color television sets. Automation of video production is occurring but with few of the benefits of digital technology inherent in the computer industry. One promising new development is a video version of the computer network. The Video Local Area Network (V-LAN) permits the same degree of transportability over video control signals that a computer network allows over data.

The use of 3D computer graphics for animation started in the early 1980s and interfaces and standards for it are just beginning to be defined. Whereas 2D animation was used almost strictly for entertainment and commercial purposes, 3D animation has become a valuable tool in the scientific and engineering world. Although animation is historically associated with film and video, the use of 3D computer graphics to generate animation is requiring unprecedented computer to video interfaces.

The dissimilarity of hardware is at the root of most problems faced in automating the animated video production process. The engineer's workstation, the animator's workstation, and the videographer's equipment typically have different screen resolutions. In addition, the two workstations usually output red, green, and blue (RGB) signal values at a scan rate of 60 hertz. The videography equipment requires separate color and luminance signals synchronized and interlaced at a scan rate of 30 hertz. Getting an image from one device to the next often requires as much specialized equipment as do the systems themselves. The capabilities, limitations, and compatibility of each individual component must be considered. Automation is made possible only when all of the hardware interfaces are resolved. If this can be accomplished, the next step is the development of automation software.

Rendering and single-frame recording of animation can be automated through software. The development of the V-LAN mentioned earlier provides the capability to control the video production process through software and audio editing can be automated as well. The complexity and speed involved in automating these processes requires fast processors, fast programming languages, and creative and efficient software design.

FUTURE DIRECTIONS

The ultimate expression of automation would be an integrated system designed to facilitate the engineering visualization process from beginning to end. Such systems are yet to be developed but the general direction of the related industries makes their eventual appearance inevitable. Many graphics terminals in use today have the capability to support multiple screen resolutions and to accept a video synchronization signal from an external source. Some can generate their own "sync" for direct output to video. These systems are the first step towards an integrated hardware system. The software

tools are also beginning to appear. Video edit control software that uses the previously mentioned V-LAN is now commercially available. Single-frame recording software is available which drives videotape animation control hardware. Most of the hardware and software on the market now is in its first development generation.

User interfaces for these emerging tools are following the trend towards intuitive operations. Intuitive interfaces reduce the learning curves associated with complex software and, if well designed, can result in faster operations. This will be especially important to the success of integrated systems where different types of users are involved. Consistency between the interfaces for the engineer, the animator, and the videographer will result in a smoother flow of ideas and information and faster production turnaround.

Engineering visualization technology is rapidly advancing. Hardware interfaces and software formats are now being developed which will streamline the process of documenting the engineering process through animation and video. The merging of engineering analysis and simulation, 3D animation, and video is a natural step which will enhance the engineering environment. The NASA Mission Support Directorate's Animation and Video Production Facility hopes that the insight it has gained over several years of integrated efforts may represent a worthwhile contribution to the direction of this effort.

References:

Bentz, Tom, "Computers In Editing," Video Manager, Torrance, CA.,Vol. XI, No. 10, October, 1988, pp. 48-50.

Davis, Susan, "The Computer Connection," Video Manager, Torrance, CA., Vol. XI, No. 7, July, 1988, pp. 19-24.

Robertson, Barbara, "Graphics Leaps Into Space," Computer Graphics World, Westford, MA., Vol. 10, No. 8, August, 1988, pp. 44-46, 48.

Robinson, Thomas L., Sabionski, Gunter R., "Visualization Engineering Gives NASA A Window on the Future," Computer Pictures, Clifton, NJ., Vol. 6, No. 3, pp. 60-64.

Schwartz, Howard, "Audio for Video," Video Systems, Overland Park, KS., Vol. 14, No. 8, August, 1988, pp. 24-25.

Multi-Tasking Computer Control Of Video Related Equipment

Rod Molina Bob Gilbert

Exchanging audio-visual information is a daily part of human communications. The video medium is the most popular choice for quickly and effectively conveying simple or complex information. However, like other communication methods, certain conventions must be followed in order to prevent video from becoming a collection of useless information.

A completed video presentation originates from raw audio-visual material that must be organized into a continuous flow of information. The physical equipment necessary for the post-production process is usually dedicated computer-based devices that perform synchronization and control, generate visual effects, and combine audio and video signals. Most video production studios are made up of these separate devices independently performing their specific tasks.

The flexibility, cost-effectiveness and widespread availability of personal computers now makes it possible to completely integrate the previously separate elements of video post-production into a single device. Specifically, a personal computer, such as the Commodore-Amiga, can perform multiple and simultaneous tasks from an individual unit. Relatively low cost, minimal space requirements and user-friendliness, provides the most favorable environment for the many phases of video post-production.

Computers are well known for their basic abilities to process numbers, text and graphics and to reliably perform repetitive and tedious functions efficiently. These capabilities can now apply as either additions or alternatives to existing video post-production methods.

A present example of computer-based video post-production technology is the **RGB CVC (Computer & Video Creations) WorkSystem.** A wide variety of integrated functions are made possible with an Amiga computer existing at the heart of the system.

EDITING

Beginning with video editing functions, the Amiga-based editing system operates using either SMPTE Time Code or Control Track as a reference for editing points. With SMPTE, the preferred reference format, a high degree of accuracy is obtained. The **CVC WorkSystem** has full control of transport (rewind, fast-forward, play, stop, still and search) and editing functions (Insert, Assemble, Goto, Preview, Perform and Review) while maintaining a constant update of system status information. The time-consuming tasks of logging specific SMPTE Time Code values, calculating and storing pertinent edit information (edit points, match-frames, running-time and preroll) and searching through raw footage are now built-in features. The task of processing and managing the data necessary for edit decision lists and

cataloging tapes can now be delegated to specific database management software.

Custom interface hardware allows the **CVC WorkSystem** to communicate directly with the transport of a single machine or up to 32 separate transports. Normally incompatible machine formats and brands can now function within a unified system. This network environment provides a common device communications link that gives the ability to handle multiple applications and devices individually or simultaneously within a consistent operating environment.

CHARACTER GENERATION AND GRAPHICS

When incorporated properly, captions, illustrations and backgrounds can greatly enhance the impact of a video production. Almost every video production, simple or complex, requires the use of some type of character or graphics generator. Character generators are usually independant, dedicated computers that perform the sole function of adding text and possibly graphics to video. Choosing a specific character generator can be difficult. They range in price from a few thousand to a few hundred thousand dollars. The features of various character generators are also widespread. However, the general criteria that most will use in making a selection is ease-of-use, quality, and reasonable cost.

The **CVC WorkSystem's** graphics processing power and direct video compatibility provides a natural ability to fit the character generator description. Sophisticated graphics and text are easy to achieve at a fraction of the traditional cost. The real significance though, is that the graphics and character generating process can simultaneously occur from the same unit that is also performing the editing functions for a continuously flowing editing process. It is no longer necessary to switch between machines (and needing to know the subtleties and differing functions of each device in the process). Instead, the average video post-production project can be completed from one location just by switching between simultaneous applications.

ANIMATION AND EFFECTS

Besides static text and graphic images, animation capabilities are also provided with the **CVC WorkSystem.** Both 2-dimensional and 3-dimensional animation can be created quickly to allow greater enhancement and effect. Imaginary and real images can be digitally manipulated to create the popular effects and simulations that are common to many present day video productions. The quality of the animation cannot yet replace those from dedicated high-end systems but the compromise is offset by the minimal effort and time necessary to achieve results that remain pleasing and above all, cost-effective. Once again, the single system, single operating environment approach provides a tremendous amount of flexibility.

OTHER POSSIBILITIES

The **CVC WorkSystem's** open-architecture provides a great amount of expandibility. The power of the system is mainly accessed through additional applications software; keeping the necessity for extra hardware peripherals to a minimum. Simply put, the limit to the system's capabilities is dependant on the available software.

Because of its personal computer nature, other applications software exists to perform tasks that are indirectly but beneficially related to the video production process: word processing, scripting, storyboarding, database management, desktop publishing and accounting to name a few.

More direct applications could involve MIDI (Musical Instrument Digital Interface) software to help in creating soundtracks and sound-effects for the finished video. With the appropriate software the video and audio aspects of the production become a completely synchronous process.

Quality video production should no longer be considered an extravagance. The techniques and processes that were once supposedly limited to elite organizations and people are now accessible to all types of professionals. The **CVC WorkSystem** can be applied to many areas requiring the benefits of a reasonably-priced, multi-featured computer and video work-station.

Marketing agencies can now afford an in-house system to present clients with immediate ideas and recieve instant feedback. If necessary, full commercial video productions can even be produced completely on-site.

Corporate training videos can become easier and less costly to produce with a system that does not require specially trained operators.

Schools and Universities can now incorporate up-to-date video communications training into their curriculum. Unlike previous video systems, the expandable design will be well protected from obsolescence.

Scientific research institutes can also benefit from the barrage of capabilities that the **CVC WorkSystem** can help make data presentations more widely understood.

Of course, the **CVC WorkSystem** is not proclaimed as the answer to all video post-production needs, but it supplies enough significant features to make a truly integrated computer/video system a cost-effective reality. A single computer in a video studio can now be integrally involved in every step of the creative process: as a word processor creating a script and project proposal, as a spreadsheet creating the project budget, as a tape synchronizer for audio and video transports during editing, as a database to log scenes, transcribe dialog, and manage an edit decision list, as an audio sequence controller (MIDI) for electronic musical instruments used in the soundtrack and special effects, as a video effect generator and character generator, and finally as a business machine for invoicing and accounting when the project is finished.

Broadening the Interface Bandwidth in Simulation Based Training

Larry E. Somers
MICROEXPERT Systems, Inc.
24007 Ventura Blvd. Suite 210, Calabasas CA 91302

ABSTRACT

Currently most computer based simulations rely exclusively on computer generated graphics to create the simulation. When training is involved, the method almost exclusively used to display information to the learner is text displayed on the CRT. MICROEXPERT Systems is concentrating on broadening the communications bandwidth between the computer and user by employing a novel approach to video image storage combined with sound and voice output. An expert system is used to combine and control the presentation of analog video, sound, and voice output with computer based graphics and text.

We are currently involved in the development of several graphics based user interfaces for NASA, the U.S. Army, and the U.S. Navy. This paper will focus on the human factors considerations, software modules, and hardware components being used to develop these interfaces.

INTRODUCTION

Advances in military and aerospace technology continue to result in increasingly complex systems requiring quick, accurate decisions under increased cognitive loads. The amounts, variety, and rate of information flow is, many times, so overwhelming that anticipated performance benefits are not realized (Rouse 1987).

Recent advances in both video and audio storage technology are providing additional resources for communications channels between computer and user. These tools may well contribute to potential solutions of the problem. This article outlines an approach we have taken in combining these tools for the development of user interfaces, including intelligent human-machine interfaces for simulation based intelligent tutoring systems (ITS).

HUMAN FACTORS

User capacities and needs have been described as a major consideration in designing user interfaces (Shneiderman 1987). The use of several media devices can help to better meet the needs and match the capacities of the user. Described below are several of the more important factor we have considered in developing a multimedia interface.

Cognitive Load. A measure of the complexity, or difficulty of a task is the number of resources it requires (Moray 1977). As described by Baecker (Baecker 1987) the cognitive load of a task correlates with such factors as:

- learning time
- fatigue
- stress
- proneness to error.

It is important that the interface help minimize the cognitive load on the user. Thus, for example, the design should consider the different loads imposed in making menu selections with a one, two, or three button mouse, respectively. It may turn out that the one-button mouse has the lowest load, since there is no overhead in determining which button to select. However, in the larger context, it may turn out there is a greater penalty in, for example, an increased number of menus or menu selections that must be provided.

Interference. Degradation in the performance of one task can occur due to competition for cognitive resources by another task during the same time period. Problem solving requires attentive behaviors that usually involve large numbers of cognitive resources. As a result, problem solving during an ongoing simulation is highly susceptible to interference. For example a tutor that provides text for coaching during a simulation could easily interfere with the simulation reducing, instead of improving, the user's performance. In such situations an alternate communications channel using voice

output or auditory cues may provide a better approach to prompting the learner without interfering with their performance.

In working with a simulation based intelligent tutoring system, there are two classes of problems that confront the user: operational and functional. Operational problems have to do with the means of operating the ITS itself. Functional problems deal with learning to perform the tasks the tutor was designed to teach. Operational problem solving often interferes with functional problem solving.

One objective of the user interface is to minimize operational problem solving. All resources expended at this level are diverted from the functional problem for which the computer was adopted in the first place. Design features such as consistency, compatibility, icon and menu design must be considered. For example operators of certain types of radar learn to access radar target information by using a joystick to position a cursor on the target and then pressing the joystick button. We have designed a simulation to train radar operators that not only simulates this operation but also provides additional information about radar symbols and controls using a very similar procedure. If, for example, the learner desires information about a symbol he does not recognize on the simulated radar display, he need only position the cursor on the symbol, using the joystick, and press the help button on the keyboard. This type of learning requires only slight stimulus generalization and is therefor easily learned by the student.

The overhead of functional problem solving can also be reduced by careful design. Information should be presented using symbols, jargon, and metaphors that are, as much a part of the users repertoire and experience as possible. In training radar operators we have employed two expert systems, a scenario expert and an interface expert. The interface expert compares the actions of the scenario expert with the actions of the user. When a discrepancy occurs the interface expert provides visual or audio coaching, during the scenario, without the learner having to request help in any specific way. Transcripts and recording made of radar instructors as they trained operators were used to design the voice output which includes training and operation related jargon already familiar to the trainees. The result is very similar to the classroom training the operators receive in which an instructor stands behind a student and provides coaching as the student operates the radar console.

Skill Acquisition. Simulation based training generally focuses on skill development. Training procedures, including help systems, are a part of the user interface. Their design should encourage development of skills in an isolated, non-threatening way. It is important that voice and sound output, for example, not be punishing to the learner, especially by drawing attention to the learner from his peers. The result is often an avoidance or aggression response by the learner which will decrease skill acquisition.

There is some evidence that skill is acquired more rapidly in an isolated learning situation (Schneider 1985). This may not hold for specific cases and requires testing for final validation. High-fidelity simulations are ultimately important in order for the advanced student to learn fine discriminations. However, for the novice it is often important to reduce the complexity of the simulation so that the student can more easily learn to make important preliminary discriminations. In training radar operators, the complexity of the simulation scenario is controlled by the interaction expert. As the student becomes more successful at solving the scenario correctly, the complexity is increased by adding additional targets and target types and by changing target vectors. If a student has difficulty with a specific scenario, the scenario is simplified so that important stimuli are isolated and the student can more easily focus on appropriate discriminations to be learned.

Mental Models, Analogy, and Metaphor. The underlying conceptual model of the software is considered to be a more important factor in user-friendliness then what is generally called "look and feel" of the system (Liddle 1989). The mental model which the user applies in trying to understand and predict systems behavior is an important consideration in the design of the interface. Users make use of analogy between systems components and previously learned stimulus-response paradigms, when operating a system. To the extent that the user interface can be designed using one or more carefully chosen metaphors familiar to the user, the interface will be perceived as user-friendly. In designing the user interface to multimedia database, in which the user can access analog video images, graphics, voice, sound, and text, we have employed the metaphor of a Library. A metaphor of a card catalog is used to specify

the information used for a database search. Following the search a graphical representation of library books on a shelf, representing the results of the search, is displayed on the screen. By pointing the cursor at a book and clicking, with a mouse, the information, be it text, sound, voice, or image, is displayed to the user. Though still in the prototype stage, preliminary user acceptance has been very positive so far.

S-R Compatibility. When a systems cause-and-effect behavior matches the user's expectations and previous experiences, it has good stimulus-response (S-R) compatibility. Two main factors to be considered are spatial congruence and custom. Having good spatial congruence between items in a menu and the layout of function keys provides good S-R compatibility. The use of the color red to indicate danger or a stop action is an example of how custom can be used to provide good S-R compatibility. In a similar fashion the user interface should be designed to make use of customs specific to the individuals that will utilize the system. Through careful knowledge engineering it is sometimes possible to uncover customs peculiar to the target group of users. To the extent that these customs can be incorporated into the interface it will be perceived as user friendly.

INTERFACE COMPONENTS

The diagram shown in figure 1, below, illustrates the functional modules we have used in developing intelligent human-machine interfaces. Each module is a unit of replaceable code with specified inputs, outputs, and functions to perform. Furthermore the interface, itself, can be seen as a module in the development of a larger intelligent tutoring system. In this way other groups are able to work separately on different modules of the ITS.

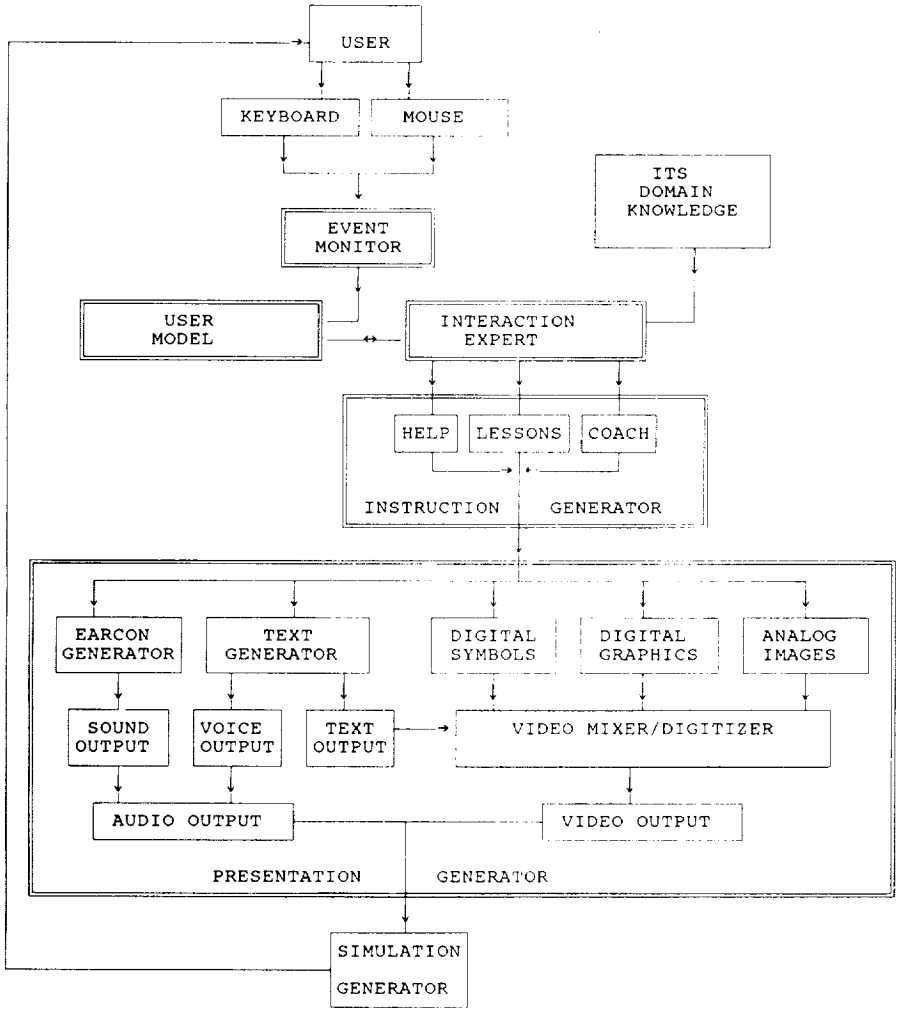


FIGURE 1.
Interface Components

Task Analysis.

While not represented as a separate interface component, a careful task analysis is essential to the development of the other components in the system. Intelligent Tutoring Systems attempt to capture and explicitly represent the knowledge that constitutes the expertise being taught. Our knowledge engineering efforts have focused on a task analysis that not only identifies the knowledge components to be represented, but creates a curriculum structure that associates knowledge components with each other and with the goals of the instruction.

During the knowledge engineering phase of development complex, high-level tasks are identified and decomposed into mid-level and then low-level unit tasks. For each unit task it is important to identify a measurable behavior associated with the task, the stimulus conditions upon which that behavioral response should be made, and the heuristics that describe the relationships between stimuli and responses. The process is an adaption of the goal-lattice structure described by Lesgold (Lesgold 1988). Each high-level task serves as the root node of a tree. Simple lessons are designed to teach the unit tasks of each tree. Many of the mid-level and unit tasks identified in one task tree are also common to other, separate, task trees. Figure 2, below, shows this architecture symbolically.

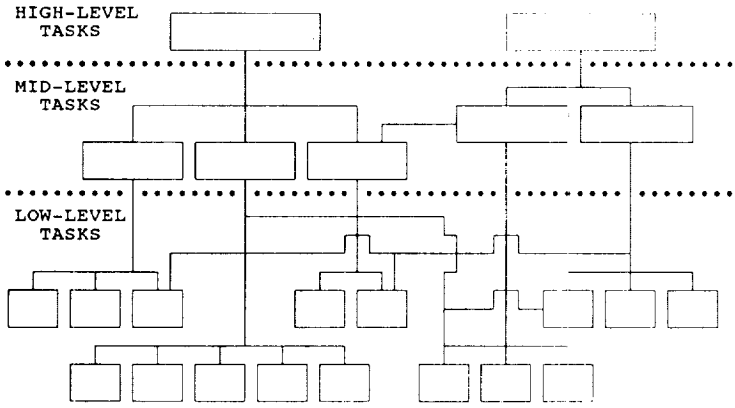


Figure 2.

Task Analysis

The resulting task trees and interconnections make up a curricular structure for the ITS which is accessed in the interaction expert. Tasks can be taught using a depth-first search, a breadth-first search, or both. Research is being carried out to determine, among other things, under what condition a specific search should be carried out.

User Input.

Prototype development has been carried out on a Symbolics LISP machine, DEC MicroVax. User input has been limited to a mouse pointing device and keyboard. We are currently developing a new type of wireless pointing device to be implemented when porting the interface to a PC. We are also considering voice input devices for entering commands on the PC.

Event Monitor.

The event monitor measures user and simulation event actions over time. Multiple timing functions are available to measure the elapse time between a task stimulus event and a specific user response (task time), between the start of sequential tasks (intratask time), to measure input from the keyboard and mouse, to determine the current task to be performed, the current position of simulation related objects on the display, and which object the cursor is pointing to at any given moment. Information measured by the event monitor is then stored in the user model.

User Model.

The user model is used to store task performance related data about the user. For each task performed, the time required by the user to complete the task is stored. The sequence of user performed tasks is also stored and used to calculate a task efficiency and task similarity (compared to an expert) rating. The time period between presentation of successive task stimulus conditions is also measured and provides an indication of the cognitive load on the user. This provides a user-specific fact base that is used by the interaction expert to adapt to individual user requirements and needs.

Also stored in the user model is data related to the users presentation preferences. As is described below, information can be presented to the student in a variety of modes, textual, graphical, voice, and sound. The user model is designed to measure the users preference for a specific mode of presentation as defined by his performance following the presentation.

The users teaching history is also tracked in the user model. Thus the tasks that have been taught, the presentation modes that have been used, the students task performance, and his presentation preferences are stored here and available to the interaction expert.

Interaction Expert.

The interaction expert is the interface rule base. Rules are designed to compare the users task performance with that of an expert. An expert system, designed as a separate component of the ITS (not shown), generates expert solutions that are available to the interface expert. The expert's solution is compared with the users solution to determine the tasks to be taught. By traversing the curriculum lattice the interaction expert determines related tasks that should be taught as well as different paths (viewpoints) from which to teach. The user model is then consulted to determine what paths have not been previously attempted for that user and what presentation mode should be tried.

Instructional Generator.

The instructional generator is primarily a database of instructional components designed to teach specific tasks. Instructional modules are designed to provide several instructional strategies; discovery learning, coaching, and Socratic dialog. Thus, several instructional modules are available for each task. Modules are also designed to differ in their emphasis of a specific presentation media. For example, coaching is available for a given task by presenting text on the video display or through voice output using a text-to-speech converter.

Presentation Generator.

The presentation generator consists of the media devices used to present information visually or auditorily along with software used to control these devices and integrate components.

Visual Channel. Both analog and digital, bit-mapped video images are available for display to the user. Currently different video display terminals are used for each. We are experimenting with both video digitizing boards and video mixers to combine both types of images onto one display.

Video. A unique video storage device, the VIEWBOX 2000, is being used to capture and display analog, RS-170, video images. The device uses a standard 20-Mbyte hard disk with a modified controller to store over 2400 RS-170 video images. Random access times are approximately 200 msec and sequential access times are under 100 msec. making a "pseudo-animation" possible. A standard video camera is used to capture images. Software drivers in the presentation generator are used to control the device over the computers RS 232 port.

Graphics. Graphic displays are highly machine dependant. Interfaces are currently being designed on both Symbolics and DEC MicroVax computers, using monochrome graphics, and on PC's using EGA color graphics. Currently simulation are graphics based and the VIEWBOX is used to display visual information that does not lend itself well to graphical display due to processing requirements and capabilities. We are experimenting with using the VIEWBOX to provide background scenery overlayed with graphics in the hopes of combining both in the future.

Symbology. Icons and symbols are separate graphical components the interface uses to help the learner make important discriminations during the simulation. Simulations are designed with varying complexities. Novices are provides simulations of very low complexity with ample use of symbols, such as pointers. While it is generally agreed that high-fidelity simulations are needed, it is possible too provide to much fidelity early in the learning process.

Text. Under the control of the instructional generator text can be displayed in a window on the video display or sent to a text-to-speech converter and presented as speech. In the later case the presentation generator formats the text string to control pitch, rate, and other parameters.

Audio Channel

Producing Speech Electronically. Generation of speech and sound (earcon) output from a computer requires special hardware components. Three major techniques for production of speech have evolved over the years: formant (resonant frequency) synthesis; linear predictive coding; and waveform sampling. Most commercial text-to-speech devices use one of the first two because they require smaller storage and slower data rates. However with as computer memory continues to decrease in cost, computer systems such as the Atari and Apple's MacIntosh are imbedding the hardware and software needed to sample and reproduce waveforms.

Synthetic Speech. The automatic conversion of text to synthetic speech has advanced remarkably in the last several years. A number of commercial devices are now available, ranging in cost from approximately $100 up to $35000. Progress in this area has resulted from advances in linguistic theory, acoustic-phonetic characterization of English sound patterns, perceptual psychology, mathematical modeling of speech production, and computer hardware design (Klatt 1987). Never-the-less a number of scientific problems remain that prevent current systems from achieving the goal of completely human-sounding speech.

The quality of voice output improves greatly in devices costing over $3000 (Kaplan et al 1987). In the $3000 - $4000 price range two text-to-speech devices stand out. Originated by Dennis H. Klatt, speech synthesis expert at MIT, DECtalk by Digital Equipment Co. has a broad range of voices including a child's voice and a female voice. In evaluations by Nusbaum et al (1984) listeners understood synthetic speech produced by DECtalk 97.7% of the time as compared to 99.4% for human speech. A rival system also originated by Klatt, the Prose 2000 by Speech Plus Inc. has similar quality but offers only a male voice and is slightly less expensive. Studies by Logan et al. (1986) indicate listeners have an error rate of 6% listening to the Prose 2000 - 3.0 compared to 1% error in understanding natural speech. Both devices can be controlled thorough the computers RS-232 Serial Port and require a data rate of approximately 100 bits, based on a typical rate of 12 phonemes per second.

We are currently using the Prose 2000 for text-to-speech conversion in several of our interface. A major advantage to these type of devices is the ability to use variables to store speech output. The major drawback of these devices is that they are limited in their ability to produce other complex sounds that would be useful for generating auditory cues.

Voice Sampling. A second method of producing digitized voice output is by sampling the waveform of human speech. Waveform sampling uses a common analog-to-digital

conversion and requires about 64000 bits per second for uncompressed speech (8000 samples per second to capture up to 4000 Hz, multiplied by 8 bits per sample). Thus storage requirements would be 8K/second. Using a dedicated microcomputer containing a 20 megabyte fixed disk approximately 2500 seconds of speech could be digitally recorded using this method. Using data compression techniques, this number could be doubled. The results are a digital recording of the speech that is almost indistinguishable from the original source.

We are currently using an Antex Model VP 620E, PC compatible digital audio processor (Antex Electronics, Gardena, CA) to provide digital audio in some interfaces. While this type of device eliminates the ability to easily store speech components as variables, the high quality sound makes the device ideal in many teaching situation and where sophisticated auditory cues are desired.

Earcons. Sound is increasingly being used to convey information in computer interfaces. The term Earcon (Sumikawa 1985) has been used to define sounds that serve as the auditory equivalent of Icons. Similar to voice generation, earcons can be produced by sampling specific sounds or synthesizing sounds with a tone generator. Gaver (1986) has classified auditory icons into three groups: 1) symbolic, such as telephone bells and sirens, 2) nomic, in which the sound is a physically caused by the source such as the sound arriving mail makes in a mailbox, and 3) metaphorical such as a change of pitch used to represent falling or a hissing sound to represent a snake. Symbolic sounds are, perhaps, easiest to produce on most computers since they do not require the ability to sample sounds. However they generally require the greatest amount of learning on the part of the user. For this reason they should be used judiciously. Symbolic sounds have been shown to be effective when used as an alerting cue prior to emergency messages produced by synthesized voice (Hakkinen 1984). Anecdotal evidence from our current research supports these finding but also suggests that overuse of sound stimuli results in confusion of the user. We are now beginning to experiment with sampled sounds to produce nomic and metaphorical earcons which should require less learning by the user.

CONCLUSION

A generic intelligent multimedia interface has been described. While research is still ongoing in many cases we have reach so interesting preliminary conclusions. We originally believed that selecting different presentation modes, e.g. voice or text, would be useful for adapting to specific types of learners. However results so far suggest that user performance improves much quicker when several modes, e.g. voice and text, are combined. This makes sense in light of the fact that the learner then comes under multiple stimulus controls.

A second factor, eluded to above, that became immediately noticeable was that earcons and auditory cues can easily be over used and become distracting to the user. However, when designed carefully, and used fastidiously, they can be of significant value in gaining the learners attention and improving his performance.

REFERENCES

Baecker, R.M., Buxton, W.A.S., "Cognition and Human Information Processing," READINGS IN HUMAN-COMPUTER INTERACTION, Morgan Kaufmann Publishers, Los Altos, CA, 1987, pp.209.

Hakkinen, Markku T., Williges, Beverly H., "Synthesized Warning Messages: Effects of an Alerting Cue in Single- and Multiple-Function Voice Synthesis Systems," HUMAN FACTORS, Santa Monica, CA, 26, 2, April, 1984, 185-195.

Kaplan G., Lerner, E., "Realism in Synthetic Speech," in HUMAN-COMPUTER INTERACTION, R.M. Baecker & W.A.S. Buxton Eds. Morgan Kaufman, Los Altos, 1987, p. 414-419.

Klatt, Dennis H., "Review of text-to-speech conversion for English," J. ACOUST. SOC. AM. 83(3), 1987, pp.737-793.

Lesgold, Alan, "Toward a Theory of Curriculum for Use in Designing Intelligent Instructional Systems," in LEARNING ISSUES FOR INTELLIGENT TUTORING SYSTEMS, 1st Ed., Springer-Verlag, New York, NY, 1988, pp. 117-118.

Logan, J.S., Pisoni, D.B., "Preference Judgements Comparing Different Synthetic Voices," J. ACOUST. SOC. AM., Suppl. 1, 79, 1986, p. S24.

Liddle, D., in LaPlante, A., "Ease of Use Involve More Than Graphical Interface," INFO WORLD, Menlo Park, CA, 11(7), Feb. 13, 1989, p.1.

Moray, N., WORKLOAD MEASUREMENT, New York, Plenum, 1977.

Nusbaum, H.C., Pisoni, D.B., and Schwab, E.C. "Subjective Evaluation of Synthetic Speech: Measuring Preference, Naturalness, and Intelligibility," Speech Research Progress Report 10, Indiana Univ., Bloomington, IN, 1984, pp. 391-408.

Rouse, William B. Norman, Geddes, Norman D., & Curry, Renwick E., "An Architecture for Intelligent Interfaces: Outline of an Approach to Supporting Operators of Complex Systems," Human-Computer Interaction, Vol. 3, 1988, pp. 87-122

Schneider, W., "Training High-Performance Skills: Fallacies and Guidelines," HUMAN FACTORS 27(3), 1985, pp. 285-300.

Shneiderman, Ben, "Human Factors OF Interactive Software," DESIGNING THE USER INTERFACE, Addison-Wesely, Menlo Park, CA, 1987, pp. 4-11.

Sumikawa, D.A., Blattner, M.M., Joy, D.I., Greenberg, R.M., "Guidelines for the Syntactic Design of Audio Cues in Computer Interfaces," Lawrence Livermore National Laboratory, 19th Hawaii International Conference on System Sciences, Oct. 3, 1985.

Animation Graphic Interface for the Space Shuttle Onboard Computer

Jeffrey Wike
Paul Griffith
MICROEXPERT SYSTEMS INC.
24007 Ventura Blvd, #210, Calabasas, CA 91302
(818)712-9934

ABSTRACT

Graphics interfaces designed to operate on space qualified hardware challenge software designers to display complex information under processing power and physical size constraints.

Under contract to Johnson Space Center, MICROEXPERT Systems is currently constructing an intelligent interface for the LASER DOCKING SENSOR (LDS) flight experiment. Part of this interface is an graphic animation display for Rendezvous and Proximity Operations. The displays have been designed in consultation with Shuttle astronauts. The displays show multiple views of a satellite relative to the shuttle, coupled with numeric attitude information. The graphics are generated using position data received by the Shuttle Payload and General Support Computer (PGSC) from the Laser Docking Sensor.

Some of the design considerations include crew member preferences in graphic data representation, single versus multiple window displays, mission tailoring of graphic displays, realistic 3D images versus generic icon representations of real objects, the physical relationship of the observers to the graphic display, how numeric or textual information should interface with graphic data, in what frame of reference objects should be portrayed, recognizing conditions of display information-overload, and screen format and placement consistency.

INTRODUCTION

While much research is being funded to advance the state-of-the-art in realtime graphic workstations, these systems are not appropriate for onboard graphic displays to assist crew members in space. Hardware aside, issues concerning display content, efficiency and practicality must be addressed when considering graphic technology in space. MICROEXPERT Systems Incorporated is conducting research in the use of expert systems and graphical data portrayal on microcomputers to aid Orbiter crews in interpreting sensor data output in space. The application, funded by NASA Johnson Space Center, is to develop a human interface to the LDS. The LDS is a laser radar designed to measure the relative position between NASA's Orbiter (Shuttle) and a target (Space Station or satellite) during rendezvous and docking operations. The project software, called the Laser Docking Sensor Associate (LDSA), displays the LDS data under LDSA control. The system assists the crew by monitoring the data for fault and safety problems and provides crew input to the sensor. This paper outlines the approach taken in designing the graphic interface of the LDSA, and identifies peculiarities typical in designing graphic interfaces for flight qualified hardware.

SYSTEM CONSTRAINTS FOR FLIGHT QUALIFIED GRAPHICS

The new orbiter onboard computer that allows graphic displays is the Payload and General Support Computer (PGSC). The system is a GRiD CASE Model 1530 laptop computer. It is equipped with a 12.5 MHz 80386 32-bit processor with 80387 math co-processor. The display is a

10 inch diagonal 640X400 pixel, backlit LCD display. The interface to external data is via RS-232 or RS 422 serial ports. The operating system is GRiD MS-DOS Version 3.21D.

The PGSC imposes the following limits on graphics applications and design:

o The processor size severely confines the amount of detailed graphic animation that can be displayed.

o The display does not support color graphics.

o The physical display screen size limits the detail in which the graphics can be presented.

o Communication via serial port restricts the amount of data available to the system. This limits the types of graphic applications the system can support. Graphic portrayal of externally digitized images for example, cannot be directly transferred in reasonable time.

o Input to the system is through the keyboard only. No mouse, track ball, voice or touch screen is available for the system. This limits user input to keyboard only and creates a need for menus, function keys and mnemonics.

o Because the PGSC is not currently connected to the orbiter General Purpose Computer (GPC), applications run on the PGSC are restricted to those which do not involve orbiter safety and mission data.

CHARACTERISTICS FOR ORBITER MMI DESIGN

As a prelude to building an Man Machine Interface (MMI) for the LDS, MICROEXPERT proceeded through a knowledge gathering process to learn more about the system being represented, the presentation preferences of users, and representation standards.

Gathering knowledge about a system yet to be built increases the difficulty of gathering definitive system information and expertise. In a large departmentalized organization it is always time consuming to locate the 'experts' with the proper knowledge required to impact system display design. When a new system is being built there is no expert and the knowledge has to be assembled from a variety of sources.

MICROEXPERT's knowledge engineers made several design trips to JSC to talk to the large variety of groups and departments with an interest in Shuttle displays. Personnel who participated in the design included four astronauts plus engineers from Proximity Operations and Shuttle Display Groups. We showed them preliminary designs and elicited comments. The final version of the display design resulted from iterative design and user input. In some cases, rules of display design yielded to astronaut expectations.

Our investigation into design preferences and requirements led to the following observations:

o Astronauts prefer graphical representation of data. Three dimensional graphic models that depict orientation are more meaningful to the astronauts than numeric data. Engineers prefer to see numeric and tabular data, and do not see the graphics as very meaningful.

o Numerical data should be displayed only to the accuracy that is relevant. Just because data may be available to three places after the decimal point, if the crew member can only affect a change in the data to the nearest integer, that is the only value necessary to display. The 'insignificant' significant figures only clutter the screen.

o Although uniformity of numerical data makes for a symmetrically pleasing display, crew members prefer receiving the exact data as required.

o Critical data, for example range and range rate in our application, can be emphasized by displaying them in large numbers on the graphic screen in head-up display fashion. Not only does this permit the crew member to easily find numeric data without changing the

graphic view, but allows other crew members to monitor the values over the shoulder of the prime user.

o Critical information that appears as a demon or warning should require positive confirmation by crew members using the system.

o Displays should be stuffed with as much information as possible to avoid changing display screens. However, astronauts should be able to declutter the screen during certain operations of the display.

o Information displayed graphically on the screen should be placed in multiple or selectable frames of reference. Astronauts prefer that most graphic displays be in shuttle coordinate orientation. For proximity and docking, all readouts should be 'fly-to' oriented. This means that the goal of the display should be to put the target in the middle of the crosshairs, or zero out all the numerical data.

o A conflict arises between generic graphic displays, and mission specific displays. Generic displays have the advantage of providing ease of training, standardization, and reusability of software. Graphics that pertain to a single mission requirements can provide a much greater level of detail and specificity to the graphics, and impart more information to the crew member. In other words, there are no absolutes for screen design in real space applications.

GRAPHICS APPLICATION: LASER DOCKING SENSOR

The above considerations were incorporated into the design of the graphic interface to the LDS. The purpose of the LDS is to improve measurement accuracy over the current docking methods using the KU band radar, the Crew Optical Alignment Sight (COAS) and external telemetry. The system goal is to achieve soft docking with a target, reduce docking time requirements, and conserve fuel while maintaining safety.

The LDS measures range, azimuth, elevation, roll, pitch, yaw and associated rates to a docking target. To achieve accuracy over a dynamic range, the LDS design calls for a complex integration of several measurement systems.including the Distance Measuring Equipment subsystem, which is a laser radar with multiple tones to measure the range and a doppler to measure range rate and the Long Range Bearing System (LRBS), consisting of an illuminator and camera to capture a video image of the target for determining bearing beyond 80 feet, Short Range Bearing System (SRBS) for calculating bearing and attitude, and several optic and microprocessor subsystems.

The LDS communicates with the PGSC via an RS-232C link. It outputs fixed formatted data packets at a 1HZ update rate. The PGSC can send to the LDS the following mode commands: STANDBY, SEARCH, BREAK TRACK, SELF TEST, CALIBRATE, and SEND VIDEO. Normally the LDS operates autonomously without input from the PGSC. The LDS also sends the data packet to the Payload Data Interface (PDI). The LDS connects to the aft flight deck switch panel via panel discrete mode selection input.

PGSC DISPLAY DESCRIPTION

The PGSC runs MICROEXPERT's realtime LDSA program. The LDSA has four main functions:

- o Communication with LDS
- o Display of the LDS data
- o LDS data analysis using expert systems
- o Satellite target recognition.

The LDSA MMI presents the LDS data in graphical and tabular form with critical data enlarged. The relative position of the target is presented from more than one perspective and coordinate system. The menu enables user input.

The LDSA expert system checks the data for validity, trends and dangerous situations. The target recognition software validates the target and calculates its attitude.

MICROEXPERT designed two interfaces for this system: one mission specific, the other a generic proximity operations display. Both

MMI's were designed around the knowledge gained from on site interviews, MICROEXPERT's experience in complex displays for tutoring, and general principals of display and expert system design.

The mission specific interface consists of several full screen graphical views with scales and numerical readouts. These windows are:

o The front view from the perspective of the LDS (see Appendix A) displaying a three dimensional wireframe model of the target vehicle. The model is scaled, transformed and oriented in roll, pitch and yaw from the LDS measurement. The model appears in real time as the satellite would to a mission specialist observing it through the COAS.

o The side view depicting the position of the shuttle and the target vehicle from a point on the azimuth axis.

o The readout screen, listing all the data, sensor status and a history of the LDS status and mode changes.

The generic interface (see Appendix A) consists of one main display screen with six main windows. In the graphics windows icons portray the relative positions of the vehicles. The target icon always includes a halo to indicate the deadband. Scales to the side of the graphics indicate the measurement, its rate, and direction of change. The windows are:

o Side view indicating the Z and X coordinates of the target in graphics and a linear scale.

o Top view graphing the target relative to the shuttle in X and Y as seen from above the shuttle (-Z). The field of view of the LDS is outlined by a dashed box.

o Sensor data listing measured parameters (range az., el., etc.) sent from the LDS in a columnar readout. The values are converted to the cartesian coordinate system of the shuttle and displayed.

o P/L Relative displaying target attitude values in columnar readouts with corresponding rate values.

o LDS FOV plotting the target in the LDS's field of view.

o Status and Warning indicating the mode of the LDS and suggestions from the expert system.

CONCLUSIONS

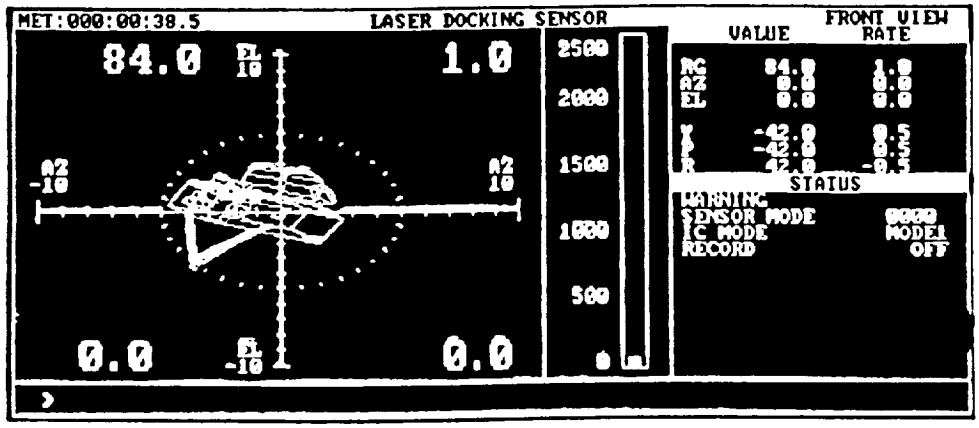
Designing animation graphic interfaces for space applications creates considerations that may impact the design of the interface and disallow the current state-of-the-art in color animation graphics. In spite of this, graphic interfaces can be applied to onboard, realtime software applications with strongly positive results.

The 80386 processor in the PGSC is capable o running a variety of sophisticated programs that could aid Orbiter crews. The terminate-and-stay-resident display, interface software, and data analysis routines in the PGSC run "simultaneously" by servicing interrupts to share processor time. Displaying multiple 2D views, or 3D wireframes can provide a graphical representation, while omitting detail prohibited on small machines.

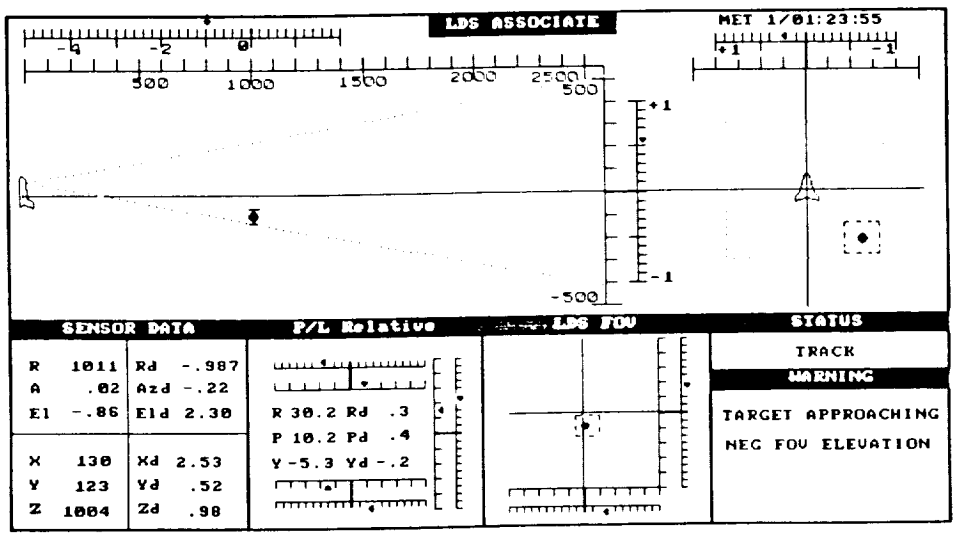
On board graphic software for data representation is relatively new to the Shuttle program. Animated graphic representation o data brings a more intuitive understanding of the data to crew members, and should be carried into more onboard software systems. Integrating crew member desires into the display design creates an efficient, tailored display that is will provide graphics to better aid the crew. Because different players have different needs and display interests, graphics standards should be created for the PGSC and adapted for all payload support software. This will improve and reduce the cost of display design, increase acceptability, and enhance training.

ACKNOWLEDGEMENTS

Our thanks to Silvio Nunes, Jane Chang and Dave Stevens for their programming support.



CGA Mission Specific Display



EGA Generix Proximity Operations Display

OPERATIONAL COMPUTER GRAPHICS IN THE FLIGHT DYNAMICS ENVIRONMENT

James F. Jeletic

NASA/ Goddard Space Flight Center
Flight Dynamics Division
Code 552.2
Greenbelt, MD 20771

ABSTRACT

Over the past five years, the Flight Dynamics Division of the National Aeronautics and Space Administration's (NASA's) Goddard Space Flight Center has incorporated computer graphics technology into its operational environment. In an attempt to increase the effectiveness and productivity of the Division, computer graphics software systems have been developed that display spacecraft tracking and telemetry data in 2-d and 3-d graphic formats that are more comprehensible than the alphanumeric tables of the past. These systems vary in functionality from real-time mission monitoring systems, to mission planning utilities, to system development tools. This paper discusses the capabilities and architecture of these systems.

1. INTRODUCTION

Since the mid 1960s, Flight Dynamics Division personnel have been performing spacecraft orbit and attitude determination and a variety of mission planning, monitoring and analysis functions. These functions have often been based on the analysis of large volumes of numerical data. These data represent a wide variety and range of geometric values, some as easily interpretable as the position of the sun in a sun sensor's field-of-view, others as abstract as spacecraft attitude expressed in quaternions.

In the past, operators and analysts have been presented these data values via monochrome screens of alphanumeric tables. Today these displays are now 2-d and 3-d color graphic representations of the data. In an ongoing effort to increase the efficiency and effectiveness of this working environment, the Flight Dynamics Division has invested in an endeavor to utilize computer graphics technology as a means to present flight dynamics data in a more comprehensible format.

This paper discusses how graphics technology has been applied to the flight dynamics environment. Presented in detail are graphics software systems that are currently in use in the Flight Dynamics Operations Area. These systems have been separated into three distinct categories. The first, real-time mission monitoring systems, encompasses distributed processing software that receives and graphically displays real-time spacecraft telemetry data. These systems are used for ensuring the health and safety of a spacecraft and verifying the quality of experiment data. The second category, non-real-time planning tools, includes passive standalone software systems that are used for various mission planning and analysis activities. The final category, system development tools, contains high level subroutine packages used by Division programmers to create frequently incorporated graphical displays in a cost effective manner.

2. MISSION MONITORING SYSTEMS

The Flight Dynamics operations personnel are often required to interpret tracking and telemetry data as it is received on the ground. The interpretation of the data is necessary to ensure the integrity of experiment data, verify attitude maneuvers, and monitor the health and safety of a spacecraft. Computer graphics systems have been applied to four specific applications to assist analysts with this interpretation process. These applications are further discussed in this section.

2.1 TCOPS WORLD MAP

2.1.1 BACKGROUND

One of the most common real-time analysis problems faced by Flight Dynamics operation personnel is the determination of a spacecraft's position above the earth and whether that location is within communication range of ground or satellite-based antenna. To help visualize this problem, a world map display was incorporated into the Trajectory Computations and Orbital Products System (TCOPS), the Flight Dynamics Division's institutional orbit determination system.

2.1.2 CAPABILITIES

The underlying principal for the world map display is to generate a 2-d Cartesian projection of the earth then overlay orbit tracks of various spacecraft onto this projection. The orbit tracks are propagated and

the current location of the spacecraft is updated as real-time position data is predicted analytically. Communication zones are drawn as a set of contours that take into account any interference that may be due to obstacles blocking either ground or space-based antenna. These obstacles are both tangible (e.g., mountains or buildings) and abstract (e.g., atmospheric interference). The world map display also includes electromagnetic radiation contours, a sunrise/sunset terminator line and sun icon. The system also predicts shadow constraints and possible communication obstruction due to solar interference (see Figure 1) [4].

2.1.3 ARCHITECTURE

The TCOPS world map incorporates a distributed processing approach (see Figure 2). Spacecraft orbit vectors are retrieved from spacecraft ephemeris files by a FORTRAN program (WMDRV) which is executed on a National Advanced Systems (NAS) 8063 mainframe computer under the MVS operating system. The orbit vectors are then transmitted over a bisynchronous 9600 baud communications line to an IBM PC/AT compatible workstation.

TCOPS WORLD MAP ARCHITECTURE

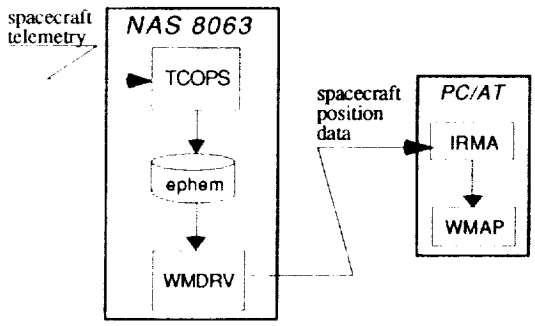


Figure 2.

The PC is configured with a Digital Communications Associates (DCA) IRMA communications board and an ATI Technologies, Inc. Enhanced Graphics Adapter (EGA) Wonder board (resolution of 640 X 350 pixels). This specific version of the EGA board is compatible with the closed circuit television (CCTV) system at the Goddard Space Flight Center, allowing the image to be transmitted to multiple control centers. A FORTRAN program (WMAP) on the PC, executed under DOS, then generates and continually updates the world map display using an orbit propagator to predict the location of the spacecraft [4]. All text and graphics are produced using the Media Cybernetics, Inc. HALO graphics package.

2.2 3-D MON

2.2.1 BACKGROUND

In contrast to the simplicity of the world map system and its related analytical support is the problem of verifying, in real-time, such items as: relative position and orientation of a spacecraft (and its appendages) to celestial bodies; objects and targets along an instrument's boresight; and solar lighting constraints.

To alleviate the time consuming and difficult task of determining such alignments by examining numbers, the Flight Dynamics/Space Transportation System 3-D Monitoring System (3-D Mon) was developed to display real-time spacecraft data with some degree of photographic realism. 3-D Mon presents a 3-d model representation of the Space Shuttle, its payloads and surrounding environment using near real-time Shuttle telemetry (received every two to five seconds) to compute the orbit and attitude of the models [8]. The system can also accept other satellite telemetry streams for spacecraft other than the Shuttle.

2.2.2 CAPABILITIES

The primary capability of the 3-D Mon system is to generate realistic 3-d images of the Shuttle, the Remote Manipulator System (RMS), and the Shuttle's payloads based on Shuttle telemetry data. These objects are shown at their relative sizes, orientations, and positions. All of these objects can be displayed as solid, flat shaded objects, with shading based on light sources located at the sun and/or viewpoint. The viewpoint light source prevents objects from appearing as silhouettes when the sun and viewpoint are positioned on opposite sides of the model. The objects also can be depicted in a wireframe representation if system performance needs to be increased or if a transparent object provides an improved analytical view [2]. The capability for Gouraud shading is currently being incorporated.

The next capability of 3-D Mon is to merge these spacecraft images with accurate representations of the surrounding environment. The earth is displayed in its accurately scaled size and position and is rotated appropriately. Land masses can be displayed as filled or outlined, with or without day/night shading. Interference zone contours and longitude/latitude lines also can be overlaid onto the earth's surface. Images of the celestial bodies (sun, moon, Mars, etc.) and other celestial objects (galaxies, quasars, etc.) are represented as 2-d icons or alphanumeric characters, respectively, at their relative positions. Celestial body positions are based on ephemeris files that precisely predict their location. The sun and moon icons also are displayed in their properly scaled size, with lunar phases (full moon, crescent moon, etc.) displayed upon request. Stars are rendered as groups of pixels whose sizes are varied proportionally to the brightness of the star. Vectors may be added that represent the direction of the sun, earth, targets, spacecraft velocity, etc. to provide a relative indication of motion with respect to the universe [2]. Figures 3 and 4 are images generated by the 3-D Mon system that merge both spacecraft and environmental data.

Interactive capabilities for analysts are also provided by the 3-D Mon system. An analyst can toggle any of the aforementioned system configurations, whether for system performance or analytical considerations. The analyst can specify the current

view in a variety of ways, which include predefined views (RMS wrist camera view, rear Shuttle cockpit window view, communications satellite view, etc.) or a user-specified view where the user can select the viewpoint and point of interest. Analytical information is provided to the operator for any object selected interactively. Playback modes are provided for analysts to review previous scenarios in either a slow motion or frame-by-frame (data record-by-data record) mode [6].

2.2.3 ARCHITECTURE

To achieve the required image update rates, several key design concepts were incorporated in both hardware and software. The 3-D Mon system is a distributed processing system that consists of a FORTRAN program executed on a NAS 8063 computer under the MVS operating system and a set of C programs executed on a Silicon Graphics IRIS 4D/60 Turbo workstation under the UNIX System V operating system. (See Figure 5.) The mainframe program - the Data Acquisition and Transmission (DAT) program - acquires the spacecraft telemetry and environmental data and strips out or processes key parameters necessary to generate the graphics displays. The program then transmits these parameters to the IRIS workstation over an asynchronous 2400 baud line. The IRIS software receives the data from the mainframe computer and then generates the display using calls to IRIS Graphics Library routines. User interaction is conducted with either a mouse to control pop-up menus or a dial box to facilitate zooming, panning, rotating and trucking of the images [2].

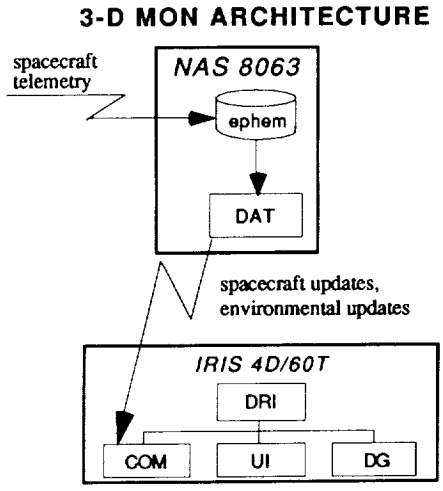


Figure 5.

The IRIS-resident software consists of three major subtasks that execute as concurrent UNIX processes - communications (COM), user interface (UI) and display generation (DG). These tasks are monitored by a parent task (DRI) and communicate with each other via UNIX pipes. The multitasking approach allows data receipt and display updating to occur simultaneously. This approach also allows the screen to be updated while a user interactively selects system options on a set of pop-up menus [2].

A simple 2-d version of 3-D Mon also exists at the Goddard Space Flight Center and is referred to as the 2-D Graphics Monitoring System (2DGMS). Modeled after another similar 2-d system in use at the Johnson Space Center, 2DGMS provides three predefined views along the Shuttle's x, y and z axes. This system has an architecture similar to the TCOPS world map system (residing on a PC/AT) and is executed simultaneously with 3-D Mon to provide additional visual support if needed.

2.3 PAYLOADS MM

2.3.1 BACKGROUND

Under NASA's Shuttle-Attached Payloads Program, government organizations and educational institutions can place scientific experiments in the cargo bay of the Space Shuttle [10]. Associated with these experiments are several constraints affecting the safety of the instrument and also the integrity of the data collected. Examples of such constraints are: no oxygen molecules can impact an instrument to avoid damage to its crystal lining; no data can be collected while the earth is occulting an instrument's field of view to avoid erroneous data values; and no ultraviolet light can enter an instrument's field of view to avoid damage to spectrometers [9]. The problem of monitoring all of these constraints simultaneously in real-time prompted the need for the Attached-Shuttle Payloads Mission Monitoring System (PAYLOADS MM).

2.3.2 CAPABILITIES

The PAYLOADS MM system generates six types of 2-d displays that depict the instrument environment. These displays are used to determine which objects, either real (sun, moon, etc.) or abstract (radiation regions, Shuttle velocity vector, etc.), are within the field of view of the instrument or are causing interference between an antenna and a communications satellite. Unlike the previously mentioned 3-D Mon system, photographic realism does not make a significant contribution to the analysis of such constraints, therefore 2-d rather than 3-d images are sufficient. Figures 6 and 7 present two types of displays generated by the PAYLOADS MM system.

Similar to the functionality of the 3-D Mon system, orbit and attitude of the Shuttle model are derived from near real-time Shuttle telemetry data. Additional payload telemetry streams are also captured and used for detailed information about the configuration of the instrument. The telemetry data are normally received at time intervals varying from two to 30 seconds [9]. Environmental data are retrieved from ephemeris files or computed by highly accurate analytical routines on an as needed basis.

The six types of displays can be cycled through, and simultaneously updated on up to six graphics devices. This capability allows all six displays to be viewed concurrently or one display to be configured in multiple ways. The dwell time for each display can be modified interactively or the display can be

suppressed entirely from the cycle. Other interactive capabilities include: the selection of an object to obtain additional information on that object; the selection of multiple objects for computation of angular separation; and the ability to zoom in on the image based on a user-defined outline.

2.3.3 ARCHITECTURE

The first design of the PAYLOADS MM system, a mainframe based design, encountered serious performance problems due to the number of graphics devices used and the large amount of graphics processing needed for each device. To eliminate the performance problems, a distributed processing approach similar to the 3-D Mon and TCOPS systems was used (see Figure 8). Spacecraft and payload telemetry data are retrieved and processed by a FORTRAN computations program (COMP) executing on the NAS mainframe computer. Parameters necessary to generate the displays are then computed and written in real-time to an interface dataset (IDS). These parameters are then accessed by a communications program (COM) that transmits the data to an IBM PC/AT compatible workstation (configured with IRMA and EGA boards) using the same communications protocol incorporated in the TCOPS system. Up to eight sets of communications programs with corresponding PC workstations can be operating simultaneously (see [1]). Three FORTRAN subsystem programs reside on the workstations and are executed under the DOS operating system. The three programs - Initialization (INIT), User Interface (UI), and Display Manager (DM) - were designed as three separate executables to avoid memory limitations [7].

PAYLOADS MM ARCHITECTURE

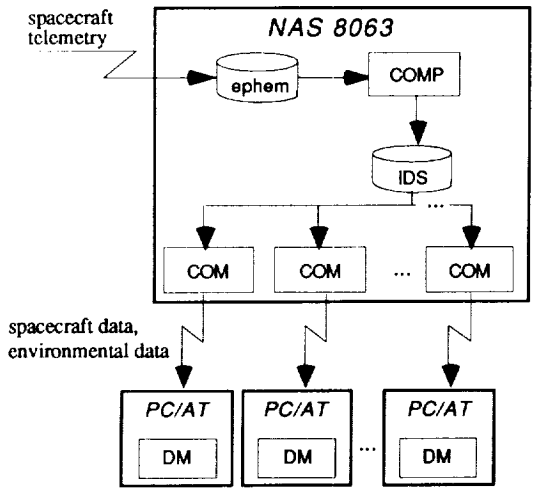


Figure 8.

The UI subsystem is first executed to allow users to initialize colors, image dwell times, etc. Next, the INIT subsystem is executed to graphically generate display skeletons that contain static textual and graphic information such as borders, coordinate grids and text legends. The display skeletons are saved to a Random Access Memory (RAM) drive for fast retrieval/access times. The DM subsystem is then invoked to process the most current data record sent from the mainframe and to control the generation of all displays. When a display is to be refreshed, the DM reads in the specific display skeleton and generates all the dynamic graphics and text onto the skeleton. The image is then displayed via a double buffering algorithm to provide an animation effect [7]. All graphics and text displays are generated using the Media Cybernetics, Inc. HALO graphics package. All user interface screens are developed using the West Chester Group Screen Generator Package.

As a result of this alternative architecture approach, remote usage of the system is now possible. Users who do not have access to a bisynchronous communications line directly connected to the NAS computer can run the DCA IRMA Remote software emulator package. This package emulates the IRMA hardware and communications protocol and converts the transmission to asynchronous messages that can be transmitted or received over a normal telephone line via HAYES V-series Smartmodem 9600 modems.

2.4 HUD

2.4.1 BACKGROUND

The Attitude Heads-Up Display (HUD) is a near real-time system that varies from those systems mentioned previously in this paper. Instead of using spacecraft data to produce images of the spacecraft in its surrounding environment, HUD attempts to allow analysts to see the spacecraft and its environment from the spacecraft navigator's perspective. This perspective decreases the difficulty of determining how a spacecraft is moving, what objects sensors are viewing and how the spacecraft's hardware is reacting during a maneuver. For example, data received from a sensor that is scanning the celestial sky are displayed in a window that corresponds to that sensor's field of view. Data received from an actuator are displayed in a format that indicates the level at which the actuator is operating and how safely it is functioning.

2.4.2 CAPABILITIES

The HUD system displays one graphics image that is updated every time a spacecraft telemetry record is received. Depending upon the spacecraft and its complement of sensors and actuators, the updates can be received as often as 1/8th of a second to every 10 seconds. The graphics image is configured similarly to the dashboard or heads-up display generated by aircraft flight simulators. (See Figure 9.)

Sensors that track solar system objects (sun sensors) and stars (fixed head star trackers) are shown as windows that display the object as it is viewed by the spacecraft in its appropriate location. Analysts can then see if the star trackers are locking in on a star. Thrusters are displayed as a series of lights aligned in the same configuration as they exist on the

spacecraft. The lights are "turned on" when the thrusters are being fired. Sensors and actuators whose excessive operation can be hazardous to the spacecraft's health are displayed as various bars and potentiometers. The colors of the bars change as their operation reaches or exceeds safety levels. The color green indicates a safe level of operation; yellow indicates a warning that the level of operation is approaching the safety limit; and red indicates an unsafe level of operation. An attitude directional indicator, similar to those found in airplanes, shows the orientation of the spacecraft with respect to an inertially fixed coordinate system.

2.4.3 ARCHITECTURE

The HUD system is currently in a prototype phase. Eventually a distributed processing architecture similar to that used in the PAYLOADS MM system will be incorporated into the HUD system. To date, only the PC graphics program has been developed.

3. MISSION PLANNING TOOLS

Flight Dynamics Division analysts are responsible for determining various mission constraints and timelines as part of their premission planning activities. Often, this planning requires the study of environment and spacecraft parameters over a given period of time. In the past, this information has been generated in tabular form as records that are time incremented and contain the data as a series of numbers and flags. Although these data are highly accurate, the presentation format makes quick analysis of trends and time-oriented parameters difficult. To assist with these types of mission planning activities, two computer graphics applications have been developed and are described below.

3.1 MPGT

3.1.1 BACKGROUND

Prior to the launch of a satellite or a Shuttle- attached payload, the Flight Dynamics Division performs several analytical studies that are used to optimize the data collection time for a mission. These studies compute numerical values that contain such statistics as: the amount of Tracking Data and Relay Satellite System (TDRSS) contact time per orbit; the number of orbits per day that pass through a given radiation region; and the percentage of time in sunlight of a given orbit. As a utility to assist analysts with quick analysis of such details, the Mission Planning Graphical Tool (MPGT) was developed. MPGT also provides analysts with a means to produce a graphical picture of the overall spacecraft environment. From this information alternate orbit selections that may better fulfill the mission objectives can be more easily chosen for further investigation.

3.1.2 CAPABILITIES

MPGT produces 2-d and 3-d plots of the earth with spacecraft and environmental data presented as overlays. These overlays include: spacecraft orbit tracks, ground station antenna masks, TDRSS communication contours, interference zone contours, earth and spacecraft sunrise/set terminator lines, solar and lunar ephemeris, a star chart, and an ecliptic coordinate grid. Figures 10 and 11 are images generated by the MPGT system.

All overlays are designed to be mission generic. For instance, communication zone contours and spacecraft terminators are generated analytically dependent upon the altitude of the spacecraft. Interference zone contours are specified through text-edited data files that can be altered to reflect mission specific electromagnetic contamination regions. Up to six separate spacecraft orbit tracks can be specified via Keplerian or Cartesian state vectors. Time-oriented overlays (orbit tracks, sun terminators, etc.) are based on an interactively defined Greenwich Mean Time that is of specific importance to a given mission.

3.1.3 ARCHITECTURE

The system was designed as a standalone system for an IBM PC compatible workstation executing DOS. All graphics images are produced using the HALO device independent graphics package, eliminating graphics adapter hardware requirements.

3.2 SATVIEW

3.2.1 BACKGROUND

One of the Division's attitude responsibilities involves the planning of attitude maneuvers to achieve scientific instrument pointing objectives. Some spacecraft require several attitude constraints to be satisfied simultaneously. These may include instrument target availability, thermal restrictions and power requirements. As an aid to determine if such constraints will be satisfied, the Satellite Viewing system (SATVIEW) was developed as both an attitude maneuver planning aid and a quality assurance tool.

3.2.2 CAPABILITIES

Many of the SATVIEW system capabilities are similar to those found in the 3-D Mon system. Images of the earth, stars, moon, sun and other targets are generated in 3-d while the orientation of the spacecraft model is driven from attitude data that has been previously generated. The attitude and environmental data are provided to the system in greater than real-time. This allows a 90 minute maneuver to be viewed in several seconds. The user views this scenario from any of the spacecraft instrument or sensor field of views.

A second display mode of SATVIEW allows the user to look at the universe from outside the celestial sphere. The celestial sky is drawn as a sphere centered around the spacecraft coordinate system axes. The sun, moon, stars, and the earth's outline are drawn on the sphere while the spacecraft is represented by x, y, and z spacecraft body coordinate axes. Sensor and instrument field of view outlines are drawn on the sphere while attitude and environ-

mental data are again provided in greater than real-time. The user can interactively alter the viewpoint to see the unfolding scenario anywhere outside the sphere. Figure 12 is a celestial sphere image generated by SATVIEW.

An additional feature of SATVIEW is interactive modification of the spacecraft attitude. The user can adjust the spacecraft axes such that a particular mission constraint or set of constraints are satisfied. The required attitude numbers are then returned to the user.

3.2.3 ARCHITECTURE

SATVIEW is a standalone system residing on an Silicon Graphics IRIS 4D/60T graphics workstation. All graphics images are produced by calls to the IRIS Graphics Library routines. Attitude data are produced by software on a NAS 8063 computer and downloaded to the IRIS workstation.

4. SOFTWARE DEVELOPMENT TOOLS

Due to the variance in capabilities, data formats, and dynamics between spacecraft, the Flight Dynamics Division is responsible for generating dynamic simulators, telemetry simulators, attitude ground support systems, and various mission planning tools that are specific to a given spacecraft. Since many of the displays incorporated into these systems vary from mission to mission only slightly in layout, but not in capability, software development tools have been created to increase programmer productivity. These tools are discussed in this section.

4.1 VEGAS

4.1.1 BACKGROUND

Of the numerical output generated by flight dynamics software systems, a large quantity is presented as displays of interactive alphanumeric tables or interactive X-Y plots. Some systems also display the output on world map plots. To reduce the resources required to reproduce source code that generates such complex display capabilities for each system, the Visual Environment for Graphics-oriented Analysis Systems (VEGAS) was developed. VEGAS consists of independent high level subroutine packages that produce x-y plots, text displays, and world maps (see [5]).

4.1.2 CAPABILITIES

The VEGAS X-Y Plot package provides capabilities for data to be displayed as scatter or line plots with Greenwich Mean Time axis label formats. An interactive environment is included that permits data modification, point flagging, curve fitting, zooming, panning and other orientation options. Curves can also be updated in real-time if desired [3]. These capabilities are invoked through high level FORTRAN subroutines.

The VEGAS Text Display package allows alphanumeric data to be displayed with different color and video attributes. User input is verified for type compatibility and range constraints [3]. Screen layouts are defined through text-edited template files. These files give an application programmer the ability to change the screen format without relinking the application.

The VEGAS World Map package was previously developed by another organization at GSFC. This package produces thirty world map continent projections and is invoked through a single FORTRAN subroutine. Routines are also provided for plotting contours on top of the projections.

4.1.3 ARCHITECTURE

Both the X-Y plot and World Map packages are built on top of the Template Graphics Software, Inc. machine and device independent graphics subroutine package TEMPLATE. This design allows these packages, and application software incorporating these packages to reside on the Division's IBM 4341 and DEC VAX computers. This design also allows displays produced by these packages to be generated on IBM 5080 and Tektronix 4100 series terminals.

Since TEMPLATE does not easily provide the character string capabilities needed for alphanumeric displays, the Text Display package was built on top of the IBM Graphics Access Method (GAM) package for IBM mainframe applications and on top of the DEC Screen Management Facility for VAX applications. The IBM version of the package supports the IBM 5080, 3250, and 3278 terminals. The DEC version supports VT series compatible terminals.

5. SUMMARY

The Flight Dynamics Division of Goddard Space Flight Center has committed itself to the use of computer graphics as an effective and efficient tool for comprehending mission related data. This commitment has only been accepted after various systems have proven their worth in the flight dynamics environment. From this commitment numerous graphics-oriented systems discussed in this paper were developed and have been or are currently being validated for operational use while more systems are being planned. And, as more graphics systems are created, more graphics development tools will be created, similar to those discussed in this paper, to reduce software development costs.

ACKNOWLEDGEMENTS

I would like to thank Kelly Franks, Randy Frisch, Greg Marr, Greg Shirah, David Weidow and other members of the NASA Goddard Space Flight Center and Dale Fink, Gary Hunt, Michelle Langrehr, and Ernie Pittarelli of the Computer Sciences Corporation for generating some of the figures used in this paper and also providing technical support.

REFERENCES

1. Brown, C., Franks, K., Bugenhagen, J., Mucci, D., Shirah, G., Weidow, D., "Attached Shuttle Payloads Broad Band X-ray Telescope Detailed Design Document," CSC/SD-88/6125, Computer Sciences Corporation, Greenbelt, Maryland, October 1988.

2. Chang, K., Garrahan, J., Langrehr, M., Pittarelli, E., and Tamkin, G., "Flight Dynamics/Space Transportation System 3-D Monitor System Release 2 System Description," CSC/SD-88/6066, Computer Sciences Corporation, Greenbelt, Maryland, August 1988.

3. Green, D., Pittarelli, E., Hendrick, R., Campos, M., Buhler, M., Durbeck, R., Jeletic, J., Shoan, W., "Programmer's Guide to the VEGAS Graphics Utilities," CSC/SD-87/6018, Computer Sciences Corporation, Greenbelt, Maryland, December 1987.

4. Hardie, B., "System Description for the Trajectory Computation and Orbital Products System (TCOPS) User Interface," CSC/SD-88/6070, Computer Sciences Corporation, Greenbelt, Maryland, July 1988.

5. Jeletic, J., Shoan, W., "Flight Dynamics Graphics for the Space Station Era.," TEMPLATE User's Network Conference, Arlington, Virginia, March 1987.

6. Langrehr, M., Buchanan, L., Chang, K., Garrahan, J., Pittarelli, E., and Tamkin, G., "Flight Dynamics/Space Transportation System 3-D Monitor System Release 2 User's Guide," CSC/SD-88/6006, Computer Sciences Corporation, Greenbelt, Maryland, June 1988.

7. Shirah, G., "Solving a Real-Time Performance Problem of the Attached Shuttle Payloads Mission Monitoring System Through Prototyping," NASA/Goddard Space Flight Center, 552.2, Greenbelt, Maryland, December 1988.

8. Wallace, R., and Buchanan, L., "SPIF/FDF Interface Graphics Support System Requirements and Specifications," CSC/TR-85/6702, Computer Sciences Corporation, Greenbelt, Maryland, June 1985.

9. Weidow, D., "The Shuttle-Attached Payloads Operational Support System," NASA/Goddard Space Flight Center, 552.2, Greenbelt, Maryland, December 1986.

10. Wetmore, R., Anderson, C., and Coon, G., "Shuttle-Attached Payloads Operational System Requirements and Functional Specifications," CSC/TR-85/6001, Computer Sciences Corporation, Greenbelt, Maryland, May 1985.

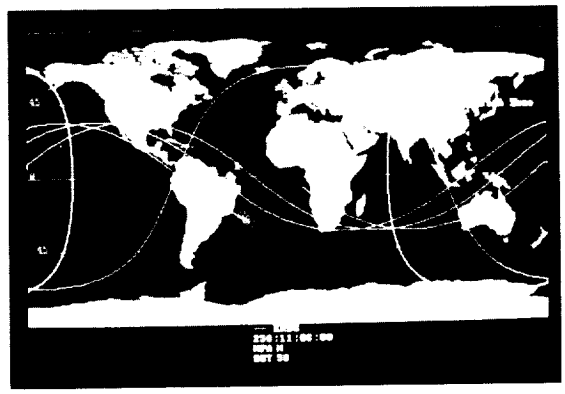


Figure 1. A Trajectory Computations and Orbital Products System world map plot displaying coverage of the Earth Radiation Budget Satellite (ERBS).

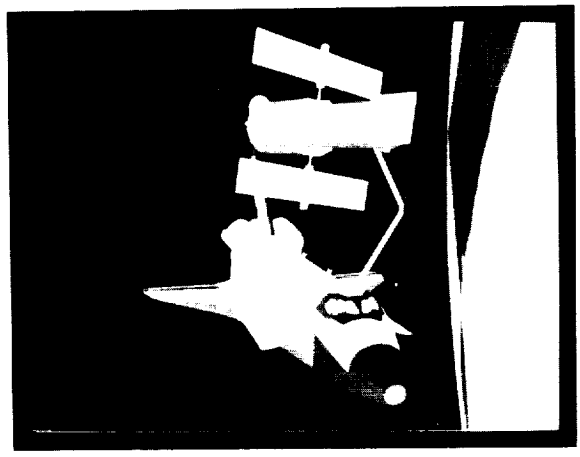


Figure 3. The deployment of the Hubble Space Telescope as displayed by the 3-D Mon system.

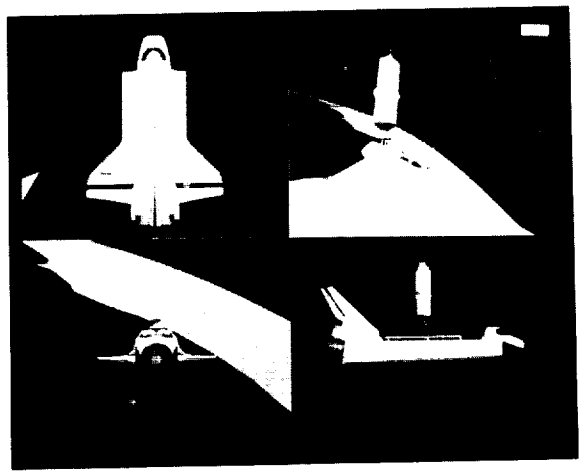


Figure 4. The deployment of a Tracking Data and Relay Satellite as displayed by the 3-D Mon system. The top right viewport displays a view from the rear cockpit window. The earth and sun position vectors and the Shuttle velocity vector are also displayed.

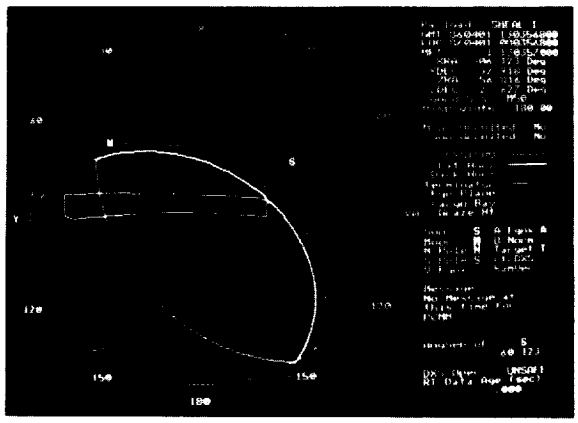


Figure 6. The PAYLOADS MM Celestial Sphere display illustrating the view along the Shuttle's -z axis including the position of the earth, the earth's atmosphere, celestial objects, and instrument field of view outline [1].

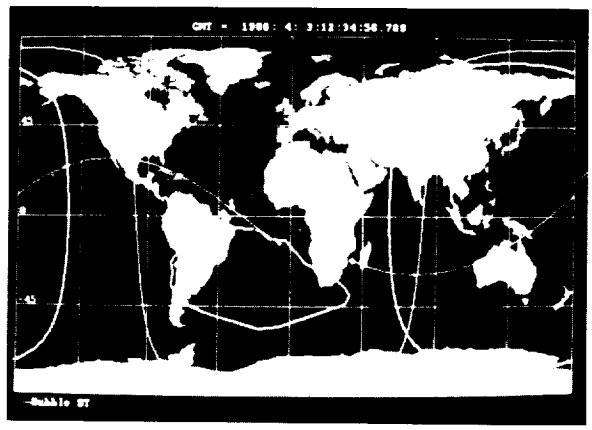


Figure 10. A 2-d world map plot produced by the Mission Planning Graphical Tool configured for orbit studies of the Hubble Space Telescope.

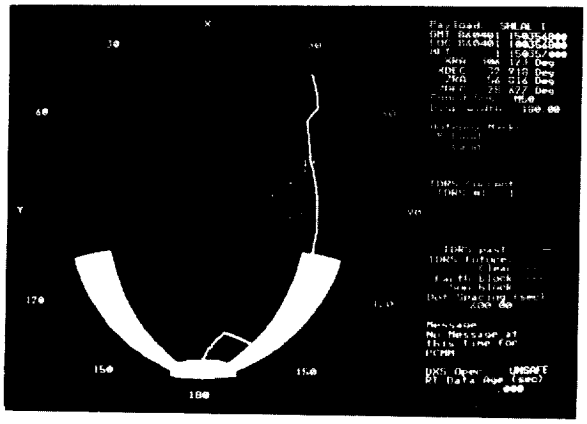


Figure 7. The PAYLOADS MM TDRS display illustrating the view along the Shuttle's -z axis including antenna masks and past, current and future positions of a TDRS [1].

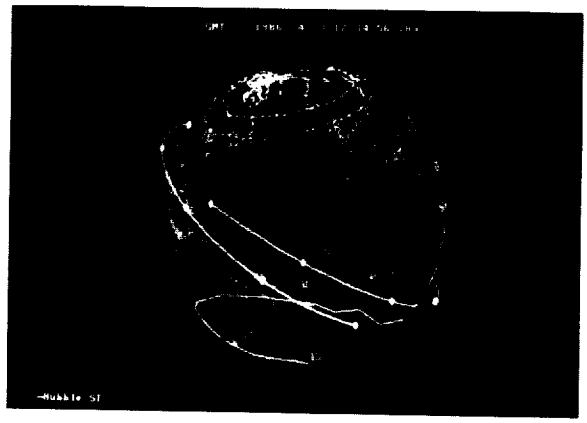


Figure 11. A 3-d earth plot produced by the Mission Planning Graphical Tool configured for electromagnetic interference studies of the Hubble Space Telescope.

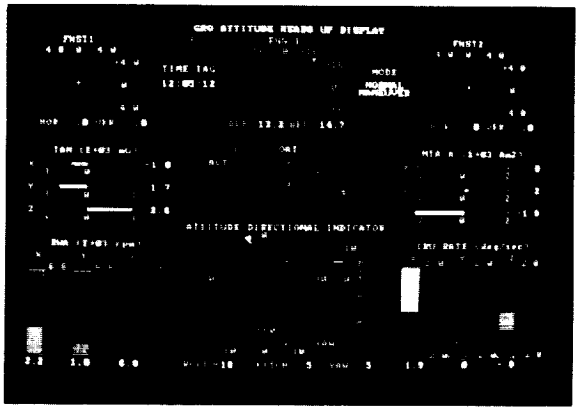


Figure 9. The Attitude Heads-Up display configured for the Gamma Ray Observatory satellite.

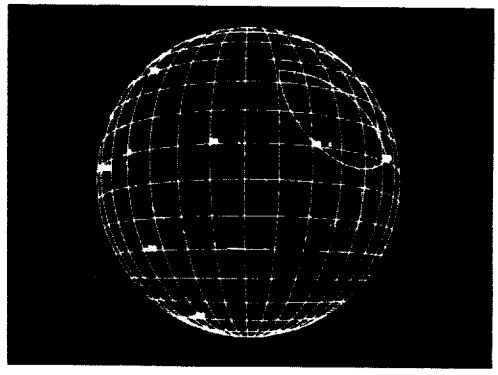


Figure 12. An interactive celestial sphere display produced by the SATVIEW utility.

OMV MISSION SIMULATOR

Keith E. Cok
TRW Defense Systems Group, R11/2850
One Space Park
Redondo Beach, CA 90278
(213)812-2530

ABSTRACT

The Orbital Maneuvering Vehicle (OMV) will be remotely piloted during rendezvous, docking, or proximity operations with target spacecraft from a ground control console (GCC). This paper describes the real-time mission simulator and graphics being used to design a console pilot-machine interface.

A real-time orbital dynamics simulator drives the visual displays. The dynamics simulator includes a J2 oblate earth gravity model and a generalized 1962 rotating atmospheric and drag model. The simulator also provides a variable-length communication delay to represent use of the Tracking and Data Relay Satellite System (TDRSS) and NASA Communications (NASCOM).

Input parameter files determine the graphics displays. This feature allows rapid prototyping since displays can be easily modified from pilot recommendations. Different subsets of OMV telemetry data can be shown to determine the information necessary for pilot operations.

A series of pilot reviews are being held to determine an effective pilot-machine interface. Pilots fly missions with nominal to 3-sigma dispersions in translational or rotational axes. Console dimensions, switch type and layout, hand controllers, and graphic interfaces are evaluated by the pilots and the GCC simulator is modified for subsequent runs. Initial results indicate a pilot preference for analog versus digital displays and for two 3-degree-of-freedom hand controllers.

INTRODUCTION

The OMV is designed as a reusable unmanned spacecraft. Initially deployed from the space shuttle, it is capable of staying in orbit for months while receiving periodic on-orbit maintenance and refueling. The OMV is used to deliver, retrieve, reboost, or maneuver satellites between the shuttle or space station and a specific orbit.

The OMV flies autonomously to within 1000 feet of a target spacecraft. A pilot then remotely controls the OMV in rendezvous, docking, or proximity operations. The OMV will be operated by NASA personnel from a ground control console (GCC) located at the Johnson Space Center.

The GCC sends pilot commands to the OMV via NASA Communications (NASCOM) and two Tracking and Data Relay Satellites (TDRS). The OMV downlink transmissions consist of telemetry and two video camera transmissions. The communications link can transmit up to 32 kilobits/second of telemetry and 1 megabit/second of compressed video signal. The communications link has an approximate 3-second round-trip delay time.

The OMV docks with the target spacecraft using either the remote manipulator system (RMS) grapple docking mechanism (RGDM) or a three-point docking mechanism (TPDM) for those spacecraft that have a flight support system (FSS) interface.

The OMV prime contractor, under NASA Marshall Space Flight Center, is TRW. The OMV is scheduled for deployment in November 1993. Its potential first mission is in conjunction with the Waves in Space Plasma (WISP) project.

OMV flight operations will be conducted from either of two identical GCCs. A GCC provides pilot control of the OMV during all flight operation phases. Each GCC consists of switches, hand controllers, two terminals and keyboards, data processing equipment, and two monitors displaying information from the on-board docking and pan/tilt/zoom (PTZ) video cameras. The pilot manipulates hand controllers for OMV maneuvers and utilizes switches for OMV or console commands.

The GCC must provide a pilot-machine interface that gives adequate information to avoid information overload, and minimizes pilot errors. TRW was given the task of building a prototype GCC (PGCC) to simulate man-in-the-loop, real-time remote OMV teleoperations. The PGCC is the tool used to establish the console pilot-machine interface.

SIMULATOR OVERVIEW

Simulator Models

The PGCC was developed as a representative operational pilot station used for preliminary design evaluations and crew reviews. The OMV program concluded that to evaluate a pilot-machine interface fully, it was necessary to simulate a dynamic docking environment which integrates flight telemetry with hand-eye coordination. Space environment and OMV models are included in the simulation.

The simulator dynamically models the space environment. The environment models include a J2 oblate earth gravity model and a generalized 1962 rotating atmospheric density and velocity model. A drag model is based on a cylindrical approximation for the OMV and target bodies.

Each body is characterized by 6-degree-of-freedom (DOF) equations of motion including effects of position, velocity, attitude, translational and rotational rates, moments of inertia, centers of mass, and gravity gradient torques. Each target satellite is in free drift and has no control system. Only the OMV has thrusters and a flight control system.

Mission date and time parameters position the sun, moon, and earth in the simulator reference frame. Other mission parameters determine orbit position and velocities. Positions of the OMV during the simulation determine sun occlusion, camera sun intrusion, and communication zones of exclusion. They also affect lighting conditions and shading. Without these real-world conditions, valid data cannot be taken.

The simulator models several OMV subsystems. These include the fuel system, radar, and two video cameras. For example, the pilot may select either a hydrazine or GN_2 thruster system during flight. Each alternative has its own fuel tanks and rates of consumption. The hydrazine tanks are manifolded while the GN_2 tanks are independent.

Each fuel system has its own set of thrusters. Input parameter files determine the location, force vector, and impulse moment of each thruster. A particular thruster is rendered useless when the fuel tank feeding that thruster is empty. Deviation in thruster force is modeled by varying the force vectors in a parameter file. Simulator logic is used to model the less efficient first few microseconds of burn. A thruster pulse size, initialized by an input parameter, determines the minimum burn allowed. Individual thrusters can be failed on or off. If a thruster is failed off, no force or fuel is spent. However, if a thruster fails on, fuel will be burned and corresponding impulse moments will occur.

Pilots maneuver the OMV by commanding thruster burns in one or more axes. The simulated on-board computer receives the axis thrust commands and uses a jet select table to compute thruster burn times. The simulator provides two jet select tables. The real OMV utilizes identical jet select information which is uplinked to the vehicle during preflight checkout.

The simulator also models the OMV radar subsystem. A pointing vector from the radar mount to the target is computed. This vector takes into account the OMV position, gimbal limits, and radar field of view. The simulator computes the azimuth, elevation, azimuth rate, and elevation rate from the pointing vector. The radar also models the radar-to-target surface range and range rate. Radar noise and bias are introduced into the range and range rate data for greater realism. The models also provide maximum and minimum radar cutoff points at selectable distances.

The simulator models the docking (boresight) and PTZ cameras. They both produce black and white video. The pilot operates either a joystick or switches on the PGCC console to tilt, pan, or zoom the PTZ camera to a commanded position with corresponding slew rates.

Each camera has a 30-degree half-angle field of view. Gimbal stops limit the PTZ camera range of motion. Each camera is equipped with a sensor to detect sun brightness. If sun intrusion should occur, the shutter of the camera will close, blinding that camera.

Contact detection and limited dynamics are modeled in the simulator. Since modeling full contact dynamics between all surfaces of the OMV and its target is impractical without additional computing power, the simulator detects contact only between the open or closed TPDM latches and target trunnions. The simulator computes contact dynamics with a method of "soft constraints." This technique allows solids to penetrate each other at the point of contact. The algorithm then computes the restoring normal and tangential forces based on the depth of penetration. Damping forces also may be added. In addition, sliding (Coulomb) and viscous

friction may be applied. Linear and angular momentum is conserved upon contact for complete 6-DOF motion.

The OMV model contains a flight control system. The system uses the earth centered inertial (ECI) or local-vertical local-horizontal (LVLH) reference frames. A three-axis linear control law fires thrusters if either attitude or attitude rates exceed a selectable deadband. Attitude or rate hold is disabled for an axis if a pilot commands a maneuver in that axis. In addition, an automatic attitude maneuver capability is built into the simulator. The simulator rotates the OMV by firing thrusters to the desired attitude commanded by the pilot.

The OMV uses two high-gain antennas (HGA) to communicate with the TDRSS spacecraft. The simulator maintains a pointing vector from each HGA to each TDRS. Communication zones of exclusion are based on the orbit, ECI satellite positions and velocities, earth occultation, and HGA gimbal limits.

Simulator Interfaces and Architecture

The simulator provides several interfaces in addition to the pilot-machine interface. The simulator operator has a telemetry and data display on a side terminal. The operator can introduce anomolies from either this terminal or from an event file. The event file, read in at initialization, is a list of commands and events that occur at some specified time into the simulation. The operator also receives history and contact report files for post-simulation analysis. The history file contains all OMV and target state vector information, switch inputs, and environment information. The contact report file contains time-stamped contact information.

Nearly all simulator data is initialized by input parameter files. These files determine values such as fuel and thruster characteristics, orbit position, environment data, mass properties, and size of the OMV and target. They also initialize such other data as the number of targets, placement of the video camera, radar characteristics and all simulator control information.

Orbit characteristics determine initial orbit placement and rates. This data can be specified in osculating mean of 1950 (OM50), rectangular mean of 1950 (RM50), inertial mean of launch date (IMLD), or target relative reference frames. State vector integration and derivatives are computed using quaternions. Forces and accelerations due to gravity, torques, and thrusters are computed using the Adams-Moulton integrator.

The simulator maintains its own time with software interrupts. Each major subsection is given a constant delta time each cycle to perform its tasks. For example, the input subsection reads the joysticks and switches every 50 milliseconds. The on-board computer (OBC) subsystems are executed every 250 milliseconds and graphic displays are updated every 200 milliseconds. This approach simplifies the software architecture, eliminating separate processes and semaphores. However, one slow subsection can degrade the entire simulation.

The simulator hardware consists of a MicroVAX 3600, Chromatics CX2000 with frame grabber and a 24-bit z-buffer. The CX2000 drives two 1280 x 1024 pixel 19-inch monitors. A Q-bus Direct Memory Access (DMA) connects the MicroVAX with the CX2000. The simulator drives two pilot consoles, each containing hand controllers and up to 48 switches. The simulator is built from approximately 17,000 lines of FORTRAN.

PILOT-MACHINE INTERFACE

Interface Description

The main PGCC task is to define a pilot-machine interface: the physical console and graphic displays. The console interface consists of console dimensions, hand controllers, and placement, function, and choice of switches. The console ergonomics are designed to accommodate the 95th percentile man and 5th percentile woman (Figure 1).

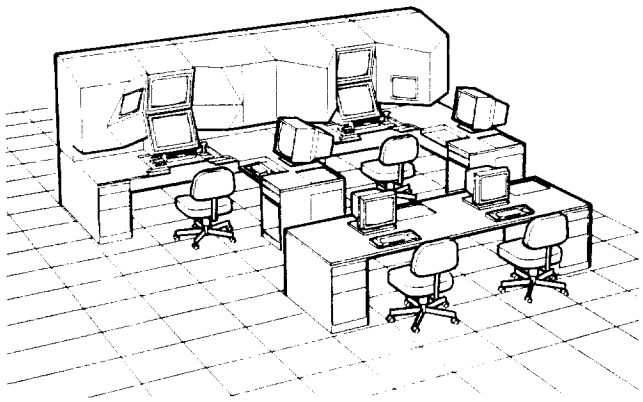


Figure 1. Prototype Ground Control Console

The selection, placement, and style of telemetry and video data form the second part of the pilot interface. A language was created to express overlay characteristics and to allow easy reconfiguration. Input parameter files, written in this language, define the color, placement,

style, etc. of each overlay. In this way, per-simulation customization can take place. In addition, alternate styles of display, graphic or text, can both be accommodated (Table I). Merely changing the input files drastically alters the "look and feel" of the pilot-machine interface. Figure 2 shows a current set of piloting overlays.

Table I. Overlay Definition File

```
*
*  TPDM Docking Overlay
*
BEGIN_ICON NEAR_FIELD
   OVERLAY 0
   OFFSET 100.0 73.242
   SCALE 1.0 1.0 1.0
   ROT 0.0
   SUB_ICON
         COLOR WHITE
         OFFSET 0.0 0.0
*     Vertical Ranging Marks
*     10 feet out
         LINE -4.736 2.0 -4.736 -2.0
         LINE 4.736 2.0 4.736 -2.0
*  3 feet out
         LINE -15.787 2.0 -15.787 -2.0
         LINE 15.787 2.0 15.787 -2.
*  Minimum docking range
         LINE -21.714 2.0 -21.714 -2.0
         LINE 21.714 2.0 21.714 -2.0
   END_SUB_ICON
END_ICON
*
BEGIN_ICON DOWN_THRUST
   OVERLAY 1
   SUB_ICON
         OFFSET 3.5  1.0
         ROT 270.0
         SCALE 1.0 1.0 1.0
         FILLED
         COLOR CYAN
         ARROW
         ARC 90.0
   END_SUB_ICON
   SUB_TEXT
         HEIGHT 2
         EXPAND 1.0
         RIGHT
         STRING Rate:
   END_SUB_TEXT
END_ICON
```

DOCKING INTERFACE

The pilot operates hand controllers and switches to guide the OMV to a dock with the target vehicle. The OMV is equipped with one of two types of grappling mechanisms depending on the target vehicle interface. Two standard mechanisms include the RGDM or the TPDM. The current simulator configuration models the TPDM with the Hubble Space Telescope. After the pilot maneuvers the OMV within the docking envelope, the three TPDM latches can be independently closed, ensnaring the trunnions mounted on the aft of the Space Telescope.

The pilot uses the docking target located on the back face of the target satellite as a guide when docking. The target, in relation to the docking overlay, gives the pilot relative translation and rotation information. When the docking target fills the docking overlay, the target trunnions are within the grapple capture envelope.

Each TPDM latch mechanism is equipped with two sensor beams. When the trunnion breaks a sensor beam, the corresponding grapple beam overlay changes color. Using the overlays and video, the pilot can accurately determine the position and attitude of the target relative to the OMV.

Attitude errors discernible from the Space Telescope docking target are larger than the TPDM will accommodate. Therefore, the docking overlay is built to give the pilot information on maximum attitude and translational docking allowances. With this overlay, the pilot can back out, if necessary, to realign the OMV with the target for a safer dock. If the docking target should exceed the overlay, the pilot can expect the latches to contact the trunnions. The overlay provides the allowances at the minimum docking range (when the trunnion are just within the docking envelope) and at the point when the trunnions are centered over the second (inside) beam.

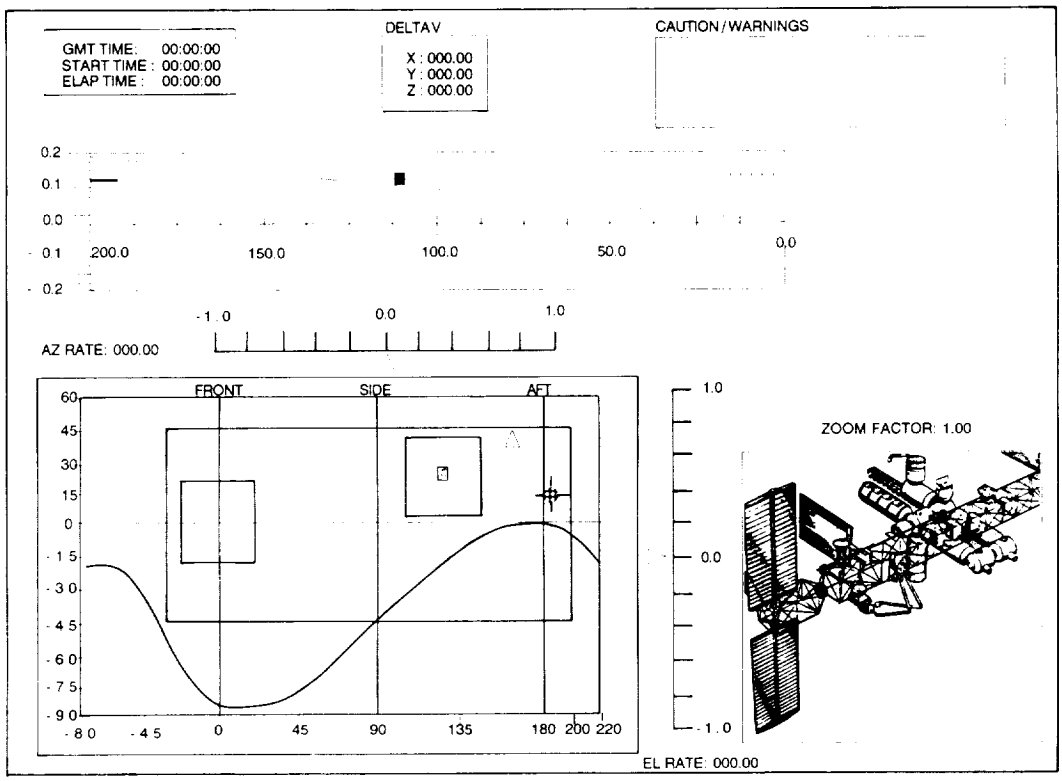
Astronaut comments indicate that range and range rate information is especially important within the radar cutoff point. Since acceptable latch closure rates are 0.1 foot/second along any axis and 0.5 foot/second about any axis, it is important the pilot get an accurate "feel" for the OMV's closing rate. Therefore, ranging aids were built into the docking overlay.

PILOT REVIEW

Approach

The first in a series of simulator reviews was held in August 1988. Thirteen people from TRW, Johnson Space Center, and Marshall Space Flight Center, including two astronauts, were available as pilots. The pilots ran through a sequence of training procedures to familiarize themselves with switch layouts, OMV thruster sensitivity, docking procedures, and overlays. After being "qualified," each pilot ran a set of simulations emulating various mission phases. Initial conditions ranged from nominal to 3-sigma cases in translational or rotational rates. Overlays were explained prior to each training procedure. Piloting tips were

Upper Monitor



Lower Monitor

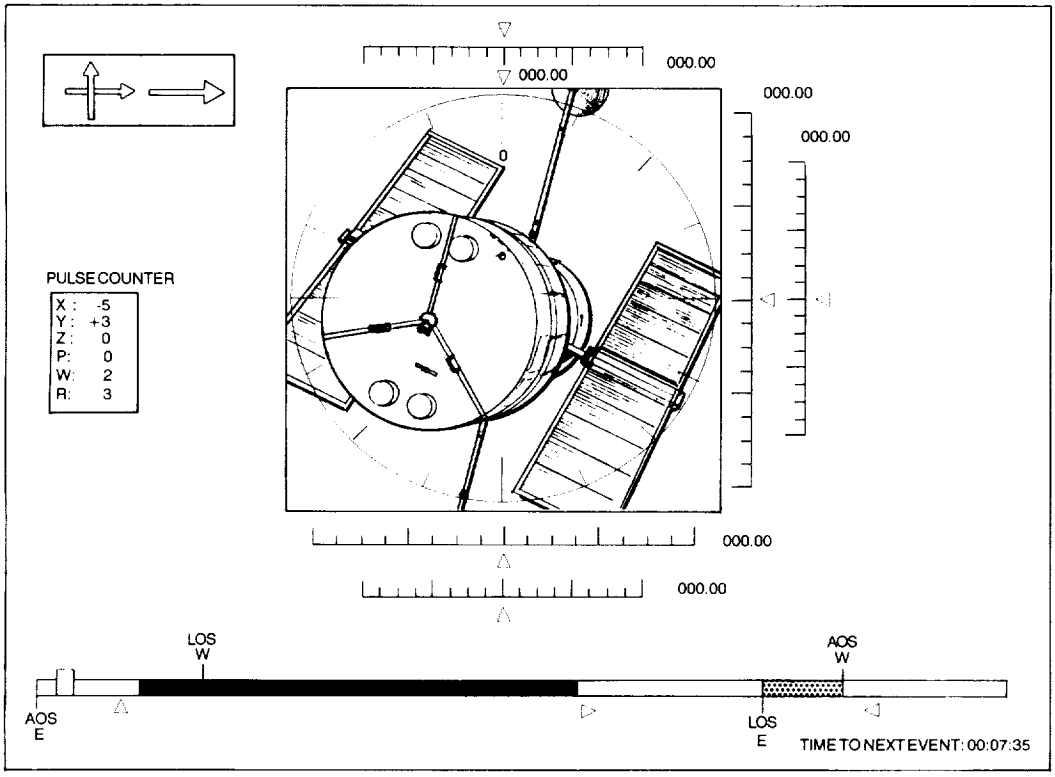


Figure 2. Pilot Overlays

provided and any questions were answered during the simulation. Pilots flew simulations during eclipse and docked with spinning targets. A history log was kept of each procedure and simulation for analysis. After each training procedure and simulation, pilots were debriefed. The total flight time exceeded 40 hours. Training time was limited to approximately 1 hour per pilot. The time for each run varied between 10 and 30 minutes.

The first review focused on two variables: text versus graphic displays and type of hand controller. Although these were the primary concerns, other feedback was also noted.

Review results were based on observations during flight simulations and pilot feedback gained from questionnaires and discussions. The evaluation focused primarily on the reasons for the success or failure to reach the simulation goal.

Initial Results

The review clearly showed a pilot preference for a hybrid of primarily graphic overlays mixed with some text. There were varying opinions expressed on the graphic versus text attitude direction indicator (ADI) format. In future reviews, pilots will select an ADI format from a palette of four displays. Digital range and range rate will be added to the enlarged analog radar display. The radar display will be enlarged to detect azimuth and elevation rates more easily.

Some of the overlays are placed directly on top of the video. These were difficult to see at times due to the underlying video color. Since the video contrast varies during orbit, there is a need to dynamically change the color of the overlays during simulation. One overlay color may be acceptable during one mission phase but not during another.

Pilots flew with targets spinning at 1.0 degree/second. It was apparent that the piloting techniques vary sufficiently to warrant another type of docking overlay. Specific aids for matching target spin rates and tracking rotating targets will be included with the standard ranging information and docking allowance overlays.

Overall, the pilots liked the console ergonomics. Most preferred an adjustable tilt monitor. They were pleased with the monitor size and resolution. Pilots flew with both types of displays and hand controllers. One console had two 3-DOF hand controllers and the other had one 6-DOF controller with a different assortment and arrangement of switches. Switches varied in type, shape, color, and mounting. Pilots indicated that switch shape, size, or mounting did not aid in correct switch selection. Most pilots preferred flush-mounted switches.

Unverified piloting switch commands are indicated by flashing switches. The switch light changes color after the command has been verified or executed. This scheme worked well; most pilots did not prefer any other method.

Most pilots were trained to fly with two 3-DOF hand controllers and preferred to continue using them rather than the one 6-DOF controller.

CONCLUSION

It is evident that a full dynamic simulation is prerequisite to gaining useful data. Comments on an interface from an unrealistic simulator would have limited use. Likewise, trained pilots are needed to produce valid conclusions and avoid review comments which merely reflect unfamiliarity with the simulator, overlays, or piloting techniques.

The choice of pilot missions also influences the quality of gathered information. Carefully planned missions which stress pilot or OMV performance are most useful; during nominal missions, nearly all displays either work well or are never used.

By holding a series of pilot reviews and by building prototype displays, agreement will be reached on an acceptable pilot-machine interface. It is expected that having a community consensus on an OMV pilot-machine interface will prevent problems during the acceptance phase of the GCC project.

THE USE OF GRAPHICS IN THE DESIGN OF THE HUMAN-TELEROBOT INTERFACE

Mark A. Stuart and Randy L. Smith

Lockheed Engineering and Sciences Company
2400 NASA Road 1
Houston, Texas 77058-3711

The Man-Systems Telerobotics Laboratory (MSTL) of NASA's Johnson Space Center employs computer graphics tools in their design and evaluation of the Flight Telerobotic Servicer (FTS) human/telerobot interface on the Shuttle and on the Space Station. It has been determined by the MSTL that the use of computer graphics can promote more expedient and less costly design endeavors. This paper describes in detail several specific examples of computer graphics applied to the FTS user interface by the MSTL.

INTRODUCTION

Computer graphics techniques, including software prototype development programs, can serve as an aide in the design, evaluation, and development of user interfaces of many types. These systems design tools can result in the development of ergonomically well-designed workstations in less time with lower costs when compared to the use of other systems design tools.

With the system development process becoming more complex and expensive, more emphasis is being placed on the evaluation of systems during early stages of the development cycle. The design of systems that include human operators is especially complex because determining overall systems performance is dependent upon the interaction of the human operator, hardware components and software components (ref. 1). Adequately evaluating the performance of a system during the design cycle is becoming increasing more difficult when using the static evaluation tools traditionally available to the Human Factors Engineer, such as job and task analyses and mockup development (ref. 2). It is becoming more common for systems developers to use computer graphics as a design tool instead of hardware models (ref. 3) and for Human Factors Engineers to use computer graphics to enhance the use of static design tools (ref. 4).

The Man-Systems Telerobotics Laboratory (MSTL) of NASA Johnson Space Center (JSC) with support from Lockheed has extensively used computer graphics tools in their design and evaluation of the Flight Telerobotic Servicer (FTS) user interface. It is the goal of the MSTL to help design, evaluate and develop requirements for the user interface of the FTS. Goddard Space Flight Center is the lead center in the development of the FTS with other NASA centers and industry playing various roles.

The FTS will be a dual-armed teleoperated robot used to help assemble, service, and maintain NASA's Space Station. There will be an FTS control panel on both the Shuttle and the Space Station. The design of the FTS control panel is especially challenging since

it may be physically impossible to have identical control panels on both the Shuttle and the Space Station due to the physical constraints of the Shuttle. The ultimate objective in the design of the FTS control panel is that the human operator's capabilities and limitations have been best accommodated for while ensuring that overall systems goals and requirements are met. The use of computer graphics will enable NASA to iteratively design a good FTS control panel on the Space Station which does not radically differ from the FTS control panel included on the Shuttle. Radical departures from the control panel used on the Shuttle will increase the likelihood of negative transference or reversal errors. Therefore, design features which take advantage of population expectancies should be a constant feature across both control panels to ensure maximum performance.

This paper will discuss the MSTL's use of computer graphics tools that have been applied to the design and evaluation of the human-telerobot interface that will be a part of NASA's Shuttle and Space Station. Each example will begin with a statement of the objectives of the task and will then detail the approach taken by the MSTL for that particular application. The discussion of these applications will also include illustrations of the computer graphics used.

PROGRAMMABLE DISPLAY PUSHBUTTONS

The first example given will be an illustration of how computer graphics was used by the MSTL to establish a set of guidelines concerning the use of programmable display pushbuttons (PDPs) on the Space Station's FTS control panel (see ref. 5 for a detailed discussion concerning this study). The graphics tool used during this evaluation was Hypercard. Hypercard is an information management software package which allows the user to organize text, graphics and active "screen buttons" into cards. The cards can then be linked together in different user-definable stacks. The stacks can then be arranged so that high-fidelity control panel prototypes can be created with relative ease.

This phase of the FTS workstation evaluation covered a preliminary study of PDPs. Since the study of PDPs is now in the early phase of the design cycle, the focus on this evaluation was to use computer graphics as a means of testing the feasibility of using PDPs on the FTS control panel. The PDP is constructed of a matrix of directly addressable electroluminescent (EL) pixels which can be used to form dot-matrix characters. PDPs can be used to display more than one message and to control more than one function. Since the PDPs have these features, then a single PDP may possibly replace the use of many single-function pushbuttons, rotary switches, and toggle switches, thus using less panel space. Due to space constraints on the Orbiter and the Space Station, an overriding objective of the design of the FTS workstation is that it take up as little panel space as possible. It is of interest to determine if PDPs can be used to adequately perform complex hierarchically structured task sequences.

Other investigators have reported on the feasibility of using PDPs in systems design (refs. 6,7), but the present endeavor was deemed necessary so that a clearly defined set of guidelines concerning the advantages and disadvantages of PDP use in the FTS workstation could be established. This would ensure that PDP use was optimized in the FTS workstation.

The objective of this investigation was to study the performance of subjects performing a simulated manipulator task on PDP and non-PDP computer prototypes so that guidelines governing the use of programmable display pushbuttons on the FTS workstation could be created. The functionality of the manipulator on the Orbiter was used as a model for this evaluation since the functionality of the FTS at the time of this writing had not been solidified.

The graphics version of the non-PDP control panel is depicted in Figure 1. The distinguishing feature of this configuration is that traditional single-function pushbuttons are used in conjunction with a

simulated EL panel to activate commands. The EL panel was simulated in this evaluation by displaying single-function commands as they would appear on the EL panel in the upper right-hand corner of the prototyped screen.

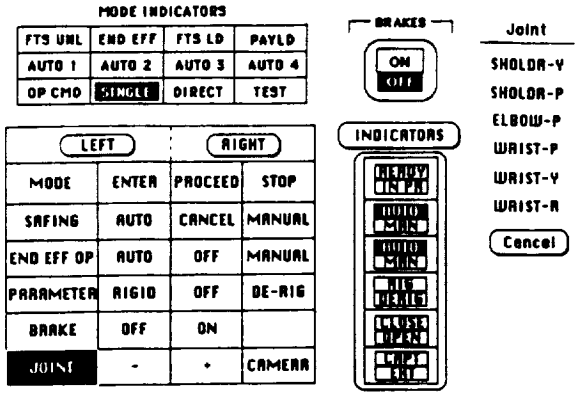


Figure 1 -- Non-PDP control panel prototype.

The graphics version of the PDP control panel is depicted in Figure 2. This control panel utilized simulated PDPs instead of single-function pushbuttons. In Figure 2, the PDPs are the twelve pushbuttons located in the lower-middle portion of the display. The portions to the left and top of the display are dynamic status indicators that were used to display various functional states.

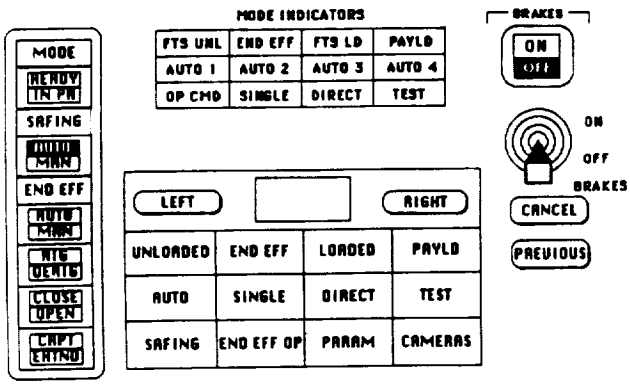


Figure 2 -- PDP control panel prototype.

When a PDP is selected, the name of that function is then displayed in a small simulated EL display located just above the PDP cluster and the options that follow within that functional category are then displayed by the PDPs. For example, when SINGLE is selected in Figure 2, the display changes to that depicted in Figure 3. In Figure 3, SINGLE is now displayed in the EL display and the PDPs have changed to list the options that follow under SINGLE. The small EL display was designed to serve as a navigational aid to help orient operators throughout performance of the hierarchically structured tasks. It was contended that the use of the navigational aid in the PDP hierarchy would be useful since a previous evaluation (ref. 8) found that navigational aids are helpful with hierarchical search tasks through menu structures on a computer.

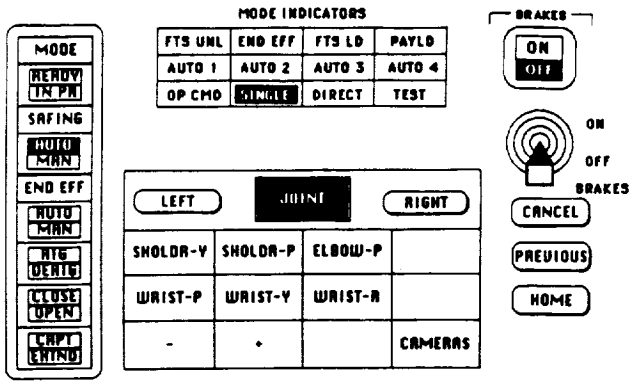


Figure 3 -- PDP control panel prototype with PDP changes and navigational aid.

After performing the task scenarios on both of the control panel prototypes, each subject was asked to select which of the two control panel prototypes were preferred. Each subject was also asked to complete a questionnaire designed to garner subjective impressions concerning the control panels. Data were collected and analyzed with the objective of determining differences in user performance and preference between the two different control panel configurations so that, ultimately, guidelines concerning the use of PDPs could be established. All numeric data were statistically analyzed with a repeated measures analysis of variance.

The ultimate objective of this investigation was to use computer prototyping to establish a set of guidelines concerning the use of PDPs on the FTS workstation. The data collected during this investigation were used to create these guidelines. It is contended that the established set of guidelines will also be generalizable to other workstations as well.

For a complete list of these guidelines, please see ref. 5. It is contended that the use of this set of guidelines will help to ensure that PDPs will be optimally designed and arranged.

The use of computer graphics proved to be invaluable during this evaluation. Graphics allowed the experimenters to iteratively try out many different design configurations before testing actual, hard-wired PDPs. Without the use of computer prototyping, it is contended that the design process would have taken much more time and money to perform as efficiently. If computer prototyping was not used by the MSTL then it would have been necessary to have completely assembled the hardware components and electrical wiring of each of the design configurations evaluated with the computer prototyping method to iteratively evaluate different design possibilities so that an optimal solution could be derived. The hardware approach would have been much more expensive and involved.

HAND CONTROLLERS AND RESTRAINT SYSTEMS

The second example will be a discussion of how graphics was used to evaluate the placement of different types of hand controllers and different types of body restraint systems within various conceptual designs of the FTS workstation on the Shuttle. The tool used during this evaluation was the PLAID graphics package. PLAID is a graphics development package created by the Graphics Analysis Facility of NASA's Johnson Space Center. PLAID enables the creation of three-dimensional, color, graphical images with accompanying animation. The feature of animation enables the MSTL to evaluate different workstation configurations with the interaction of figures of human operators which are anthropometrically correct, thereby determining anthropometric reach limits and viewing angles. PLAID also makes it possible to evaluate how well operators of varying physical dimensions can interact with different workstations.

PLAID enabled the MSTL to iteratively evaluate FTS workstation layouts within the aft flight deck and the mid-deck of the Shuttle. Figure 4 illustrates a conceptual design of the placement of the FTS workstation on the aft flight deck. (PLAID drawings are produced in color, but, due to reproduction restrictions on this document, color prints could not be included in this article. Therefore, the PLAID renderings included in this paper are, out of necessity, line drawings.) If the FTS workstation is placed in this location, it will be in close proximity to the Remote Manipulator System (RMS) control panel. This particular figure gives an indication of how two 95th percentile male operators would work together simultaneously. The reader should notice that the PLAID drawing indicates that there will be some shared work space between the two operators. This important finding was made available to the MSTL without the necessity of fabricating full-scale mockups. Different sized operators other than the ones examined in this example could also have easily been put into the aft flight deck conceptualizations for evaluation.

Figure 5 illustrates how the FTS workstation might be laid out in the mid-deck of the Shuttle. In this figure, a 95th percentile male operator is using the workstation located within the bank of lockers in the mid-deck of the Shuttle. Figure 6 is a conceptualization of how well a 5th percentile female would be able to reach the controls of the mid-deck FTS workstation. The reader should notice that in each of these two figures a restraint system that attaches to the torso of the operators is included for evaluation. This particular restraint system concept was developed by Charles Willits of NASA-Reston.

CONTROL/DISPLAY LAYOUTS

The third example will be a discussion of the use of computer graphics in the consideration of the placement of the FTS control panel in the Shuttle. At the time of this writing, it had not been determined where the FTS control panel would be located in the Shuttle. As in the discussion of the use of PLAID in

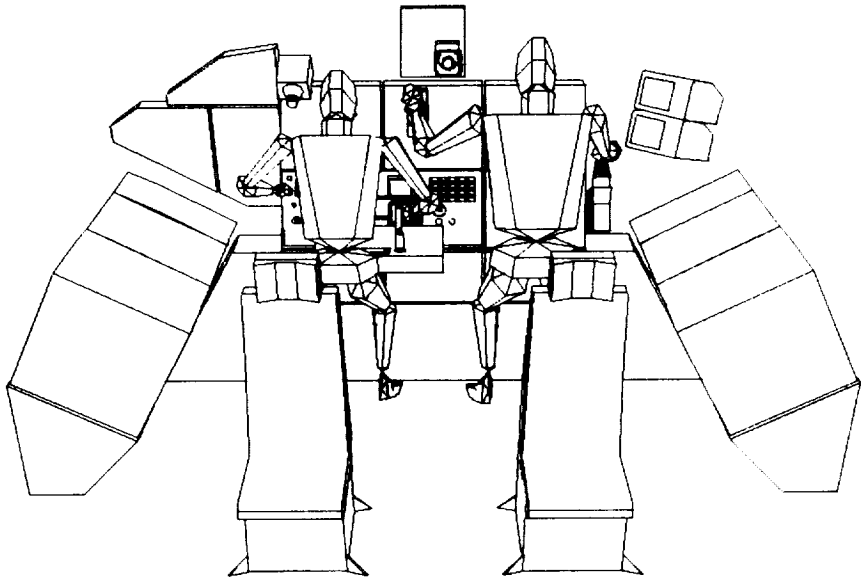


Figure 4 -- PLAID conceptualization of the placement of the FTS workstation in the aft flight deck of the Shuttle.

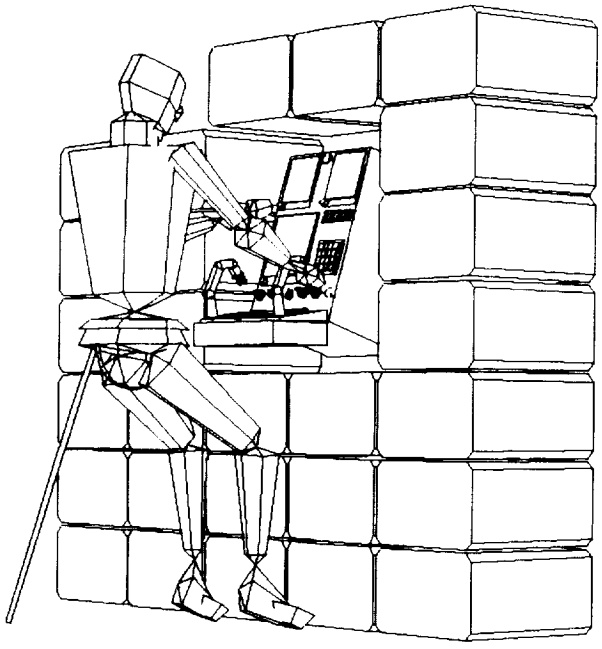


Figure 5 -- PLAID conceptualization of the placement of the FTS workstation in the mid-deck of the Shuttle with a 95th percentile male operator.

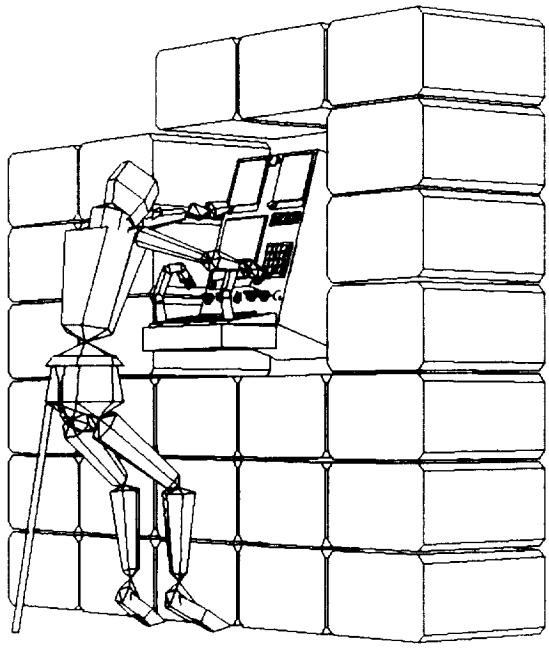


Figure 6 -- PLAID conceptualization of the placement of the FTS workstation in the mid-deck of the Shuttle with a 5th percentile female operator.

the previous section, two locations were being considered: the aft flight deck and the mid-deck. Many different design features were considered and computer graphics enabled the MSTL to quickly and inexpensively evaluate the preliminary placement of these features. Some of these features were the size and number of monitors to use, placement of control switches, the types of controls to use, and whether or not a detachable keyboard should be a part of the control panel. The graphics package used in this example was MacDraw. MacDraw is a graphics development package that is available on Apple Macintosh computer products.

The first examples given will be design considerations made concerning the placement of the FTS control panel in the aft flight deck. Figure 7 is a drawing made with MacDraw to illustrate a possible FTS control panel using aft flight deck panel A6-A2.

The second location within the Orbiter where the placement of the FTS control panel was considered was the mid-deck. There was more space available in the mid-deck for the FTS control panel, so the control panel layouts where slightly different. Figure 8 is an illustration of a control panel layout in the mid-deck.

The MSTL has determined that one advantage of the use of computer graphics is that it will allow a somewhat extensive analysis to take place before any physical mockups have been developed. After several design iterations using computer graphics, full-scale mockups with varying levels of fidelity can then be constructed.

OTHER COMPUTER GRAPHICS APPLICATIONS

The MSTL had other proposed uses for computer graphics at the time of this writing. Since these applications were still in the design stage, the drawings were not available for this publication. None the less, these applications also represent further uses of computer graphics within the field of Human Factors. For this reasons, then, these projects will be briefly described here.

One project which is currently underway is the use of Hypercard to create "pulldown" and "popup" menu-overlays on real-time video images that appear on cathode ray tube (CRT) screens. The video images will be fed from analog and digital cameras located at remote locations from test subject viewers. The video images will be the subjects' only view of remote work sites of interest. The menu-overlays will enable the MSTL to evaluate the utility of an operator using various input devices to control cameras while performing simulated FTS remote manipulation tasks.

Another project underway at the MSTL was the proposed use of computer-aided measurement tools to monitor and display various indicants of work physiology, especially mental workload. The objective here was to incorporate computer-aided data collection and display technologies so that the MSTL could evaluate the workload tradeoffs associated with various workstation components and configurations.

CONCLUSION

The consideration of the productivity, safety, and comfort of the astronaut crewmember is being incorporated into the design process of advanced NASA telerobots through the use of powerful computer-aided systems such as PLAID, Hypercard and MacDraw. The above mentioned examples serve to illustrate the invaluable role that computer-aided design technologies play in the design and development of the FTS workstation by NASA JSC's MSTL. The MSTL has determined that the use of computer graphics packages contributes to a more efficient and less costly systems design cycle. Graphics packages will continue to be used by the MSTL and should certainly exhibit increased usage throughout the field of Human Factors.

ACKNOWLEDGEMENTS

Support for this investigation was provided by the National Aeronautics and Space Administration through Contract NAS9-17900

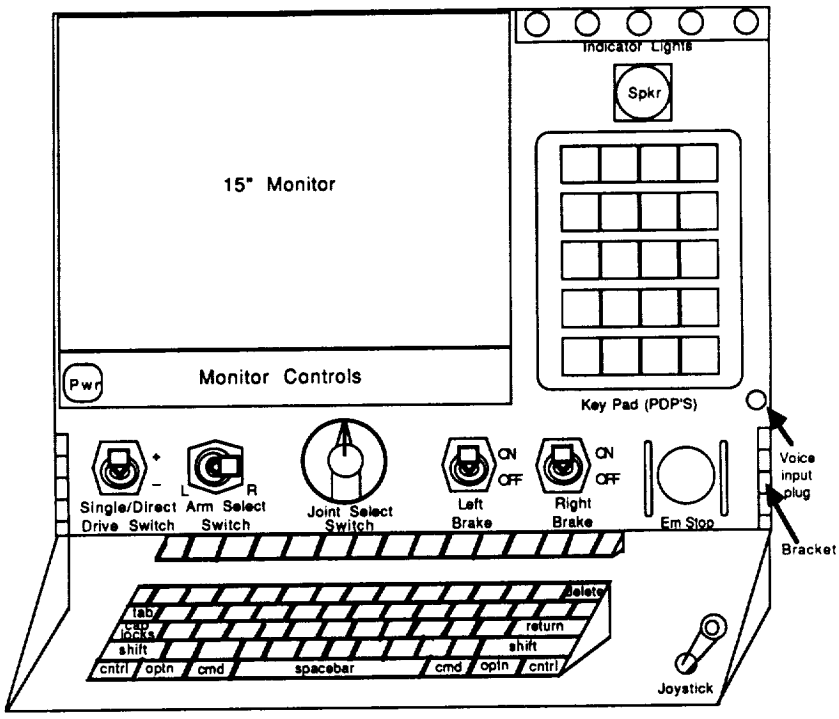


Figure 7 -- FTS control panel in aft flight deck panel A6-A2.

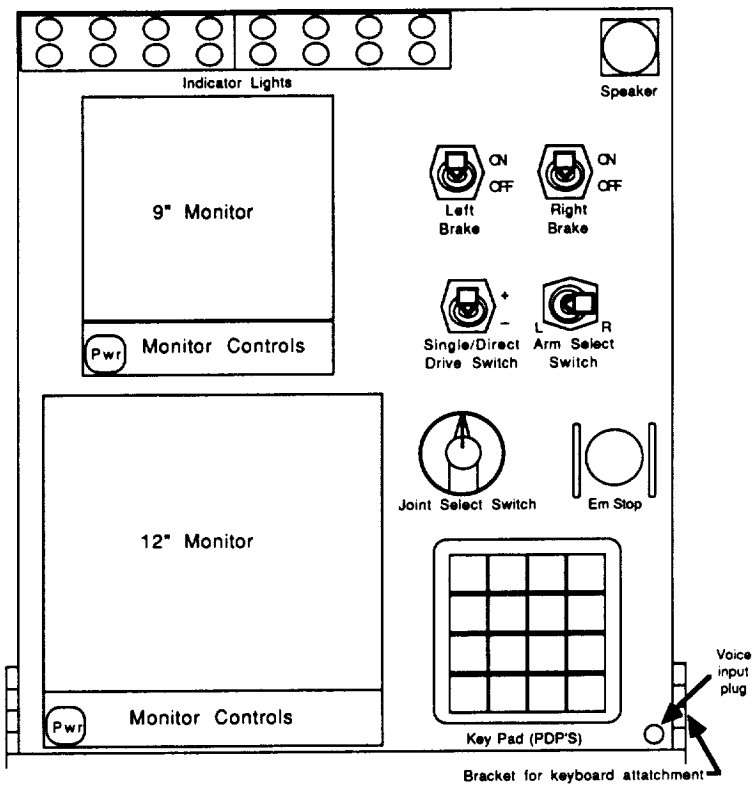


Figure 8 -- FTS control panel in mid-deck with nine-inch monitor and twelve-inch monitor.

to Lockheed Engineering and Sciences Company.

Thanks are also extended to Ervette Moore and Terence Fleming of Lockheed and Linda Dumis and Linda Orr of NASA for their invaluable assistance with the computer graphics packages described in this paper.

REFERENCES

1. Chubb, G. P., Laughery, Jr., K. R., and Pritsher, A. A. B., "Simulating manned systems," in: G. Salvendy (Ed.), HANDBOOK OF HUMAN FACTORS, John Wiley and Sons, New York, New York, 1987, pp. 1298-1327.

2. Gee, C. W., "Human engineering procedures guide," (AFAMRL-TR-81-35), Air Force Aerospace Medical Research Laboratory, Wright-Patterson Air Force Base, Ohio, 1981.

3. Gawron, V. J., and Polito, J., "Human performance simulation: combining the data," in: J. S. Gardenier (Ed.), SIMULATORS, Simulation Councils, Inc., La Jolla, California, 1985, pp. 61-65.

4. Stuart, M. A. and Smith, R. L., "Simulation of the human-telerobot interface," in: PROCEEDINGS OF THE SECOND ANNUAL SPACE OPERATIONS AUTOMATION AND ROBOTICS CONFERENCE (SOAR '88), National Aeronautics and Space Administration: Scientific and Technical Information Branch, Dayton, Ohio, 1988, pp. 321-326.

5. Stuart, M. A., Smith, R. L. and Moore, E. P., "Guidelines for the use of programmable display pushbuttons on the space station's telerobot control panel," in: PROCEEDINGS OF THE HUMAN FACTORS SOCIETY 32ND ANNUAL MEETING, Human Factors Society, Santa Monica, California, 1988, pp. 44-48.

6. Hawkins, J. S., Reising, J. M., and Woodson, B. K., "A study of programmable switch symbology," in: PROCEEDINGS OF THE HUMAN FACTORS SOCIETY 28TH ANNUAL MEETING, Human Factors Society, Santa Monica, California, 1984, pp. 118-122.

7. Burns, M. J., and Warren, D. L., "Applying programmable display pushbuttons to manned space operations," in: PROCEEDINGS OF THE HUMAN FACTORS SOCIETY 29TH ANNUAL MEETING, Human Factors Society, Santa Monica, California, 1985, pp. 839-842.

8. Gray, J., "The role of menu titles as a navigational aid in hierarchical menus," SIGCHI BULLETIN, New York, New York, 17, 3, January, 1986, pp. 33-40.

DISTRIBUTED EARTH MODEL / ORBITER SIMULATION

Erik Geisler / IBM
Scott McClanahan / Ford Aerospace
Dr. Gary Smith / IBM

NASA Johnson Space Center
Workstation Prototype Lab
FS-7
Houston, Texas 77058

ABSTRACT

Distributed Earth Model / Orbiter Simulation (DEMOS) is a network based application developed for the UNIX environment that visually monitors or simulates the Earth and any number of orbiting vehicles. Its purpose is to provide Mission Control Center (MCC) flight controllers with a visually accurate three dimensional (3D) model of the Earth, Sun, Moon, and orbiters, driven by real time or simulated data. The project incorporates a graphical user interface, 3D modelling employing state-of-the art hardware, and simulation of orbital mechanics in a networked / distributed environment. The user interface is based on the X Window System and the X-Ray toolbox. The 3D modelling utilizes the Programmer's Hierarchical Interactive Graphics System (PHIGS) standard and Raster Technologies hardware for rendering / display performance. The simulation of orbiting vehicles uses two methods of vector propagation implemented with standard UNIX / C for portability. Each part is a distinct process that can run on separate nodes of a network, exploiting each node's unique hardware capabilities. The client / server communication architecture of the application can be reused for a variety of distributed applications.

1. INTRODUCTION

This paper describes a graphics project under development by the NASA / Johnson Space Center (JSC) Workstation Prototype Lab (WPL) staff that provides a scene generation tool capable of maintaining and displaying a realistic model of the Earth and various orbiting objects. Display output may be used to drive a large screen projector or closed circuit TV. The four major components of the application will be described. The first section covers the architecture and communication between the different tasks. The second section describes the user interface that controls the system. The third section is the model manager, which is the center of the application that manipulates the 3D graphics and coordinates the simulations. The final section discusses the simulation task, which generates positional and attitude data representing an orbiting vehicle.

2. BODY

2.1. Architecture

DEMOS is based on a server/client model. The model manager is the focal point of the system. It performs the server function, servicing requests from the client processes. The clients include one user interface task and several simulation tasks. There is one simulation task per orbiting vehicle, and several vehicles may be viewed simultaneously.

The processes that comprise DEMOS are Local Area Network (LAN) transparent, as a result, each task may run on different network nodes. Communication between the tasks is accomplished by passing packets via UNIX sockets, which is compatible across multiple vendor workstations. The sockets also work within a single workstation, so full flexibility is provided in defining the topology of the application. Configuration of each task's node can be defined by the user at run time.

The application may be distributed over multiple workstations to off-load computations to machines more appropriate for that type of work. The model manager must run on the workstation containing the target graphics hardware. Eliminating nearly all other processes on the model manager workstation, allows it to run at real time priorities, thus allowing the 3D image updates to occur more frequently. The graphical user interface is dependent on the X Server, graphics hardware, keyboard, and mouse, but it does not use much CPU, so a low end workstation is acceptable. The simulation tasks are CPU bound and profit from floating point hardware.

Figure 1 shows the system configuration of DEMOS. Circular components denote system processes. Double ended arrows represent communication between processes. Rectangluar boxes represent external data files. The "Config Data" contains system initialization information. The "Model Defs" contain external scene descriptions and model geometry.

These files are read by the model manager in order to construct a hierarchical scene. The user interface process is started first and employs the services of the X server. Upon successful initialization, the user interface process starts the model manager. The model manager in turn starts the Sun, the Moon, and any number of orbiting vehicle simulations. The system is shutdown in reverse order. The model manager terminates all simulation processes before terminating itself. The user interface is shutdown immediately upon a user request.

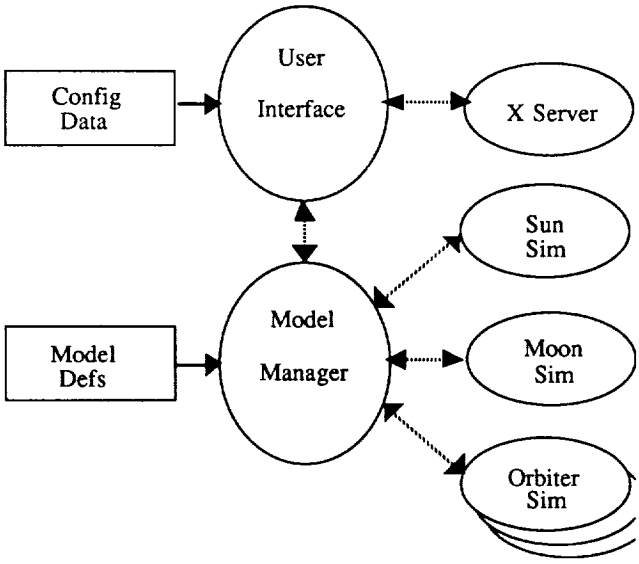


Figure 1 - DEMOS System Configuration

Each task in DEMOS uses the same packet send and receive subsystem. This prevents any task from blocking for I/O on socket operations. Since many packets may be sent at once, a queue holds the packets that the UNIX kernel cannot keep internally. The socket "streams" protocol is used to guarantee packet delivery and ordering. The queuing system also handles sending a partial packet. These packets can vary in size. At the receive end, the same subsystem isolates the application from incomplete packets by building and returning only complete packets.

2.2. User Interface

The user interface task provides the single point of contact between a user and DEMOS. It provides full control of DEMOS, including the ability to initiate and shutdown the system. The user interface does not have to be present after initializing DEMOS. The user can logoff the user interface and DEMOS will continue running. The system allows only one user interface to run at a time to ensure system consistency. System advisories are normally sent to the user interface, but if it is not present, then they are queued by the model manager until a user logs on.

There are two versions of the user interface for DEMOS - a graphical version, "xruif", and a command line interpreter version, "shuif". Xruif is based on the X Window System, so it is dependent on a graphics terminal controlled by an X server. On the other hand, shuif will run on any ASCII terminal, allowing more portability. The only difference between shuif and xruif is the user interaction. The low level areas of the two user interfaces share the same services.

2.2.1. Shuif

The command line interpreter version of the user interface is similar to the UNIX shell ("sh" or "csh"). It prompts for input from the keyboard, parses the input line for the command, and then executes the command.

A generic command line interpreter subsystem was created in the process of developing shuif. The first word of the input line is taken as the command name with the rest of the line being the arguments to the command. The command definitions, which are table driven, include the invocation, or callback, subroutine, as well as help information. The command line interpreter can be recursively nested to simulate submenus of commands. This subsystem also uses the "select" system call to block for I/O pending on any file descriptor. Each file descriptor has a corresponding callback routine which is called to process its data. I/O.

"Autotype" is a feature that allows shuif to run a complete user interface session in batch mode, reading commands from a file and echoing them to the screen as if they were typed by the user. This is useful for hands free demonstrations and test scripts. A "wait" command is included in shuif that suspends the user interface until an event occurs, such as waiting for the list of models to arrive before requesting a model to be loaded. Autotype files use the wait command to synchronize events within the system.

"Playback" is similar to autotype in that it allows the user working interactively with shuif to run a sequence of commands from a file. This is a convenience feature for redundant commands and to modularize operations involving a series of commands. Playback can also be invoked from an autotype file.

To complement the autotype and playback features, shuif can record (to a file) everything typed in by the user. Recording can be turned on or off at any time. Recording to an existing file appends the new commands.

Shuif provides commands for entering data for the simulation and base date values. When a simulation is started or the base date is set, the user has the option of using a data file or typing in all of the values. The data files can be created by a separate command that prompts the user for the values. All data files are validated when they are created and read.

2.2.2. Xruif

Xruif, the graphical user interface to DEMOS, is based on X11 Release 2. Since X is becoming a windowing standard, X clients are source code portable across many vendors' workstations. Xruif uses the X-Ray toolbox developed by Hewlett-Packard. The latest version of X-Ray is from the Release 3 tape of X11 from MIT. Xruif currently runs on a Sun 3/60 with the MIT X server in either monochrome or color.

The user interface style of xruif was not intentionally based on any existing application or style. It is based on the available X-Ray editors.

Xruif is composed of a single window divided into tiled panels. At the top is the title, followed by the main menu, then a work area, and an advisory panel at the bottom.

The work area is a reserved space where transient panels reside. All of the work area panels are the same size, even though their contents may not fill the panel. The work area panels are composed of a selection lists, (such as models, viewport configurations, viewport mappings and eyes, active vehicles, and the on-line help screens), or data entry screens for the base date and simulation values. These panels are activated by a main menu selection, and only one panel can occupy the work area at a time. When no panels are visible in the work area, a simple panel with the "work area" label in the center is left visible.

The user interface gadgets in X-Ray are called editors. Many of the X-Ray editors are used in xruif. The title bar editor contains a graphically offset single line of text with a selection box at each end. Each panel in xruif contains a title bar with a selection box containing a question mark for displaying help information on that panel. The push button editor is a matrix of oval buttons containing a label that is selected with the mouse. The main menu is comprised of push buttons. The list editor is a rectangle with an optional title bar at the top, optional scroll bars on the side, and a list of text that can be scrolled and selected (highlighted) with the mouse. This editor is used extensively in xruif. The text editor is a data entry field with a prompt to one side. It is used to enter simulation and base date values. A message box editor pops up a window containing an icon, some text, and some push buttons. It is used to force answering "Are you sure?" questions. A group box editor is simply a rectangle with a label at the top to surround a group of editors and visually associate them.

The title panel also uses the static raster editor to display pixmaps, such as the NASA logo, the WPL logo, and the DEMOS icon. The DEMOS icon is also used to represent the xruif window when it is closed.

There are several advantages to using separate panels. First, it modularizes each component of the user interface. Any panel can easily be modified and rearranged without affecting the other panels. Second, the X window events "entry" and "leave" are used to determine when the pointer goes into or out of a panel. This allows each panel to do its own input processing instead of having to handle all inputs in one routine. Third, each panel can size and create its own editors. Finally, panels can be redisplayed independently, each handling its own "expose" events.

Xruif employs an on-line help facility for information on each panel. The title bar of each panel has a help icon, which, when selected, brings up the help panel in the work area, overlaying the previous panel. When the help panel is terminated, the previous panel is restored. The help panel contains scrollable help text, plus a help index listing all help screens. Selection of a help index item displays that panel's help screen. The help screens are loaded at initialization from ASCII text files. For DEMOS, help screens were formatted by the "nroff" utility and can be easily customized by the user.

2.3. Model Manager

The model manager is responsible for servicing user interface requests, loading data models, managing simulations, and generating accurate visual displays of a modeled scene. Its implementation employs the PHIGS standard, as well as PHIGS+ extensions for lighting and shading. The Raster Technologies PHIGS+ subsystem off-loads a number of graphic functions including model hierarchy management, traversal, and rendering/display. Many functions are performed in firmware. Using this graphics architecture, the model manager is able to concentrate on a variety of control functions associated with managing multiple, asynchronous simulations. The model manager is composed of three major elements: Scene Construction, Simulation Management, and View Generation.

2.3.1. Scene Construction

After communications have been established with the user interface task, the model manager begins by constructing an in-memory tree representation of a selected scene hierarchy. A user selects a scene by choosing a top-level description file. The model manager reads this verb-based description file in order to build an internal representation of the scene. Description files may reference other description files. In this manner, a complex external model hierarchy may be defined. The model manager will recursively read these files until the entire scene tree is built. Using this technique, generic scenes may be developed and processed by a general modelling subsystem.

Besides model hierarchy construction, description files provide additional information which is attached to the model's geometric definition. This separation of model geometry and model attributes allows models to be tailored for rendering performance versus realism. Each file has a specified type, which determines how the remaining commands are to be interpreted. A variety of types are currently supported: 'model', 'eye', 'camera', 'scene', 'light', and 'ghost'.

A 'model' description file provides the following information: the model units, initial placement, display options (polygon, vector, polyline), shading method (flat, Gouraud), surface properties, color model, hidden-line/hidden-surface options and an optional reference to a data file containing the actual geometric model. Model attributes are inherited from parent models. Typically, a top level model node provides overall model information and attributes while children nodes reference individual submodels and define how they are geometrically related to their parent.

An 'eye' or 'camera' description file provides the definition of viewing parameters for a single viewpoint. The only distinction made between eyes and cameras is that cameras represent physical optical devices while eyes define a synthetic viewpoint. Both are treated as submodels, positioned relative to their parent node. By attaching eyes and cameras to a geometric model, a wide variety of views can be supported. This viewing mechanism forms a major element within the model manager. A majority of the model manager's computational effort is spent maintaining selected views. The following information can be defined for an eye or camera: camera position, camera orientation, perspective reference point, view distances (front, view, back), projection type (parallel, perspective), and viewing window parameters.

A 'scene' description file defines global scene characteristics. Specifically, it provides the following information: scene lighting method (ambient, diffuse, specular, none), true or pseudo color display indicator, background screen color, viewport edge characteristics, viewport titles flag, initial viewport definition(s) and background colors, screen aspect ratio, Normalized Projection Coordinate the (NPC) window and Device Coordinate (DC) viewport in which NPC window will be mapped. Many of the developed scenes refer to a common scene node since this information rarely changes. Changing the scene lighting and the number of viewports can drastically affect scene display rates. The DC viewport provides the capability to place the graphic display into a selected portion of the screen. This becomes important when the RGB signal is converted to video via converter boxes such as Genlock or RGB Technologies VideoLink.

A 'light' description file defines a single light source. Ambient, infinite, point, and spot light types are supported. Depending on the light type, a number of lighting characteristics may be defined, such as color, location, direction, concentration exponent, and cone of influence. Adding additional lights seems to have only a minimal computational effect on the overall rendering process. DEMOS currently employs a single infinite light source - the Sun. Additional lights might be added to have the orbiter always visible to the user even though it is positioned on the dark side of the planet.

A 'ghost' description file defines an object which assumes and maintains a position relative to its immediate parent node. As its name implies, a ghost object is an invisible object which cloaks another object. Ghost objects are semi-attached to their parent. This is, they only receive positional updates; attitude transformations are not applied. These type of objects are typically used to establish a set of viewpoints associated with an orbiting vehcile. Since these viewpoints only accept positional updates, and therefore move along with an object, they are capable of viewing rotational (attitude) changes to the object in which they are connected. This feature provides a flexible viewing mechanism and is used to support visual verification of spacecraft orientation, as well as, Earth rotations from a point in space.

During description file processing and hierarchical scene tree construction, a set of linear lookup tables are developed in order to minimize the model editing process. Each table entry contains a unique object name followed by a tree node pointer. The use of this pointer eliminates unnecessary tree traversals by the model manager when updating an object's position and attitude. In addition, the PHIGS structure ID is obtained from the tree node, and is used to perform PHIGS editing. The reason in-memory scene tree structures are edited along with the PHIGS structures is to facilitate the formation of a particular view from an eye or camera. The PHIGS specification allows structure inquiries to obtain this information; however, the PHIGS implementation currently used does not support this operation. Combining an in-memory tree structure and the sorted lookup tables provides an efficient framework for model editing. The linear table provides efficient model searching while the in-memory tree structure provides the necessary model hierarchy.

After the in-memory scene tree has been constructed, it is loaded into the PHIGS Central Structure Store (CSS). The CSS provides a central database where graphics information is stored and edited. In order to construct the PHIGS database in a contiguous manner, the model manager recursively traverses the in-memory scene tree and loads each model and light node into the CSS. If a child node representing a model, ghost, or light is referenced within the current node, a PHIGS structure execution command is issued to link

this child node to its parent. Eye and camera nodes are ignored during this PHIGS loading process and are managed separately.

2.3.2. Simulation Management

The processing architecture of the model manager is based on a state machine approach employing time and events. To manage multiple, asynchronous simulations, the model manager must maintain its own internal clock. This clock is established by the user issuing the system start time command. Once the time is set, the model manager begins propagating it by a discrete unit. In addition, the model manager automatically spawns a Sun and Moon simulation task on previously defined workstations. The user also determines how quickly time should propagate and the amount of Earth rotation per display update. This clock is used to synchronize all events within the model manager.

Simulations are initiated upon reciept of a user interface request. The model manager retains the specified simulation information within an internal state structure. These structures hold and maintain information neccessary for communications, groundtrack requests, and position and attitude requests. States transition from one state to another due to an occurrence of an event. For example, a simulation is not started until the internal clock is equal to or greater than the simulation starting time. Once started, the simulation, or monitoring element, transitions from the 'wait_to_start' state to the 'has_started' state. Typically, simulations enter a cyclic state where the model manager continually requests their next position for the current time of interest (e.g., the internal system time). Since the model manager makes all the requests, it controls the rate at which simulation elements respond. Simulation or monitoring elements never send unsolicited information. This greatly simplifies their control. In effect, simulation management is handled via a master / slave approach rather than with the client / server relationship held with the user interface. This control technique also ensures the model manager is never inundated with data from clients' simulations.

The notion of a time node was developed to maintain an accurate visual display of multiple moving objects. When time is propagated, a node is allocated and placed on the end of a time list. For each time unit, a request is generated for each active simulation element in order to update its position, attitude, or light direction. These requests are attached to the current time node. When the simulation element responds with appropriate data, the corresponding model is updated to reflect this update, and the request is removed from the appropriate time node. Once all requests for a particular time node have been removed, the time node is freed and the scene is in a correct state for the next display. If all requests for a time node have been removed, and an earlier time node still contains outstanding requests, then these earlier time nodes are destroyed - leaving only the latest information. By employing time nodes and multiple requests per time unit, the accuracy of the visual display is ensured.

2.3.3. View Generation

The model manager spends the majority of its processing maintaining accurate visual representations of the scene being modeled. It supports a wide array of scene viewing capabilities. Under user control, the graphics screen may be partitioned into a number of viewports. Each viewport is treated as an empty slot in which an eye or camera may be assigned. Only one view (eye or camera) may be assigned to a particular viewport at any one time; however, a view may be assigned to multiple viewports. Viewports have a background color, and are outlined to indicate their screen coverage. In addition, viewports have the property of visibility. The user may wish to temporarily turn off a particular viewport to improve display rates or to ignore uninteresting views. Viewports which have an assigned eye or camera may have a small title displayed to help identify the particular view. These titles are extracted from the corresponding eye or camera nodes from within the scene tree.

During the scene tree construction phase, a default viewport configuration file and a default view assignment file are read to provide an initial viewing framework. This framework is used to view the scene prior to starting simulations. Once a scene is loaded, the user interface requests a list of all available viewport configurations and their current view assignments. Given this information, the user may freely assign views to viewports, toggle viewport visibility, or select another viewport configuration.

The model manager constructs a view for a given viewport in the following manner. First, a view is assigned to a particular viewport by copying the specified viewing parameters to the desired viewport data structure. A view mapping matrix is then computed for this viewport. The next step involves the actual view generation, given an arbitrary viewing position in modelling coordinates. Since the eye coordinate system is fixed, it is necessary to transform the world coordinate system into this eye coordinate system. The scene tree contains all information concerning model hierarchy, and is therefore used to compute this transformation by traversing the scene tree backward from the eye or camera node to the tree's root node. Initially, the eye's orientation is set to the identity matrix. This matrix is then transformed by applying inverse transformations while traversing up the tree. Once the root node is reached, a final orientation matrix has been formed, and it is then associated with the corresponding viewport.

The viewing computation is then completed by loading the newly computed viewing represen-

tation and allowing PHIGS to traverse the hierarchical model contained in the CSS. In order to minimize the view construction, only the assigned, visible, viewports are computed.

2.4. Simulation Components

The simulation tasks provide the model manager with the necessary data to maintain an accurate representation of the Sun, the Moon, any number of orbiting vehicles, and the orientation of the Earth within the M50 coordinate system (the basic JSC inertial coordinate system).

Currently, three types of simulation tasks are supported: a Sun simulation, a Moon simulation, and an orbiting vehicle simulation. Simulation tasks are started by the model manager via a remote procedure call. They are typically deployed on workstations providing floating point hardware. Once a simulation has successfully started and has established communication with the model manager, it is sent a packet containing all information required to begin processing.

A simulation component of DEMOS consists of up to five functional elements:

1) Compute the Rotation-Nutation-Precession (RNP) matrix. The RNP matrix relates the M50 coordinate system to a coordinate system fixed to the Earth.
2) Generate from one to ten orbits worth of ground tracks for the orbiting vehicle
3) Determine the position over the Earth of the orbiting vehicle.
4) Determine the attitude of the orbiting vehicle axes relative to an Earth-fixed coordinate system so that the vehicle maintains a pitch, roll, and yaw of zero degrees relative to the UVW local orbital reference frame (U is a unit vector in the direction of the radius vector, W is a unit vector in the direction of the angular momentum vector, and V is the unit vector which forms a right-handed system). (Note that the body axis system for this application has the x-axis out the nose of the orbiter, the z-axis out the top of the orbiter, and the y-axis out the left wing).
5) Determine the location of the Sun and the Moon.

2.4.1. Computation of the RNP Matrix [12]

The fundamental transformation matrix for the simulation component is the RNP matrix. It incorporates all of the precession, nutation and rotation changes that have affected the orientation of the Earth in inertial space since 1950. It relates the orientation of an axis system fixed to the Earth relative to the M50 coordinate system. The user interface task provides the base-time-of-interest values. These include: the year, month, day, hour, minute and second. The time difference between Ephemeris Time and Universal Time Corrected is also provided.

Once the input base time has been obtained, the computation of the RNP matrix proceeds as follows:

1) Calculate the Julian Universal Date and the Julian Ephemeris Date.
2) Compute the three precession angles.
3) Compute the precession transformation matrix, P.
4) Compute the nutation angles.
5) Compute the nutation in longitude.
6) Compute the nutation in obliquity.
7) Compute the nutation transformation matrix, N.
8) Compute the rotation transformation matrix, R, which orients the X-axis (through Greenwich) for the base time of interest.
9) Compute the RNP matrix by multiplying the R, N, and P matrices together.
10) Perform the z-axis rotation to rotate the RNP matrix back to December 31, 0 hours, 0 minutes, 0 seconds of the previous year.

This fundamental RNP matrix is employed to transform an M50 vector (for a given time) into an Earth-fixed vector. This is extremely important for the generation of ground tracks or for positioning a vehicle over the surface of the Earth.

2.4.2. Generation of Ground Tracks

The user interface task provides the number of orbits worth of ground tracks that are to be displayed. This number is passed to the ground track simulation element where 180 sets of Earth-fixed latitude and longitude points are generated for each orbit. These points are passed to the model manager which then displays the ground tracks on the 3D Earth Model.

Each Earth-fixed latitude and longitude point is computed as follows:

1) Propagate the state vector to the particular time along the ground track (the delta time between propagation steps is the period of the orbit divided by 180 points).
2) Calculate an updated RNP matrix using the time of the state vector.
3) Transform the propagated M50 position vector into and Earth-fixed position vector using the RNP matrix.
4) Calculate the Earth-fixed latitude and longitude from the Earth-fixed position vector.

The ground track simulation element either uses a two-body propagation method or a modified Analytic Ephemeris Generator (AEG) propagation method to generate the state vectors from which the latitude and longitude points are computed. The user specifies which propagation method is desired when the initial state vector is entered.

2.4.3. Computation of Position

The user interface provides values for the initial simulation. The object could be the shuttle orbiter, the Space Station, or any other satellite of the Earth. The user decides the choice of units (e.g. feet, meters, or Earth radii) and the initial time of the 'state'. This 'state' can be entered either as M50 position and velocity vectors or as M50 Keplerian orbital elements (semi-major axis, eccentricity, inclination, longitude of the ascending node, argument of perigee and mean anomaly). Finally, the choice of propagation method is entered. This can be either two-body or AEG propagation. When the simulation element is initialized, additional orbital parameters are computed which will be utilized in the propagation of the position and velocity. In the case of two-body propagation [6], the additional parameters include unit vectors in the two-body orbital plane. An AEG propagation is initialized by computing a set of 'invariant' elements which can be used to propagate position and velocity including the effects of the J2, J3, and J4 gravitational harmonics of the Earth. The AEG propagation method is a scaled down version of Edgar Lineberry's Analytic Ephemeris Generator [8] which is used in the MCC to support missions. The scaled down version computes the effects of the short-period terms on the orbital elements. The version implemented for DEMOS does not include the calculation of drag effects.

When a time is given to the position computation element, Get_position, the position and velocity are propagated to that time. This time may be sent to Get_position either by the ground track computation element or by the model manager. Once the position and velocity are propagated, a series of coordinate transformation routines rotate these vectors into an Earth-fixed coordinate system. The Earth-fixed position vector is then used to calculate the Earth-fixed latitude and longitude.

2.4.4. Computation of Attitude [9]

After the position is obtained, the pitch, roll, and yaw attitude angles of the orbiting vehicle relative to the Earth-fixed reference frame are computed to maintain the vehicle attitude of its body axes relative to the UVW local orbital reference frame. This "UVW hold" attitude causes the shuttle to appear on the graphics screen with its nose parallel to the ground tracks and the plane of the wings perpendicular to the radius vector. This allows the payload bay doors to be visible to a viewer looking down on the shuttle as it orbits the Earth.

2.4.5. Computation of the Sun Position [2,3]

The model manager has the capability to provide lighting over the surface of the Earth, by knowing where the Sun is relative to the Earth-fixed coordinate system. The Sun position computation element, Get_sun, calculates the Earth-fixed position of the Sun by the following steps:

1) Calculate the precession angles and the precession matrix P, for the time of interest.
2) Calculate the Mean Longitude of Perigee for the Sun relative to the Mean Equinox of Date.
3) Calculate the Mean Anomaly for the Sun relative to the Mean Equinox of Date.
4) Calculate the eccentricity of the Earth's orbit around the Sun.
5) Solve Kepler's equation for the Eccentric Anomaly of the Sun.
6) Calculate the True Anomaly of the Sun.
7) Calculate the Mean Obliquity of the Ecliptic.
8) Calculate the longitude of the Sun in the Ecliptic plane.
9) Calculate the magnitude of the radius vector from the Earth to the Sun.
10) Compute the position vector of the Sun in Ecliptic coordinates.
11) Apply the precession matrix, P, to this Ecliptic vector to compute the position vector for the Sun in M50 coordinates.
12) Rotate this M50 position vector using the RNP matrix to Earth-fixed coordinates and extract the Earth-fixed latitude and longitude of the Sun.

2.4.6. Computation of the Moon Position [2,3]

The model manager can move one of its viewpoints sufficiently far from the Earth-Moon system so that both the Earth and the Moon are visible in the same view. If the system time is accelerated the Moon can be seen to orbit the Earth.

The Moon position computation element, Get_moon, calculates the Earth-fixed position of the Moon by the following steps:

1) Compute the precession angles and the precession matrix, P.
2) Calculate the nutation angles.
3) Calculate the Ecliptic latitude, Ecliptic longitude, and parallax of the Moon using Fourier Series Expansions (sine and cosine terms) of combinations of the nutation angles.
4) Compute the magnitude of the radius vector from the Earth to the Moon.
5) Compute the position vector of the Moon in the Ecliptic plane.
6) Rotate the Ecliptic position vector into the M50 coordinate frame using the precession matrix, P.
7) Rotate this M50 position vector using the RNP matrix to Earth-fixed coordinates and then extract the Earth-fixed latitude and longitude of the Moon.

Eventually, the position, velocity and attitude information for the orbiting vehicle will be obtained over the LAN from the Mission Operations Computer (MOC) or Calibrated Ancillary System (CAS). The internal units for the simulation component have been kept compatible with the Ground Based Space Sys-

tems (GBSS) internal units on the MOC to ease this transition.

3. CONCLUSION

DEMOS is a successful implementation of 3D modelling employing accurate simulations of the Earth, Sun, Moon, and any number of orbiting objects. It provides a visualization tool which has the capability to simulate / monitor orbiting objects and to display a realistic scene in an acceptable time period. A flexible viewing system allows flight controllers to view objects from a variety of viewpoints. Vehicle cameras and synthetic eyes may be defined to inspect spacecraft activity from arbitrary view positions. The distributed architecture provides the framework for future application extensions. Application software employs the latest workstation standards, maximizing its lifecycle while minimizing any rehosting costs. Simulation techniques are implemented from proven algorithms.

4. ACKNOWLEDGEMENTS

The authors wish to thank Randall Barnett of Lincom Corporation for his PHIGS assistance and software techniques. His optimized algorithms improved view generation times considerably. We would like to also thank the Mission Planning and Analysis Division (MPAD) graphics lab for their generous supply of high fidelity graphics models of various spacecraft, which helped assimilate realistic scenes.

5. REFERENCES

1. Computer Science Corporation, FLIGHT DYNAMICS / SPACE TRANSPORTATION SYSTEM 3-D MONITOR SYSTEM RELEASE 2 SYSTEM DESCRIPTION, CSC/SD-88/6066, Contract NAS 5-31500, Task Assignment 58 214, NASA Goddard SFC, Greenbelt, Maryland, August 1988.

2. Escobal, P. R., METHODS OF ASTRODYNAMICS, John Wiley & Sons, Inc., New York, 1968.

3. EXPLANATORY SUPPLEMENT TO THE ASTRONOMICAL EPHEMERIS AND THE AMERICAN EPHEMERIS AND NAUTICAL ALMANAC, H. M. Stationery Office, London, 1961.

4. Foley, J. D., Van Dam, A., FUNDAMENTALS OF INTERACTIVE COMPUTER GRAPHICS, Addison-Wesley Publishing Company, Philippines, 1982.

5. Gettys, Jim, Newman, Ron, Scheifler, Robert W., XLIB - C LANGUAGE X INTERFACE, Massachusetts Institute of Technology, Cambridge, MA, 1987.

6. Herrick, Samuel, ASTRODYNAMICS, Volume I, Van Nostrand Reinhold Company, London, 1971.

7. Hewlett-Packard Company, PROGRAMMING WITH THE XRLIB USER INTERFACE TOOLBOX, February 1988.

8. Lineberry, Edgar, INVARIANT ORBITAL ELEMENTS FOR USE IN THE DESCRIPTION OF MOTION ABOUT AN OBLATE EARTH, JSC Internal Note No. 74-FM-84, December 4, 1974.

9. MATHEMATICAL STANDARDS AND GUIDELINES, IBM GBS Programmer's Guide, Section 6, September 1, 1976.

10. Mortenson, M. E., COMPUTER GRAPHICS: AN INTRODUCTION TO THE MATHEMATICS AND GEOMETRY, Industrial Press Inc, New York, NY, 1989.

11. O'Reilly and Associates, XLIB PROGRAMMING MANUAL FOR VERSION 11 RELEASE 2 OF THE X WINDOW SYSTEM, Volume I, O'Reilly and Associates, Newton, MA, April 1988.

12. Schulenberg, C. W., DESCRIPTION OF A SELF CONTAINED SUBROUTINE WHICH ANALYTICALLY GENERATES INTERPLANETARY COORDINATE SYSTEM TRANSFORMATIONS REFERENCED TO THE MEAN OF 1950.0 EPOCH, TRW Note 70-FMT-853, Sept. 30, 1970.

Prototype Part Task Trainer a Remote Manipulator System Simulator

David Shores
Barrios Technology Inc.
1331 Gemini
Houston, Texas 77058

Abstract

The Part Task Trainer program (PTT) is a kinematic simulation of the Remote Manipulator System (RMS) for the orbiter. The purpose of PTT is to supply a low cost man-in-the-loop simulator, allowing the student to learn operational procedures which then can be used in the more expensive full scale simulators. PTT will allow the crew members to work on their arm operation skills with out the need for other people running the simulation. The controlling algorithms for the arm were coded out of the Functional Subsystem Requirements Document to ensure realistic operation of the simulation. Relying on the hardware of the workstation to provide fast refresh rates for full shaded images allows the simulation to be run on small low cost stand alone work stations, removing the need to be tied into a multi-million dollar computer for the simulation. PTT is not intended to replace the full scale simulators but to augment the training process and reduce the load of the full scale simulators, especially when the student is learning new procedures and is error prone. PTT will allow the student to make errors which in the full scale mock up simulators might cause failures or damage hardware. On the screen the user is shown a graphical representation of the RMS control panel in the aft cockpit of the orbiter, along with a main view window and up to six trunion and guide windows. The dials drawn on the panel maybe turned using the dials on the dial box to select the desired mode of operation. The inputs controlling the arm are read from a chair with a Translational Hand Controller (THC) and a Rotational Hand Controller (RHC) attached to it.

INTRODUCTION

Part Task Trainer (PTT) is a kinematic simulator for the shuttle remote manipulator system(RMS). This paper will discuss what PTT does, it's history, uses, operation, design and the future of the program.

The controlling algorithms for the arm are coded from the functional subsystem software requirements document (FSSR) to ensure operation as close to the real arm as possible. Five of the computer supported modes and one of the non-computer supported modes are modeled. These modes supply the student with training in the major RMS modes of operation.

HISTORY

PTT started out as two separate programs on two separate machinès. The graphics were done on an IMI500 in wire frame and the simulation on an HP9000. When the Silicon Graphics 4D/60 was announced it was decided these two programs could be merged and provide better functionality. The controlling algorithms were coded from the FSSR and merged with already existing display code. This allowed us to deliver a limited working version in two months.

USES

PTT will be used in the training cycle for the crew members. It will provide inexpensive hands on training in an environment were mistakes can cause no damage to hardware. In the full scale simulator if the student makes a mistake damage to the equipment could be costly. But with PTT the worst damage only means restarting the simulator not rebuilding the hardware. PTT is not meant to replace the large scale simulators, but to augment them. The large scale simulators are expensive to run (computer time, support personnel), but PTT needs no support personnel, it is all self contained. It will allow the crew members more time to work with the arm and learn the different modes of operation. It will be used to maintain proficiency of operation, warm up for the integrated simulations and flight specific training. It is also used for engineering studies of reach limits and space station assembly.

OPERATION

PTT started out originally to be only a single joint operation simulator. But with the capability of the machine for floating point operations it was decided to include the computer supported modes. In single joint mode the operator is working with only one joint at a time. Therefor, the movements of the end effector will be an arc rather than a straight line as in the computer supported modes

In single joint mode the user selects a joint with the joint knob and inputs a positive or negative rotation with a toggle switch on the chair. There is no joint software reach limit checking done since this mode is used to drive the arm out of reach limits. In four of the computer supported modes (orbiter unloaded, end effector, orbiter loaded and payload) the translational hand controller (THC) and the rotational hand controller (RHC) are used to control the point of resolution (POR). The POR is the point about which the rotations are calculated, generally this is the tip of the end effector or a point located inside the payload. The translations translate the POR in a straight line along the axis of the coordinate system and the rotations are done about the POR. If the arm is in orbiter unloaded mode the coordinate system used is the orbiter's. In orbiter loaded mode the coordinate system is the orbiter's plus any offset added by the user for the POR. In end effector mode the coordinate system is the tip of the end effector. In payload mode the offset is added to the end effector position for the final POR. The THC provides positive and negative input on all three axis. The RHC provides positive and negative rotations for pitch, yaw and roll.

The last computer supported mode, operator commanded (OCAS), deals with the POR the same way as the other four. The difference is in the input for movement. In OCAS the user enters the position and attitude desired for the end effector. If it is a valid position and attitude, meaning the arm can reach it, the software attempts to drive the POR to this position and attitude in a straight line. The software does no checking for reach limits, singularities or interference when checking the final position and attitude. But reach limits and singularities are checked when the arm is being driven to the new position and if one occurs the user must deal with it. Interference between models is left up to the user just like the real RMS.

During the simulation the user has the option of practicing the grapple and release operation. When the grapple trigger is activated a list of possible grapple figures is checked to determine which one should be grappled. The grapple fixture must be within the constraints of the real arm, these are $-4 < [x,y,z] < 4$ and $-15 < [\text{pitch, yaw, roll}] < 15$. If the grapple is determined to be valid the arm is drawn to the grapple fixture and the payload is relinked dynamically to the arm. In other words the software determines the new position and attitude of the payload relative to the arm for the drawing hierarchy. This procedure can be reversed when a release is done. The new position and attitude relative to NULL is calculated for the payload and the hierarchy is changed to reflect this change. When the arm is going through a grapple or release sequence it takes approximantly the same amount of time as the real arm does to help reduce negative training.

These are the basic modes of operation for the RMS. Now we will discuss the link to the graphics interface.

DESIGN

The interface to the simulator is a graphic representation of the control panel for the RMS in the aft cabin of the orbiter, an alpha-numeric terminal, a buttons and dials box, the mouse and a specially designed chair with an RHC and a THC attached to it. The lower left quarter of the screen contains the panel. This is used to indicate which mode of operation is active. The actual panel has three dials for the mode control. These dials have been mapped to the dials box. Movement of the dials is reflected by the dials on the screen. The other buttons and toggle switches on the panel which are needed for the simulator are mapped to the buttons box. The alpha-numeric terminal is used to simulate the auxiliary display in the aft cabin. Two of the possible screens have been modeled. The DISP94 and SPEC96 screens. These are useed for input for operator commanded mode and position and attitude information display. The buttons and dials box is used for moving the camera around in X, Y, and Z, manipulating the dials on the panel, changing the active camera, and the switch input for the panel. The mouse is used for the toggling between wire frame and shaded views, and enlarging any of the view windows up to full screen or back. Also, in setup mode it is used to adjust models, cameras, lights and other operational information. The chair with the RHC and THC is used for arm control input. The chair communicates with the simulation over an RS-232 line.

The top left quarter of the screen is the main view window. This window contains the view from the active camera. The active camera can be changed to preset camera positions with the buttons or moved around in X, Y, and Z with the dials. The right side of the screen is used for special purpose windows. These windows are views of the trunions on the payload and the associated guides in the payload bay. These windows provide an unseeable view in the real world. They assist the student when doing single joint operations.

There are two types of models used in PTT. Those predefined by the software, the panel, and those built by another program and read in, the orbiter. 98% of the models read into PTT were created with the in house model building package Solid Surface Modeler (SSM).

The predefined models are the models used to draw the panel. These were hand designed and placed on the screen. Only the parts of the panel which change are updated. If the parameter dial is turned the parameter dial on the display will change as well as the number readout, but nothing else is updated. The models read into PTT are drawn in the view window. All of the models are update in the main view every time. Each of the special purpose windows has a list of models associated with them, if any of the models in the list are moved then the window is updated. Otherwise it is left unchanged.

The models read in are linked together in an hierarchy which tells the program where to draw each model. Each node in the hierarchy has a flag which is set if the position and attitude change. If so, every node that is a child to this node must be redrawn.

One of the design goals deals with the speed of updating the screen. Only drawing the models which move allows the graphics engine to do as little work as possible when updating the screen. Another design goal was to minimize negative training. Since mistakes on orbit can be costly or even dangerous all training is done as close to the actual procedure used in flight as possible for consistency. Some examples of this are labeling the dials and buttons on top instead of underneath. All the switches and knobs in the orbiter are labeled on top. Also the length of time is takes the grapple/release sequence. Since the arm can be moved with this operation is taking place the time in the simulator is approximantly the same amount used for the real arm so the user does not get in the habit of moving the arm to soon.

The justification for PTT is simple. Most of the code was already written but used in different programs. By using this code the maintenance of the code is relatively easy. It also means enhancements to the code are just as easy. The cost of operation is minimal. Once the student has an introduction course there should be no more need for instructors. Also the cost of the machine is small to the cost of the large simulators.

The future of PTT looks promising. It will go into the training cycle in April. So far everybody dealing with training who has seen PTT has liked it and are anxious to get it into the training cycle. As for program enhancements another view window and dynamics have been discussed. We are hoping to get a Silicon Graphics 240GTX which is a 4 processor parallel machine. We feel these enhancements would greatly improve the ability of the simulator. We also have several different versions of PTT. One allows the student to work with a two arm configuration. Another version of PTT is being developed for the Space Station Freedom arm.

With the ease of use, ease of modifications and speed of the simulator PTT should be very useful for training, maintaining proficiency and engineering studies.

SPACE STATION FREEDOM INTEGRATED FAULT MODEL

by Fred J. Becker
Lockheed Engineering and Sciences Company
2400 NASA Road 1, C87, Houston, TX 77058-3711

ABSTRACT

This paper describes a demonstration of an integrated fault propagation model for Space Station Freedom. The demonstration uses a HyperCard graphical interface to show how failures can propagate from one component to another, both within a system and between systems. It also shows how hardware failures can impact certain defined functions like reboost, atmosphere maintenance or collision avoidance. The demonstration enables the user to view block diagrams for the various space station systems using an overview screen, and interactively choose a component and see what single or dual failure combinations can cause it to fail. It also allows the user to directly view the fault model, which is a collection of drawings and text listings accessible from a guide screen.

Fault modeling provides a useful technique for analyzing individual systems and also interactions between systems in the presence of multiple failures so that a complete picture of failure tolerance and component criticality can be achieved.

1.0 INTRODUCTION

This paper illustrates a HyperCard user interface for a failure propagation model of the Space Station Freedom integrated systems. It uses as an example a typical session of investigating the failure tolerance of the integrated Space Station Freedom systems. It also provides some background on how the failure model and HyperCard interface was developed.

The failure propagation model was coded and solved using Digraph Matrix Analysis, a proprietary software toolset. The results were transferred from a VAX to a Macintosh and there provided the required data to the HyperCard graphical environment.

This project was performed for the Guidance, Navigation, and Control Systems Branch of the Avionics Systems Division of the Johnson Space Center as a prototype for failure modeling tools which can be used for Space Station Freedom.

2.0 ABOUT FAILURE MODELING

The failure model which is behind the HyperCard user interface contains information about failure propagation in Space Station Freedom systems. This information is accessed by the HyperCard stack on command from cues provided by the user.

The failure model is in the form of a directed graph (digraph), which is a network model of a system pictorially representing failure propagation throughout the system. The digraph consists of nodes representing system components, and arrows representing failure propagation from one component to the next. Inputs to a node represent all things that component depends on for proper functioning. Conversely, outputs from a node represent failure propagation from that component to other components.

AND gates are used to indicate functional redundancy. For example, a computer might be supplied with electrical power from two busses. This would be drawn as two node inputs to an AND gate feeding the computer. Both nodes must fail before the computer fails.

Digraphs can be used to model anything from electrical diagrams to logic diagrams, fluid diagrams,

mechanical systems, procedures or end-to-end functions. Failure tolerance can be studied, when the system digraph is completed, by looking at the failure propagation results to see either what failures a given component failure can cause or what other failures can fail a given component. The latter method (the more difficult problem) is how this demonstration presents the information. Single failures or double failures which can result in the failure of the target component are displayed on the screen.

3.0 SPACE STATION FREEDOM FAULT MODEL

The overall fault model used in this demonstration contains about 600 nodes representing orbital replacement units, data lines, piping, tankage and other components of the Space Station Freedom systems. The purpose of this demonstration is to provide a relatively small model which nevertheless illustrates the highly integrated nature of the Space Station Freedom systems. Therefore, the models have been kept fairly high-level. Several high level functions, felt to be the most critical, are also modeled by showing their dependence on the systems. Systems and functions included in this model are:

Data Management System (DMS)
Guidance, Navigation and Control (GN&C)
Communications and Tracking (C&T)
Thermal Control System (TCS)
Environmental Control and Life Support System (ECLSS)
Propulsion
Electrical Power System (EPS)
Extravehicular Activity System (EVAS)
Reboost
Attitude Control
Docking
Collision Avoidance
Fire Suppression
Atmosphere Maintenance

The initial fault modeling was done by drawing the digraphs on paper, based on the system block diagrams. The failure propagation modeled was based on judgements about whether a particular functional connectivity implied any failure propagation. The resulting drawings were then translated into text listings for use with the Digraph Matrix Analysis software, which computed the failure "reachability". Next the block diagrams were created on HyperCard stacks, using button names which corresponded to the digraphs. Of the total model, 545 buttons were chosen for display. The following section shows how the result works.

4.0 HYPERCARD DEMONSTRATION TUTORIAL

4.1 Navigating Through the Stack

Figure 1 illustrates the opening card. This card is the first card the user sees. It contains an illustration of Freedom Station as a visual cue that the user is at the top level, plus a title bar across the top, a home button, and credits across the bottom. This card gives the user four options--INTRODUCTION, RUN MODEL, VIEW MODEL and FINISH. Clicking on INTRODUCTION will take the user to a section containing tutorial text. The RUN MODEL button starts the demonstration. The VIEW MODEL button allows the user to view the fault model drawings and listings. Clicking on FINISH will exit HyperCard.

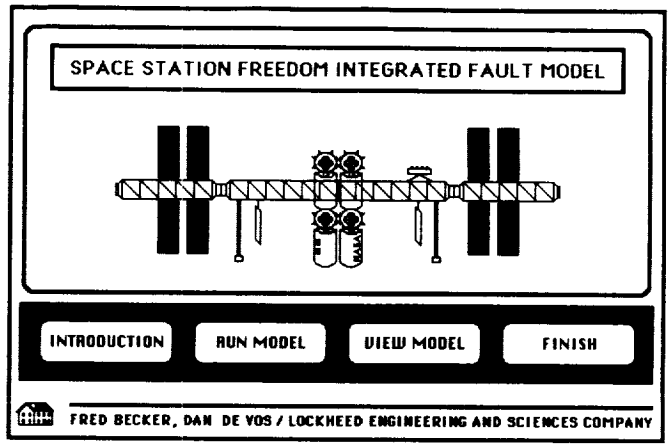


Figure 1
OPENING CARD

Figure 2 shows the introduction card. Instructions printed across the top tell the user what to do. This card guides the user to four areas of tutorial information:

PART 1
HOW TO USE THIS DEMONSTRATION

PART 2
ABOUT DIRECTED GRAPH FAULT MODELING

PART 3
WHAT HAS BEEN MODELED?

PART 4
SYSTEM FAULT MODELING IN THE SPACE STATION FREEDOM PROGRAM

Part 1 helps the user learn to use the actual demonstration. Part 2 explains in general how directed graphs are used to model failure propagation. Part 3 details how the space station systems were modeled, and what source materials were used. Part 4 then gives a perspective on how fault modeling may be used in designing and

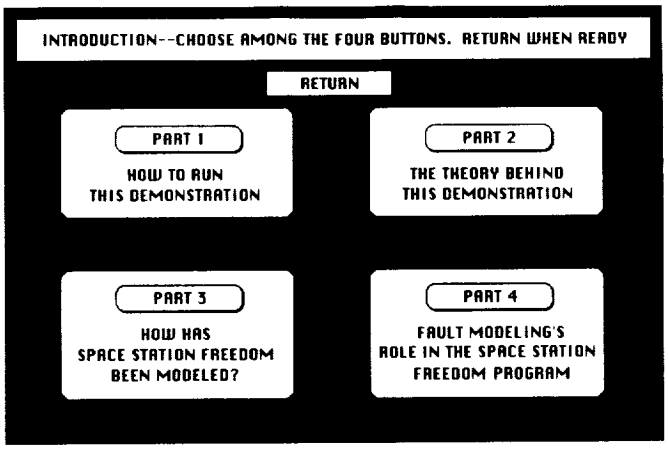


Figure 2
INTRODUCTION CARD

operating Space Station Freedom. These cards all contain text in a scrolling field. The RETURN button on these four cards will return the user to the introduction card. The RETURN button on the introduction card will then allow the user to return to the opening card. The introduction portion of the demonstration is optional. The user can go directly to the demonstration if desired by clicking on RUN MODEL on the opening card.

Figure 3 shows the result of clicking on RUN MODEL from the opening card. It has two buttons to allow the user to chose between two types of demonstration. The HARDWARE button goes into a demonstration in which individual hardware items are chosen as the ultimate targets of other hardware failures. The FUNCTION button takes the user into a demonstration which shows how Freedom Station's functions can be affected by various hardware failures. It was felt that an entire card dedicated to this choice would emphasize to the user the distinction between the two types. The RETURN button on this card allows the user to return to the opening card if desired. This card will be referred to as the "hardware or functions?" card.

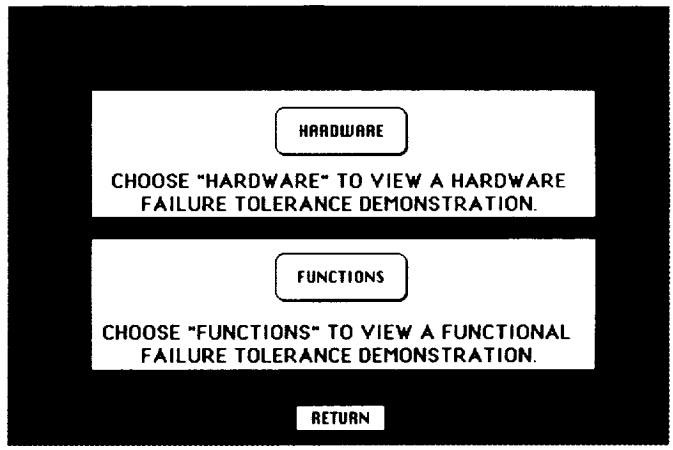


Figure 3
"HARDWARE OR FUNCTIONS?" CARD

Figure 4 shows the systems hardware card, identified by its title bar across the top of the screen. The user will see this card when the HARDWARE button is chosen from the "hardware or functions?" card. This card has a selection of Space Station Freedom systems from which the user will go into the demonstration. The user can view the block diagrams for these systems by clicking on the desired system. The SHOW DATA button allows viewing of singleton and doubleton data from the previous target chosen (an advanced feature). The VIEW DIGRAPHS button allows the user to view the entire digraph listing, taking the user to the same card as does the SHOW MODEL button on the opening card (a short cut from this card to the digraph). A message box across the bottom informs the user to click on one of the system boxes. MENU BAR will turn off the Macintosh menu bar, which is useful on

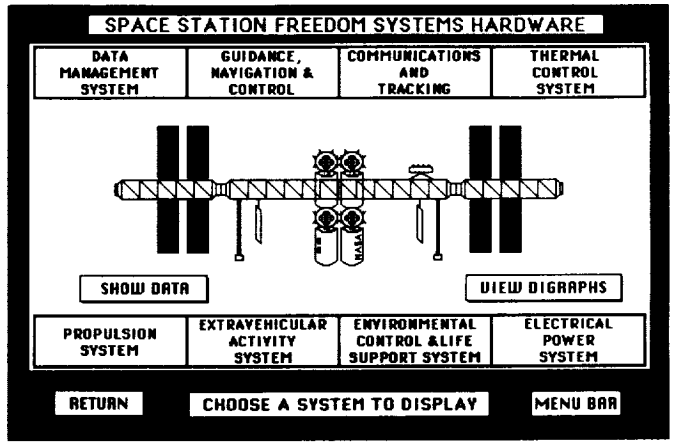


Figure 4
SYSTEMS HARDWARE CARD

Macintosh SE's or when making a presentation. RETURN will take the user back to the opening card.

Figure 5 shows the functions card. This card will allow the user to see how selected functions can be affected by various hardware failures. It is reached by clicking on the FUNCTIONS button from the "hardware or functions" card. When one of the six

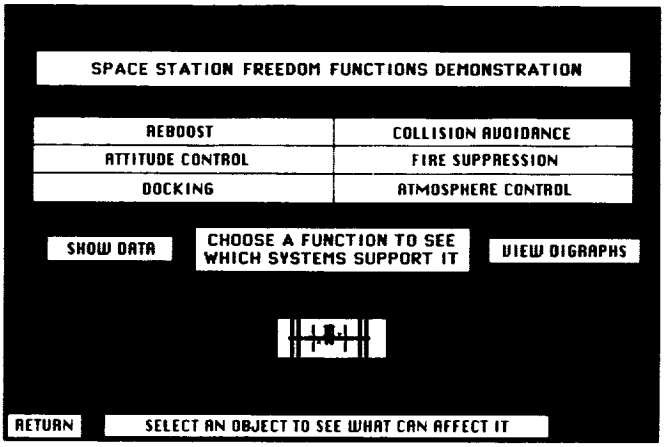


Figure 5
FUNCTIONS CARD

function buttons is clicked on, the associated systems which have hardware supporting that function will be highlighted across the bottom of the card. The user can then jump over to those system drawings to view the particular hardware combinations which can cause the loss of the chosen function. The remaining buttons have the same function as on the systems hardware card.

4.2 Starting the Demonstration

Figure 6 shows a typical system block diagram card, reached by clicking on one of the systems listed on the systems hardware card. This particular card shows the Guidance, Navigation and Control System block diagram. Each object in this card represents a node in the integrated Space Station Freedom fault model. By clicking on one of the objects, the user will be able to view all single and double hardware failures anywhere in the station which can propagate and eventually cause the failure of that object. These objects are the lowest level of detail contained in this demonstration, and when the user clicks on one of them the demonstration becomes active.

Other buttons on this card include an ACRONYMS button for displaying and removing a scrolling field

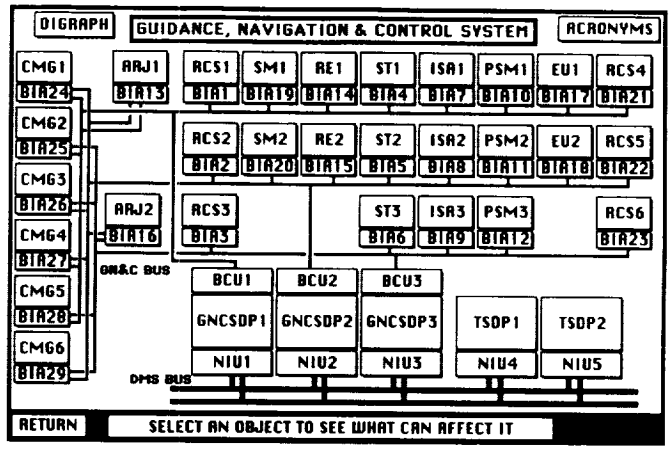


Figure 6
GN&C SYSTEM BLOCK DIAGRAM

containing the full names of the objects on the screen, a DIGRAPH button for viewing the associated digraph drawing or listing directly, and a RETURN button for returning to the systems hardware card.

When the user has clicked on a target object, in this case the Star Tracker #1 (ST1), buttons for the various systems will appear across the bottom of the screen (DMS, GN&C, C&T, etc.) with instructions on what to do. See **Figure 7**. Certain system buttons will be highlighted. These are the systems containing failures which can propagate to the target object, and are therefore a guide to viewing the

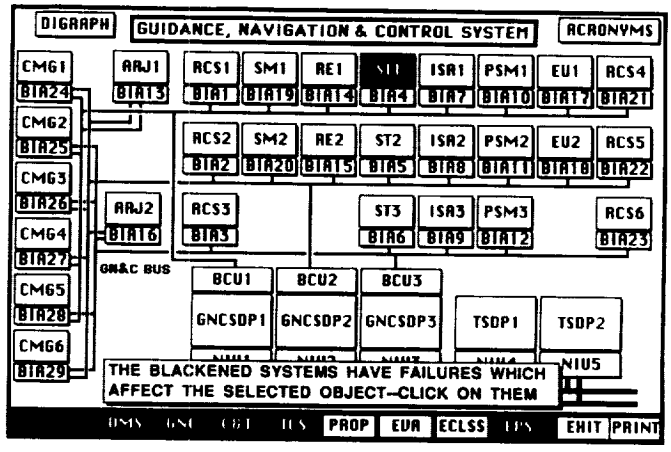


Figure 7
SYSTEMS WHICH REACH ST1

failure sources. When the user clicks on one of these highlighted systems buttons, the card for that system will be displayed.

As shown in **Figure 8**, the user has gone ahead and clicked on the GNC system button, which is the same card from which the target was already selected. Clicking on another system would have taken the user to that card. The GNC card remains, but two buttons, SINGLETONS and DOUBLETONS will appear at the bottom of the card. Clicking on one of these buttons will highlight one of the singletons or doubletons on (or partially on) this card which can reach the original target, ST1.

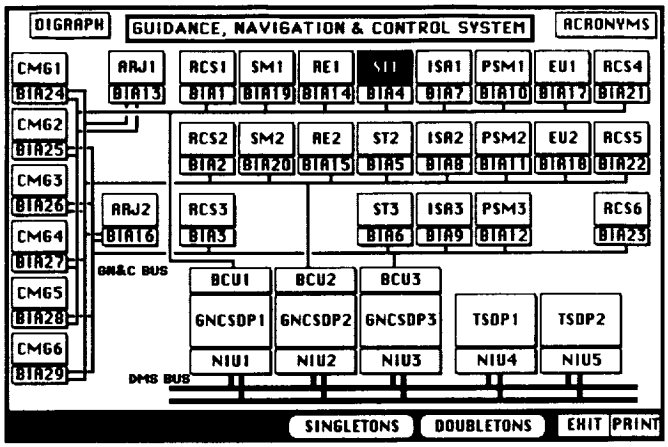


Figure 8
GN&C CARD, READY TO DISPLAY
SINGLETONS AND DOUBLETONS TO ST1

In **Figure 9**, the user has clicked on SINGLETONS, and the first singleton, Bus Interface Adapter #4 (BIA4), has been highlighted. A SHOW NEXT SINGLETON button will appear at this time. This button allows the user to cycle through all the single point failures. The components which can fail the original target will be independently highlighted. The original target remains highlighted as a reference.

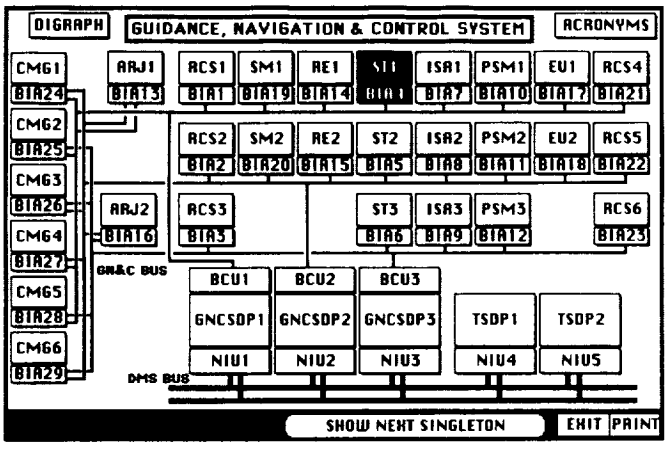


Figure 9
BIA4 IS A SINGLETON TO ST1

Doubletons are viewed in a similar way. **Figure 10** shows how two components, BIA13 and BIA16, are highlighted as a double-point failure which causes loss of the target, ST1.

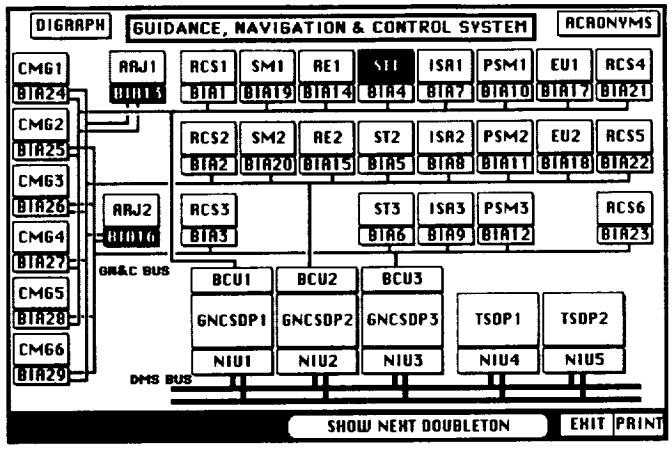


Figure 10
BIA13 AND BIA16 COMPRISE A
DOUBLETON TO ST1

However, when a double-point failure involves components on separate cards, another button, GO TO OTHER DOUBLET, will appear to allow the user to view the other half of the doubleton (a doubleton is made up of two "doublets"). As shown in **Figure 11**, the user has cycled through viewing doubletons until BIA16 is highlighted. BIA16 plus some other component off-screen are a double-point failure which can reach the target, ST1.

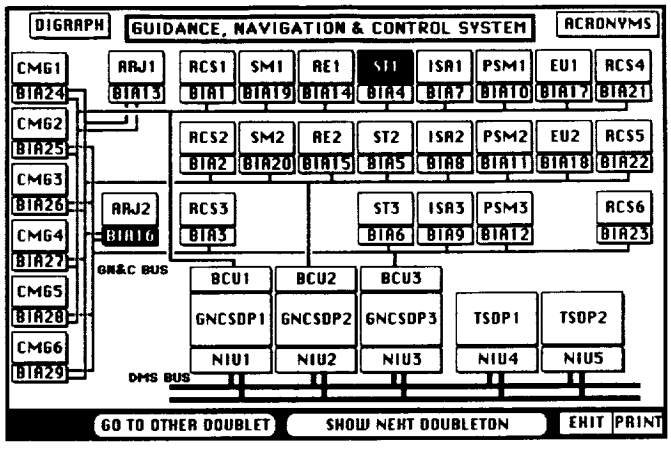


Figure 11
DISPLAY FOR AN OFF-SCREEN DOUBLET

By clicking on GO TO OTHER DOUBLET, the user will see the Electrical Power System block diagram with a single component highlighted, as shown in **Figure 12**. In this example, the alpha joint #1 (AJ1) is the second component of the pair. Thus BIA16 plus AJ1 failing together, will cause loss of ST1. A little inspection will show that BIA16 causes loss of Alpha Rotary Joint Driver #2 (ARJ2) which in turn causes loss of Alpha Joint #2 (AJ2). So the failure propagation path here involves loss of both

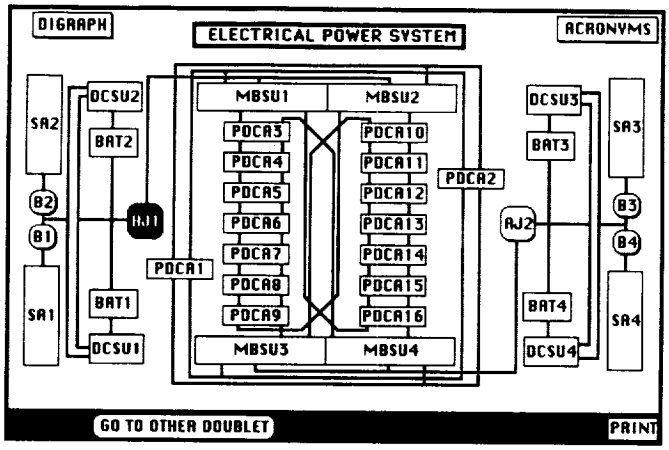


Figure 12
AJ1 IS THE OTHER DOUBLET

alpha joints and hence eventual total loss of power--which will naturally fail ST2. More complicated failure paths for other targets and sources might require viewing the entire fault model, as explained later.

There are other singletons and doubletons which reach ST1, not all shown here. These are viewed in the same way. At any point, the user can exit the process and choose another target. This is accomplished by clicking on EXIT until everything has been progressively reset. When the user is ready, clicking on RETURN will return him or her to the opening card.

4.3 Viewing the Fault Model

While running the demonstration, the user might be surprised that a particular singleton or doubleton can reach a given target. Failure tolerance is not always intuitively obvious. This demonstration allows the user to investigate a particular failure scenario further by viewing the fault model directly and rapidly tracing a failure path back from the target to the singleton or doubleton. Eventually, an understanding of the particular fault model contained in this demonstration can be attained. The speed with which the path can be traced demonstrates some of the advantages of computer graphical fault modeling.

The digraph topview card used for this purpose is shown in **Figure 13**. This card contains a guide to viewing the Space Station Freedom fault model drawings and source listings. It is reached by clicking on the VIEW MODEL button on the opening card, or the VIEW DIGRAPHS button on either the systems hardware or functions cards.

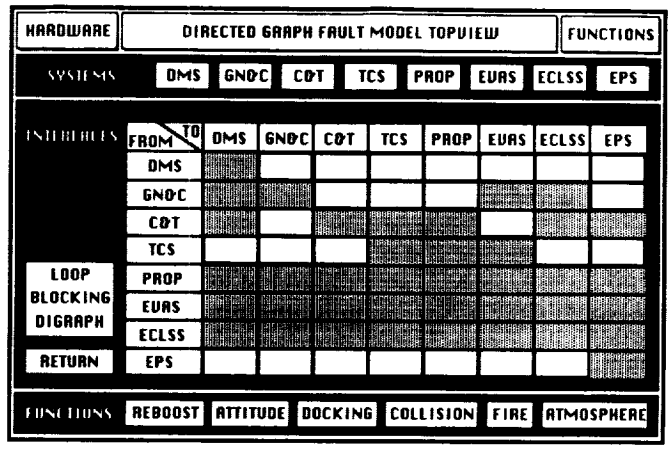


Figure 13
FAULT MODEL TOPVIEW CARD

The card contains a number of buttons which identify portions of the digraph. By clicking on any of these buttons, the user can view the associated portion of the digraph. This is useful for understanding how the fault model really works. The user can look through the digraphs to find the target chosen on a previous run, and trace back the failure propagation along the directed graph.

The systems buttons are across the top of the card. Interfaces between systems are identified in a matrix in the center of the card. Critical functions buttons are found across the bottom of the card. The loop blockage digraph has a dedicated button. The HARDWARE and FUNCTIONS buttons at the top will return the user back to the original cards. RETURN will take the user back to the opening card.

As an example, suppose the user wanted to trace the failure path from the doubleton pair BIA16, AJ1 to the target, ST1 as encountered in one of the examples above (**Figures 11 and 12**).

Figure 14 shows one of the digraph portion illustrations from which to start, in this case part of

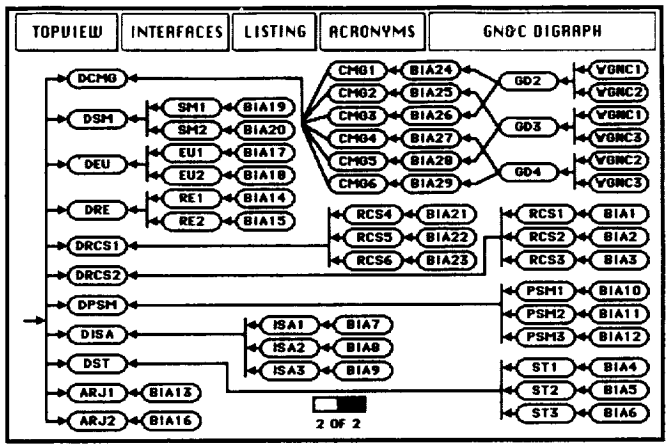


Figure 14
GN&C DIRECTED GRAPH

the GN&C system. A short-cut from the GN&C block diagram would have been to use the DIGRAPH button from that card (not available while the demonstration is active).

There are several active buttons across the top of **Figure 14**. The TOPVIEW button allows the user to return to the TOPVIEW card. ACRONYMS lists the acronyms found on this card. INTERFACES goes to the interface digraph for the GN&C System, and LISTING links to the digraph text listing.

Note from this figure that BIA16 fails ARJ2. On this drawing, ARJ2 fails nothing else. To see if ARJ2 fails anything in any other systems, the user will click on INTERFACES.

Figure 15 shows the result. In this case, all the GN&C interfaces fit on one card.

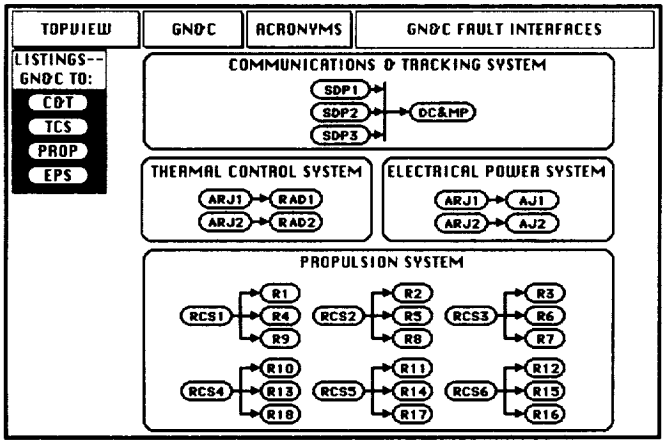


Figure 15
GN&C INTERFACES DIRECTED GRAPH

The interface digraphs such as this allow the user to gain a visual understanding of failure propagation paths between systems. A system button (GN&C here) allows the user to jump back to the source system digraph, if needed. To go to one of the interfacing systems, the user just clicks on the desired component belonging to that system.

ARJ2 can be found on the GN&C to EPS interface digraph. Here it is seen that that ARJ2 fails AJ2. It is also found in the GN&C to Thermal Control System (TCS) digraph. In this case, the user first tries the EPS digraph, and clicks on the AJ2 component to go there.

Figure 16 shows the EPS digraph. It can be seen that AJ1--one of the doubletons--is on this card as well as AJ2. By inspection, it can be seen that AJ1

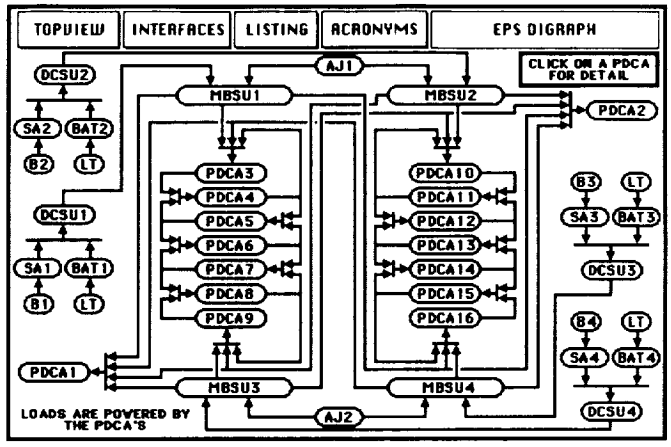


Figure 16
EPS DIRECTED GRAPH

and AJ2 failing will cause failure of Main Bus Switching Units (MBSU's) 1 through 4. Loss of all these will cause loss of power flow to all the Power Distribution and Control Assemblies (PDCA's). By clicking on EPS TO GN&C from the topview card, the card shown in **Figure 17** reveals that several BIA's are failed by losses of various PDCA's.

Going back to the GN&C digraph, **Figure 14**, it is seen that the target, ST1, is failed by loss of BIA4. Then, by going back to **Figure 17**, it is seen that BIA4 is indeed powered by PDCA1. Thus, the path from AJ1, BIA16 to ST1 has been found.

A check of the TCS and TCS interfaces digraphs does not find any such direct paths (**Figures 18 and 19**).

In a more developed tool, it would be useful to provide for automated tracing of these failure

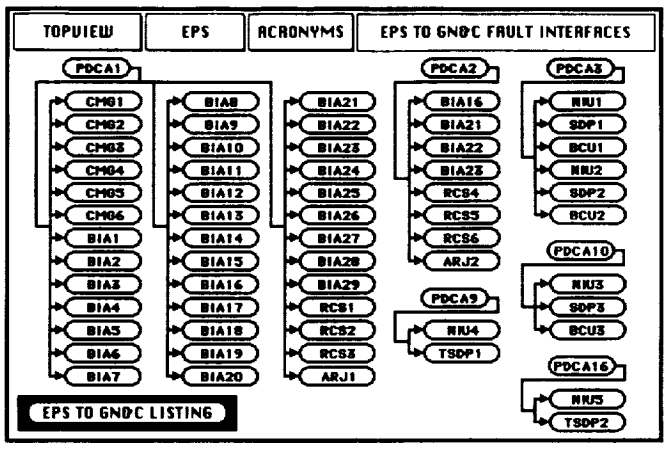


Figure 17
EPS TO GN&C INTERFACES
DIRECTED GRAPH

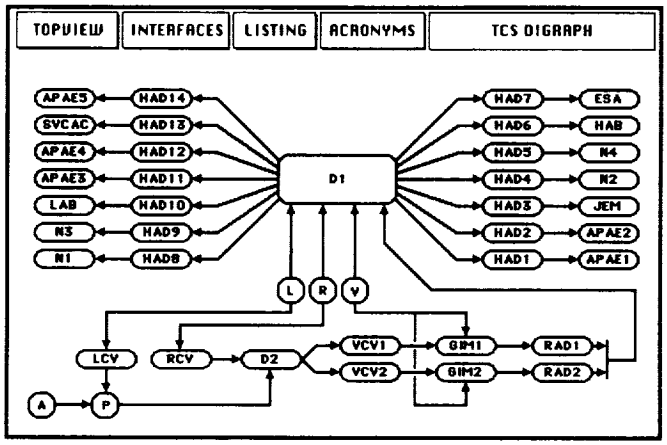


Figure 18
TCS DIRECTED GRAPH

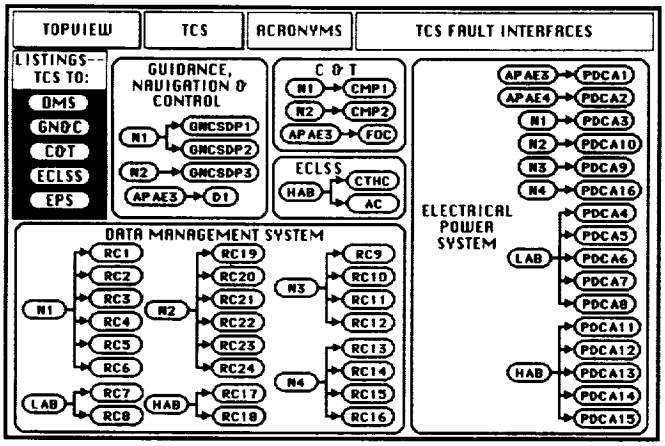


Figure 19
TCS INTERFACES DIRECTED GRAPH

propagation paths, known commonly as "cutsets."

A typical function digraph is shown in **Figure 20**. The function digraphs contain small "pushbuttons" which allow the user to go to the associated system card containing that object. The function digraphs are drawn in a fault tree structure. The linkage to digraphs shows one way that hardware failure modeling can be used in conjunction with functional modeling.

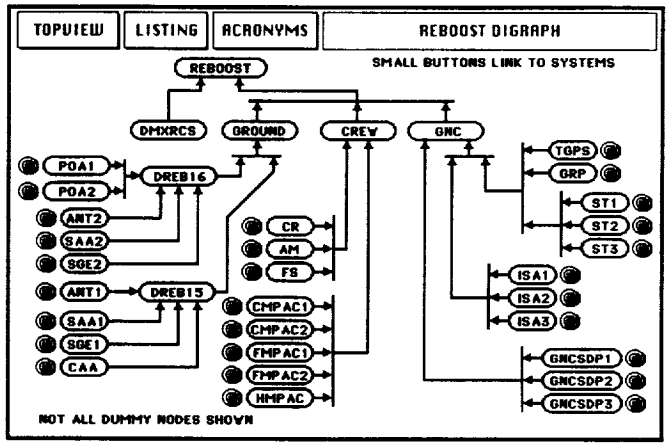


Figure 20
REBOOST FUNCTION DIRECTED GRAPH

Figure 21 shows one of the digraph listings cards. These cards contain the actual text listings of the Space Station Freedom fault model directed graph. The button highlighting seen in the demonstrations is driven by the failure propagation modeled in the total set of such listings. The syntax for these listings is of the form "A,B,C", meaning that A can reach B with C. If C is a "1", then the node A is a singleton

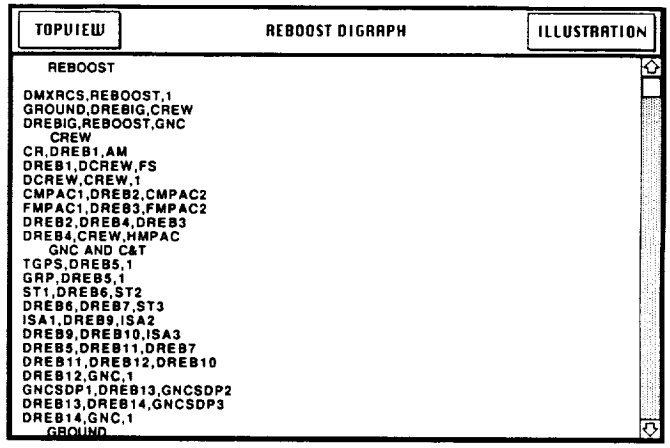


Figure 21
DIRECTED GRAPH LISTING EXAMPLE

to node B. The field can be scrolled up and down, using the scroll bar on the right, to view various portions of the listing. The ILLUSTRATION button will display an illustration of the digraph associated with this card, and the TOPVIEW button will return the user to the digraph topview card.

5.0 HYPERCARD DEMONSTRATION WORKINGS

This project had three separate phases: creating the HyperCard block diagrams and special button scripts to allow the demonstration to run, running the digraphs using Digraph Matrix Analysis (DMA) codes, and formatting the DMA results so they could be used by the HyperCard graphics demonstration. Reference 1 details the development of the demonstration. Reference 2 provides more information on DMA.

There are actually three HyperCard stacks used in the demonstration: one for the Introduction, one for Running the Demonstration, and one for Viewing the Model. This conserves the active memory in use by the Macintosh at any one time. The total memory occupied by all three stacks is about 670 kilobytes.

There are 98 cards in total: six for navigating, four for tutorials, 12 for system block diagrams, one for the functions demonstration, three for utility purposes, 14 for system digraphs, 11 for interface digraphs, seven for function digraphs, and 40 for digraph text listings.

The demonstration runs by accessing data stored in separate files on the Macintosh. The 1056 singleton and doubleton files are stored in a folder called "STATION_DATA." This folder occupies about 2.2 MB. In the demonstration, each time the user clicks on a target object, two of these files are copied into the HyperCard stack for use in highlighting the correct objects to show single and double-point failures.

6.0 FAILURE MODELING IN THE SPACE STATION FREEDOM PROGRAM

The distributed nature of the Space Station Freedom Program makes the system integration problem different from that on any previous NASA program. There is no longer a simple division of work by large hardware elements. Rather, work is divided into both hardware elements and distributed functional systems. The interfaces between the functional systems are more varied and complex than those between distinct elements.

In view of this, there is a need for new approaches to satisfying the program failure tolerance requirements and capturing the knowledge of how redundancy management evolves in various systems--both during design and operations. Failure modeling is one way to accomplish these goals.

7.0 CONCLUSIONS

The HyperCard tool has proven valuable as a means for prototyping various graphics concepts and for interfacing displays to fault modeling tools. The demonstration shows the basic methodology which would be performed in doing more detailed and accurate fault modeling. It contains a tutorial section, a block diagram section from which failure tolerance can be interactively displayed, and an illustration section by which the large directed graph fault model can be viewed. The Space Station Freedom systems, as modeled here, are indeed highly interdependent.

ACKNOWLEDGEMENTS

The fault modeling and demonstration was a joint effort by F. J. Becker and D. M. De Vos, a Lockheed summer-hire. D. M. De Vos's contributions were particularly essential for developing the software which interfaces the DMA files to the HyperCard stack and for co-creating the HyperCard script which drives the demonstration.

Thanks go to our NASA task monitor, J. T. Edge, and Lockheed task manager, W. H. Geissler, for their leadership in guiding this project.

REFERENCES

1. Becker, Fred, "Space Station Freedom Integrated Fault Model Report," LESC-26314, Lockheed Engineering and Sciences Company, Houston, Texas, December, 1988.

2. Sacks, Ivan, "Digraph Matrix Analysis," Analytic Information Processing, Inc., November, 1988.

GRAPHICAL PROGRAMMING AND THE USE OF SIMULATION FOR SPACE-BASED MANIPULATORS

Debra S. McGrath and James C. Reynolds
The MITRE Corporation
1120 NASA Road 1
Houston, TX 77058

ABSTRACT

Robotic manipulators are difficult to program even without the special requirements of a zero-gravity environment. While attention should be paid to investigating the usefulness of industrial application programming methods to space manipulators, new methods with potential application to both environments need to be invented. These methods should allow various levels of autonomy and human-in-the-loop interaction and simple, rapid switching among them. For all methods simulation must be integrated to provide reliability and safety. Graphical programming of manipulutors is a candidate for an effective robot programming method despite current limitations in input devices and displays. A research project in task-level robot programming has built an innovative interface to a state-of-the-art commercial simulation and robot programming platform. The prototype demonstrates simple augmented methods for graphical programming and simulation which may be of particular interest to those concerned with Space Station applications; its development has also raised important issues for the development of more sophisticated robot programming tools. This paper discusses both aspects of the project.

1. INTRODUCTION

1.1 Inherent Difficulty of Robot Programming

Programming robotic manipulators for safe and reliable execution in the face of inevitably uncertain conditions is difficult for a number of reasons [7]. Many robots today still provide only rudimentary control instructions that are roughly equivalent to machine language for computers. The only space-based robotic arm, the Remote Manipulator System (RMS), can only be programmed, as opposed to teleoperated, by setting the joint values. Obviously, the impossibility of envisioning the movement of the arm by mentally solving the forward kinematic problem makes this method unsatisfactory.

More advanced industrial manipulators provide sophisticated control languages like VAL II or Karel with point-level instructions and modern branching constructs for structured programming. Most often, these languages are not used because it is difficult for programmers to reliably envision spatial operations and exceptions even at the point level. A "bug" introduced off-line can have much more disastrous effects in a robot program than, for example, in a word processor.

Consequently, most robots are programmed with a teach pendant. This is an easy method and appropriate for simple, repetitive tasks. Its disadvantages are many: on the factory floor it requires down-time; complicated tasks like assembly are almost impossible to program; there is little if any branching capability, especially on complex feedback from machine vision systems or force/torque sensors. With regard to its use in space-based robotics, this last disadvantage is decisive. In addition, the experience of the RMS shows that space-based manipulators are more likely needed for complicated, "one-of-a-kind" tasks than for simple, repetitive operations, and on the Space Station Freedom, NASA plans to use the Flight Telerobotic Servicer (FTS) for assembly.

1.2 Task-level Robot Programming

1.2.1 Requirements for Task-level Robot Programming

The difficulties and limitations of current methods of programming robotic manipulators both on Earth and in space have motivated considerable research in developing new methods. One line of research beginning in 1976 [13] and continuing with impressive momentum during recent years has the goal of developing systems that allow manipulators to be programmed at the "task" level. An example of a space-oriented task command is PLACE ORU-1 IN PAYLOAD-BIN-2. A task-level robot programming system would translate this command into a sequence of motions and sensing operations that would reliably and safely accomplish the task. A necessary component of such a system is a model of the workspace and manipulator, including geometry with tolerances, kinematics, and dynamic attributes like mass and required forces and torques for assembly. The key challenge to building such a system is that any model is inaccurate to some extent, and therefore manipulator motion occurs in the context of uncertainty [8].

1.2.2 Programming Methods Must Allow Various Levels of Autonomy

One advantage of a task-level programming system for space-based robotics is that it provides the building blocks for various levels of autonomy. This is essential for the astronaut who uses the system to control a manipulator to gain confidence in its reliability and safety. At a lower level of autonomy the sequence of point-level motions and sensing operations generated by the task-level system can be examined, simulated, or executed one at a time under close monitoring; at a higher level of autonomy a number of task-level commands could be combined into a more complicated script. Using advanced ideas of augmented control [2], teleoperation could take over at any time.

1.3 The Need for Simulation

As mentioned above, it is absolutely imperative that a robot programming system provides simulation capabilities. The use of simulation is as vital to programming manipulators as the use of a symbolic debugger is vital to programming large data manipulation applications for a conventional computer system. Despite a long-time and often world-class emphasis on simulation for training, NASA has not baselined simulation capabilities for the FTS. Real-time, three-dimensional (3D) simulation of manipulators in space is necessary for astronauts to gain confidence in any form of autonomous control.

1.4 Graphical Programming

All of the research at Stanford, MIT, and Carnegie Mellon in task-level robot programming has assumed a keyboard-based textual interface with the manipulator. Until recently this was justified by the relatively low resolution and slow speed of graphics workstations, and the resulting difficulty in specifying robot motions and effects. The last few years have demonstrated that high resolution, three-dimensional graphics displayed in real-time can be a practical component of a desktop-based robot programming system, while even more revolutionary simulation capabilities like stereoscopic displays and three-dimensional input devices are realizable in the near future.

For space the need for non-keyboard devices to input manipulator commands that can be simulated in real-time 3D is even more critical. Keyboard input is simply too awkward in zero gravity, and robot programming requires too much knowledge of the workspace not to be facilitated by contemporary interface components like menus and mice.

1.5 The Task-level Robot Programming Prototype

For the last year and a half the MITRE Corporation, with initial funding from NASA, has been conducting research in approaches to building a task-level robot programming system (TLRPS). A prototype has been built that includes an innovative interface to a state-of-the-art commercial simulation and robot programming software platform (Deneb Robotics' IGRIP) running on an advanced graphics workstation (Silicon Graphics' IRIS 4D/70GT). The interface is used to control a Microbot Alpha manipulator performing pick-and-place and bin-filling operations in an appropriately simple plastic blocks world. Feedback is provided by a 2D vision system capable of recognizing circles, triangles, and squares.

The prototype demonstrates simple augmented methods for graphical task specification and the use of simulation for testing commands that may be of interest to those concerned with Space Station Freedom robotic applications. In the course of building the prototype, limitations in today's 3D display and non-textual input devices became apparent, suggesting new requirements for tomorrow's applications.

2. SIMULATION

2.1 Need for Simulation on board the Space Station Freedom

Simulation of planned RMS teleoperation is conducted rigorously prior to Shuttle missions using world-class high fidelity simulation facilities. Experience has shown that many uses of the RMS were not predicted and yet were critical to mission success. On the Space Station Freedom, over much longer duty cycles, this may be even more true for its manipulators. If this is the case and control of space-based manipulators advances from continual man-in-the-loop teleoperation to some degree of autonomy, then simulation capabilities on board will be a necessity.

2.2 Uses of Simulation

2.2.1 Complete Simulation before Execution

This type of simulation is used today for the off-line programming of industrial manipulators and could be used for ground-based programming of complicated planned manipulator operations on the Space Station Freedom. The programmer describes the manipulator actions desired, either at a point-level or a task-level, and then views the complete simulation that was specified. If there are problems like joint limits exceeded or collisions detected, the program is altered and simulated again. From this loop of program, simulate, re-program, will develop a correct program that can with confidence be downloaded and executed by the actual manipulator. This use of simulation has been implemented in MITRE's task-level robot programming system prototype, building on top of IGRIP's simulation and robot-specific translation capabilities.

2.2.2 Simulation Simultaneous with Execution

The task-level robot programming system prototype uses the simulation capability of IGRIP simultaneously with execution by the Microbot manipulator to implement collision detection, reachability checks, and simple grasp planning. The simulation is run ahead of the execution by the robot so that any actions that result in undesirable effects are not physically carried out. This type of operation would allow continual supervision of the manipulator without teleoperation, and it is believed that the strain on the astronaut in a space-based implementation would be significantly reduced. It is important that the world model and the graphical display be updated to reflect changes in the workspace after manipulator actions. Ideally, this would be in real-time; in MITRE's task-level robot programming system prototype, updates can occur only after the task or collection of tasks is completed, and the Microbot returns to home position.

3. GRAPHICAL PROGRAMMING

3.1 Commercial programs

The ability to graphically program a robot is limited by the current state-of-the-art in input devices and displays. Although true 3D displays and pointing devices are in development, they are not widely available. We are therefore constrained to showing a projection of a 3D scene on a 2D graphics screen. The latest generation of graphics workstations lets us show perspective views of 3D scenes and even translate and rotate them in real time, but they are still only 2D projections.

Trying to point to an arbitrary location in 3 dimensions on a 2D screen poses problems. It is difficult to determine the dimension corresponding to depth into the scene, or distance from the user (see Figure 1). Several methods are used to help remedy this problem. The S-GEOMETRY 3D graphics software from Symbolics attempts to alleviate the problem by allowing optional cursors for selecting points [12]. These cursors have extra lines drawn to help the user determine the current 3D location. One cursor, the arm cursor (see Figure 2), has lines drawn from the cursor point to the intersection of

each of the x, y, and z planes. A second cursor, the box cursor (see Figure 3), has lines drawn from the cursor point to the intersection of the three planes (as in the arm cursor) and lines drawn to the coordinate axes. Cursor motion in combination with mouse buttons determine the 3D motion of the point. Even with these methods, accurately specifying a 3D location requires concentration and can be difficult if the screen is cluttered with objects.

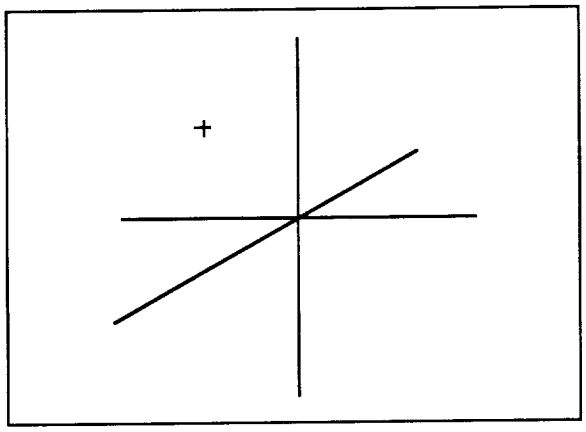


Figure 1 - Crosshair Cursor

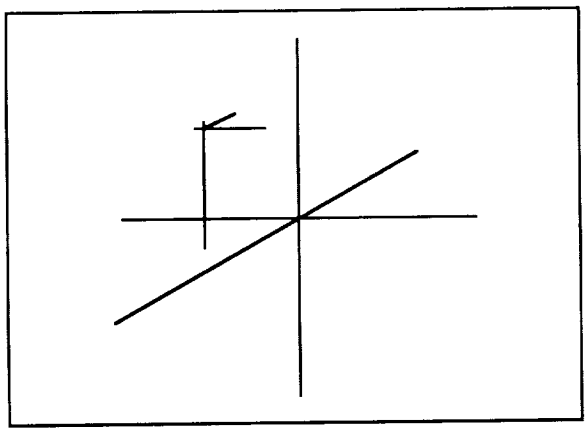


Figure 2 - Arm Cursor

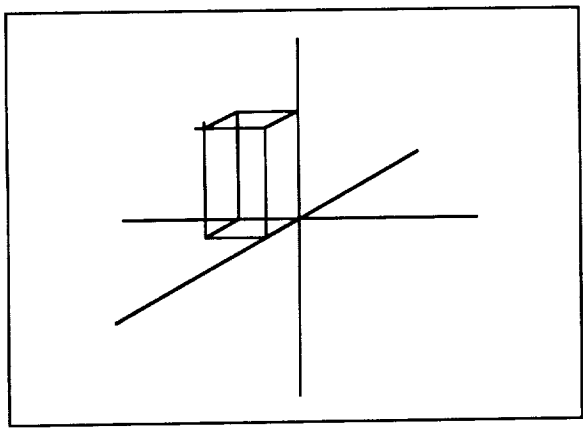


Figure 3 - Box Cursor

Commercially available robot programming and simulation packages typically allow the user to program the robot by specifying a desired position and orientation for the manipulator's toolpoint. The toolpoint is most often the tip of the robot gripper and its position and orientation in space is called a "pose". A pose can be specified by giving the x, y, and z coordinates and the yaw, pitch, and roll angles or by selecting predefined locations in the robot's workspace. These predefined, named location are called tagpoints or reference frames by the various robot programming packages [3, 4, 9]. To avoid the 3D pointing problems mentioned above, the tagpoints are defined in advance by specifying coordinates or by aligning a new coordinate system with components of objects in the workspace. Thus a tagpoint can be coincident with the local origin of an object or aligned with a vertex, for example. The robot programming package can perform the inverse kinematics to translate a pose into a set of joint angles, or a configuration, for the specific robot. Motion between configurations can be constrained to be a straight line in cartesian space or can be joint-interpolated, meaning the joint angles will change linearly between configurations but the resulting path of the toolpoint will be curved.

Robot programming and simulation packages generally provide a Pascal-like language in which robot programs can be written and simulated. Figure 4 is an example of a program written in GSL, the programming language in IGRIP from Deneb Robotics. These languages are robot-independent and are therefore ideal for prototyping and simulation, where different manipulators may be tested. If a translator is available, the programs in these language can be translated into the language understood by the robot controller hardware or software. For example, the program in Figure 4 can be converted to a form that a Microbot Alpha I robot can use, as in Figure 5. It is obvious that programs written in the high-level languages are much easier to write, test, and debug than are robot-specific programs.

```
program moveit
----------------------------------------

VAR
 round_cap_1_1, pos_1:  POSITION
----------------------------------------
begin
  UNITS = ENGLISH
  $motype = JOINT
  move link  6  by  50 relative nosimul
  move near round_cap_1_1 by 3
  move to round_cap_1_1
  move link  6  by  -30 relative nosimul
  grab round_cap_1_1 at link 6
  move away  3
  move near pos_1 by 3
  move to pos_1
  release round_cap_1_1
  move link  6  by  30 relative nosimul
  move away  3
  move home
  move link  1  by  60 relative nosimul

----------------------------------------
end moveit
```

Figure 4. GSL Program

IGRIP also has a menu-driven graphical robot programming capability, which provides an interactive method of producing GSL programs [4]. Menus provide the ability to set up parameters and choose manipulator motion commands. When a motion "move to tagpoint" type command is chosen, the user has the option of pointing to tagpoints on the screen or choosing the named point from a list, as well as keying in the name. Motion commands are carried out by the simulated robot as they are specified. An entire robot program can be generated in this manner and then tested, translated, and downloaded to the actual manipulator for execution. However, because this programming is at the point level, the use of menus can become tedious compared to simply writing the program directly in GSL.

```
@STEP 199,0,0,0,0,0,1136
@STEP 199,-1345,815,503,1025,623,503
@STEP 199,0,390,176,0,0,176
@STEP 199,0,0,0,0,0,-681
@STEP 199,0,-390,-176,0,0,-176
@STEP 199,-690,693,-408,-431,431,-408
@STEP 199,0,415,50,0,0,50
@STEP 199,0,0,0,0,0,681
@STEP 199,0,-415,-50,0,0,-50
@STEP 199,690,-1515,-98,-595,-1057,-1234
@STEP 199,1351,0,0,0,0,0
```

Figure 5. Microbot Alpha Program

3.2 TLRPS prototype and extensions

3.2.1 Specifying parts

An important step up from manually creating a robot program by specifying individual points on the manipulator's path is the ability to specify the objects to be manipulated and their goal positions, with an appropriate path automatically generated. This is the capability that the TLRPS adds to the commercial programming package. For a simple pick and place task, the object to be moved and its goal location must be specified. In the TLRPS prototype an object to be manipulated is chosen from a menu of all known objects in the workspace. The object is then highlighted on the graphics screen to allow the user to confirm the choice. A goal location can then be chosen from a menu of predefined locations and the tagpoint is highlighted for confirmation. Optionally, the user can specify an x-y location, in inches from the origin, for the goal, with z and rotations defaulted to reasonable values for a pick-and-place operation. A tagpoint for the specified goal is created and highlighted for confirmation.

Once an object and a goal have been chosen a sequence of motions is automatically generated to move the object from its original location to the goal. Checks are made along the way to ensure that all locations are reachable by the robot and neither the robot nor the object being moved will collide with other objects in the workspace. The user needs only to specify the object and the goal; the TLRPS generates the intermediate steps needed to safely move the object.

With manual robot programming or the menu-driven programming provided by IGRIP, a complete robot program must be written, tested, translated, and downloaded before the actual manipulator can be used. The TLRPS prototype allow more interactive control of the manipulator. The user may choose to test the task to be complete by simulation only. However, the TLRPS provides the ability to simultaneously simulate the task and execute it with the actual robot. This was discussed in greater detail in Section 2.

3.2.2 Dragging objects

Although specifying operations by naming objects or choosing them from a menu is certainly an advance from point-by-point programming, a more intuitive interface would be to allow the user to point to an object to be moved. Pointing to an object in 3D is not a problem. An object can be chosen by placing the cursor over any portion of the object facing the user. The function providing selection of objects with a mouse under program control is not available in the simulation and robot programming package currently being used for the TLRPS prototype, so this capability has not been implemented. Another desirable option, which is not implemented in the current prototype, is to allow the user to drag an object on the screen from its original location to a new location and have the TLRPS generate the equivalent manipulator motions to move the object without collisions. With this option there still is the problem of visualizing the 3D motion on a 2D screen.

3.3 Input Devices for 3D Manipulation and New Display Technology

Even with a more functional and open software platform to use in building a graphical interface to the control of a manipulator, the inherently two dimensional input and display technology of today's workstations would be severely limiting. There are, however, laboratory efforts and even a few advanced commercial products that demonstrate this will not be true in the near future. With respect to space-based robotics and Space Station Freedom in particular, it is of utmost importance that plans should be made now to provide hooks and scars so that these rapidly developing technologies can eventually be used.

Complex interaction in zero gravity with the computer control of dynamic physical devices such as a manipulator requires a large bandwidth of information that would be difficult if not impossible to communicate using a keyboard. Current NASA thinking foresees voice recognition technology as an alternative. The use of this technology will be an important advance, but for robotic control it will have to be implemented together with to the interactive display concepts (menus and highlighting) described above, or else too much workspace knowledge (names, dimensions, dynamic attributes) will be required of the astronaut operator.

The most natural control and programming of a manipulator, though, can only be expressed with three-dimensional input devices. If an object needs to be grasped, the operator should only have to move his hand appropriately and the effect should be displayed on the screen. This is, in fact, possible today. There are two commercial products, a glove-like device (DataGlove™ by VPL Research) and an optical gesture sensing device (by Sensor Frame Corporation) that could be used to build an interface to a manipulator. Ames Research Center is already using the first device in conjunction with demonstrations of their head-mounted display technology. An alternate device for manipulator programming that is within the realm of today's technology, although not commercially available, is a small manipulator replica that could provide true three-dimensional input for graphical display.

Equally important, especially to NASA, is the development of true three-dimensional displays. This is necessary for remote

teleoperation as well as supervised autonomy. It has been demonstrated that the remote control of robots in response to ordinary video feedback is extremely difficult. True 3D displays may be built using stereoscopic or holographic technology; there are many laboratory efforts currently in progress with the objective of developing these displays.

4. ISSUES

4.1 Uncertainty

Most robot programming and control systems assume a perfect world: error-free sensors, perfect object models, and robots that can be positioned precisely where desired. The real world, unfortunately, falls far short of perfection. The robot and environment can be engineered to minimize these errors and uncertainties, but they will never be perfect. Robot programs need to handle differences between models and actual objects and handle errors in sensor data and manipulator control. Practical methods for dealing with these uncertainties need to be developed. Brooks [1], Erdmann [6], Durrant-Whyte [5], and Volz, Xiao, and Desai [14], among others, have published work upon which a practical system might be based.

4.2 Path planning

The current version of the TLRPS prototype does not have any true path planning capabilities. To move an object from one position to another the system follows a fixed set of steps with a few parameters for initial object location and goal. The manipulator first opens the gripper and moves to a pose directly above the object to be moved, then moves straight down over the object and closes the gripper to grasp the object. Next the manipulator moves straight up, then to a pose directly above the object's goal location, then straight down. Then the robot opens the gripper and moves straight up again. This sequence of motions is sufficient to handle any simple pick-and-place operation in two dimensions, where the objects are all roughly the same height and can be grasped from the top.

Some minor modifications of this same plan would enable the system to handle a more extensive collection of objects and limited three dimensional placement. To cover a larger set of tasks, however, these heuristics should be replaced by algorithmic path planning. This would add the capability, given the initial and goal poses, to compute a path for the manipulator through the free space or unoccupied volume of the workspace avoiding collisions. Some of the commercial robot programming package vendors are developing this kind of path planning capability.

4.3 Human-in-the-loop

Any method of robot programming must allow various levels of autonomy and human-in-the-loop interaction and simple, rapid switching among them. While it is desirable to automate as much as possible to free the astronaut from tedious and time-consuming chores, we still need to allow the human to immediately and safely assume control if it becomes necessary. Once a crisis situation has ended, the system should be able to resume autonomous operations with as little input from the human as possible and preferable without having to re-plan an entire task from scratch. Sheridan [10], Stark, Kim, and Tendick [11], and Conway, Volz, and Walker [2], among others, have all made some suggestions and progress along these lines, but there is still much work to be done.

5. CONCLUSIONS AND FUTURE WORK

Graphical programming of manipulators is an effective approach despite current limitations in input devices and displays. The prototype TLRPS described in this paper includes an interface that takes advantage of these graphical techniques. In the future we would like to investigate both the use of voice recognition technology with the interactive display methods described in this paper and true three-dimensional input devices that promise a more natural way of programming manipulators. We feel this is an especially important technology area for NASA to develop and exploit.

REFERENCES

1. Brooks, Rodney, "Symbolic Error Analysis and Robot Programming," *International Journal of Robotics Research*, Vol. 1, No. 4, Winter 1982, pp. 29-68.

2. Conway, Lynn, et al., "Tele-Autonomous Systems: Methods and Architectures for Intermingling Autonomous and Telerobotic Technology," in *Proceedings 1987 IEEE International Conference on Robotics and Automation*, Raleigh, NC, 1987, pp. 1121-1130.

3. Deneb Robotics, Inc., *GSL Graphics Simulation Language Reference Manual*, Version 1.5, Deneb Robotics, Inc., Troy, MI, July 1988.

4. Deneb Robotics, Inc., *IGRIP Simulation System User Manual*, Version 1.5, Deneb Robotics, Inc., Troy, MI, July 1988.

5. Durrant-Whyte, Hugh, "Uncertain Geometry in Robotics," *IEEE Journal of Robotics and Automation*, Vol 4, No. 1, February, 1988, pp. 23-31.

6. Erdmann, Michael, *On Motion Planning with Uncertainty*, Master's Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1984.

7. Lozano-Perez, Tomas, "Robot Programming," *Proceedings of the IEEE*, Vol. 71, No. 7, July 1983.

8. Lozano-Perez, Tomas and Brooks, Rodney, "An Approach to Automatic Robot Programming," A.I. Memo No. 842, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, April 1985.

9. Silma, Inc., *CimStation User's Manual*, Revision 3.0, Silma, Inc., Los Altos, CA, 1986.

10. Sheridan, Thomas, "Human Supervisory Control of Robot Systems," in *Proceedings 1986 IEEE International Conference on Robotics and Automation*, San Francisco, CA, 1986, pp. 808-812.

11. Stark, Lawrence, et al., "Cooperative Control in Telerobotics," in *Proceedings 1988 IEEE International Conference on Robotics and Automation*, Philadelphia, PA, 1988, pp. 593-595.

12. Symbolics, Inc., *S-Geometry*, Graphics Division of Symbolics, Inc., Cambridge, MA, October 1986.

13. Taylor, Russell, "The Synthesis of Manipulator Control Programs from Task-Level Specification," AIM-382, Stanford Artificial Intelligence Laboratory, Palo Alto, CA, July 1976.

14. Volz, Richard, et al., "Contact Formations and Design Constraints: A New Basis for the Automatic Generation of Robot Programs," unpublished report, the University of Michigan and the Jet Propulsion Laboratory, 1988.

Using an Instrumented Manikin for Space Station Freedom Analysis

Linda Orr, NASA/Johnson Space Center
Richard Hill, Lockheed Engineering and Sciences Corporation

One of the most intriguing and complex areas of current computer graphics research is animating human figures to behave in a realistic manner. Believable, accurate human models are desirable for many everyday uses including industrial and architectural design, medical applications, and human factors evaluations. For zero-gravity (0-g) spacecraft design and mission planning scenarios, they are particularly valuable since 0-g conditions are difficult to simulate in a one-gravity Earth environment.

At NASA/JSC, an in-house human modeling package called PLAID is currently being used to produce animations for human factors evaluations of Space Station Freedom design issues. This paper will present an introductory background discussion of problems encountered in existing techniques for animating human models and how an instrumented manikin can help improve the realism of these models.

BACKGROUND

The difficulty in creating realistic models of people lies in the complexity of the human body. There are over 200 degrees of freedom in the body structure [6]. For purposes of human modeling for task planning and motion studies, the body can be graphically represented as a series of rigid body joints and linkages. For many movements the human model can be adequately represented by a subset of 30-40 degrees of freedom if it is not necessary to model each finger, toe, spinal disc, etc. for a study [4]. Even with such simplification of body structure, however, the approach to animating human movement in a realistic manner remains a complex issue. With 30-40 degrees of freedom in a model, redundant solutions for a desired motion are possible, some of which may be more comfortable and intuitive for a human to perform than others are. (Fig. 1.)

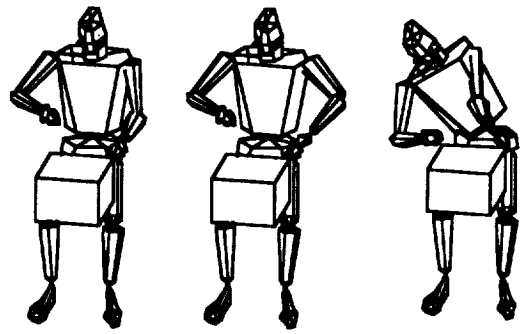


FIGURE 1

Redundant solutions for left hand reach with fixed feet locations.

There are basically three methods of modeling human motion for animated graphics display output: a guiding (keyframe) system, a program level or algorithm-based system, and a task level system [7]. Each method has its strengths and weaknesses.

Method 1: Guiding System

The guiding system is the traditional tool of computer animators dealing with human motion. Under this system, a user sets up a series of "keyframes" explicitly describing key actions of interest. For example, in Fig. 2a, a crewmember is modeled in an initial position configuration at time t_0. At time t_x, he/she has assumed a new position configuration of interest (Fig. 2b). The

program is then instructed to calculate a number of frames (n) showing the in-between frames from t_o to t_x, usually at a rate of 30 frames per second for video output. The program can use a simple linear interpolation to compute the new position of link L at each frame between t_o and t_x, based on the distance of travel of link L during that time interval. Linear interpolation tends to make the motion of the figure appear jerky and unnatural, however. The motion can be given a smoother appearance by using a spline intepolation instead of a linear assumption.

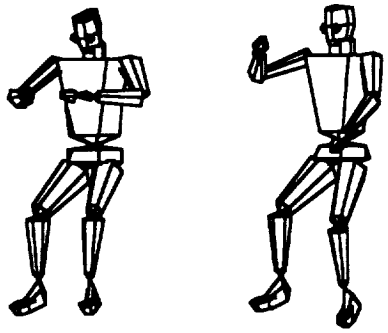


FIGURE 2

Crewmember at initial time to and later time tx.

Having the computer calculate in-between frames and check joint limits for solution feasibility can help the user relieve some of the tedium involved in animating human figures. This approach can be satisfactory when simple motion is all that is required. Where subtle changes of motion are desired, however, guiding systems require a lot of manual set-up time since they require more keyframes to define fully the action of interest. Much iteration is usually required to "tweak" the motion for it to look correct to a viewer. The motion generated is therefore highly dependent on the powers of observation of the animator.

The guiding method is particularly time-consuming to set up for three-dimensional animated studies since perspective views of the models and their work environments can be misleading. For graphically directing motion from one specific point to another, some guiding system users turn perspective off and look at 2-dimensional views for better precision in positioning body segments. This approach requires a view change to locate the third dimensional coordinate, and a decomposition of the movement into two or three orthogonal rotations, depending on the joint being manipulated. The view change and mental decomposition require additional set-up time.

Method 2: Algorithm-based System

In an algorithm-based system, physical laws are applied to human parameters. Typically, these systems assume rigid body mechanical links with joints modeled as spring and damper systems. The most commonly used algorithms are direct/inverse kinematics and direct/inverse dynamics algorithms borrowed from robotics applications.

The direct kinematics approach can be described as: given a set of joint angle information, determine the position and orientation of an end effector such as a hand or foot. Once position and orientation are determined, they can be differentiated to obtain joint velocities and accelerations. A simple example of a direct kinematics algorithm is the Denavit-Hartenberg matrix method [2]. The inverse kinematics problem is to determine appropriate joint angles given position and orientation of a desired end effector, and an example of such an algorithm is one described by Hollerback and Sahar [3].

The inverse kinematics approach is useful in reach evaluations for human factors studies. Given information on lengths of body segments, such algorithms can determine if Crewmember A at location (x,y,z) can reach button B without requiring the system user to predetermine (or guess) the desired joint angles. Since human beings have joint limits that restrict some motions, a good human modeling program will check joint limits for each frame of animation. Joint limit checking improves the animation result by eliminating solutions that are not humanly feasible to perform. The problem with joint limit checking is that it tells you nothing about the "naturalness" of the motions.

For dynamics analyses, the direct dynamics problem is described as determining the trajectories of the end effector(s) given appropriate initial conditions of force and

torque parameters. The inverse dynamics solution is to determine the initial forces and torques on joints required to produce known resultant forces and torques at time t_x. For human modeling, the direct/indirect dynamics algorithms borrow heavily from robotics applications. The most commonly used dynamics algorithms generally fall into one of two categories [4]: Lagrange's equations of motion based on kinetic and potential energies for nonconservative systems, and Newton-Euler formulations based on Newton's second law for determining the total force vector and Euler's equation for determining the total torque vector.

A major drawback to modeling human motion with algorithms is that human motion is not purely kinematics or purely dynamics: it is a combination of both [1]. Dynamics simulations should produce accurate motion animations if the dynamic model is sufficiently detailed. Often, however, technically feasible but unnatural looking solutions are a result of dynamic modeling since it is difficult to come up with enough equations of motion, constraints, etc. to eliminate redundant solutions.

An additional problem with dynamics modeling of humans is that spring and damper functions, not constants, are required to describe humans accurately with spring/damper analogies. Determining these functions requires collection, storage and reduction of empirical data and such data is generally not available. Data supplied from cadaver studies can be of questionable value when applied to simulations of living people. Existing data from live subjects is usually limited to studies of specific motions or tasks and may not be universally applicable to all motion situations.

For realistic-looking animations based on algorithms, information may also be needed on motion comfort levels and preferred motion. For example, to retrieve an object dropped on the floor, does someone simply bend straight-legged from the hips or does he/she bend the knees and stoop part way? The result is that even with a reasonably detailed algorithmic model, the system user is still required to tweak the model to make its motion appear more natural to a viewer.

Method 3: Task Level System

This method uses Artificial Intelligence (AI) techniques to describe the performance of a task at multiple levels. For animation purposes, this requires applying a set of facts to rules about task actions. For a given task, high level AI commands, rules and descriptions of actions are used to describe the behavior of the human model in terms of events and relationships. The high level AI system transforms the behavior model into low-level instructions such as algorithm references or key values for parametric keyframe creation; these low-level instructions are then used to create an animation of the task performance [4,7].

Successful task performance interpretation requires knowledge of the task environment and objects within it. This knowledge usually involves an object oriented database that not only contains information about an object's geometry and mass attributes (e.g., density, specularity, thermal properties) but also how it is put together, how it behaves and whether it inherits properties from related objects. An example high level task command might be, "Put the book on the table." A task performance system must contain rules defining how the verb "put" is translated into a human motion, object information such as book dimensions and table height, information regarding which person is to put the book on the table, and the current state of the animation environment (Must someone first pick up the book or is he/she already holding it? Is the person close enough to use a simple arm reach to place the book on the table or must he/she walk across a room to complete the task?). A more sophisticated system could also check an anthropometric database for information about the individual performing the task to determine arm length and strength factors that might affect the task outcome.

Sophisticated task performance systems will take many years to develop. Rules for task performance must be created and iterated to perfect; knowledge-based object descriptions must be input to a database so the system can access the information needed for task simulation. The lengthy development time for perfecting task performance behavior rules and the problems of organizing the large

database required for such a system are its chief drawbacks.

DISCUSSION OF MANIKIN DEVELOPMENT

Each of the three animation methods mentioned has strengths and weaknesses. At present, the authors see the PLAID human modeling effort eventually evolving into a program with heavy emphasis on task performance and algorithm-based methods with a guiding system user option. However, such a sophisticated modeling program will take years to develop. In the meantime, PLAID animators use a combination of guiding and kinematic algorithm methods to evaluate human factors issues for the Space Station Freedom Program.

Reach algorithms and joint limit checking are an integral part of PLAID's anthropometrics features but still require a large amount of user set-up time for some motion studies. The reach algorithm works quite efficiently when used to evaluate simple reaches to a predefined vertex on a person or object. A significant area of difficulty arose during some complex reach studies for the NASA Man-Systems Integration Standards (MSIS) document [5], however.

The MSIS is a 4-volume set of man-systems integration design considerations and requirements for development of manned spacecraft. Volume IV is specifically dedicated to Space Station Freedom human factors design issues. PLAID anthropometric features were used in the MSIS to help determine maximum reach envelopes of 5th percentile female and 95th percentile male astronaut candidates. For simple reaches, the existing PLAID features were straightforward to set up and manipulate. (Fig. 3). User set-up of imaginary 0-g maximum side-reach envelopes in free space with a foot restraint presented significant complications, however.

In Figure 4, the human model is initially positioned in a 0-g configuration with arms reaching above the head as far as possible and feet restrained in a foot restraint. The model is then positioned to sweep out an envelope in his/her lateral plane and identify points on that envelope. This motion is quite complex and eventually involves waist and/or hip twist, knee flexion, ankle flexion, etc. Since points on the envelope are in free space and are unknown to the system user, the reach algorithm (which requires a known destination vertex on a person or object) cannot be used. The user must therefore manipulate the various degrees of freedom on a joint by joint basis. Altering one joint affects the links downstream from it so the process is tediously iterative. Since the figure was being viewed on a computer screen, an inherently 2-dimensional display device, the user was required to make frequent view changes to ensure he/she understood exactly how the human model was currently positioned. For additional studies of complex motion a more user-friendly set-up procedure is obviously needed.

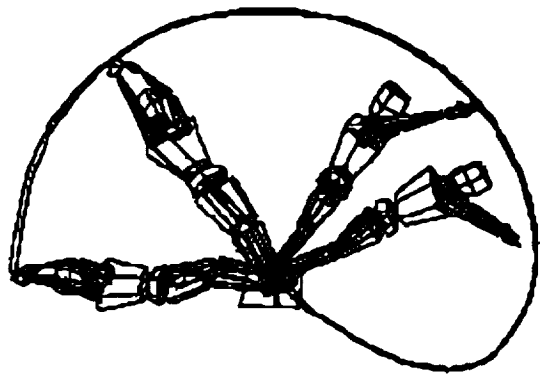


FIGURE 3

Simple forward/backward reach envelope with foot restraint for MSIS document.

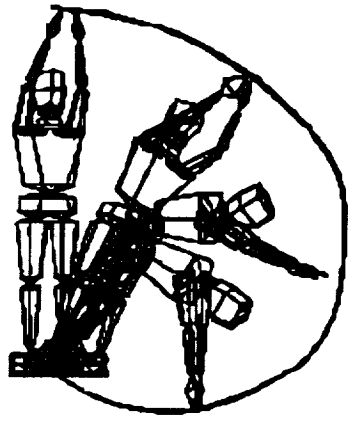


FIGURE 4

Complex side-reach envelope with foot restraint for the MSIS document.

A faster, more intuitive input device for positioning complex human movements in free space is an instrumented manikin. Such a device is currently being developed by the Graphics Analysis Facility at JSC for use with PLAID human modeling features. The manikin is a modified crash dummy with wirewound linear potentiometers instead of accelerometers for its instrumentation. It is approximately 48 inches tall and has 38 measurable degrees of freedom. To model actual human movement capabilities more closely, the standard crash dummy mechanical structure was modified to provide shoulder and thigh twist and was given a flexible neck.

The manikin is a truly 3-dimensional input device that can provide the computer with multiple position and orientation inputs simultaneously. It can be manipulated by a user in hands-on fashion to a desired configuration, where friction in the joints retains their positions once the user lets go. Alternatively, set screws can be used to lock the joints if preferred; for example, the user may want the legs configured in a 0-g orientation for an entire study. Mechanical joint limit stops equivalent to or slightly exceeding normal human limits are built into the structure.

The manikin is initially placed in a 1-g standing position and calibrated. When the user has manipulated the manikin to a new configuration, relative displacement voltages undergo an AC/DC conversion and signals are sent through an RS232 interface to the computer program. The input is converted to degrees for segment displacement information and then joint limits are checked by software to ensure position validity. Since PLAID body segment lengths are normalized, they can be read if desired from a user specified database of astronaut applicant data compiled by the Johnson Space Center's Anthropometrics and Biomechanics Laboratory. Thus, the manikin can be used to manipulate positions of different sized human models without mechanical or electrical reconfiguration.

CONCLUSION

By using the instrumented manikin, a user has a combination of algorithm and guiding methods available for setting up the desired study parameters. The user can utilize the power of algorithms as much as possible to simplify set-up procedures, yet have an effective way to tweak the human model for creating complex, subtle motion keyframes.

As a long-term animation system goal, an AI-based task performance system with heavy reliance on efficient algorithms is anticipated. While this system is being developed, however, human modeling analysts still need an effective tool to blend the individual strengths of guiding and algorithm methods. Even when the long-term system is in place, users will probably continue to demand an efficient way to modify the motion analysis output if desired. The instrumented manikin can be an effective tool for providing this option.

REFERENCES

1. Badler, Norman I. "A representation for natural human movement"; Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, 1986.

2. Denavit, J. and Hartenberg, R.S. "A Kinematic Notation for Lower Pair Mechanisms Based on Matrices", JOURNAL OF APPLIED MECHANICS, Vol. 22, 1955.

3. Hollerbach, J.M. and Sahar, G. "Wrist-Partitioned, Inverse Kinematic Accelerations and Manipulator Dynamics", INTERNATIONAL JOURNAL OF ROBOTICS RESEARCH, Vol. 2, No. 4, 1983.

4. Magnenat-Thalmann, N. and Thalmann, D. (1988) "Course Notes on Synthetic Actors", SIGGRAPH '88, Atlanta, Georgia, 1988.

5. NASA-STD-3000, Man-Systems Integration Standards.

6. Zeltzer, D. "Motor Control techniques for figure animation", IEEE COMPUTER GRAPHICS APPLICATIONS, November 1982.

7. Zeltzer, D. "Toward an integrated view of 3-D computer animation", THE VISUAL COMPUTER: THE INTERNATIONAL JOURNAL OF COMPUTER GRAPHICS 1,4 (1985).

THE DEVELOPMENT OF THE CANADIAN MOBILE SERVICING SYSTEM KINEMATIC SIMULATION FACILITY

by:

G. Beyer, B. Diebold — CAE Electronics Ltd., P.O. Box 1800, Montreal, Que. Canada

W. Brimley, H. Kleinberg — Spar Aerospace Ltd., 1700 Ormont Dr., Weston, Ont. Canada

ABSTRACT

Canada will develop a Mobile Servicing System (MSS) as its contribution to the U.S./International Space Station Freedom. Components of the MSS will include a remote manipulator (SSRMS), a Special Purpose Dexterous Manipulator (SPDM), and a mobile base (MRS).

In order to support requirements analysis and the evaluation of operational concepts related to the use of the MSS a graphics based kinematic simulation/human-computer interface facility has been created.

The facility consists of the following elements:

(a) A two-dimensional graphics editor allowing the rapid development of virtual control stations.

(b) Kinematic simulations of the space station remote manipulators (SSRMS and SPDM), and mobile base.

(c) A three-dimensional graphics model of the space station, MSS, orbiter, and payloads.

These software elements combined with state of the art computer graphics hardware provide the capability to prototype MSS workstations, evaluate MSS operational capabilities, and investigate the human-computer interface in an interactive simulation environment.

This paper describes the graphics technology involved in the development and use of this facility.

1.0 INTRODUCTION

The Mobile Servicing System (MSS) will be Canada's contribution to the U.S./International space station. The MSS will play an important role in performing the following functions on the space station:

- ❑ Space station construction and assembly
- ❑ Transportation (External on the space station)
- ❑ Payload Handling (Deployment, retrieval, and berthing including the orbiter)
- ❑ Attached payload servicing (in the extravehicular environment)
- ❑ Space station maintenance (in the extravehicular environment)
- ❑ Crew extravehicular activity (EVA) support
- ❑ Space station safe haven support

1.1 MSS System Configuration

The space segment of the MSS comprises three elements; the MSC (Mobile Servicing Centre), the SPDM (Special Purpose Dextrous Manipulator), and the MMD (MSS Maintenance Depot).

The MSC comprises two sub-elements called the MRS (Mobile Remote Servicer) and the MT (Mobile Transporter). The MT is to be supplied by the United States and provides the MSC with translation, corner turning, and plane change capability. The MRS comprises a number of major systems. The MBS (MRS Base System) provides the structure which interfaces with the Mobile Transporter and accomodates payloads and the remaining systems of the MSC. The relocatable SSRMS (Space Station Remote Manipulator System) is provided as a system of the MSC.

Figure 1 illustrates some of the MSS equipment described above.

The intent of this paper is to describe the applications of graphics technology to the MSS systems design process and to the creation of the MSS Kinematic Simulation Facility.

The role of the facility within the overall systems design and space operations process will be described. Following this, technical details regarding the current uses, and hardware and software configuration of the facility will be discussed.

2.0 ROLE OF THE GRAPHICS WORKSTATION IN MSS SYSTEMS DESIGN AND OPERATIONS

The development of the MSS Kinematic Simulation Facility was driven by the need to rapidly prototype and evaluate candidate configurations and capabilities of manipulators and control stations in a cost-effective manner.

This facility forms an integral part of the systems design process providing input at all stages of the design. The following will describe the uses of the graphics workstation in relation to this process.

2.1 Derivation of Requirements

The ability to visualize abstract concepts allows a systems designer to gain insight into the system being designed. The knowledge gained thus allows the designer to define, refine, and verify system requirements in a more efficient manner.

The preliminary definition process consists of performing task analyses based on operational concepts and proposing designs or prototypes as implementation solutions to the requirements specified in the formal program requirements documentation.

Rapid prototyping capabilities allow the formulation and creation of many competing design concepts in a cost-effective manner.

All components of the MSS must operate within the environment imposed by the physical and logistical infrastructure of the space station. As a result, influences external to the MSS have a significant impact on the design process. A graphics workstation based simulation facility provides the capability to simulate these external influences and assess their impact on design solutions.

2.2 Pilot Evaluation

A prototype control station may be evaluated using such criteria as accuracy of control, user preference, response time, the ability to learn and re-learn to use the workstation, and the ability to transfer training between operators.[1] Pilot evaluation consists of allowing qualified personnel such as astronauts and human factors specialists to interact with the simulations and evaluate the various prototoypes in a realistic environment.

The results of the evaluations are quantified through the use of questionaires designed to elicit relevant comments and impressions from the reviewers.

2.3 Iteration

The iteration process consists of refining the design by integrating the best features of each prototype identified by the pilot evaluators into a new proposed design and re-submitting the design for evaluation.

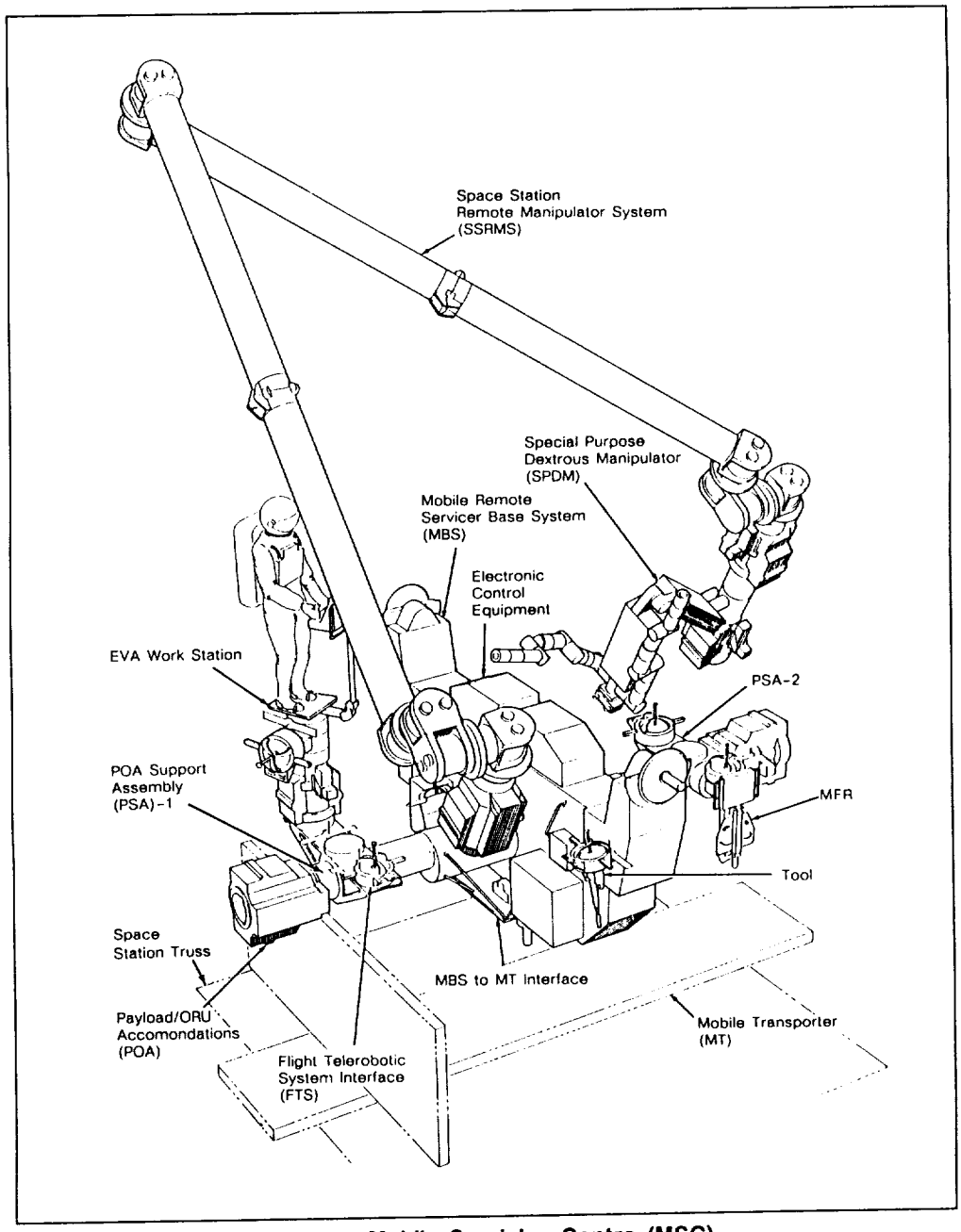


Figure 1 Mobile Servicing Centre (MSC)

2.4 Final Design

Eventually the prototypes will converge on a preferred configuration. The final design of the workstation may necessarily be a tradeoff between such factors as the capabilities of the available technology, requirements for interface commonality with other systems, operator preferences, and the impact of acceptable operational procedures.

3.0 MSS KINEMATIC SIMULATION FACILITY OVERVIEW

The intent of this section is to describe how the functional components of the facility are combined to provide an integrated simulation of MSS systems and how this capability is utilized in the development of the MSS.

3.1 Uses

The primary use for the MSS kinematic simulation facility is threefold:

- ❑ Operations Analysis

Operations analysis includes trajectory planning, reach analysis, viewing analysis, and evaluating the effectiveness/capability of the MSS to perform in the space station environment.

The outputs of operational analyses may effect the design of the MSS by providing critical information regarding the length of booms, and number, placement, and characteristics of joints. Information may also be obtained which impacts the space station design for operations using the MSS. In addition, viewing analyses assist in the preliminary definition of camera locations and quantities.

- ❑ Human-Computer Interface Development

Telerobotic applications rely on the ability of a human operator to directly control or supervise an operation. The definition, placement, size, colour, and functionality of the controls associated with the MSS will determine the ease with which the MSS will be operated. Evaluations of human-computer interface concepts are being performed in parallel with the systems design activities.

- ❑ Animated video production

Animated videos have been found to be an efficient method of conveying operational concepts and MSS capabilities. In addition, videos are excellent vehicles through which public awareness of space activities can be broadened.

3.2 Hardware Configuration

The MSS Kinematic Simulation Facility currently consists of a Silicon Graphics IRIS 4D 70-GT with 8 Mb of RAM, a 380 Mb hard drive, mouse, keyboard, 19 inch monitor with resolution of 1280 x 1024 pixels, 96 bitplanes, and an Ethernet card for communication with other hosts.

The operator's primary input devices are the keyboard, mouse, and a 6 degree of freedom handcontroller used for control of the manipulator. Other input devices such as a touch screen, trackball, discrete switches, and a voice recognition system may be added and evaluated serially or in parallel.

The hardware dedicated to the video recording function consists of an optical disk recorder, sync generator, IRIS genlock card, RGB Encoder, VHS editing video recorder, and an NTSC monitor.

Figure 2 depicts the current hardware configuration of the facility along with typical input/output device options.

3.3 Software Configuration

The facility operates in a Unix/C based stand-alone IRIS environment. Communication with other hosts for off-line processing is available however the stand-alone performance of the IRIS has been found to be sufficient for the current operational analyses and levels of simulation complexity. The iteration rate of the simulation varies between 3-5 Hz. depending on the number of simultaneous 3D windows being displayed and the processing demands of the kinematics.

Figure 3 depicts the current software configuration of the facility.

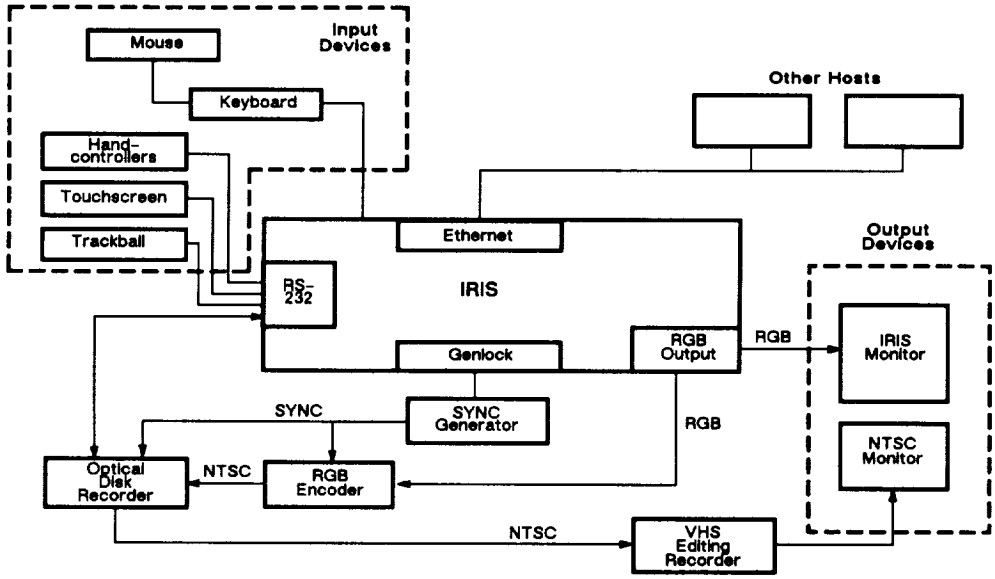


Figure 2 MSS Kinematic Simulation Facility Hardware Configuration

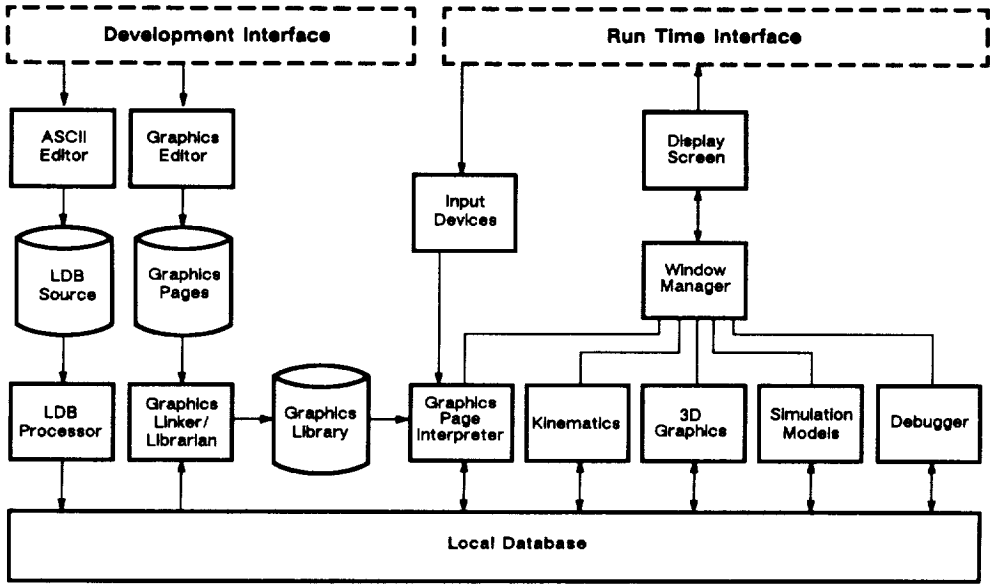


Figure 3 MSS Kinematic Simulation Facility Software Configuration

3.3.1 Simulation Environment

The MSS Kinematic Simulation Facility currently consists of the following simulation components:

- Two-dimensional virtual display and control panels
- Kinematic model of the space station remote manipulators
- Three-dimensional graphics animation of the space station, MSS, orbiter, and payloads
- Simulations of the operation of MSS systems.

TIGERS (The Integrated Graphics Environment for Real-Time Systems), a product of CAE Electronics, provides the simulation environment through which the MIKE (Manipulator Interactive Kinematics Evaluator) kinematic model and MIKEGRAF 3D animation software, produced by Spar Aerospace, are integrated with the 2D virtual display and control panels. Models of MSS systems are easily added to the simulation with all communication transpiring through a local database residing on the IRIS or a remote database residing on a remote host.

The simulation variables defined in the databases can also be displayed and controlled by the 2D virtual instruments created using the TIGERS graphics editor. The graphics editor pages are processed by a linker/librarian and then displayed by the graphics page interpreter as windows controlled by the TIGERS window manager. The 3D animations are also displayed as windows allowing the number of animation views, their size and location on the screen to be dynamically modified.

3.3.2 Two Dimensional Graphics Editor

Stylized panels and displays can be created using the on-screen graphical interface of the editor. Most inputs are made via pop-up menus and a standard 3-button mouse. The graphics editor provides a wide range of drawing tools, raster and vector fonts, and multiple dynamic attributes that can be applied to graphical elements of displays to make them respond to changing simulation variables. Examples of dynamic attributes include: color, size, position, rotation angle, and digital and alphanumeric readouts. Graphical elements can be combined into objects and stored in libraries for use on several display panels.

3.3.3 Virtual Displays and Controls

The virtual displays and controls created using the graphical editor provide the user interface for the MSS Kinematic Simulation Facility. The current interface consists of a parent screen, primary and secondary control areas, pulldown menus, and virtual control panels containing virtual instruments which interact with the MSS system simulations. Virtual instrumentation created using the editor includes digital readouts, icons, virtual pushbuttons, status indicators, and data input and feedback sliders. Figures 4 through 7 illustrate some typical prototype control panels developed for MSS applications.

3.3.4 System Simulations

Although some control over the attributes of the workstation prototype is available from the run-time services of the 2D graphical display manager, additional functionality may be achieved through the use of simulation routines which explicitly control the 2D graphical attributes. These routines would be required to simulate various menuing schemes or logic related to systems or subsystems driving the user interface.

3.3.5 Three Dimensional Graphic Modelling

Three dimensional (3D) graphical models of the environment may be created and rendered in individual windows under the control of the window manager. Typical objects used in operational analyses include the space station, orbiter, manipulators, payloads, and free flyers.

The 3D views are used to simulate out-of-window views from the orbiter or space station, views originating from various closed circuit television (CCTV) cameras, or synthetically generated images created from the space station master object database.

The graphical objects are created from combinations of the available 3D primitives which include boxes, cylinders, cones, spheres, and generic objects created by manipulation of

vertices and polygons. Attributes such as the position and orientation of a 3D object can be dynamically modified by generic simulation routines.

Objects may be individually rendered as wire frame, filled polygon, or Gouraud shaded polygons. From an operational analysis perspective, it has been found to be advantageous to allow concurrent display of wire frame and filled polygons as the former allows the operator a see-through capability which may assist in determining the relative position of objects. Object ordering is achieved through the use of the IRIS z-buffer which operates in double buffered mode.

The IRIS 4D-70 allows the use of multiple light sources to illuminate the workspace and allow some measure of realism to the view. It cannot however, adequately simulate the effects of shadowing and glare which have a significant effect on visibility in space.

Orthogonal or perspective viewpoints are available for display in each window. Viewing parameters including pan, tilt, roll and field of view, may be modified from a simulation routine which relies on inputs from the 2D virtual control domain or from alternate input devices such as hardware switches or buttons. In addition, the attachment location of a viewpoint (camera) may be tied to any object such as a manipulator which allows an assessment of viewing capabilities from manipulator cameras.

Graphical objects may be bound to other graphical objects to simulate the chaining of manipulators or the acquisition and maneuvering of payloads.

The 3D graphical displays are under the control of the window manager which provides the ability to:

- ❑ simulate graphics over video by the overlaying of 2D over 3D graphics.
- ❑ simulate split screen operation and resizing by manipulation of 3D windows using window manager services.

3.3.6 Kinematic Simulations

The configuration of the manipulator is controlled by a kinematic simulation routine which performs the inverse kinematics required to convert from a commanded point of resolution (POR) to the set of joint angles required to achieve the configuration. The set of joint angles along with the base position and orientation uniquely defines the manipulator configuration and may be used by the 3D graphical rendering routines to draw the manipulator.

The kinematic simulation currently implemented on the prototype has the following characteristics:

- ❑ "N" degrees of freedom: The SSRMS will consist of a 7 DOF manipulator.
- ❑ Bi-directional Control: The SSRMS will have the capability to attach itself to grapple fixtures which supply power and data transfer to both the base and end effectors. This feature means that the SSRMS can relocate itself by "walking" off of the MSC and operating from a grapple fixture at a remote location.
- ❑ Coordinate Re-referencing: The SSRMS may be controlled in any desired reference frame. Reference frames are selectable by the operator.
- ❑ Control Modes: Three main control modes are associated with the SSRMS kinematic model.

  Single Joint Mode: Manipulator joints may be controlled individually by the operator using a suitable input device.

  Automatic Mode: The manipulator configuration may be controlled by an automatic trajectory planner thus allowing the execution of pre-planned trajectories.

  Manual Augmented Mode: The manipulator POR may be controlled by a human operator using a handcontroller.

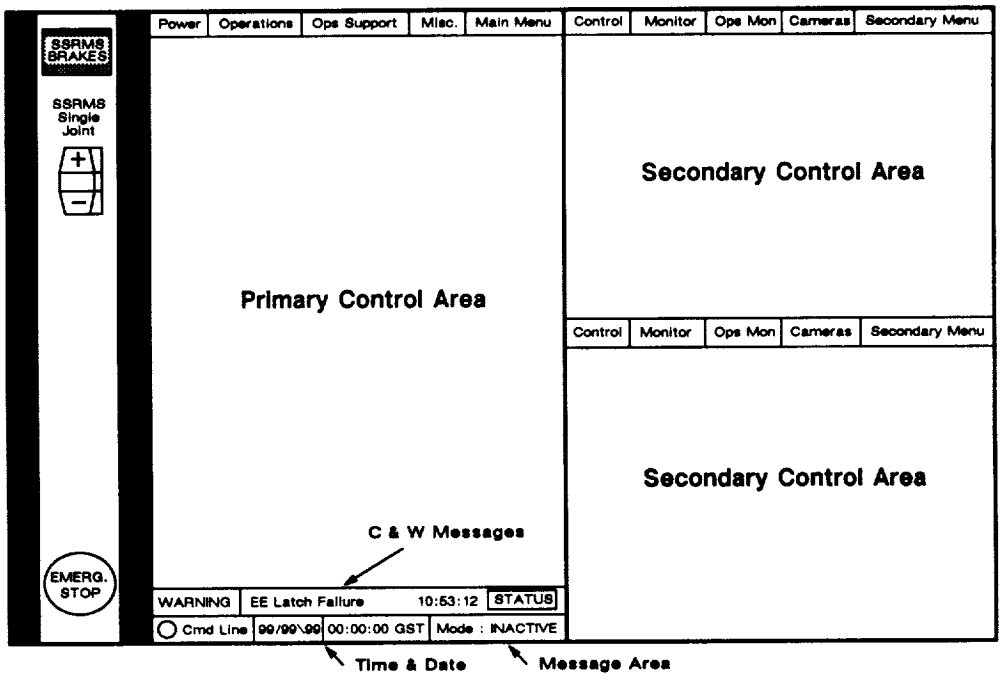
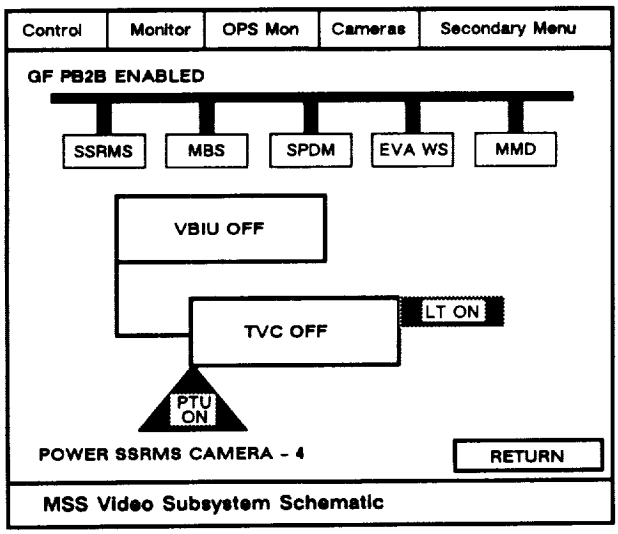


Figure 4 Graphical Display Template



NOTE: Figure is representative of the information required for display, not the implementation of how function will be displayed.

Figure 5 MSS Camera Power Control Panel Selection

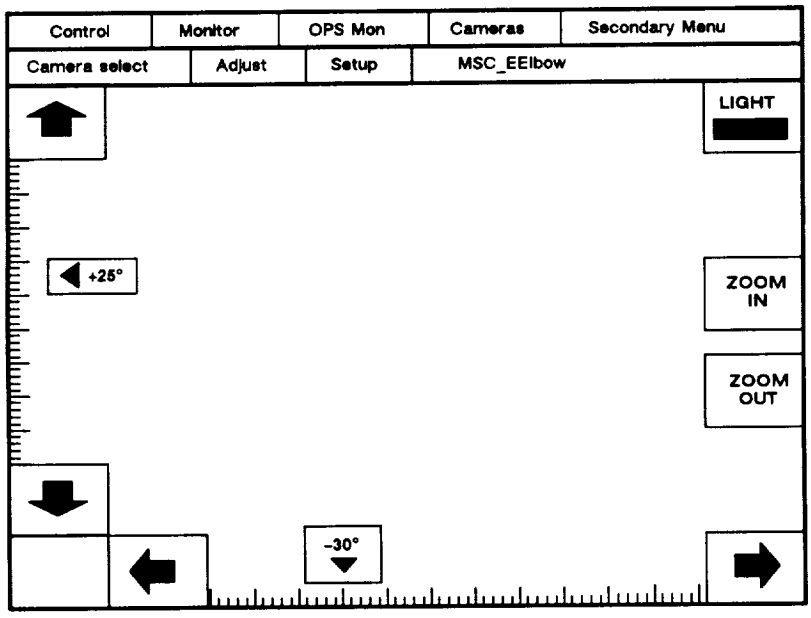


Figure 6 MSS Camera Control Panel

NOTE: Figure is representative of the information required for display, not the implementation of how function will be displayed.

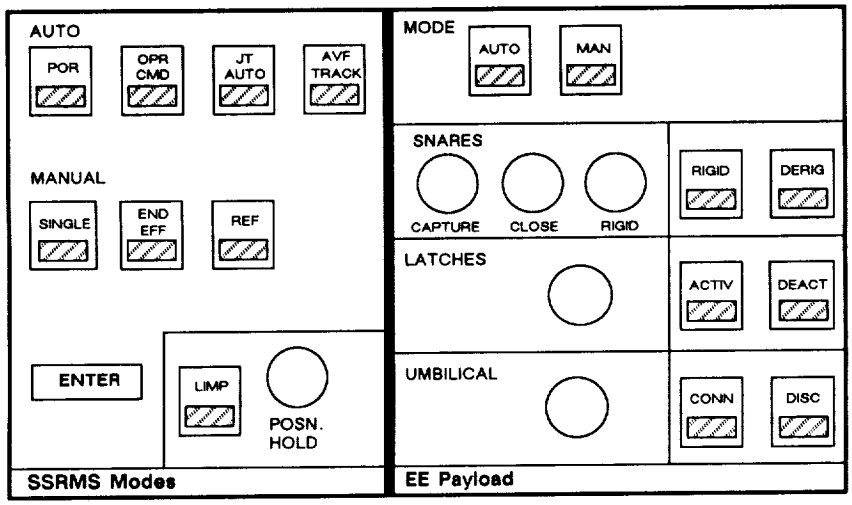


Figure 7 Manual End Effector Control - SSRMS Mode and Payload End Effector Panels

3.3.7 Post Processing Functions

Post processing software is included in the facility which allows analysis of simulation runs. Current post processing capabilities include workspace analysis and video recording and playback.

An interactive workspace analysis program allows multiple instances of manipulator positions to be superimposed. The investigator may then visually determine the point of closest approach between the manipulator and another object. The line of closest approach may be quantified by calculating the distance between the two points as defined with a 3D cursor.

After the definition of a manipulator trajectory the set of parameters relevant to the operation may be saved to disk for off-line trajectory post processing analyses or the creation of videos.

Animated videos may be created for engineering presentations or public relations. The video software will read the disk file, redraw the image, and sent the appropriate commands to the optical disk recorder for recording of the video image. The images thus stored are spliced together on a VHS editing tape recorder for production of the final video.

4.0 FUTURE DEVELOPMENTS

The MSS Program will achieve a higher level of simulation capability with the development and delivery of the MDSF (Manipulator Development and Simulation Facility). The MDSF will provide all the functionality of the MSS Kinematic Simulation Facility along with the following additional features:

- ❑ Real-time dynamic simulation of generic manipulators with multiple operator control stations
- ❑ 3D graphics editor
- ❑ Generic instructor station for operator training
- ❑ Record and Playback functions
- ❑ Simulation of elastic deformation of bodies
- ❑ Collision detection algorithms

5.0 CONCLUSION

An implementation of computer graphics technology in the design of a complex system has been presented, specifically related to the development of the Canadian Mobile Servicing System and its IVA human-computer interface. The systems design methodology, hardware and software configuration, and current and future uses of the facility have been discussed.

The application of currently available graphics technology provides systems designers and operations analysts with the ability to visualize and simulate the capabilities of a complex system in a cost-effective manner. The integration of input/output devices with the simulation facility provides a high degree of interactiveness allowing the testing and verification of concepts throughout the design process in a realistic environment.

ACKNOWLEDGEMENTS

The work presented has been performed at SPAR Aerospace Limited, Remote Manipulator Systems Division, by CAE and SPAR personnel, under contract to the National Research Council of Canada, Space Division.

The authors would like to thank the NRCC, SPAR Aerospace Ltd, and CAE Electronics Ltd, for permission to present and publish this paper.

REFERENCES

[1] "Space Station Information System (SSIS) Human-Computer Interface Guide," Version 2.0, NASA USE-1000, May 1988

SOFTWARE SYSTEMS FOR MODELING ARTICULATED FIGURES

Cary B. Phillips

Computer Graphics Research Laboratory
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, Pennsylvania 19104-6389

Abstract

Research in computer animation and simulation of human task performance requires sophisticated geometric modeling and user interface tools. The software tools for a research environment should present the programmer with a powerful but flexible substrate of facilities for displaying and manipulating geometric objects, yet insure that future tools have a consistent and friendly user interface.

Jack is a system which provides a flexible and extensible programmer and user interface for displaying and manipulating complex geometric figures, particularly human figures in a 3D working environment. It is a basic software framework for high-performance Silicon Graphics IRIS workstations for modeling and manipulating geometric objects in a general but powerful way. It provides a consistent and user-friendly interface across various applications in computer animation and simulation of human task performance. Currently, Jack provides input and control for applications including lighting specification and image rendering, anthropometric modeling, figure positioning, inverse kinematics, dynamic simulation, and keyframe animation.

1 Introduction

The great promise of computer graphics is visualization, the ability to answer difficult problems and convey complex information through computer generated images. The problem for researchers in computer graphics is how to generate images which convey such useful information. Recent advances in computer hardware have revolutionized the capabilities of graphics simulation systems. Today's hardware is capable of displaying large numbers of graphics primitives in real time. The task now is to take full advantage of these new graphics capabilities in software modeling systems.

This charge applies especially to software for modeling geometric objects. The importance of visualization in geometric modeling is quite obvious, but the application of computer graphics goes beyond simply generating static synthetic images. A modeling systems should give it user the ability to *manipulate* the models, and the models should behave in a way which conveys information back to the user.

User interface toolkits such as X-windows provide good tools for designing interfaces to many types of software programs, but they do not provide adequate tools for constructing interfaces for manipulating geometric objects. Developers of software for computer animation and simulation need similar sorts of tools for assisting in the higher-level task of modeling, displaying, and manipulating complex geometric figures.

Jack is a system being developed at the University of Pennsylvania to support research in human task performance in the Computer Graphics Laboratory. Its goal is to provide a consistent and easy-to-use programmer utility for modeling, displaying, and manipulating complex articulated structures, and at the same time provide a consistent and convenient user interface across various applications. Jack runs on the 4D Series Silicon Graphics IRIS Workstations, and its intended user community consists primarily of engineers with a basic understanding of robotics and geometric modeling concepts.

Jack is very general in its ability to model articulated figures, but its primary purpose is to support

human factors analysis. The object modeling facilities in Jack are designed to handle the special difficulties of modeling and manipulating human figures in a 3D working environment. There are many sources of support for this project, each with its own emphasis and application:

- NASA Johnson Space Center and Lockheed Engineering and Management Services, Graphics Analysis Facility of the Man/Systems Division: primarily Space Shuttle and Space Station applications, with major interest in reach, fit, and view analyses, with active interest in strength models, zero-gravity dynamics simulation, and language-based task processing.
- NASA Ames Research Center: the A^3I project to simulate all aspects of a helicopter mission is the application, with Jack providing the pilot model and forming the basis for workload computations. The pilot's mission and tasks are provided by an external AI-based simulator.
- Army Research Office, the Human Engineering Laboratory at Aberdeen Proving Grounds: application to multi-operator vehicles, with a primary interest in evaluation of reach, comfort, strength, workload, and cooperative behavior.
- Pacific Northwest Laboratories, Battelle Memorial Institute: application to control a mobile robot mannequin used to test suit designs for permeability to chemical and biological agents, with a primary interest in animation control, safe path determination, collision avoidance, and motion feasibility.
- State of Pennsylvania Benjamin Franklin Partnership: technology development in Artificial Intelligence methods to aid human factors evaluation.
- National Science Foundation: representations and systems to assist in the interactive and automatic generation of natural, animated human motion.

This project greatly benefits from its home in a Computer and Information Science Department because computational tools and techniques are essential for such a broad spectrum of human performance problems.

This paper gives an overview of Jack. It explains many of the features built into Jack, including its object modeling facilities as well as its interaction mechanism for manipulating figures through direct manipulation and inverse kinematics.

2 The Software Engineering Aspect

In order for computer graphics to fulfill its great promise, software developers must be careful to craft their systems to be effective *tools*. The graphics images must be merely the means to the end and not a burden to support.

This is especially important in a research environment, where researchers need to experiment with new types of algorithms and techniques, yet produce software tools which will be usable by non-programmers. In such an environment, it is particularly important to develop flexible and extensible software tools, since it is not always possible to fully anticipate future demands of the software. In this type of environment, the modeling software cannot be a "black box" because it is frequently necessary to extend it or customize it in various ways.

Jack is designed to be easily extensible. It is a command driven system: the user typically executes commands by selecting items from pop-up menus, but commands may also be entered explicitly from the keyboard or read from command files. Jack is very modular in that each command or group of commands roughly corresponds to a specific utility. During development, certain unnecessary utilities may be omitted, drastically reducing the overhead of the development process.

Jack follows a well-defined mechanism for defining commands and controlling input from the user. This mechanism makes it very easy to customize Jack for specific applications. Many times such applications evolve into sophisticated standard utilities. The simplicity with which commands may be written and added to the menus make it easy to manage the software as it grows and encorporates new utilities.

3 The Peabody Object Representation

The heart and soul of geometric modeling software is the representation for geometric objects. This involves much more that just the shape of the objects. The models must represent information about how the objects *behave*, how they simulate the behavior

of the real-world objects they represent. The purpose of the graphics facilities in the software is simply to convey information about the models. If the underlying model is not rich in information, the graphics facilities will have little use.

Jack is primarily a user interface which controls the interaction with articulated figures represented by a system called peabody. The name *peabody* refers to both the internal data structures representing the geometric objects and to the external language for describing and storing them. Peabody objects are described in text files, and Jack can be viewed as a graphical editor for constructing and manipulating these objects. By analogy, this editor is to the geometric objects what a word processor is to English prose.

Peabody represents *figures* composed of rigid *segments* connected by *joints*, which may also be under the influence of certain *constraints*. The segment is the basic geometric primitive. The state variables of each segment represent its mass and moment of inertia, as well as its surface geometry, which is a boundary representation. Segments also represent material properties such as reflectance parameters and light emission values.

Joints connect segments through attachment frames called *sites*. A site is a local coordinate frame specified with respect to the base coordinate frame of the segment to which it belongs. Segments may have any number of sites. Joints connect sites belonging to different segments within the same figure. Constraints are pseudo-joints which express relationships between arbitrary sites in the environment.

3.1 Articulated Figures

One of the most important features of the peabody representation is its model for the articulation of the figures. Figures may be composed of any number of segments connected arbitrarily by joints. Since each segment may have any number of sites, and the sites may be located anywhere on the segment, the "skeleton" of the figure is easy to define even if the segment does not have a distinct proximal and distal end.

The joints may have arbitrary degrees of freedom, which peabody represents as zero to six rotational or translational components. A zero degree of freedom joint is like a bolt and is not manipulatable. Each degree of freedom may have upper and lower limits on it range of motion, and the limits are enforced during interaction. The degrees of freedom also represent stiffness and dampening information for dynamic simulation.

The user treats figures as arbitrary collections of segments connected by joints, without imposing a predefined hierarchy upon them. Jack encourages the user to think of the geometric objects as an arbitrary graph of segments connected by joints. It computes the global position and orientation of each segment internally by first computing a spanning tree of the environment. This tree need only be recomputed when a new joint or segment is created or deleted, i.e. when the topology of the environment graph is altered.

Since this tree is computed internally, the user does not have to think of a figure as a strict hierarchy with a predefined root. This simplifies the operation of "rerooting" a figure, either to attach a figure to another figure, or to change the point of attachment of a figure to the world coordinate frame. This scheme makes it easy to specify transformations with respect to arbitrary reference frames.

The arbitrary figure root is important for manipulating figures which must maintain contact which certain points in space. This is especially true of manipulating figures in a zero gravity environment, where figures may be attached through arbitrary foot or hand restraints. When a figure is attached to a hand restraint in zero gravity, a bend in the elbow results in the movement of the entire body, rather than a movement of the hand and arm, which remains fixed at the point of attachment.

3.2 Human Figure Models

Although Jack is primarily a tool for human factors analysis, it makes no formal distinction between human figures and other geometric objects. All objects are described by peabody, and the human figure models used are described by data files designed to model the human figure in the specific ways. This model is external to the software itself, which makes it possible to model figures with differing degrees of complexity. Different body models may be used in different circumstances. For example, engineers in the Graphics Analysis Facility at JSC have developed a model for the Extra-Vehicular Activity suit which has restricted ranges of motion.

In addition, the actual geometry of the individual segments is defined separately from the topology of the figure, which is defined in terms of the locations of the joint centers. This makes it possible to use different body geometries with the same underlying topologies. Jack currently has three basic body geometries. Each model consists of 31 segments with 29 joints. The

first model consists of 109 polygons and represents a very crude, almost stick-figure, approximation to the human body. An intermediate model consists of 408 polygons and resembles a robot-like figure. Finally, a highly complex model has been derived from laser scan data of human subjects. This model consists of 4571 polygons.

3.3 Anthropometry

The syntax of peabody language loosely resembles data structure definitions in a traditional programming language. The peabody language employs a powerful mechanism for parsing arithmetic expressions. These expressions may be used as part of the definition of the figures, so that figures may be parameterized.

The ability to parameterize figures allows Jack to easily model human figures of arbitrary anthropometric proportions. An auxiliary facility called SASS is a spreadsheet program which allows the user to create peabody human figure models of arbitrary anthropometric sizes, based either on percentiles from specific populations or from actual numerical values. It generates parameters for girth, joint limits, and centers of mass. Currently, SASS uses NASA trainee population data from the NASA Man-Systems Integration Manual, Chapter 3 (NASA-STD-3000).

4 The Jack Window System

Jack uses the Silicon Graphics IRIS window manager, **4Sight**, which run under the Unix operating system. This window manager allows the user of the workstation to create multiple windows and run different graphics programs simultaneously. Jack creates windows which provide views of geometric objects. These windows may be moved and reshaped just like the other window manager windows. This allows Jack to be used as a "tool." The user may easily shift back and forth between using Jack and using the underlying operating system.

Jack derives most of its input from a three-button mouse, with a little input required from the keyboard. It is a menu-driven system, and commands are generally executed by selecting items from the pop-up menus. Although Jack maintains unique names for all geometric objects, it is usually possible to refer to objects by pointing at them with the mouse. Most of the keyboard input is in the form of single keystrokes to invoke certain options. Very little typing is required, although it is possible to control Jack completely without the mouse if necessary. Jack avoid being too cryptic in its keystroke bindings by displaying information about the bindings whenever the user is in a position to need them.

The execution philosophy of Jack is to select a high level operation first, and then select the operands. The user executes commands from a menu, such as **move figure**, then he pick the appropriate object by pointing at it with the mouse. Finally, he specifies the *value* of the operation, i.e. a transformation, which is usually manipulated interactively. Most operations such as moving are terminated by hitting a special key, such as the escape key.

4.1 Jack Windows

The parameters of each Jack window are easily to tailor for specific applications and situations. By default, Jack displays the screen in a visually informative way by drawing a ground reference plane grid, giving a perception of the orientation of the world coordinate system. Jack draws orthogonal projections of the figures in the scene on each of the coordinate axis planes. These projections roughly resemble shadows from three infinite orthogonal light sources and serve as quick reference cues for the position and orientation of the figures. The projections are drawn in a darker color than the figures themselves, so they do not distract from the rest of the scene. Since all three projections are closely placed on the screen, the user can quickly reference the orientation and relative placement of neighboring objects in the scene. These projections may be easily disabled, and may also be enabled on a segment by segment basis.

Most aspects of the display are optional. The user may choose to display the vertices, edges, or faces of each segment. The edges are drawn in wireframe. The face may be shaded and z-buffered, illuminated by multiple light sources using the lighting model hardware of the IRIS workstation. With the IRIS hardware, there is no significant performance penalty involved in the shading, and the user is free select either shaded or wireframe display. The sites associated with each segment may also be displayed. Sites are drawn as a labeled *xyz* coordinate axis frame. This allows the display to be tailored to suit a particular application, since all forms of display may not be appropriate for all tasks.

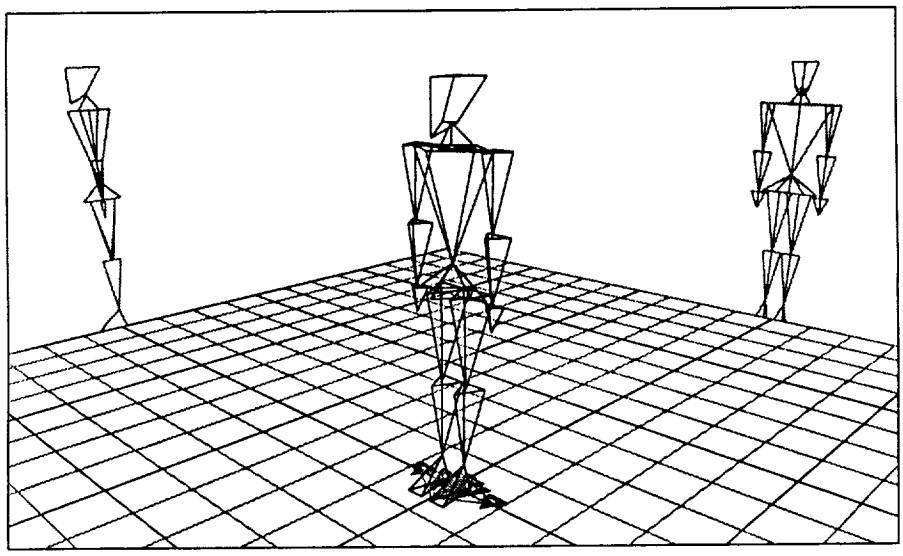


Figure 1: The Jack screen, with a human figure model

4.2 Viewing Facilities

It is especially important for modeling systems to provide good facilities for change the view. In the real world, when someone is presented with an object to observe, the natural reaction is to look at it from different directions, either by picking it up and moving it around, or if it is too large to handle, by walking around it and looking at it from all sides. Software which attempts to convey geometric information through images should provide a similar ability.

Jack has a flexible way of manipulating the view. The view in each Jack window is described by the global position and orientation of a specific site on a peabody figure. By default, a "camera" figure accompanies each window to represent the view. Moving the view in a window corresponds to moving the camera figure in a special way. This is especially beneficial in the Jack windowing environment, since the user may create different windows, each with a different camera. The camera figures may be displayed just like any other figure, so it is possible to see and manipulate in one window the camera of another window.

Jack allows the user to change the view by what it calls *sweeping* and *panning*. The sweeping operation moves the camera in circular arcs centered at a constant reference point, called the *view reference point*. The view may swing horizontally or vertically, or it may zoom in and out, all controlled by the mouse. This is beneficial for viewing a particular point in space from different directions. Panning is the opposite of sweeping: the location of the camera remains fixed while it pivots up or down. The view reference point changes as the camera turns. This is useful for looking side to side, but it is also an easy way to move the view reference point around in space.

The view in each Jack window may alternatively be "attached" to any site on any figure. This makes it possible to attach the view to the eyes of a human figure model and *see* in the window what the figure *sees*. This is particularly beneficial during the animation of the motion of a figure. The figure may be manipulated from a secondary point of view, and the animation may be played back both from the point of view of the figure or from a fixed point.

Another application of the viewing mechanism is positioning light sources. A special light source positioning facility temporarily attaches the view to the light and then allows the user to adjust the view. The user sees in the window where the light shines.

5 Object Manipulation Facilities

An important characteristic of truly manipulatable computer models of geometric figures is quick turnaround time between the user's decision to position the figure in a certain way and the time he actually accomplishes his goal. Most figure positioning tasks in an interactive 3D environment either require great precision, such as moving objects to tangency

or contact, or are very general, in which case rough positions are sufficient and precision is not an issue. Generally, the more detailed the positioning task, the more input may be required from the user. It is important for modeling software to be able to handle both cases well because the user should not be forced to enter complex input for a simple positioning task.

Jack provides several facilities for moving and manipulating objects. The user may use great precision when necessary, but may also quickly and intuitively move objects around in the workspace without the overhead of a complex positioning algorithm. The direct manipulation scheme is useful for position objects with gross movements. The inverse kinematics facility allows the user to position objects automatically to achieve multiple simultaneous goal positions.

5.1 Direct Manipulation

Jack uses a "view based" 3D direct manipulation scheme. All of the rotational and translational input is obtained from the mouse, but the movement of the mouse by the user is coupled to the current view of the object being manipulated, so that there is a direct and intuitive relationship between the direction the user moves the mouse and the direction the object moves.

For 3D translation, the user selects an axis of translation by holding down a button on three button mouse, but after the axis has been selected, the direction of movement of the mouse which cause movement of the object along that axis is determined by the line which that axis in space makes on the screen. The user may also translate objects in a plane by holding down two mouse buttons simultaneously, in which the movement of the object is constrained to lie in that plane and the location of the object is determined by the point in the plane which lies underneath the mouse cursor in the current view. The effect is a intuitive way of translating objects, since the object "follows" the mouse on the screen.

Rotations in 3D are accomplished with a rotation "wheel", which is a graphical icon describing the axis of rotation. The user selects the axis of rotation by holding down a single mouse button, and the wheel appears to demonstrate the selected axis. The rotation is accomplished by moving the mouse around the perimeter of the rotation wheel. The effect is also fairly intuitive, since the user moves the mouse in circles around the object to cause the object to rotate.

The rotation and translation mechanisms are used both for moving figures in the world coordinate frame and for manipulating the displacements of joints. Joints may have either rotational or prismatic components in their degrees of freedom, and the user may manipulate the joint using the direct manipulation facilities. If the joint has limits, the limits are obeyed during the interaction, and the joint is not allowed to violate the limit.

5.2 Inverse Kinematics

The direct manipulation facilities in Jack make it easy to position entire figures and manipulate individual joints by hand. But many positioning tasks involve manipulating many joints simultaneously until a certain condition is satisfied, such as tangency or point-to-point contact. Jack has a sophisticated inverse kinematics facility which uses a gradient descent algorithm to solve for a set of joint angles, within the defined joint limits, which satisfy a number of geometric "goals". The user-selectable parameters of the "reach" are the goal site, the end effector, and the set of joints to be manipulated during the reach. The objective function may encorporate a weighted combination of position and orientation. During the solution of a multiple goal reach, each goal may have a separate weighting factor, which specifies the relative importance of each goal if the goals are not all simultaneously reachable.

There are several variants of the reach algorithm. First, the *active* reach attempts to model the behavior of a real human subject performing a reach. It attempts to solve the reach with af user-specified chain of joints, but if the goal is not reachable, joints are added to the joint chain, working towards the body root, until the chain includes all joints between the end effector and the root.

Another variant of the reach algorithm is a *pointing* reach, which is useful for orienting the head and eyes of a human figure for looking at a particular point in space. The user input is similar to the ordinary reach, but the algorithm manipulated the joints so that the line of sight of the end effector is directed towards the goal.

5.3 Keyframe Animation

Jack has a sophisticated keyframe animation subsystem which allows the user to define *groups* and *actions*. Groups are sets of "things which change over time", typically joints and constraints. Actions are primitive sequences of changes to the values of the elements of a group. *Keyframes* are sets of values for

the elements of a group. A *scene* is a collection of possibly overlapping actions.

The animation facility may be used directly for keyframing known movements or interpolating between specific positions. It may alternatively be used as a means of collecting, storing, and playing back motions generated from external means, such as from external dynamic simulation software. Typically, the dynamic simulation produces output at specific time slices, which may be greater or less then the desired frame rate for playing back the motion sequence. The features of the animation system allow this to be easily controlled.

References

[1] Badler, Norman I., Jonathan D. Korein, James U. Korein, Gerald Radack, Lynne S. Brotman, "Positioning and Animating Human Figures in a Task-Oriented Environment," *The Visual Computer 1,* No. 3, 1985.

[2] Grosso, Marc, Richard R. Quach, and Norman I. Badler, "Anthropometry for Computer Graphics Human Figures." Technical Report, Dept. of Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1989.

[3] Phillips, Cary B, and Norman I. Badler, "Jack: A Toolkit for Manipulating Articulated Figures" Proceedings of ACM/SIGGRAPH Syposium on User Interface Software, Banff, Alberta, Canada, 1988.

[4] Phillips, Cary, "Programming With Jack," Technical Report, Dept. of Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1989.

[5] Phillips, Cary, "Using Jack," Technical Report, Dept. of Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1989.

[6] Zhao, Jianmin and Norman I. Badler, "Real-time Inverse Kinematics with Joint Limits and Spatial Constraints" Technical Report MS-CIS-89-09, Dept. of Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1989.

HUMAN TASK ANIMATION FROM PERFORMANCE MODELS AND NATURAL LANGUAGE INPUT

Jeffrey Esakov
Norman I. Badler
Moon Jung
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, Pennsylvania 19104-6389

Abstract

Graphical manipulation of human figures is essential for certain types of human factors analyses such as reach, clearance, fit, and view. In many situations, however, the animation of simulated people performing various tasks may be based on more complicated functions involving multiple simultaneous reaches, critical timing, resource availability, and human performance capabilities. One rather effective means for creating such a simulation is through a natural language description of the tasks to be carried out. Given an anthropometrically-sized figure and a geometric workplace environment, various simple actions such as reach, turn, and view can be effectively controlled from language commands or standard NASA checklist procedures. The commands may also be generated by external simulation tools. Task timing is determined from actual performance models, if available, such as strength models or Fitts' Law. The resulting action specifications are animated on a Silicon Graphics Iris workstation in real-time.

1 Introduction

Simple computer animation is not so simple anymore. What was once acknowledged as a "good" animation is no longer acceptable. Animations are not necessarily things which are "looked at" for aesthetic purposes but are being used for practical applications in science and engineering analyses. Human figure animation, in particular, is receiving considerable attention as new display systems and robust animation software bring motion control and rendering capabilities to a widening range of users. Animations are created to evaluate the ability of people to fit or work in designed environments, determine whether work places satisfy their functional requirements, and analyze human task performance in a given situation. With the expanded role of animation and increased viewer sophistication, the tools for developing animations for these analytic purposes have become considerably more complex.

To gain control over complexity, animation tools are becoming "task oriented." A system which allows a process to be described at a level best suited for the action allows the user to specify the action in the least restrictive, and most natural, manner [4, 23]. This important benefit becomes crucial as the animation tools shift out of the animation production houses and into other industries and laboratories; human factors engineers often lack the manual and artistic skills necessary for the specification of animation.

The solution to this problem is two-fold. New users must be educated, but also, the vocabulary recognized by the tools must be modified. Certainly, the obvious conclusion is that the tools must understand a "task level" vocabulary. Even with that higher level of understanding, communication would still be limited as the user not only lacks the vocabulary, but also the language for communication.

The ideal language for communication is one with which the user is most comfortable. Natural language parsers, however, are complex programs [3]. Furthermore, integrating such a program into the animation environment introduces several interfacing problems [5].

We shall describe here a prototype system in which task animation is driven via natural language. We

focus on the interface between the natural language parser and the motion generator. The paper is organized as follows. Section 2 discusses how we currently limit the scope of the problem and describes the domain in which our animations are created. Section 3 describes relevant research. Section 4 discusses how the parser and motion generator are integrated. Section 5 describes the technique which is used to fill in the timing information tacitly embedded in the natural language commands.

2 Problem Domain

Since our goal is to investigate the linkage between language and task animation, initially the task domain is limited to "simple" reaches and view changes. (Karlin [17] investigated more complex motions; these will be added to the system vocabulary later.) A "simple" reach is one which requires no locomotion, only movement of the arm or upper body. A view change is a change in the orientation of a figure's head (i.e. the figure's view of the world changes). While seemingly very easy, these tasks already demonstrate much of the essential complexity underlying language-based animation control.

2.1 Task Environment

The tasks to be performed and animated all center around a control panel (i.e. a finite region of more or less rigidly fixed manually-controllable objects). By using a control panel, it is obvious that many everyday tasks can be simulated. Some control panels encountered in a normal day-to-day routine are typewriter keyboards, elevator panels, light switches, and car dashboards. We will use as a generic example the remote manipulator system control panel in the space shuttle (Figure 1) as it contains a variety of controls and indicators.

The purpose of creating the task animation is for task performance analysis. In particular, we want to determine if some person, X, can perform a task, and if so, we want to view the task performance. However, task performance depends on who is executing the task. If X has short arms, then he might not able to reach the control panel. Therefore, included in our task environment is the ability to specify the anthropometric "sizing" of the people to be included [15]. The size is based on a percentage of some population data (e.g., NASA crew member trainees [1]). For example, a 50%-ile man represents the average man in some body of data, whereas the 95%-ile man represents a man whose size parameters are in the 95^{th} percentile. Similar data should exist for women over some population. Figure 2 shows 50^{th} and 95^{th} percentile men and women based upon available data [21].

3 Relevant Research

Zeltzer [26] first gave names to the various "levels" of computer animation: "guiding level," "production level," and "task level." Using his nomenclature, the type of system we describe here is a "task level" system. His system for controlling the walk of human figure [25] is a specialized system for a particular task to be performed (i.e., walking). For now, our "skills" consist of reaching and viewing.

The Story Driven Animation System [22] accepts modified natural language input and creates the corresponding animation. The emphasis in this work is on story understanding and the ability to *choose* the correct key frames. Similar high level (intelligent) selection among existing key frames is also demonstrated by Fishwick [11, 10]

MIRALOGIC [19] is an interesting approach to embedding a high-level of understanding within an animation system. Through the use of this expert system, the user can specify rules for setting up an environment and the system will identify inconsistencies or potential problems and suggest possible solutions.

ASAS [20], and the other object-oriented systems it exemplifies [19], can also implement task-level semantics through task decomposition. A task can be decomposed procedurally.

These systems all address a different type of problem than that which is being addressed here. The tasks in our system are specified in natural (or any syntactically-described artificial) language with the purpose of examining task performance. As such, it is easy to change the tasks as well as the anthropometric parameters describing the performers.

4 Integrating Language and Motion Generation

The primary focus of this work is to examine how natural language task specification and animation can be combined in an application-independent manner. The burden of this requirement falls upon the link

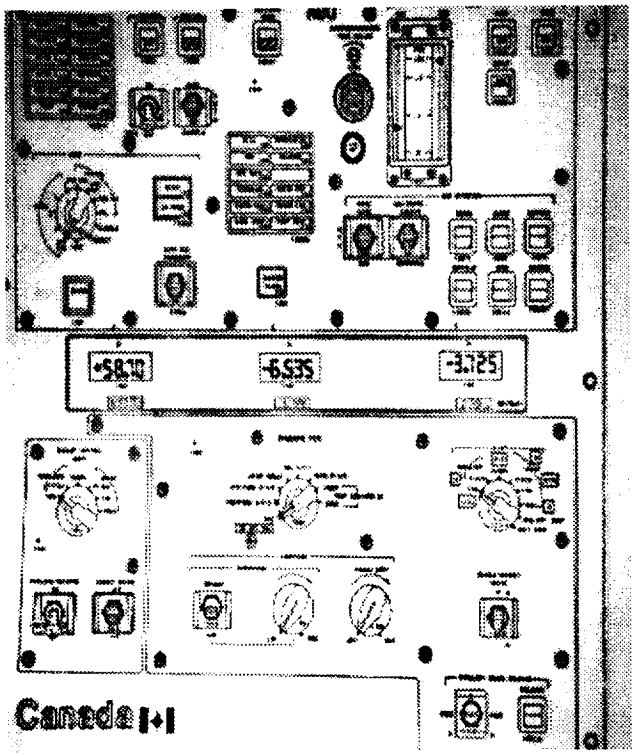
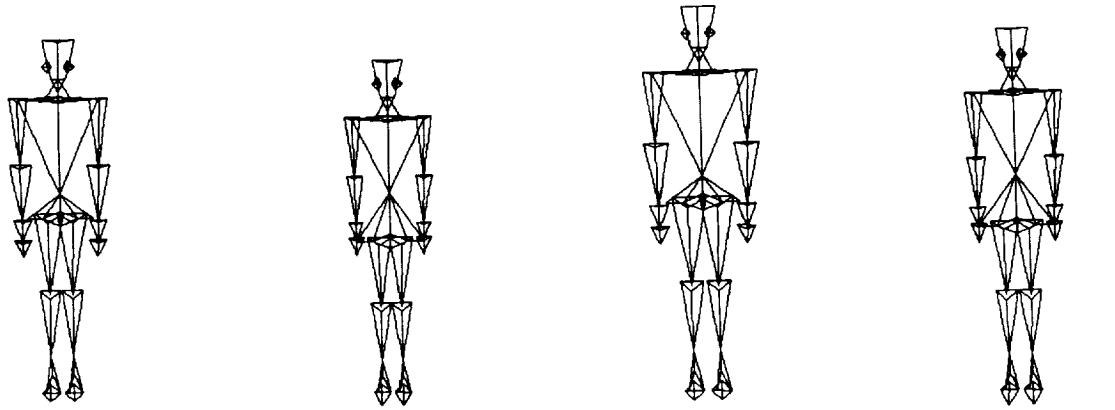


Figure 1: Space Shuttle Remote Manipulator System Control Panel



(a) 50^{th}% man (b) 50^{th}% woman (c) 95^{th}% man (d) 95^{th}% woman

Figure 2: Anthropomorphically Valid Articulated Figures

between these two environments. To illustrate the situation, we will discuss a sample natural language script actually used to create an animation:

```
J is a 50 percent man.
S is a 50 percent woman.
J, look at switch twf-1.
J, turn twf-1 to state 4.
S, look at tglJ-1.
J, look at twf-2.
S, turn tglJ-1 on.
S, look at twf-3.
S, turn twf-3 to state 1.
J, look at twf-3.
J, look at S.
S, look at J.
```

This type of script is common in performing checklist procedures such as those done in airplanes or space shuttles [2]. The verb "look at" represents a view change and the verb "turn" involves a simple reach. (The parser accepts a larger variety of syntactic constructions than illustrated by this example [5].)

The two primary problems are specifying reach and view goals, and connecting object references to their geometric instances.

4.1 Specifying Goals

A goal for a reach task is the point which the hand should touch. For this particular type of task, such a goal has three positional degrees of freedom, although there are situations in which rotational degrees of freedom may be considered as well. A view goal is a point in space toward which one axis of an object must be pointed.

Within an animation environment, such goals represent points in space (for position goals) or coordinate reference frames (for position and rotation goals) ultimately specified numerically with respect to a coordinate system. Within the natural language environment, the goals are not coordinates, but rather are represented by objects as in, for example, the commands:

```
J, look at switch twF-1.
S, turn switch tglJ-1 on.
```

The information regarding the exact locations of the switches is basically unimportant at the language level. Somehow, the switch name `tglJ-1` must be mapped to the appropriate switch on the panel in the animation environment. The same process must be followed for the target object toward which an object axis must be aligned in a view change. This problem reduces to one of object referencing.

4.2 Object Referencing

In general, all objects have names. Although the names may be different in the animation and language environments, providing a map between the names is not difficult. This, of course, assumes there is a one-to-one correspondence among the names. Such a requirement, however, defeats the goal of independence between the environments.

The problem domain specifically includes control panels. From a task specification perspective, a control panel is a very complex object consisting of many features such as controls, indicators, etc. From a computer graphics perspective, the most salient feature of the control panel is its appearance, not necessarily the detailed geometry of the individual switches. An object such as a control panel can most efficiently be represented as a single textured object which can then be mapped onto a polygon. The alternative of representing each individual switch would require a large number of polygons and an extensive amount of digitizing work to obtain a visually adequate representation of the switches.

By allowing each environment to represent the panel in a manner that is best suited for the way in which it will be referenced, the one-to-one correspondence among names is lost. The many objects in the task specification environment all correspond to a single texture mapped panel. A method is needed which will allow the construction of a mapping of feature names in the task specification environment to texture map locations in the animation environment.

We used a paint program as the basis for such a tool. Since a paint program allows one to create the texture maps in image space, additional input was required to specify the polygon on which the image is to be mapped. With that information, important locations on the texture map could be identified and given attributes (e.g., switch or indicator, rotary control or push button, etc.), and the corresponding locations on the polygon were calculated. The output of this tool provided input to both the semantic knowledge base and the geometric database.

4.2.1 The Knowledge Base

The knowledge base needs to contain information about object names and hierarchies, but need not

be concerned with actual geometry or location. Furthermore, as the task specifications and object definitions become more complex, the knowledge base can contain causality relationships. For example, turning switch `tglJ-1` to `on` may cause some other object to move or change state [5]. We use a frame-like knowledge base called DC-RL to store semantic information [8].

Object information must be entered into the knowledge base manually, as it can differ for each control panel, but the name mapping program described above can be used to specify the linkages into the animation environment.

For example, here is a section of an actual map file.

```
{ concept ctrlpanel from panelfig
  having (
    [role twF-1 with
          [ value = ctrlpanel.panel.twf_1 ]]
    [role twF-2 with
          [ value = ctrlpanel.panel.twf_2 ]]
    [role twF-3 with
          [ value = ctrlpanel.panel.twf_3 ]]
    [role tglJ-1 with
          [ value = ctrlpanel.panel.tglj_1 ]]
    [role tglJ-2 with
          [ value = ctrlpanel.panel.tglj_2 ]]
  )
}
```

The names `twF-1`, `twF-2`, `tglJ-1` correspond to the names of switches manually created in the existing knowledge base panel description called `panelfig`. These names are mapped to the corresponding names in the animation environment (e.g., `ctrlpanel.panel.twf_1`, etc.) and are guaranteed to match as the actual object within the animation environment is automatically generated.

4.2.2 The Geometric Database

The geometric database is called the Peabody Environment Network (or just `peabody`). In peabody, a figure is composed of a set of *segments*, each of which may have geometry associated with it. The geometry within each segment is defined within its own local coordinate system. *Joints* connect segments at attachment points called *sites*. A joint is actually a transformation between sites and hence sites have an orientation as well as a location. Segments can have any number of sites and it is through those sites that the different interesting points on the texture map are identified for the animation environment.

The relevant part of the peabody description of the panel figure is shown:

```
figure ctrlpanel {
  segment panel {
    psurf = "panel.pss";
    site base->location =
         trans(0.00cm,0.00cm,0.00cm);
    site twf_1->location =
         trans(13.25cm,163.02cm,80.86cm);
    site twf_2->location =
         trans(64.78cm,115.87cm,95.00cm);
    site twf_3->location =
         trans(52.84cm,129.09cm,91.43cm);
    site tglj_1->location =
         trans(72.36cm,158.77cm,81.46cm);
    site tglj_2->location =
         trans(9.15cm,115.93cm,94.98cm);
  }
}
```

This entire file is automatically generated based upon the map file. Since the panel is a rigid object with no movable parts, no joints are required. The location of each site (each of which represents a different switch) was calculated in the paint program (which created the file) by applying the texture mapping transformations normally applied when the image is rendered.

4.3 Creating an Animation

Mapping objects from the task description environment to the animation environment provides one of the crucial links needed for creating an animation. The language processor provides another link. Our Motion-Verb Parser (MVP) [5] uses both a subset of natural language and an artificial language (NASA checklists) for its syntax. Information obtained during the parse is stored in the semantic knowledge base DC-RL. The natural language task descriptions that are included in the problem domain are such that a single animation key frame can be developed from a single command. Each part of speech fills in slots in an animation command template.

Figure 3 shows the relationship between the task specification and the animation commands. A "turn" command specifies a reach which can be solved using inverse kinematics; a "look at" command specifies an orientation change which can also be solved using inverse kinematics [6, 14]. Frames from an animation created using the script shown in Section 4 are shown in Figure 4.

```
J look at switch twf-1.    ⟹  point_at("ctrlpanel.panel.twf_1","J.bottom_head.between_eyes",(1,0,0));
J turn twf-1 to state 4.   ⟹  reach_site("ctrlpanel.panel.twf_1","J.right_hand.fingers_distal");
S look at tglJ-1.          ⟹  point_at("ctrlpanel.panel.twj_1","S.bottom_head.between_eyes",(1,0,0));
S turn tglJ-1 on.          ⟹  reach_site("ctrlpanel.panel.twj_1","S.left_hand.fingers_distal");
```

Figure 3: Natural Language Input and Animation Commands

5 Default Timing Constructs

Given that the basic key frames can be generated based upon a natural language task description, creating the overall animation can still be somewhat difficult. Techniques for creating motion by animating the solution algorithm such as those done by Badler, Manoochehri and Walters [6], Witkin, Fleisher and Barr [24], or Barzel and Barr [7] are themselves inappropriate for task performance analysis. Instead, the positions created must be taken for what they are: the desired configuration of the body at a particular time. The exact time, however, is either unknown, unspecified, or arbitrary.

The timing of actions could be explicitly specified in the input, but (language-based) task descriptions do not normally indicate time. Alternatively, defining the time at which actions occur can be arbitrarily decided and a reasonable task animation can be produced. In fact, much animator effort is normally required to temporally position key postures. There are, however, more reasonable ways of formulating a guess for possible task duration.

Several factors effect task performance times, for example: level of expertise, desire to perform the task, degree of fatigue (mental and physical), distance to be moved, and target size. Realistically speaking, all of these need to be considered in the model, yet some are difficult to quantify. Obviously, the farther the distance to be moved, the longer a task should take. Furthermore, it is intuitively accepted that performing a task which requires precision work should take longer than one not involving precision work: for example, threading a needle versus putting papers on a desk.

Fitts [12] and Fitts and Peterson [13] investigated performance time with respect to two of the above factors, distance to be moved and target size. It was found that amplitude (A, distance to be moved) and target width (W) are related to time in a simple equation:

$$\text{Movement Time} = a + b \log \frac{2A}{W}$$

where a and b are constants. In this formulation, an index of movement difficulty is manipulated by the ratio of target width to amplitude and is given by:

$$\text{ID} = \log \frac{2A}{W}$$

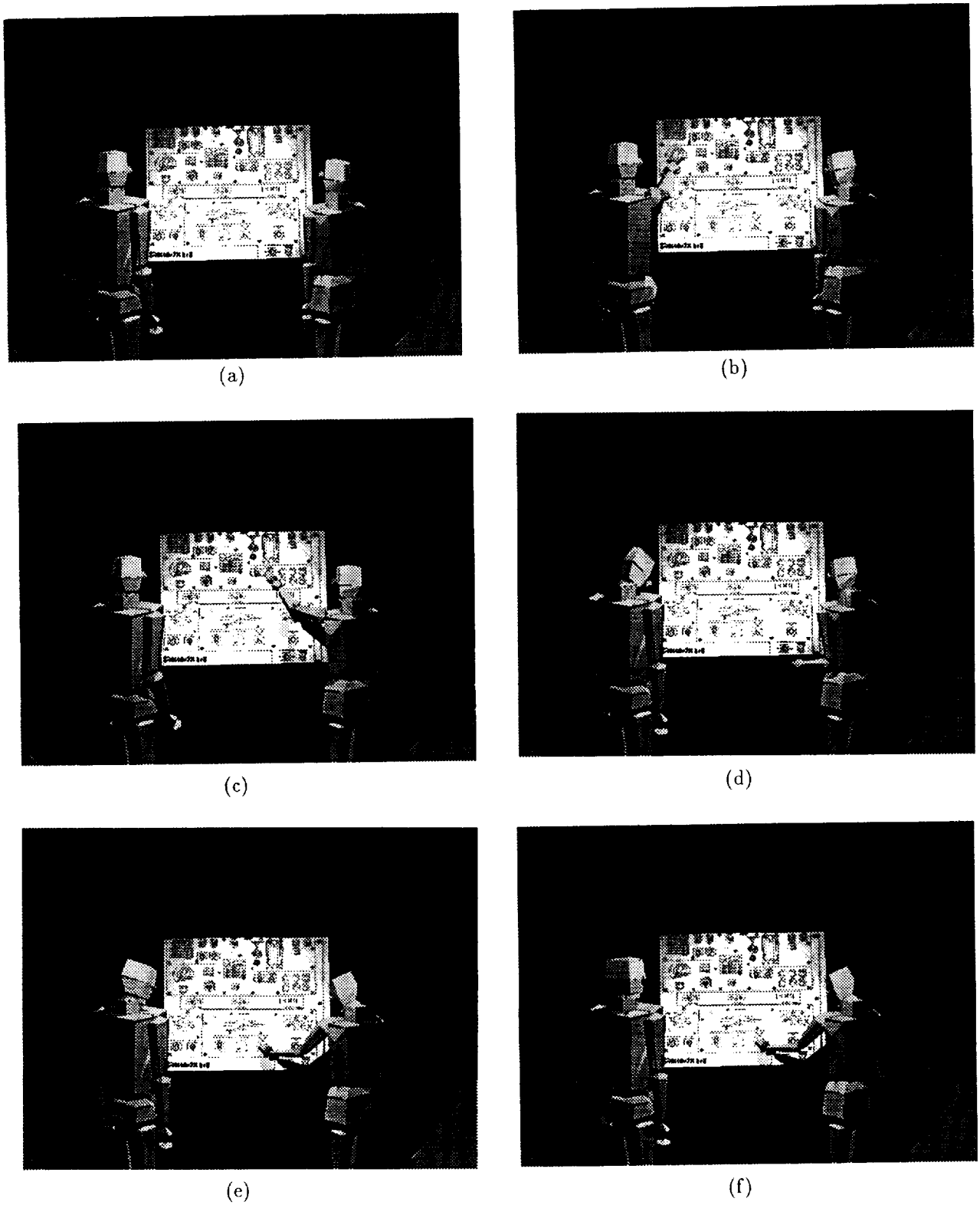
This index of difficulty shows the speed and accuracy tradeoff in movement. Since A is constant for any particular task, to decrease the performance time the only other variable in the equation W must be increased. That is, the faster a task is to be performed, the larger the target area and hence the movements are less accurate.

This equation (known as Fitts' Law) can be embedded in the animation system, since for any given reach task, both A and W are known. The constants a and b are linked to the other factors such training, desire, fatigue, and body segments to be moved; they must be determined empirically. For button tapping tasks, Fitts [13] determined the mean time (MT) to be

$$MT = 74ID - 70msec$$

Although Fitts' Law has been found to be true for a variety of movements including arm movements ($A = 5 - 30$cm) and wrist movements ($A = 1.3$cm) [9, 16, 18], the application to 3D computer animation is only approximate. The constants differ for each limb and are only valid within a certain movement amplitude *in 2D space*, therefore the extrapolation of the data outside that range and into 3 dimensional space has no validated experimental basis.

Nonetheless, Fitts' Law provides a reasonable and easily computed basis for approximating movement durations. Should a more exact model be developed, it should readily fit into a 3D computer animation environment in which default task durations must be computed.



(a) (b) (c) (d) (e) (f)

Figure 4: Animation Frames Showing "Look" and "Reach"

6 CONCLUSIONS AND FUTURE WORK

One of the goals of the Computer Graphics Research Lab at the University of Pennsylvania is to develop human task performance analysis tools specifically for users who are engineers and **not** particularly likely to be animators. Higher-level animation tools are deemed essential to the satisfaction of this goal. We have demonstrated the feasibility of building a complete pipeline of processes beginning with natural language input, proceeding through semantic resolution of simple tasks, default task time durations, and object references, and ultimately terminating in inverse kinematic positioning and rendered graphics. The pipeline confronts the issues of establishing appropriate linkages between objects, time, and actions at the language and geometric levels without adopting *ad hoc* solutions such as the selection of pre-defined key frames or the use of fixed default timings.

Of course, the model is quite incomplete in many respects, but we have work in progress in many areas, including:

- Extending the knowledge base to more complex task verbs and more general object environments.
- Extending the animation interface to include dynamics and constraints as well as inverse kinematics.
- Extending the task processor to a more general task simulator which handles temporal expressions, resource management, and task interruption.
- Extending the panel editor to permit on-line changes to panel object locations and semantics.

Ultimately the user should be able to control most of aspects of the animation (excepting the creation of the actual geometric environment) through a language-based interface. This will include the ability for parameterizing (1) bodies, (2) object and object feature locations, and (3) tasks. With this capability, experiments can be performed without descending to the key frame level for animation.

7 Acknowledgements

Many people helped developed the software described in this paper: especially Jean Griffin, Cary Phillips, Aamer Shahab, and Jianmin Zhao.

This research is partially supported by Lockheed Engineering and Management Services, Pacific Northwest Laboratories B-U0072-A-N, the Pennsylvania Benjamin Franklin Partnership, NASA Grants NAG-2-426 and NGT-50063, NSF CER Grant MCS-82-19196, NSF Grants IST-86-12984 and DMC85-16114, and ARO Grant DAAG29-84-K-0061 including participation by the U.S. Army Human Engineering Laboratory.

References

[1] *Man-system integration standards*, NASA, nasa-std-3000 edition, March 1987.

[2] *Space Shuttle Flight Data File Preparation Standards*, Flight Operations Directorate, Operations Division, NASA Johnson Space Center, 1981.

[3] Allen, James, *Natural Language Understanding*, Benjamin/Cummings, 1987.

[4] Badler, N., *A representation for natural human movement*, Technical Report MS-CIS-86-23, Dept. of Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1986.

[5] Badler, N. and Gangel, J., Natural language input for human task description, In *Proc. ROBEXS '86: The Second International Workship on Robotics and Expert Systems*, Instrument Society of America, June 1986.

[6] Badler, N., Manoochehri, K., and Walters, G., Articulated figure positioning by multiple constraints, *IEEE Computer Graphics and Applications*, Vol. 7, No. 7, June 1987.

[7] Barzel, R. and Barr, A., A modeling system based on dynamic constraints, *Computer Graphics*, Vol. 22, No. 22, 1988.

[8] Cebula, D., *The Semantic Data Model and Large Information Requirements*, Technical Report MS-CIS-87-72, Dept. of Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1987.

[9] Drury, C., Application of Fitts' Law to foot-pedal design, *Human Factors*, Vol. 17, No. 17, 1975.

[10] Fishwick, P., The role of process abstraction in simulation, *IEEE Trans. Systems, Man, and Cybernetics*, Vol. 18, No. 18, Jan./Feb. 1988.

[11] Fishwick, Paul A., *Hierarchical Reasoning: Simulating Complex Processes over Multiple Levels of Abstraction*, PhD thesis, Dept. of Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1986.

[12] Fitts, P., The information capacity of the human motor system in controlling the amplitude of movement., *Journal of Experimental Psychology*, Vol. 47, No. 47, 1954.

[13] Fitts, P. and Peterson, J., Information capacity of discrete motor responses, *Journal of Experimental Psychology*, Vol. 67, No. 67, 1964.

[14] Girard, M. and Maciejewski, A., Computational modeling for the computer animation of legged figures, *Computer Graphics (Proc. SIGGRAPH 85)*, Vol. 19, No. 19, 1985.

[15] Grosso, M. and Quach, R., *Anthropometry for Computer Graphics Human Figures*, Technical Report, Dept. of Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1988.

[16] Jagacinski, R. J. and Monk, D. L., Fitts' Law in two dimensions with hand and head movements, *Journal of Motor Behavior*, Vol. 17, No. 17, 1985.

[17] Karlin, R., *SEAFACT: A semantic analysis system for task animation of cooking operations*, Master's thesis, Dept. of Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, December 1987.

[18] Langolf, G. D., Chaffin, D. B., and Foulke, J. A., An investigation of Fitts' Law using a wide range of movement aplitudes, *Journal of Motor Behavior*, Vol. 8, No. 8, 1976.

[19] Magnenat-Thalmann, N. and Thalmann, D., MIRANIM: An extensible director-oriented system for the animation of realistic images, *IEEE Computer Graphics and Applications*, Vol. 5, No. 5, October 1985.

[20] Reynolds, C., Computer animation with scripts and actors, *Computer Graphics (Proc. SIGGRAPH 1982)*, Vol. 16, No. 16, 1982.

[21] Reynolds, Herbert M., The inertial properties of the body and its segments, *NASA Reference Publication 1024: Anthropometric Source Book*, Vol. 1, No. 1.

[22] Takashima, Y., Shimazu, H., and Tomono, M., Story driven animation, *Proc. of Computer Human Interface and Graphics Interface*, 1987.

[23] Wilhelms, J., Toward automatic motion control, *IEEE Computer Graphics and Applications*, Vol. 7, No. 7, April 1987.

[24] Witkin, A., Fleisher, K., and Barr, A., Energy constraints on parameterized models, *Computer Graphics*, Vol. 21, No. 21, 1987.

[25] Zeltzer, D., Motor control techniques for figure animation, *IEEE Computer Graphics and Applications*, Vol. 2, No. 2, September 1982.

[26] Zeltzer, D., Towards an integrated view of 3-D computer animation, *Proc. Graphics Interface '85*, 1985.

SES CUPOLA INTERACTIVE DISPLAY DESIGN ENVIRONMENT

Bang Q. Vu & Kevin R. Kirkhoff

Lockheed Engineering & Sciences Company

INTRODUCTION

The Systems Engineering Simulator, located at the Lyndon B. Johnson Space Center in Houston, Texas, is tasked with providing a real-time simulator for developing displays and controls targeted for the Space Station Freedom. These displays and controls will exist inside an enclosed workstation located on the space station. The simulation is currently providing the engineering analysis environment for NASA and contractor personnel to design, prototype, and test alternatives for graphical presentation of data to an astronaut while he performs specified tasks. A highly desirable aspect of this environment is to have the capability to rapidly develop and bring on-line a number of different displays for use in determining the best utilization of graphics techniques in achieving maximum efficiency of the test subject fulfilling his task.

The Systems Engineering Simulator now has available a tool which assists in the rapid development of displays for these graphic workstations. The Display Builder was developed in-house to provide an environment which allows easy construction and modification of displays within minutes of receiving requirements for specific tests.

SOFTWARE DESIGN

Program Structure

The Display Builder is compiled to run under UNIX AT&T System V on a Silicon Graphics' IRIS 4D/70 GT graphics workstation. It has fourteen modules, and nearly 11,500 lines of C source codes. Four modules are dedicated entirely to 2-Dimensional (2D) graphics; four are dedicated to 3-Dimensional (3D) graphics, and the rest to user interface and display list management. The executable size is roughly 400K bytes.

Data Structure

The display list is implemented as a doubly linked list. Each node in the list contains various house-keeping data as well as an union (set) of all structures (records) representing 2D and 3D primitives. The advantage of using the union feature of C is that although the primitives have variable length, they all fit into a node, thus greatly simplifying the task of data management.

USER INTERFACE DESIGN

Any software, especially an interactive graphics application system, is often judged primarily on its success to deliver its functionalities to the users. Even if the system is computation-intensive, what good is it if it fails to communicate effectively with the human operator? In the worst case the acceptability of the whole program may be invalidated because even the experienced users shy away from a poorly-designed User Interface (UI). The Display Builder employs a direct-manipulation UI popular on many modern interactive graphics systems.

UI Design Strategy

The Display Builder's UI lets the user manipulate objects directly on the screen. This type of interface is popular because it is easy to learn, and easy to use; however, it is also one of the most difficult to implement. Direct-manipulation UI is often complex due to stringent performance requirements (rapid actions and feedbacks), elaborate graphics, asynchronous input devices, and various ways to give the same command (keyboard and mouse). Currently the most successful strategy to create a reliable UI is the Iterative Design method. Under this method, prototypes of the Display Builder were iteratively tested and modified based on users' comments to generate the final UI.

Command Language

Command languages appear in all computer systems. A command language is the set of actions a user is allowed to have and the methods through which he can request a particular action. In designing the Display Builder's command language, the following issues were resolved:

Command Style: The Builder is both keyboard-dialogue and menu driven. Dialogues are of a very simple nature and conducted inside a dialogue box. For example, the Builder may prompt for a string or a number, in which case the user would proceed to type in the requested data. It may ask for an answer, such as "yes" or "no", in which case the choices are presented and can be selected. Most of the time, the Builder is menu driven. Selecting either an answer from the dialogue box or a choice from a menu can be done by the mouse or keyboard shortcut (see FIG 1,2,3).

Command Modes: The builder has a dual mode, 2D and 3D, command language, and therefore interprets user actions in two different ways. For instance, if the Builder is in 2D mode and the user requests to move an object, he can move only 2D objects. The user switches from one mode to the other via the main menu (see FIG 2). The dual mode method is chosen because it improves the UI by cutting down the size of many menus, and contributes heavily to program maintainability by keeping modules handling 2D and 3D graphics separate.

Command Abort & Error Handling: These are usually the most critical areas of any interactive system because of the presence of an unpredictable human in the process. The Builder allows the user to abort any single- or multi-step command by pressing the ESC key. In case of user error, such as opening a nonexistent file, a beep would

sound, the file name together with an error message appears in the dialog box and the user can then correct the error or abort the command. Because the Quit action irretrievably destroys the drawing, it is implemented as a two-stage command. If the user chooses this action, the following warnings occur: a beep would sound, and a prompt appears in the dialog box. Then the user can abort the command or confirm it by clicking the mouse's right button.

Information Displays

This is one of the most subjective and therefore troublesome areas of UI design: how to display information in the most "effective" manner. Screen layout and object display are the two items of interest.

Screen Layout: The prompts for the universal command abort key (ESC), current file name, current drawing color, and current builder mode (2D/3D) are displayed in the upper left corner. The dialog box is located in the upper right corner (see FIG 1). The drawing area can be created and moved to any place on the screen. Menus appear on the right side of the screen only when needed.

Object Display: The Display Builder makes no distinction between 2D and 3D objects; they coexist in the same drawing space. The user can allocate any area of the screen, either same or separate, to 2D and 3D objects. All objects will automatically be clipped to fit inside its allocated space.

Interactive Graphical Input Techniques

The user interacts with the computer through a graphical display. He is generally not skilled in graphics, and only interested in how fast and easy it is to accomplish his task. Interactive techniques reduce the need for great manual dexterity and the effort required to draw and manipulate objects visible on the screen. Feedback, Selecting, and Positioning are the techniques of interest.

Feedback: This is an essential component of graphical interaction. Feedback techniques help to provide immediate confirmation to the extent or intention of the user's action. For instance, if the user selects multiple objects to delete, and no feedback is provided, the user is left to wonder whether he has made the right selections. The uncertainty will eventually be answered when he gives the delete command, but the efficiency of interaction is severely hampered. The Builder uses highlighting, bounding box, blinking color, prompts, and beeping noise to provide feedbacks to the user.

Selecting: the need to select primitive(s) to manipulate is one of the most basic interactive graphics techniques. The Builder supports different selection techniques depending on the mode it is in.

2D selecting: The user can select one or many 2D primitives unambiguously by pointing to and clicking on specific spots located on these primitives. These selection points appear only when required in logical places such as the center of a circle or the bottom left corner of a button (see FIG 5). Feedback is provided by immediate highlighting of selected selection points.

3D selecting: The user can select one or many 3D primitives by clicking on their names in a linear display list which appears when required. Feedback is provided by immediate blinking of selected primitive(s).

Positioning: The Builder supports the following techniques to position a new 2D primitive or relocate an existing one: grid, rubber banding, dragging, and aligning.

Grid: A grid is provided. The user has the option of choosing a background or foreground grid or no grid at all.

Rubber Banding: Most 2D primitives are created with rubber bands. For example, the user defines the first point of a line. As he moves the mouse, the Builder draws a line from the first point to the current cursor position. When the second point is defined by the second click, the rubber band disappears, and the permanent line is drawn. This technique takes the guess work out of positioning a new primitive as the user can see instantly how large and where it is on the screen.

Dragging: This technique is used with moving or copying 2D primitives. For example, the user starts by selecting the primitive(s) that he wants to relocate. Then he can "drag" their outlines to the new location. The selected primitives are permanently moved when the user defines the final location by clicking the mouse.

Aligning: There are four ways to align 2D primitives: left, right, top, and bottom. All selected primitives are automatically aligned.

3D graphics does not lend itself well to the above techniques. Currently, the only way to position a new 3D primitive or reposition an existing one is to enter new data via the dialog box. After the object has been completely defined, it will be redrawn at the new location.

FUNCTIONAL CAPABILITIES

Anything drawn inside a display is done through a primitive, for example, a line, a number or a button. A primitive is a packet of data used to construct a numerical or graphical representation of information in a display. This data is stored in the display's data file and used by the real-time software to draw each primitive. The following terms will be used to describe some features of the primitives:

SYSID: An index into a block of shared memory that interfaces between the display and the simulation.

THRESHOLD: Used to denote a caution state (yellow), and critical state (red). Thresholds are optional within each primitive.

TRANSITION STATE: In the switch primitive, the time between the user requesting a switch be activated, and getting an indication from the math model that the switch has been activated.

2D Actions & Primitives

The user can create 2D primitives in a display with minimal effort. For instance, he builds a circle by pointing to a spot on the screen where it will reside, defining the center with the first click of the mouse, rubberbanding the circle to the desired size, terminating with another mouse click. Data for that primitive may be edited at that time, or left until the entire display is created.

The following actions can be performed on 2D primitives:

Edit	change any or all data of a selected primitive. (See FIG. 5).
Delete	remove selected primitive(s).
Move	move selected primitive(s).
Copy	duplicate selected primitive(s).
Align	top, bottom, left or right justify selected primitive(s).
Save	save selected primitive(s) to a disk file.
Read	read primitive(s) from a disk file and insert them into the display list.

2D Primitive List (see FIG 3):

Header - Contains display size, font and the length, in bytes, of the display file.
Integer - Contains thresholding.
Single Precision Real - Contains thresholding.
Double Precision Real - Contains thresholding.
Hexidecimal - Contains thresholding.
Ascii message - Shows eight characters of variable text.
Static text - Shows eight characters of fixed ascii text.
Button or Switch - May be a toggle or momentary switch. These primitives are similar in function and makeup except that a momentary switch is activated when the mouse button is pressed, and de-activated when released.
Keyboard input - Allows the user to enter data into the simulation from the keyboard.
Page call - Activates new displays.
Default - Allows a user to customize a screen layout .
Indicator - Reflects the state of its related sysid.
Circular gauge - These come in two types, increasing and decreasing, and three sizes. A needle moves between the upper and lower limits in a forty-five degree, six o'clock to three o'clock pattern. The actual value is displayed in the lower right quadrant of the gauge. Contains thresholding.
Meter bars - Dynamically sized by the user, and may be horizontal or vertical. They are rectangular with a cyan bar and are functionally similar to the gauges. Contains thresholding.
Dynamic position indicator - A pointer which moves, horizontally or vertically, on a bar, proportionally between an upper and lower limit. This pointer may be designated as a caret, cross-hair, and filled or empty square, circle, triangle.
Line - Explicit in primitive name.
Circle - May be filled or empty.
Rectangle - May be filled or empty.
Polygon - Can have a maximum of ten vertices and be empty or filled.

3D Actions & Primitives

There are many ways to construct 3D objects; for example, one can build an image of a car by connecting different surfaces, or an image of a house by using various primitives such as boxes or cylinders. The Display Builder lets a user construct a 3D object by combining 3D wireframe primitives together using data entered through the keyboard.

The following actions can be performed on 3D primitives:

Edit	change any or all data of a selected primitive.
Delete	remove selected primitive(s).
Copy	duplicate selected primitive(s).
Save	save selected primitive(s) to a disk file.
Read	read primitive(s) from a disk file and insert them into the display list.

3D Primitive List (see FIG 6): There are two categories of 3D primitives: Graphic and Control.

The following graphic primitives are visible on the screen:

3D line - Must specify x,y,z coordinates.

Box - Must specify x,y,z coordinates.
Cylinder - Must specify center, diameter, length, number of wireframe lines and angle of rotation (used for defining partial cylinders).
Sphere - Must specify center, radius and number of wireframe lines.

The following control primitives position and orient a 3D object:

Header - Contains distance of the eyepoint from the origin, near and far clipping planes, angle of perspective, and whether or not the display will be z-buffered.
Begin object frame - Contains sysids for securing state vector data for real-time positioning of an object.
End frame - end of object for state data application.
Ref Frame - define the coordinate system used to build a 3D object.
Rotation - Must specify the axis of movement, and degree of movement per pass of the real-time software.
Translation - Must specify the axis of movement, and distance of movement per pass of the real-time software.
Scale - Must specify the axis of movement, and distance/degree of movement per pass of the real-time software.

CONCLUSION

Because of the highly developmental nature of the simulation workstation displays, it was essential that a display builder be created that has the capability to quickly create and modify a display, for evaluation in the simulation, in a short amount of time. This "rapid prototyping", along with a conscious effort to design and write software that is easy to maintain and add prototypes as needed, makes the Display Builder a useful and essential tool in the generation and modification of displays for use in the SES.

ACKNOWLEDGEMENTS

We would like to thank the following people who have made significant contributions over the last two years to the Display Builder: Judy Murphy, Mike Red, and Mike McFarlane.

REFERENCES

1. Foley, J. D., and Van Dam, A., Fundamentals of Interactive Computer Graphics, Addison Wesley, 1982.

2. Linton, M. A. et al., "Composing User Interfaces with Interviews," IEEE Computer, Vol. 22, No. 2, February 1989, pp. 8-22.

3. Myers, B. A. , "User-Interface Tools: Introduction and Survey," IEEE Software, Vol. 6, No. 1, January 1989, pp. 15-23.

4. Newman, W. M., and Sproull, R. F., Principles of Interactive Computer Graphics, McGraw-Hill, 1979.

5. Peltz, D. L., "3-D in Perspective," MacWorld, Vol. 5, No. 12, December 1988, pp. 108-119.

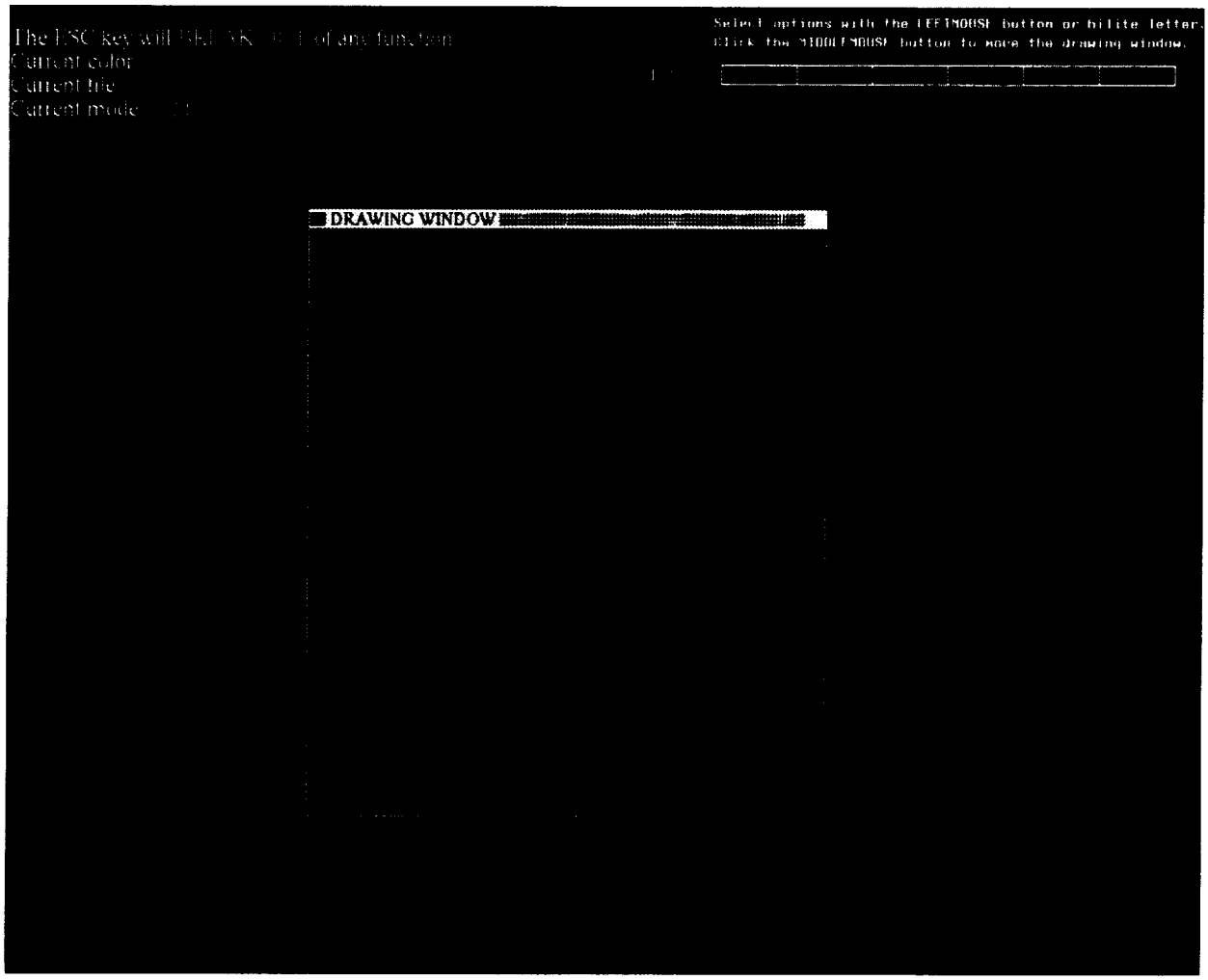


FIG. 1 SCREEN LAYOUT

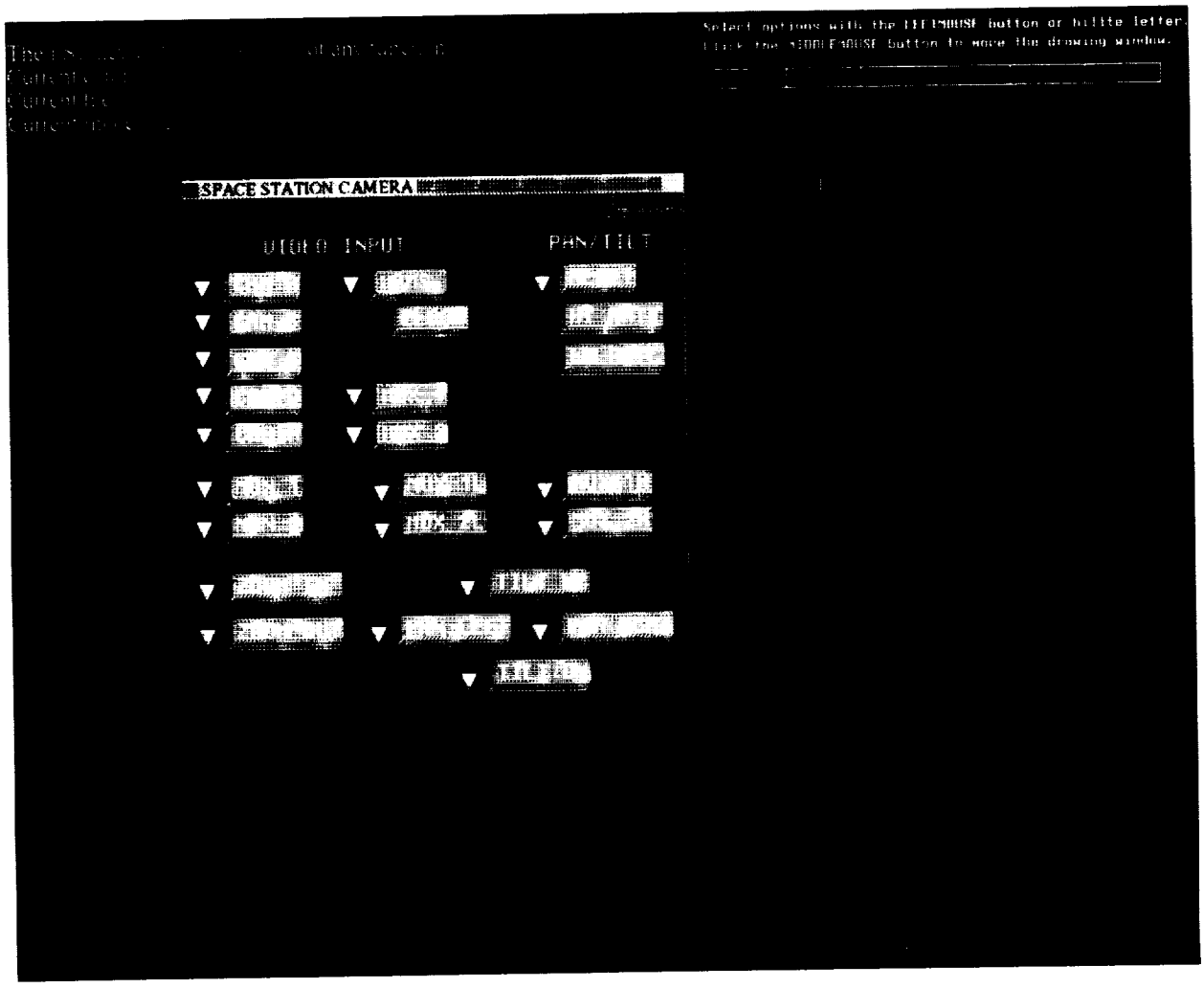


FIG. 2 DIALOG BOX WITH MAIN MENU

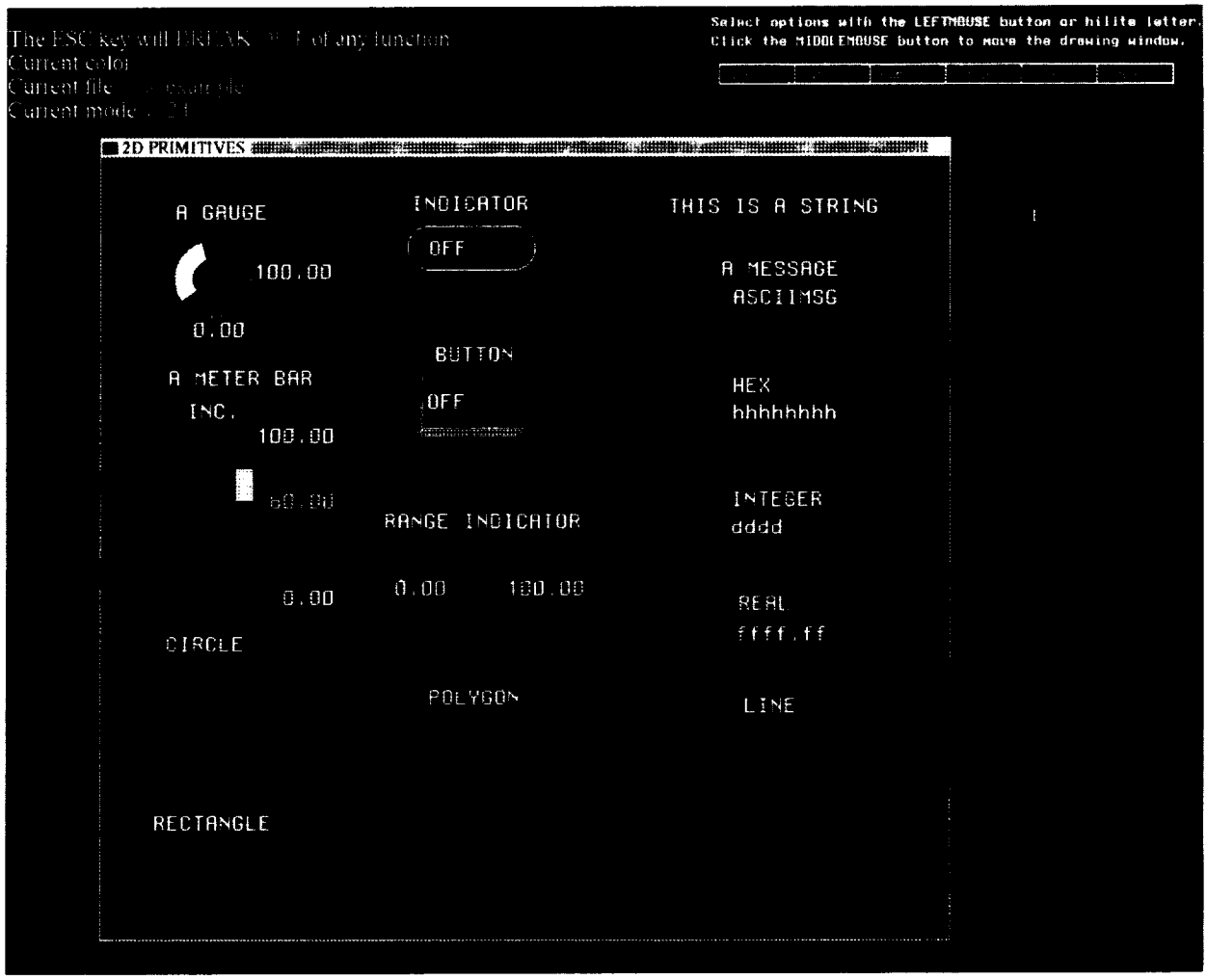


FIG. 3 2D PRIMITIVES

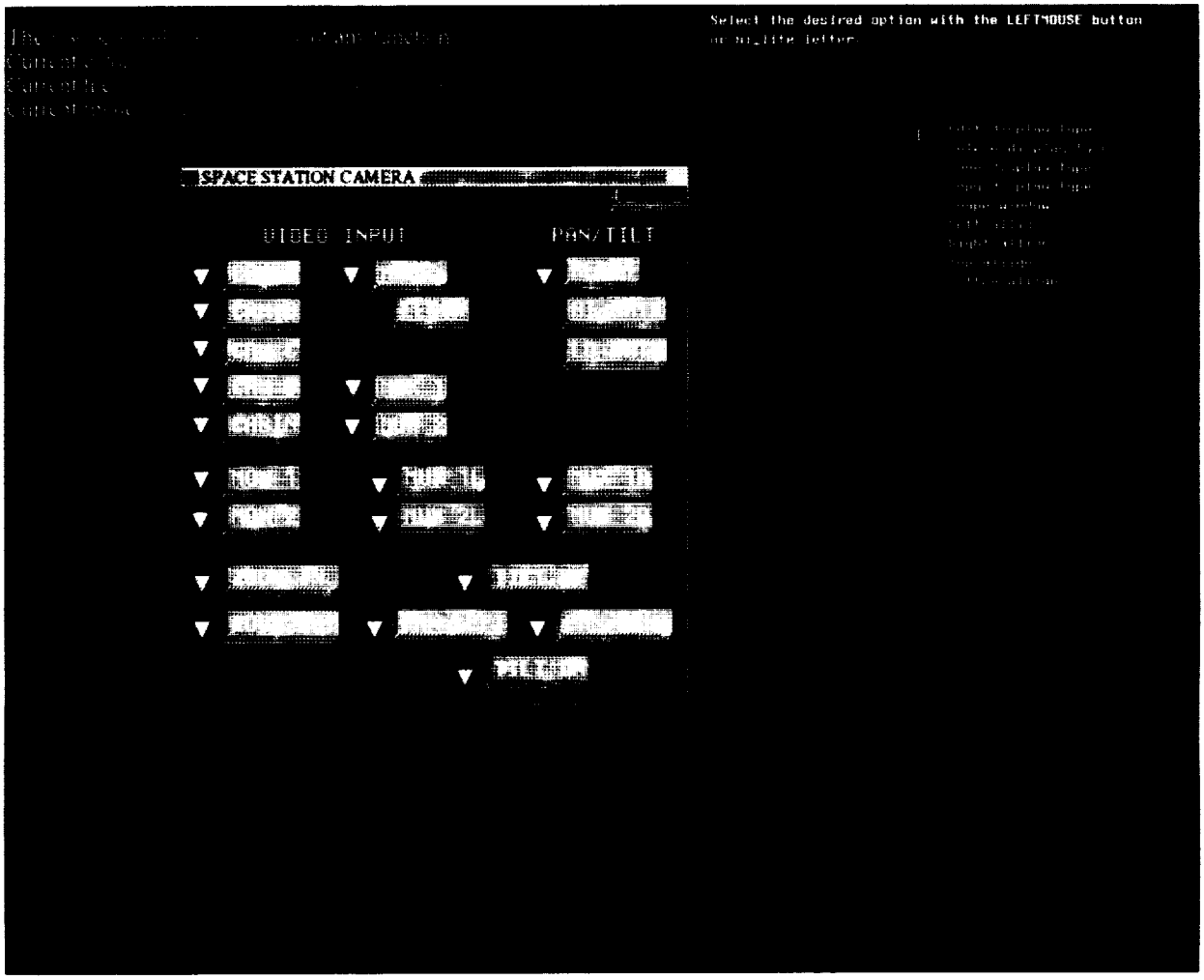


FIG. 4 2D ACTIONS

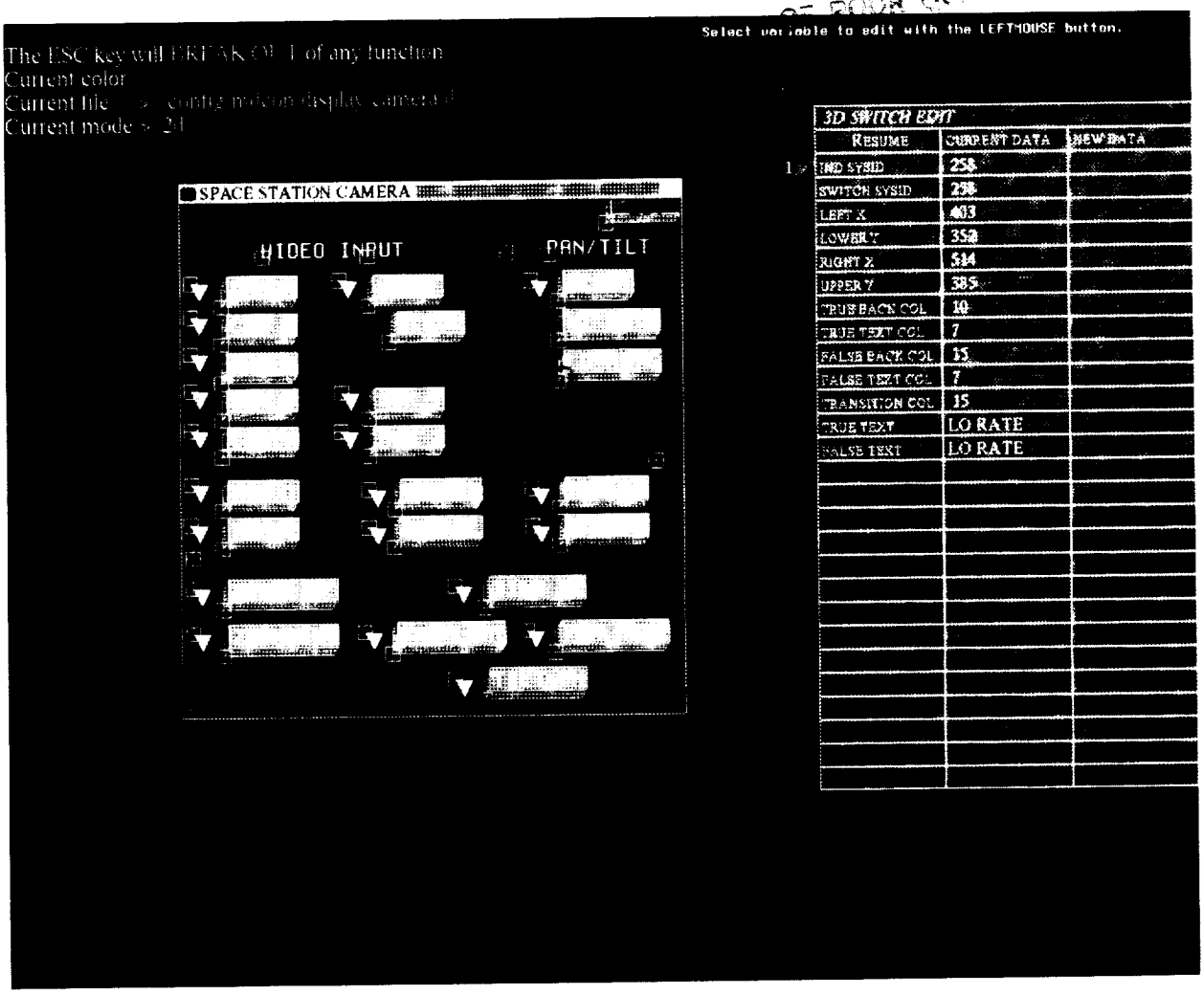


FIG. 5 2D EDITING

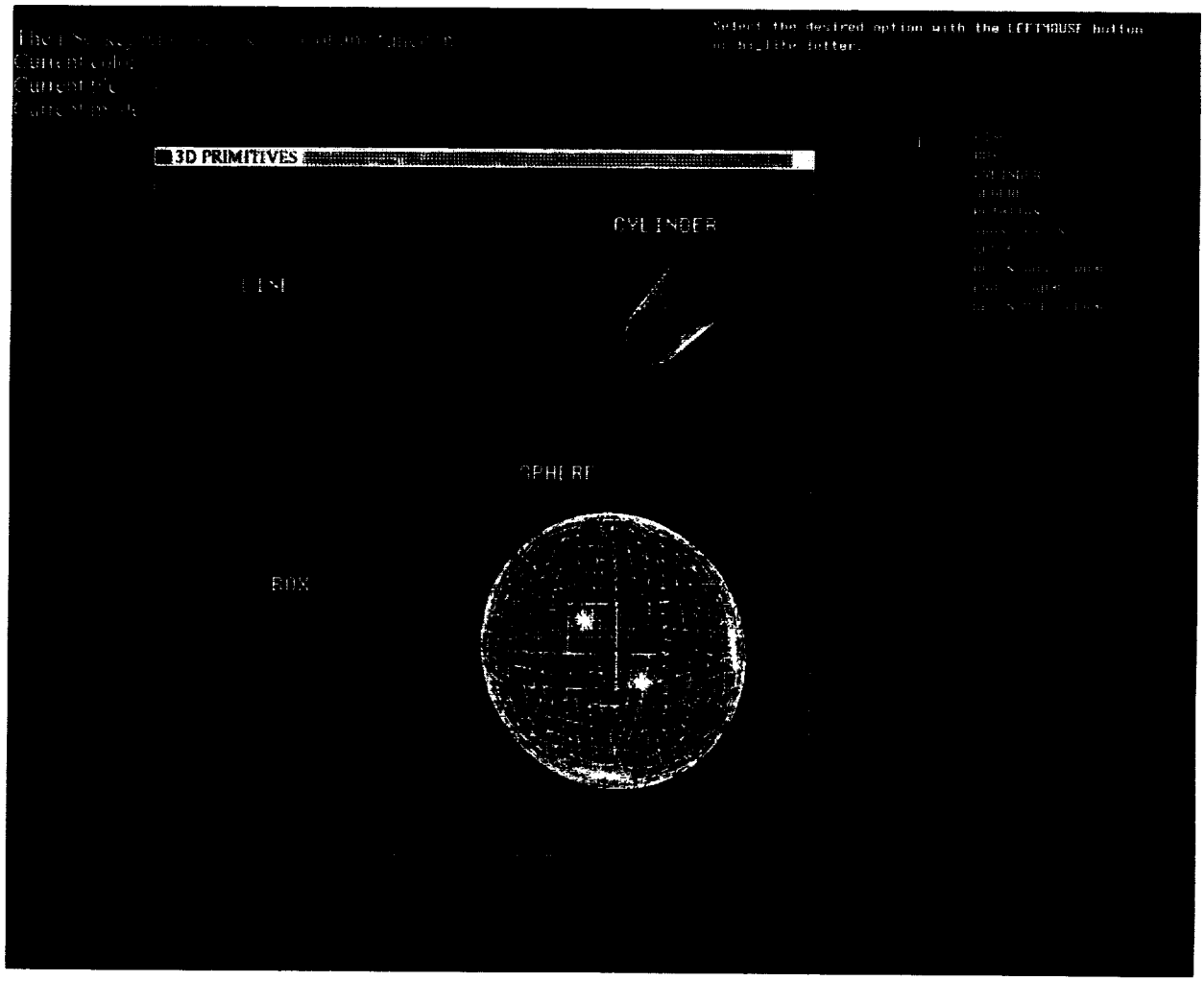


FIG. 6 3D PRIMITIVES

ISSUES IN VISUAL SUPPORT TO REAL-TIME SPACE SYSTEM SIMULATION SOLVED IN THE SYSTEMS ENGINEERING SIMULATOR

Vincent K. Yuen
Lockheed Engineering & Sciences Company
2400 NASA Road 1
Houston, Texas 77058

INTRODUCTION

In some man-in-the-loop simulations, immediate visual feedback is essential in providing the astronauts with the real world representation of their operating environment in order to successfully complete the designed task. This is especially true in space station/space shuttle docking (Figure 1) and space station/space shuttle payload hand-off (Figure 2) scenarios. More importantly, when a remotely piloted vehicle is not equipped with radar sensors to provide data describing relative motion, the astronaut has to rely entirely on visual inputs to perform his functions. Such maneuvers are impossible without the aid of adequate visual cues.

Adequate visual coverage, the field of view provided, is also of paramount importance. The visual coverage not only provides guidance for the particular maneuvers, it also dictates the feasibility of the maneuvers themselves. Due to the complex geometrical shapes of the vehicles and their attachments, together with the number of moving parts involved, collisions between parts can happen quite inadvertently. These collisions may go unchecked if visual coverage is not available to give immediate feedback.

A third aspect of the problem is providing all the pertinent views to all participants in the simulation. In scenarios involving the Space Shuttle and Space Station working in concert there may be upwards of ten window

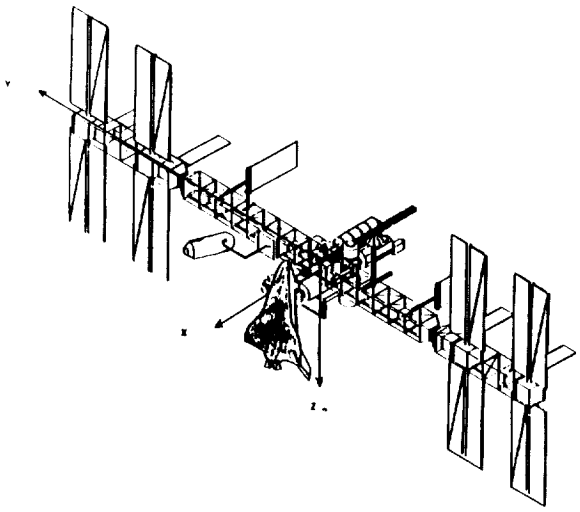


Figure 1 - Space Station Freedom / Space Shuttle Docking

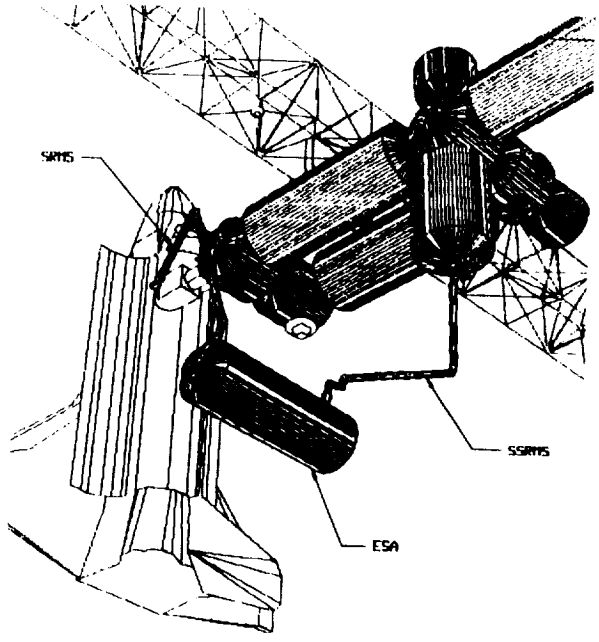


Figure 2 - Space Station Freedom / Space Shuttle Payload Hand-Off

views and seven closed-circuit-television (CCTV) monitors in use by the astronauts during one simulation.

Thus, the issues to be addressed in providing visual systems to man-in-the-loop simulations are : 1). providing real world representation for window and CCTV views, 2). providing adequate visual coverage (field of view) to adequately complete the task, 3). providing as many views to as many participants as possible during the simulation.

LABORATORY BACKGROUND

The Systems Engineering Simulator (SES) located at the Lyndon B. Johnson Space Center in Houston, Texas, has successfully addressed these issues in the development of its simulator complex which provides real time man-in-the-loop simulations for the National Space Transportation System Space Shuttle program and the Space Station Freedom program. The SES provides two crew stations for manned operation of the vehicles in the simulation. A Space Station Freedom workstation mockup is provided and a replication of the Space Shuttle aft flight deck is provided.

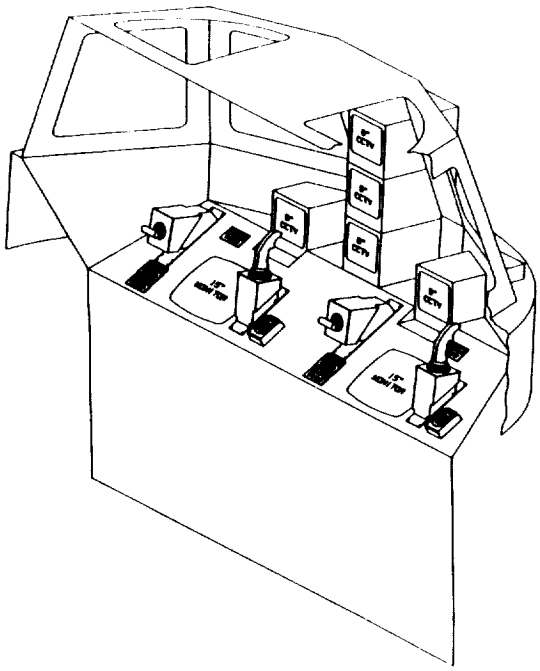


Figure 3 - Space Station Freedom Control Mockup

There are six windows and five camera monitors in the Space Station Freedom control mockup (Figure 3) and the Space Shuttle aft cockpit mockup has four windows and two camera monitors (Figure 4). Each window view is a virtual image projection of a two-dimensional scene projected by a television monitor. The virtual image will give the window view a three-dimensional perspective. Camera views are provided with pan/tilt/zoom capabilities, and the scene graphics are projected onto television monitors. This arrangement of different capabilities between window eye points and camera eye points is the design to map the visual simulation to the real world where out-the-window views are three-dimensional and camera views are two-dimensional.

The laboratory has three different scene generators providing a total of eleven channels of video signal. An Evans & Sutherland CT-6 visual generation system provides six channels, an Evans & Sutherland CT-3 system provides two channels, and a Redifusion Poly 2000e visual system provides three channels of video signal.

There are obviously more window and camera views than the eleven channels can support. A video distribution system has been developed to handle the video signal allocation and switching in a time-sharing fashion to provide video to windows and CCTVs as selected by the simulation operators. The system has two major components. A Scene Select subsystem allows simulation users to select which video channel is to be displayed on which window or CCTV. The other component is the Video Distribution Rack (VDR) which is responsible for routing the video signals from the three scene systems to the windows and CCTVs in the mockups.

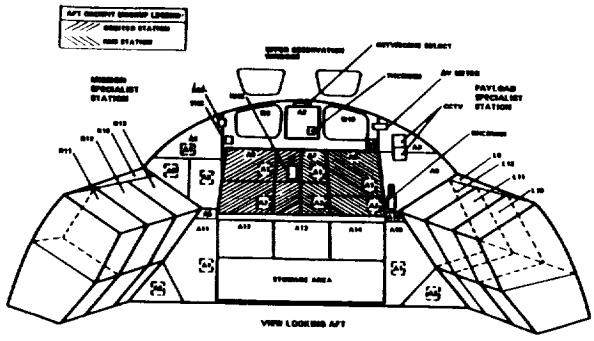


Figure 4 - Space Shuttle Aft Cockpit Mockup

VISUAL SUBSYSTEM

There are five major pieces of visual scene software needed to provide the visual simulations for the SES: Scene Drive, CT3 Interface, Poly Interface, CT6 Interface and the Scene Devices Controller (Figure 5).

Scene Drive

This element provides the mathematics between the simulation and the scene systems. The Scene Drive takes inertial state vectors of the vehicles, computes the relative state vectors, and passes them onto the scene system interfaces.

An equally important issue in the visual simulation, besides the positioning of the objects, is the positioning of the eye points. Scene Drive receives inputs that indicate which eye points are currently selected and whether pan/tilt/zoom has been commanded for camera eye points. Armed with the users' requests of monitors and the availability of video channels, the Scene Drive task determines which scene system video output channels should be routed to which television monitors for display.

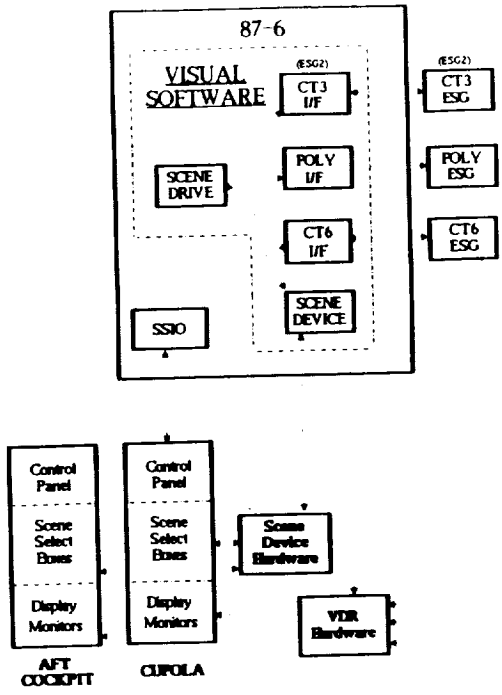


Figure 5 - Visual Subsystem Block Diagram

Scene Devices

The Scene Devices task is a background task supporting the simulations. It receives user requests for monitors from the Scene Select hardware and passes the requests to the Scene Drive task. It then receives the VDR and Scene Select commands from the Scene Drive tasks and passes the commands to the VDR hardware and Scene Select hardware respectively for hardware distribution of scene system outputs.

CT3, POLY & CT6 Interfaces

These three interface tasks receive relative state vectors of vehicles, eye points from the Scene Drive task, and data from the other parts of the simulation to drive the objects in the visual scene. Each of the tasks is the only link between the simulation and the corresponding scene system.

COVERAGE

Visual coverage is a major concern for SES Crew Stations. The coverage for the simulator is determined by the field-of-view of the real-world windows. Ideally, the simulator should provide the exact field-of-view to the astronaut as is available to him in the actual vehicle. The Space Station Freedom workstation creates an interesting problem in that there is almost a continual view across windows. The SES has developed a rotating optics system to fit around the Space Station Freedom workstation to provide this continual field of view (Figure 6). The optics system can rotate in a vertical manner to provide coverage overhead when necessary. This rotating capability allows the astronaut to select his coverage as a function of his interested area outside the workstation.

SUMMARY

The Systems Engineering Simulator has addressed the major issues in providing visual data to its real-time main-in-the-loop simulations. Out-the-window views and CCTV viewsare provided by three scene systems to give the astronauts their real-world views. To expand the window coverage for the Space Station Freedom workstation a rotating optics system is used to

provide the widest field of view possible. To provide video signals to as many viewpoints as possible, windows and CCTVs, with a limited amount of hardware, a video distribution system has been developed to time-share the video channels among viewpoints at the selection of the simulation users.

These solutions have provided the visual simulation facility for real-time man-in-the-loop simulations for the NASA space program.

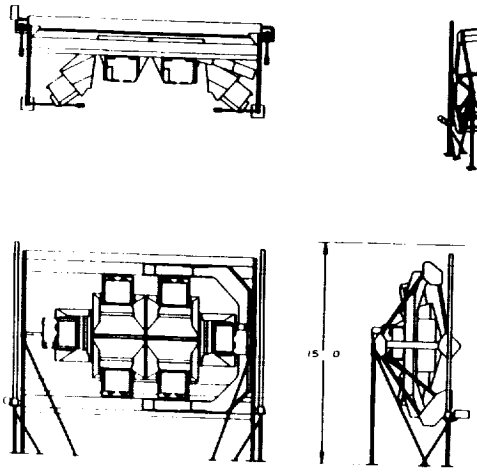


Figure 6 - Rotating Optics System

HISTORY OF VISUAL SYSTEMS IN THE SYSTEMS ENGINEERING SIMULATOR

David C. Christianson
Johnson Space Center
National Aeronautics and Space Administration
Houston, Texas 77058

ABSTRACT

The Systems Engineering Simulator (SES) houses a variety of real-time computer generated visual systems. The earliest machine dates from the mid-1960's and is one of the first real-time graphics systems in the world. The latest acquisition is the state-of-the-art Evans and Sutherland CT6. Between the span of time from the mid-1960's to the late 1980's, tremendous strides have been made in the real-time graphics world. These strides include advances in both software and hardware engineering.

The purpose of this paper is to explore the history of the development of these real-time computer generated image systems from the first machine to the present. Hardware advances as well as software algorithm changes are presented. This history is not only quite interesting but also provides us with a perspective with which we can look backward and forward.

ACRONYMS AND ABBREVIATIONS

CIG --- Computer Image Generator
CT3 --- Continuous Tone Computer Image Generator - third generation
CT6 --- Continuous Tone Computer Image Generator - sixth generation
EPU --- Edge Processing Unit
ESG --- Electronic Scene Generator
JSC --- Johnson Space Center
MMU --- Manned Maneuvering Unit
NASA -- National Aeronautics and Space Administration
OGU --- Object Generating Unit
OMV --- Orbital Maneuvering Vehicle
R520 -- Raytheon 520
SAIL -- Shuttle Avionics Integration Laboratory
SEL --- Systems Engineering Laboratories
SES --- Systems Engineering Simulator
SGS --- Surface Generator Subsystem
TOU --- Timing and Output Unit
VCU --- Vector Calculating Unit

INTRODUCTION

In the twenty odd years between the first real-time computer image generator to the present, many strides have taken place to provide realistic, full color, three-dimensional displays for use in many areas in the simulation community. NASA/JSC is rare in that it contains a snap-shot of this development approximately every ten years.

From the first of its kind to the current, real-time computer image generators provide the necessary visual displays to support the increasingly heavy demands placed on the Systems Engineering Simulator (SES). This paper explores the history of the scene generators which existed and still exist in the SES.

The hardware configuration and new technology of each graphics system is explained. The salient features and innovations of each system as they were introduced to the SES is explored. Several advances in the theory of database modeling have evolved throughout the years and real-time programming has changed from minimal to extensive.

Due to historical carryover, the terms Electronic Scene Generator (ESG) and Computer Image Generator (CIG) will be used interchangeably.

SES -- SYSTEMS ENGINEERING SIMULATOR

The Systems Engineering Simulator (SES), formerly the Shuttle Engineering Simulator, has been in continuous operation since the programs conception in 1968. The SES supports real-time man-in-the-loop computerized engineering simulation for the Shuttle, space station, and other space related programs and projects.

The two main areas of operation utilizing Electronic Scene Generators (ESG) are entry and on-orbit.

The entry simulation is hosted by a Cyber 840. The orbiter forward cockpit mockup is located in the East High Bay of Building 16.

On-orbit simulation is accomplished with the use of five SEL 32/8780 supermini digital computers and four SEL 32/75 digital computers. Mockups include an orbiter aft station, MMU station, and a cupola station. The cupola is the operations station for the space station. All of these mockups reside in Building 16. On-orbit operations which are supported include space station docking/berthing, payload handling/deployment, MMU operations, and OMV studies to mention a few.

NASA I -- THE ORIGINAL SCENE GENERATOR

> "And, in the beginning, there was texture."

NASA/JSC was instrumental in bringing to fruition the concept of real-time computer generated images. In the time before the mid-1960's, the out-the-window visual images were generated by model boards: large, scaled replicas of the simulation terrain over which a closed circuit television camera traversed. These model boards were built specifically for the purpose at hand and not easily modifiable.

A new concept emerged in the early 1960's. Although rather idealized, the scenes produced by computers could be generated in real time to satisfy the requirements to provide scenes for out-the-window displays. In August, 1964, NASA at the Manned Space Center installed the first such computer device. The dawn of real-time computer generated images began with the "Visual Contact Analog: Three-View Interim Space-Flight Simulator" build by General Electric.

As intimated above, this computer system produced three views. These views consisted of an unbounded textured planar surface for the ground. This special purpose computer, the Surface Generator System, calculated the perspective transformation of a surface texture. The optical system displayed the resulting pictures so that the environment appeared distant to the observer.

There were a few interesting details to be discovered about this system. Due to the state of the digital art at that time, several problems were solved with analog methods. The textured surface was computed in digital form without a roll angle. This made the algorithm simpler and roll was accomplished in the circular television monitor by electronically rolling the raster. Precise nonlinear sweeps were generated by the display unit to compensate for optical distortions. To avoid disturbing moire patterns, fine detail was gradually faded out of the picture with analog circuitry.

The entire computer system consisted of six pipelined processors built with pre-TTL equipment: Computer Control Corporation (3C) circuitry cards containing discrete components.

Screaming along at 5 MHz, the displays were generated at 30 frames a second. This corresponds to the current American commercial standard.

As in all computer image generators, the first processor unit is the unit with the highest programmability. The first unit of the Surface Generator System, the Program Control Unit, contained 512 48-bit words and had a memory access time of 5 microseconds.

This machine served the Guidance and Control Division for several years.

NASA II -- THREE DIMENSIONAL CAPABILITY

In February, 1968, modifications were made to the Interim Visual Space Flight Simulator and a large complement of equipment was added.

An innovation occurred in the field of computer generated images. Three-dimensional objects were added to the textured surface. In this time frame, the Manned Space Center was heavily involved with moon landings. The additional capability provided the simulation with idealized forms of lunar mountains and craters as well as the traditional realistic landing fields.

The new system consisted of the following components:

1. A Raytheon 520 (R520) general purpose computer. Flexibility was introduced by linking a general purpose computer to a set of special purpose computers. Although minimal by modern standards, the memory capacity was 8096 24-bit words core memory and 256 24-bit words of high speed memory. This concept formed the basis of flexibility in the succeeding generations of real-time visual systems.

2. A Vector Calculating Unit (VCU). This special purpose computer deviated from the classical Von Neumann computer architecture. It contained a 4096

24-bit word program memory unit and three sets of 2048 30-bit word data memory units corresponding to the X, Y, and Z components of the Cartesian coordinate system. The data memory units can be accessed in parallel, enabling dot and cross product calculations to be made quickly.

3. A set to two Object Generating Units (OGU). Each unit was capable of working on twenty objects made up of eighty faces and described by a maximum of 120 edges. For each edge of the environment, one circuit board was required.

4. A Timing and Output Unit (TOU). This unit performed three functions. It generated the master timing signals for the entire system. The TOU served as a mixing and distribution point for the video outputs from the two OGU's and routed data from the VCU to the OGU's, SGS, and displays. Test patterns were generated in the TOU for aligning and trouble shooting the system.

The Surface Generating Subsystem (which was the Visual Three-View Space-Flight Simulator) was modified to allow its operations at 20 frames a second. The NASA II system operated at a slower rate due to the constraints imposed on it by the R520 and VCU.

Of interest here is the fact that the objects which were generated came from a catalog of two-dimensional polygons and three-dimensional objects. Each OGU had the capability of generating one decahedron, one octahedron, two hexahedrons, and four tetrahedrons as well as two dodecagons, three octagons, four hexagons, three quadrilaterals, and two triangles. The maximum capability of 120 edges per OGU could not be exceeded. The combined capacity of the two OGU's was 240 edges.

The database designer had to create a scene choosing objects and polygons from a catalog of available objects and polygons. The specification of vertices for the objects and polygons followed stringent rules. The concepts of planarity and convexity were required for each planar surface. Because object topology was predefined, vertex selection required a lot of precalculation for irregular objects. Selection of the objects and polygons amounted to filling specific absolute locations in the R520 memory. Color selection followed a similar procedure.

The designer also had the choice of one quadrilateral shadow polygon and one beacon. The shadow polygon emulated the shadow created by ones ownship. It changed configuration in response of the vehicles attitude with respect to the surface and an imaginery sun. The one beacon was a two element by one line pair dot. It had the capability of flashing and the period was programmable.

The real-time programming consisted mainly of calling the subroutines which transformed objects and polygons in the correct order. At this point in time, there was not a clear distinction between database design and real-time programming. The two concepts were closely intertwined.

The NASA II system, therefore, presented an environment consisting of three-dimensional objects on a two dimensional textured surface.

NASA III -- A BETTER WAY

In November, 1971, a major innovation was incorporated into the then current visual system. The two OGU's with its combined capacity of 240 edges was replaced with an Edge Processing Unit (EPU) which increased the edge capability to 320 edges. The theory of a fixed set of objects and polygons was superseded with a more general approach of just polygons. Groups of polygons were gathered to form three-dimensional objects.

A new concept was also introduced. Edges could now be shared between an object or among objects which did not move relative to each other. This provided an addition edge capacity capability. For example, a cube has six sides and each side has four edges. With this method, a cube could be described as six sides with twelve shared edges rather than six sides with four edges each for a total of 24.

With the added flexibility that this new system brought, a need for programs to generate databases was required. The first database compiler was written by Lockheed at the NASA Manned Spacecraft Center. Because the program was written in an early version of Fortran, the syntax was necessarily field sensitive. Things had to be in the right column.

The database designer specified clusters by grouping polygons. He specified polygons by grouping vertices. Clusters had to follow some rigid constraints. NASA III used the idea of separating planes. Clusters were separated from each other by invisible, infinite planes called nodes. Modules were groups of clusters which did not move relative to each other. Each polygon was given an attribute such as color, back-face generation, and shadow or beacon generation. Given the capacity of the machine, the number of edges per polygon was completely arbitrary. One of the

test patterns which existed on the machine has sixteen polygons of twenty edges each.

The real-time program, which resided in the R520 and the VCU, was written by General Electric. It was an upgraded version of the one which resided in the R520 in NASA II but the new algorithm made the program much less complex and easier to manage.

Programming of the real-time software was rather trivial. The program was mainly driven by the database environment. It required little modification for each new environment. The environment consisted mainly of two independent coordinate systems which could contain an eyepoint or modules, one coordinate system which could contain only an eyepoint, and the ever present textured surface.

With the advent of the CT3, described below, the NASA III system was renamed the Electronic Scene Generator #1 (ESG #1).

CT3 -- A MAJOR STEP

A major step in the evolution of real-time computer generated image systems was made in November, 1976. The CT3 made by Evans and Sutherland was introduced to the SES. This system had many new attributes which deserve mentioning. This system is currently employed to a great extent in the SES laboratory.

The CT3 consists of three general purpose computers of the PDP-11 series, a visual pipeline, and a collision detection pipeline.

A central PDP-11/40, called the HOST, is interfaced to the simulation laboratory. The HOST interfaces to the visual system and the collision detection system. The main purpose of this machine is to gather the data from the simulation, format the data, and send the results to the other subsystems.

The visual subsystem is driven by a PDP-11/40. It is connected to a visual pipeline containing 10 programmable special purpose processors. Two independent channels of visual images are produced. The total capacity of the visual system is 900 polygons.

A separate collision detection system allowed the simulation to detect the intersection of impenetrable objects. This system consisted of a PDP-11/45 as well as a collision detection pipeline containing two programmable special purpose processors.

The frame rate of this system was 25 Hz. This corresponds to the European commercial standard. Although ESG #1 was operating at 20 Hz, no problem was presented. CT3 was used for on-orbit studies and ESG #1 was not.

The visual system included many new features which are described below.

Anti-aliasing and edge smoothing were added to improve picture quality. Spatial filtering was used as the algorithm.

Directional illumination was introduced to provide an illusion of sun direction, intensity, and environmental depth.

Smooth-shaded polygons simulated round or complex shaped objects. The Gouraud shading algorithm was implemented in hardware using extremely fast ECL circuitry.

Hidden surface removal by range priority was done in hardware as well. This eliminated the necessity for separating planes.

A separate modeling system was delivered which was used to create and analyze databases. This system consists of a calligraphic display system and a general purpose computer, PDP-11/40. Software packages aid the designer in producing new databases. The databases are viewed in wireframe on the display system.

This is the first time that the database modeling was implemented on a system other than the visual system. Due to the extremely heavy usage of the CT3 in the simulation laboratory, a separate modeling station is necessary.

A new language was developed for database modeling. The new compiler set, MEDUSA for the visual system and COLIDE for the collision detection system, offers the designer a capability and flexibility heretofore unknown. In the process of constructing an environment, the designer defines and names points. Subsequently, he defines polygons in terms of these points and adds attributes such as color and reflectivity. With this innovation, the database design becomes much more like programming rather than filling out specification requirements.

Another language set, VIS for the visual system, CDS for the collision detection system, and HOST for the host machine, was developed for the real-time programmer. Symbolic parameter areas are defined by the programmer and transformation sequences are specified to meet the simulation requirements. Again, the real-time programmer is more of a programmer rather than someone who

allows the data to dictate the real-time configuration. Currently, two versions of the HOST software exists. HSTSES interfaces to SES and HSTSTS interfaces to SAIL. At the present time, CT3 is in use at least 16 hours a day supporting on-orbit studies in the SES.

ESG #1 -- A NEW LEASE ON LIFE

Returning to ESG #1 (alias NASA III), the Raytheon 520 was more and more difficult to maintain and the SGS was even more difficult to maintain. During the early 1980's, this author's project was to replace the R520 with a more modern Systems Electronic Laboratory (SEL) 32/55. During this project, the SGS was removed from the system as well as some other parts of old ESG #1. The remaining units are the VCU, TOU, and EPU. The display systems were reworked. This effectively removed the texture capability from the system.

The frame update rate of ESG #1 was not changed. To have done so would have required major hardware modifications. Therefore, the frame update rate was retained at 20 Hz.

Along with the hardware changes, a new set of software packages had to be written. Rather than follow the original design for database modeling and real-time application software, it was desirable to model the software after CT3. In this effort, the database modeling software closely approximates that of CT3, given the different hardware algorithms between the two machines. In addition, the real-time software approximates the real-time software of CT3. The desired effect was the capability of a database designer or real-time programmer to move between the two machines with very little effort.

The database modeling software, ENCOM, was designed to emulate the database modeling software which exists on CT3. A complete rework was undertaken so that a programmer on CT3 could move to ESG #1 with minimal effort. The established procedure of defining and naming points followed by defining polygons in terms of these points was introduced to ESG #1.

The real-time software followed a similar theory and procedure. The real-time software on CT3 was deemed a standard and the real-time software on ESG #1 was designed to match as closely as possible. Currently, three versions exist. VISUAL interfaces to the SES, VISSTS interfaces to SAIL, and VIS is a standalone version used for local applications and development.

As of this writing, ESG #1 is actively supporting entry simulation. A set of 13 entry scenes are available. Due to its age, it is difficult to maintain and some of the electronic components are not obtainable. There is a project underway to retire it and replace it with a more modern computer image generator.

POLY 2000 -- A MEDIUM RESOLUTION, LOW COST APPROACH

A POLY 2000 built by GTI, Incorporated (GTI) was purchased in October, 1985. It was planned to add three low resolution channels to the complement which existed in SES at that time. However, the software and hardware theory was radically different.

The digital technology had advanced so greatly during this period in time that the POLY 2000 could accomplish real-time computer generated images using micro-coded, high-speed bit slice processors.

The POLY 2000 at that time was in its infancy. It did not have anti-aliasing or smooth shading. It did have diffuse reflectivity which gave the impression of a sun direction. Alphanumeric characters could be overlaid on the scene.

The POLY 2000 consisted of a general purpose computer and a set of special purpose bit-slice processors. The general purpose computer, the Alcyon, was used in database development as well as generating load modules for the POLY 2000 proper. The first major processor was the System Control Module (SCM). When the system was connected to the simulation, data was sent directly to the SCM and the Alcyon was not used. This left the Alcyon free to be used in other capacities.

Special requirements were insisted upon. The frame update rate was set at 25 Hz to match that of CT3. Multiple channels were requited. The vendor produced a system with three independent channels.

Database modeling was done on the Alcyon and the binary object files were stored on its disk. When needed, the object modules were downloaded to the SCM. The database compilers, polyi2I and polyd2D, were not compatible with any of the database modeling software which existed in the SES.

The real-time programs were written in C on the Alcyon and, also, downloaded when required. No special language was implemented; all special functions were included in a library.

These concepts were radically different than the ones established on ESG #1 and CT3. Lockheed personnel undertook the

effort to write a set of software packages which would more closely emulate the concepts of the other graphics systems. These included the database compiler (POLYS) and the real-time support software packages known as the linker (LOADER), and the loader (LOAD). The syntax of the language was modeled after those commands found in the software of the other systems.

POLY 2000e -- AN ENHANCED VERSION

Given a few years, GTI enhanced the POLY 2000 to include additional features. Their new POLY 2000e provided the following enhancements:

1. Smooth-shading - Gouraud shading was added to the flat shaded and fixed shaded polygons already in the system.

2. Anti-aliasing - Sub-pixel averaging was used as the algorithm for curing the jaggies.

3. Transparency - Eight levels of polygon transparency was incorporated into the system.

4. Depth fogging - Gradual dissolution to the background provided for fog and haze.

Essentially, the enhanced system was an entirely new system. New algorithms and hardware were exchanged for the older system. The three channels operating at 25 Hz were retained. The general purpose computer, the Alcyon, was also retained.

Because the hardware and software theories were changed, the custom-made compilers had to be changed. Extensive effort was employed to upgrade the compilers to match the new system without having to rework all the previous database and applications software.

The three channels of the POLY 2000e are currently supporting on-orbit studies in the SES.

CT6 -- A PRIDE AND JOY

As of this writing, an Evans and Sutherland CT6 is in the process of being integrated into the SES laboratory. As with the advent of CT3, major steps in computer graphics were introduced to the SES.

The general purpose computer of the CT6 is a Gould Concept 32/67. The special purpose computers are still arranged in a pipeline fashion but there are fewer major components doing much more. The configuration of the pipeline allows channelization to be performed early in the processing.

The current system has six independent channels of high resolution, high quality images. The system is capable of processing thousands of polygons and each channel can display up to 1500 polygons. Compared to the hundreds in the other systems, the increased capacity is impressive.

Not all the database need reside in active memory. Parts of the database which are potentially visible reside in memory while the rest resides on disk. When those objects not in memory become potentially visible, they are paged in from disk to memory. This provides a mechanism which essentially expands the potentially visible database many fold.

Texture returns. Not only ground texture but polygonal texture is made available. Any polygon regardless of orientation can have texture. Many texture patterns are available in the system. Texture can be produced by closed form equations or photographically derived.

Database development is now necessarily more complex. Database modeling is done on a MicroVax II. Not just one software package is enough. Several database modeling software packages are provided not only to develop databases but also to display the resulting databases on a color calligraphic display device, the PS330. Database development includes object and surface production capability. The main database compiler, DBC, deals with objects, polygons, and points. A separate linker, LNK, is used to join the intermediate binary object files. The surface feature editor, SFE, assists in creating large terrains principally used in landing scenarios. Several display packages are included such as the CT simulator, CTS, and the graphics editor, GRE.

Complexity also manifests itself in the real-time software. The increased memory and speed provided by the Gould Concept 32/67 also provides a wealth of capabilities for the real-time system. Because the Gould Concept 32/67 is a dual CPU system, the real-time tasks have been divided to optimize this feature. The user interface to the real-time process, RTS, allows the user a wide variety of commands and capability to control and configure the system.

As of this writing, the CT6 is in the last stages of integration into the SES. The database designers are working hard to supplement the databases already delivered by Evans and Sutherland. The SES is looking forward to the time when

the CT6 will be brought on-line and support the simulation studies.

CONCLUSION

This paper has explored the history of the computer image generator as they existed and still exist in the SES.

Many advances have been made in the hardware, taking advantage of the current state-of-the-art circuitry available at the time. From the discrete components to the first integrated circuits to the very large scale integrated devices, the real-time graphics industry has tried to use everything at its disposal to create the best images available. The theory on how to best utilize the hardware advances has also changed toward flexibility, programmability, and manageability.

Database design has grown from data specifications to large and complex programs. As the complexity in databases increased, the complexity in the database software increased.

Real-time software has changed radically over the duration of the real-time computer image generator. In the early stages, the simulation visual system was driven mainly by the data which it received. As more and more powerful front-end general purpose computers were available, the real-time programmer was able to enjoy more and more flexibility in the control of the visual system.

REFERENCES

1. Foley, J.D. and Van Dam, A., "Fundamentals of Interactive Computer Graphics", Addison-Wesley Publishing Co., Reading, Massachusetts, 1982.

2. Giloi, Wolfgang K.,k "Interactive Computer Graphics", Prentice-Hall, Englewood Cliffs, New Jersey, 1978.

3. Newman, W. M. and Sproull, R. F., "Principle of Interactive Computer Graphics", second edition, McGraw-Hill, New York, New York, 1979.

4. "Instruction Manual for Visual Three-View Space-Flight Simulator", General Electric, Ithaca, New York, no date.

5. "Final Report: Visual Three-view Space-Flight Simulator", General Electric, Ithaca, New York, August, 1964.

6. "Instruction Manual for Modifications to Interim Visual Spaceflight Simulator", General Electric, Ithaca, New York, no date.

7. "Final Report: Modifications to Interim Visual Spaceflight Simulation", General Electric, Ithaca, New York, February, 1968.

8. "Instruction Manual for Electronic Scene Generator Expansion System", General Electric, Ithaca, New York, January, 1972.

9. "Final Report: Electronic Scene Generator Expansion System", General Electric, Ithaca, New York, December, 1971.

10. Schumacher, R. A., "Off-line Software for Expanded Electronic Scene Generator", General Electric, Ithaca, New York, April, 1971.

11. Brues, C. T., "Off-line Software Package for the NASA III Electronic Scene Generator", Lockheed Electronics Company, Inc., Houston, Texas, June, 1972.

12. "Operations Manual for the NASA-JSC Continuous Tone Image Generation and Collision Detection System", Evans and Sutherland Computer Corporation, Salt Lake City, Utah, May, 1977.

13. "Technical Manual for the NASA-JSC Continuous Tone Image Generation and Collision Detection System", Evans and Sutherland Computer Corporation, Salt Lake City, Utah, October, 1977.

14. "Final Report: Improved Scene Generator Capability", Evans and Sutherland Computer Corporation, Salt Lake City, Utah, October, 1977.

15. "POLY 2000 System Overview", GTI Corporation, San Diego, California, January, 1984.

16. "POLY 2000 Software System Manual", GTI Corporation, San Diego, California, December, 1984.

17. "POLY 2000 Programming Manual", GTI Corporation, San Diego, California, April, 1985.

18. "POLY 2000e Programming Manual", Rediffusion Computer Graphics, Incorporated, San Diego, California, no date.

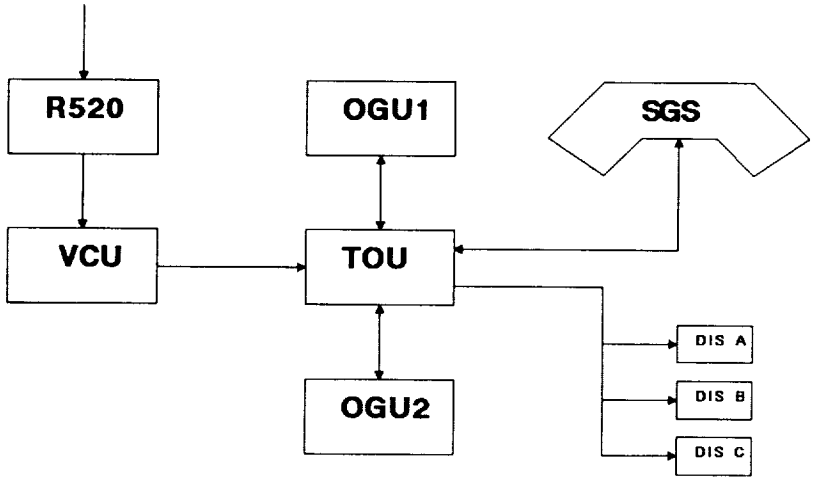


FIGURE 1: BLOCK DIAGRAM OF NASA II

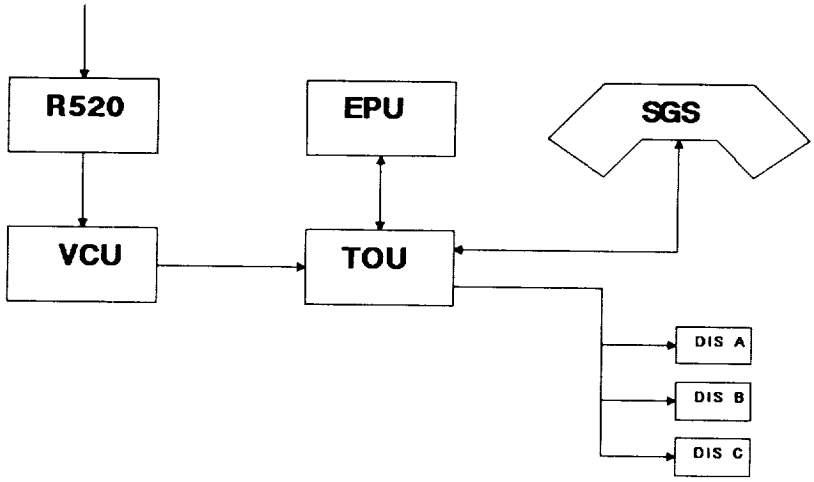


FIGURE 2: BLOCK DIAGRAM OF NASA III

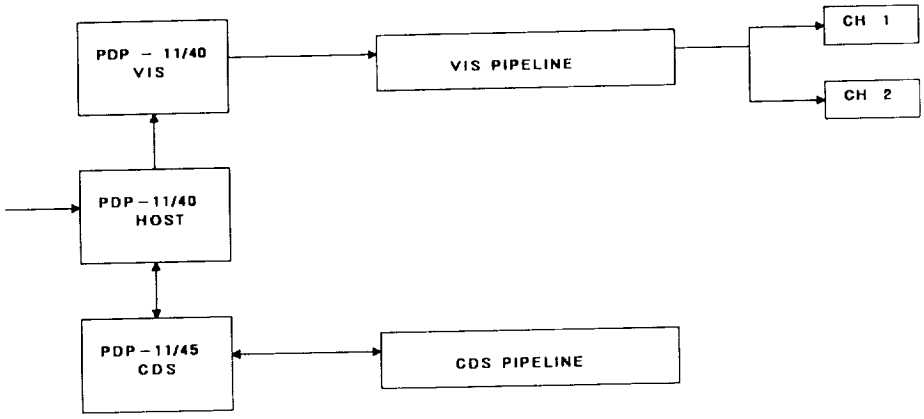


FIGURE 3: BLOCK DIAGRAM OF CT3

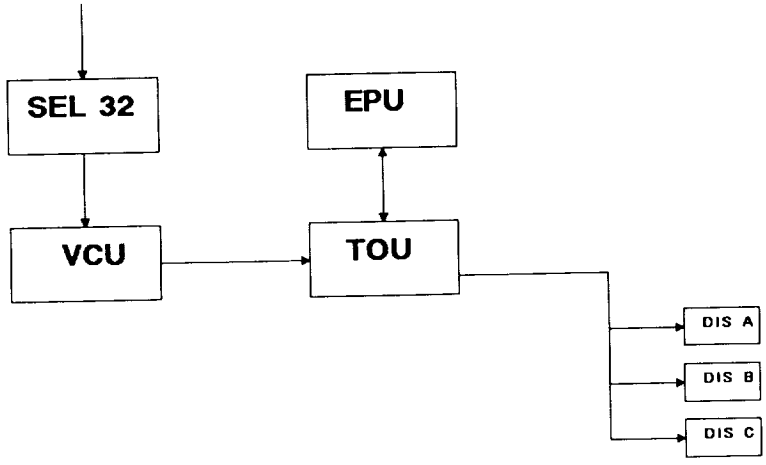


FIGURE 4: BLOCK DIAGRAM OF NASA III, UPGRADE

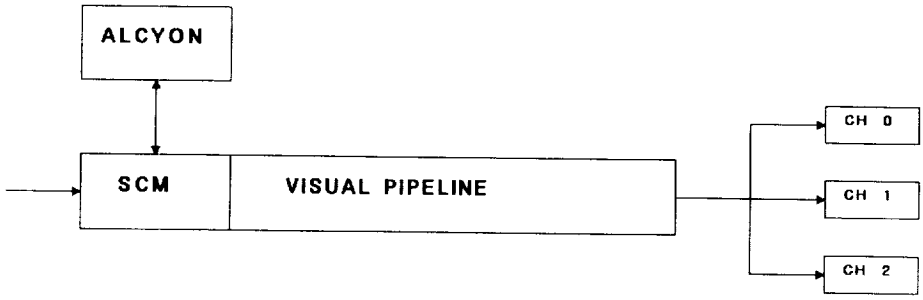


FIGURE 5: BLOCK DIAGRAM FOR POLY 2000 AND POLY 2000e

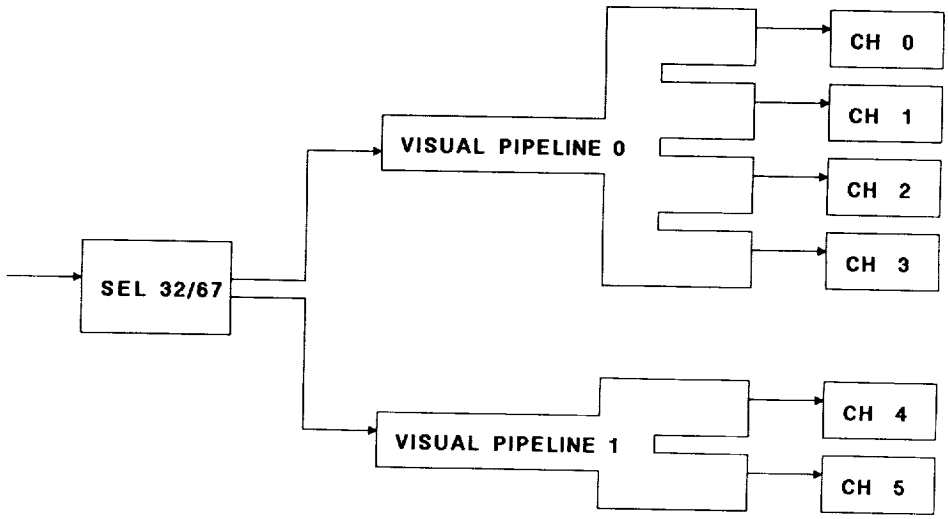


FIGURE 6: BLOCK DIAGRAM FOR CT6

COMPUTER IMAGE GENERATION: RECONFIGURABILITY AS A STRATEGY IN HIGH FIDELITY SPACE APPLICATIONS

Michael J. Bartholomew
Evans & Sutherland Computer Corporation
600 Komas
Salt Lake City, Utah 84108

ABSTRACT

The demand for realistic, high fidelity, computer image generation systems to support space simulation is well established. However, as the number and diversity of space applications increase, the complexity and cost of computer image generation systems also increase. One strategy used to harmonize cost with varied requirements is establishment of a reconfigurable image generation system that can be adapted rapidly and easily to meet new and changing requirements.

This paper examines the reconfigurability strategy through the life cycle of system conception, specification, design, implementation, operation, and support for high fidelity computer image generation systems. The discussion is limited to those issues directly associated with reconfigurability and adaptability of a specialized scene generation system in a multi-faceted space applications environment. Examples and insights gained through the recent development and installation of the Improved Multi-function Scene Generation System at Johnson Space Center, Systems Engineering Simulator are reviewed and compared with current simulator industry practices.

The results are clear; the strategy of reconfigurability applied to space simulation requirements provides a viable path to supporting diverse applications with an adaptable computer image generation system.

INTRODUCTION

One of the key problems facing high fidelity visual simulation is balancing fidelity and realism with cost and versatility. Government and Industry Aerospace Engineering and Training disciplines have typically required the highest fidelity visual imagery to maximize research and training objectives[1] [2] [3]. In order to achieve the greatest measure of fidelity for the specified objectives, visual system contractors review specifications and configure specific systems to best meet the particular requirements of a given procurement. The resultant systems are tailored for specific engineering or training applications.

As the pace and diversity of space missions increase, and the requirements of space station construction and deployment come into sharper focus, the demands placed upon engineering and training visual simulation will escalate. In their 1987 IEEE paper[4] Robert H. St. John, Gerard J. Moorman, and Blaine W. Brown concluded "Simulation was important in the design and verification of the Space Shuttle, and it will continue to be instrumental in supporting changes and improvements to Space Shuttle hardware and software as well as to the mission design and verification process." Ankur R. Hajare, in a paper presented at the 10th Interservice/Industry Training Systems Conference[5], reviewed many of the requirements for the Space Station Training Facility. Continuing evidence of this need is underscored by the pending Shuttle Mission Training Facility visual system upgrade.

One strategy for harmonizing requirements with cost, while maintaining the highest level of visual fidelity, is to design and construct the primary image generation system components with versatility as a prerequisite. This versatility, or hereafter referred to as reconfigurability, applies to hardware, software, and data base elements, and permits timely reconfiguration of one or more of the system elements to meet a wide set of well defined requirements as well as new and/or additional requirements. This paper defines and addresses the significance of reconfigurability within the framework of the Improved Multi-function Scene Generation System recently installed at Johnson Space Center, Systems Engineering Simulator.

DEFINITION OF RECONFIGURABILITY

Reconfigurability, for the purposes of this paper, is defined as the capability to reorganize one or more components of an image generation system, including hardware, software, and data base components, to meet new, different, and/or expanded requirements. The methodology applied to identifying candidate components for reconfiguration is akin to the life cycle and development methodologies espoused by Dr. Roger Pressman[6]. He indicates "system definition is the first step of the planning phase and an element of the computer system engineering process . . . attention is focused on the system as a whole. Functions are allocated to hardware, software, and other system elements based on a preliminary understanding of requirements." Reconfigurability is a key additional requirement to be taken into consideration during the system definition phase. By identifying potential contributors to reconfigurability during the system definition phase, effort can be made to modularize and further refine these elements during the design and development phases. This, in turn, permits a smooth integration and implementation of these malleable components.

FUNDAMENTAL COMPONENTS OF AN IMAGE GENERATION SYSTEM

Before proceeding to identify specific image generation system components, it is necessary to review the fundamental components of an image generation system and provide some details about the Improved Multi-function Scene Generation System. (For a thorough review of image

generation and processing theory see references and suggested reading[7][8].

As the block diagram in Figure 1 illustrates, the modeling system is the initial functional component of an image generation system. A characteristic modeling system hardware configuration includes a graphical workstation, a mini computer with mass storage and communication capability, and associated peripherals. The software components include a general purpose operating system complete with languages, text editors, and network capabilities, as well as the special purpose data base modeling software.

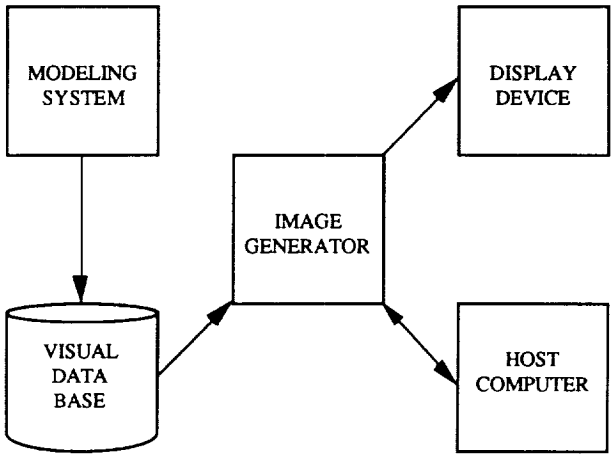


FIGURE 1
FUNDAMENTAL COMPONENTS OF AN IMAGE GENERATION SYSTEM

The modeling system is typically used in an off-line mode from the remainder of the image generation system. It facilitates the mathematical definition and construction of data base elements and organizes these elements into a visual data base (please note for the context of this paper, visual data base implies support of out-the-window or Closed Circuit Television (CCTV) views. This does not necessarily preclude other views including, but not limited to, infrared sensors or radar. For an introduction into the issues of data base correlation see references and suggested reading[9]). In addition to maintaining the mathematical representation of models and environment, the visual data base provides the framework for rapid and efficient access by the image generator.

The image generator (IG) is an highly specialized computer system typically consisting of a general purpose mini computer combined with multiple cabinets of custom image generation hardware. The hardware is controlled through a custom real time software (RTS) package that monitors IG performance as well as managing communication with the host computer.

The host computer maintains the mathematical model of the simulation, monitors operator input, and transmits position, attitude, and environmental control information to the image generator. The image generator, in turn, traverses the data base framework and displays the appropriate imagery on the display device.

THE IMPROVED MULTI-FUNCTION SCENE GENERATION SYSTEM

As the name implies, the Improved Multi-function Scene Generation System (IMSGS), an Evans & Sutherland CT6 System, installed at Johnson Space Center, Systems Engineering Simulator (SES) is dedicated to supporting many different aspects of high fidelity, large scale space simulations. The contract called for an image generation system that could integrate with existing SES simulation capabilities and augment the quality and quantity of visual imagery. Among other tasks, SES currently supports orbiter operational procedures development and testing, remote manipulator operations, payload handling, flight support and training on shuttle to proximity operations, docking and berthing techniques development, and conceptual development for the space station[10].

At the time of the CT6 installation, May 1988, the SES facilities included several networked Gould 32/87 host computer systems supporting an orbiter aft cockpit mock-up, an orbiter forward cockpit mock-up, a space station cupola mock-up, and a manned maneuvering unit (MMU) mock-up. The video feeding each of these mock-ups was derived from one of three image generation systems, each supplying one, or at most three, channels of imagery. The imagery was transmitted to the mock-ups through a sophisticated scene selection and video distribution system permitting allocation and assignment of an individual image generator channel to a specific view.

The IMSGS, as depicted in Figure 2, incorporates an Evans & Sutherland CT6 IG complete with a Gould 32/6781 mini computer. It is supplemented by a Digital Equipment Corporation MicroVAX based Modeling System complete with an Evans & Sutherland PS330 graphical workstation. The system also includes a maintenance and operation station and video switching and CCTV video post processing capabilities.

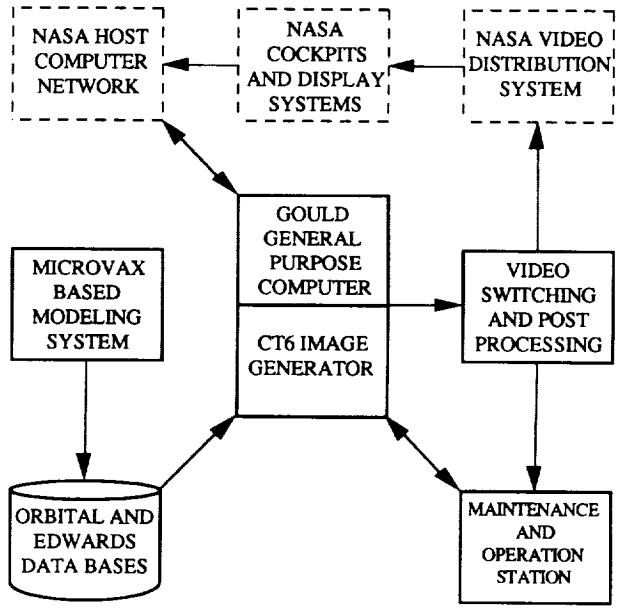


FIGURE 2
IMPROVED MULTI-FUNCTION SCENE GENERATION SYSTEM

The CT6 IG hardware supplies 6 channels of imagery. Each channel supports full texture capability and can display a standard capacity of 1500 polygons at 50 Hertz operation. Five of the six channels have a normal resolution capability of a half a million active pixels, while the sixth channel basic configuration supports three quarters of a million pixels. During terrestrial operations the hardware supports two fully independent eyepoints with up to six fields of view shared between the two eyepoints. During orbital operations each of the six hardware channels can function as a fully independent, six degree of freedom, eyepoint. One of the six channels is equipped with non-linear image mapping (NLIM) permitting pre-distortion of an image to correct for display distortions.

Acting as a front end for the CT6 IG, a Gould 32/6781 general purpose computer (GPC) system is networked to the SES host computer system and the IMSGS modeling system. With over nine hundred megabytes of disk storage, the GPC provides a repository for the visual data bases and sufficient storage space for operating system and application software needs.

During active simulations the GPC communicates with the SES host computer system at 80 millisecond intervals for position and attitude of the dynamic models as well as environmental data such as scene illumination level, sun angle, field of view, CCTV camera pan and tilt angles, and other control parameters. This information is transmitted to the IG for real time display.

When active simulations are not in session, the GPC supports stand alone simulations, diagnostic and maintenance activities on the IG, and general purpose software development. The GPC is equipped with the Gould MPX operating system, a custom real time software package supporting both host controlled and stand alone IG activity, and a comprehensive diagnostics software package to assist in fault isolation.

Assisting in both host controlled and stand alone modes, the IMSGS maintenance and operation station provides an interactive control console, or flybox, for monitoring IG activity, flying through data bases in stand alone mode, and for running diagnostics. The station houses dedicated monitors for each of the six image generator channels permitting simultaneous view of all image generator activity. There are also two dedicated CCTV monitors, one switchable CCTV monitor, and one switchable general purpose monitor.

The video supporting the maintenance and operation station is supplied via a software controlled video switching system. The video switching system controls distribution of video from the IG to the maintenance and operator station, cockpits, or to the video post processing hardware. The video post processing hardware can optionally convert RGB component video to PAL-I composite video for CCTV display, mix two channels of imagery for split screen CCTV, and overlay CCTV camera identification, pan angle, tilt angle, and camera temperature characters on the video for CCTV display.

The IMSGS modeling system supports definition, construction, modification, and display of CT6 visual data bases in an off-line mode. The modeling software includes the capability for creating new data bases, altering existing data bases, generating texture maps, automatic terrain generation from Defense Mapping Agency Terrain Elevation Data, and evaluation of IG performance through the use of a CT6 software simulator. The modeling system is connected to the GPC via an Ethernet interface, facilitating transmission of completed data bases.

A total of four operational data bases are supplied with the IMSGS. The first three data bases are orbital data bases containing the following common components: an earth model, a star field with 1,655 stars modeled with correct relative magnitudes and locations, a sun model, a moon model, and a highly detailed orbiter model.

In addition to the common elements, the first orbital data base also contains a detailed model of the Tethered Satellite Subsystem (TSS) complete with a pallet and satellite tower resting in the orbiter payload bay, and the satellite.

The second orbital data base contains a detailed and articulated model of the Remote Manipulator System (RMS) as well as a detailed model of the Hubble Space Telescope. The telescope model is visible in both stowed and deployed orientations.

The third orbital data base contains the detailed model of the RMS, a detailed model of the MB-9 version of the Space Station, a generic payload, and the Mobile Service Center (MSC) complete with a Mobile Remote Manipulator System (MRMS).

The fourth data base, a terrestrial data base, is the southern California region with a detailed representation of Edwards Air Force Base. The data base is 1,244 nautical miles by 1,244 nautical miles. The 121 nautical mile by 121 nautical mile terrain region centered about Edwards is map correlatable. Terrain elevation information was extracted from Defense Mapping Agency (DMA) Digital Terrain Elevation Data (DTED). In addition to the highly detailed Edwards AFB area, the data base is equipped with a detailed orbiter model and two detailed T-38 models.

Each of the aforementioned data bases make extensive use of algorithmic and photo-derived texture to augment scene fidelity and realism.

RECONFIGURABILITY, A CASE HISTORY

By the time the IMSGS contract was awarded in late September of 1986 the NASA Engineers who had specified the visual system requirements had already laid a great deal of the ground work for a reconfigurable image generation system. The requirements made clear the goal of reconfigurability in a number of areas, such as " the update rate shall be software selectable to run at 25, 30, 50, or 60 Hz" or "It is desirable that each channel be capable of having . . . a range of 0.25 to 1.0 megapixels . . . "[11].

The team of engineers assigned to the program, working with their NASA counterparts began the analysis, design, and implementation of the requirements specified in the contract. Some of the candidates for reconfigurability surfaced immediately, such as being able to redistribute IG hardware components to increase or decrease pixel resolution or polygon capacity. Other candidates have come to light further down stream such as the multi-tiered occultation solution. Specific examples of reconfigurable items are detailed in the following paragraphs, divided into three broad categories: hardware, software, and data base.

HARDWARE

There are numerous explicit requirements as well as suggested goals in the IMSGS contract[11] for hardware reconfigurability. Some of the more obvious items such as being able to increase memory or disk storage in the GPC or modeling system, or being able to attach other peripherals to the system, are not addressed in detail in this paper. The items deemed uncommon, or atypical, for image generation systems are detailed below.

Addressing the requirement for variable resolution, the CT6 image generator was equipped with the capability to share display processor components between channels. This permits increasing or decreasing the effective pixel resolution from 262,000 active pixels to over 1,000,000 active pixels by simply loading a different microcode initialization file. This also implies a variable line rate and pixel rate capability. The video line and pixel rates are programmable in ranges of 13 to 30 KHz and 10 to 40 MHz respectively, while the IG is equipped to run at 25, 30, 50, or 60 Hz, with display refresh rates of 50 or 60 Hz, thus allowing the use of a wide range of displays. The maintenance and operation station is equipped with multi-sync monitors able to match any line or pixel rate generated by the IG. This capability has already been put to use in the SES lab, matching the video characteristics of the old Conrac 62601 displays, as well as the newer XKD 1955 and SRL 2125 displays.

Just as display processor components can be shared between channels to focus pixel resolution, the geometric processor components can also be shared to focus polygon resolution. This permits an increase in polygon capacity from the basic 1250 polygons per channel at 60 Hz, to over 2,200 polygons per channel at 50 Hz operation.

As indicated earlier, the SES lab supports several cockpits each with a different number of displays. A sophisticated, software driven, scene selection system is in place that allows the assignment of any given image generation system channel to a particular display device in a particular cockpit. The IMSGS is required to interface with that system, and does so with the use of a software controlled video matrix switcher and video post processing capability. This video switcher can be controlled through local software commands within the IMSGS environment, or from the SES host computer system. Any one of the six CT6 channels can be routed through the switcher to provide an out-the-window, CCTV, or MMU view, as required, to any of the mock-ups.

One of the requirements of the contract stated that at least one IG channel be capable of supplying pre-distorted imagery at varying pixel resolutions for use on an unspecified display and/or projection system. This capability, known as non-linear image mapping, or NLIM, allows a digital mathematical correction of image components to ensure proper geometric relationships when displayed on a non-linear surface such as a dome. This capability works in harmony with the aforementioned display processor sharing to increase or decrease pixel resolution and is activated or deactivated through a microcode control file.

SOFTWARE

The software components identified as reconfigurable items were not as clearly defined at the requirements phase as the hardware elements, nor as straight forward to design or implement. There were the typical stated goals such as modularity and maintaining reserve capacity for future growth. There were also the not-so-obvious goals of identification and reutilization of key individual modules to help meet future requirements, or documenting critical portions of code to such a degree that a novice software engineer, with little or no image generation system background, could effectively learn and modify the software on an as needed basis. Through striving to meet these and other stated and unstated goals there were several software items that surfaced and were implemented as key reconfigurable elements.

One of the key reconfigurable software elements is a portion of the real time software package known as occultation management. This software works in harmony with the data base fixed priority relationships and existing real time object range sorting algorithms to provide an additional tier of object level occultation management. This is one of the areas where the software has purposely been designed and documented to facilitate a shopping cart approach to new requirements. By using off-the-shelf key modules and, where necessary, modifying modules that are similar in nature to the additional element(s), new capabilities can be added in a timely and consistent manner.

In like manner, the host to GPC interface communications software is designed to allow the timely addition, or deletion, of simulation control parameters. In typical simulation applications a fixed number of computer words are reserved for data communications between the host computer system and the image generation system, where each word, byte, and bit have a known fixed location and format in the data buffer. Changing the fixed format to add or delete a parameter requires modification of all software elements on both sides of the interface accessing that data buffer. By contrast, the reconfigurable solution packetizes or modularizes each control parameter by parameter type. For example, all dynamic model position and attitude data is identical in type and format, only the model identification bits vary from model to model. Adding a new model to a simulation is achieved by simply adding that packet of information to the communications block. The block is fixed length in nature, but the parameter packets can vary in any number and sequence within the data block. When new packet types are defined, an additional module is added to the communications software to handle that packet type, with no adjustments or adverse affect on other packet modules.

Similar to the concept of packetizing the control information above, the diagnostics software is organized in a modular fashion. Rather than writing a package of diagnostics unique to each image generation system configuration, or each backpanel within the image generation system, IMSGS uses a general purpose diagnostics interpreter for fault isolation within the IG. A diagnostic test is provided in the interpretive language for each applicable card type in the system. By interactively, or through a batch file, instructing the interpreter which card, function, backpanel, channel, or system to test, the appropriate diagnostics are executed within the framework established by the operator. If a particular situation demands a modification to a diagnostic, the particular diagnostic can be edited with a normal text editor to include the additional capability.

DATA BASE

As with the majority of hardware reconfigurable components, most data base components were readily identified through specific requirements in the contract. The obvious items surfaced immediately and included such

elements as: a general purpose shuttle model with variations supporting attachment of an RMS, articulated doors and solar panels, and a docking tunnel; an earth model complete with cloud cover and an atmosphere ring; dynamic moon and sun models; a star field containing a minimum of 100 specific stars with correct relative magnitudes and locations; two specific payloads including the Tethered Satellite Subsystem and the Hubble Space Telescope with variations for stowed and deployed positions; and the MB-9 version of the space station including articulated solar arrays, a highly detailed primary docking port, and a dynamic mobile service center with MRMS.

Comparable with the software items there were reconfigurable data base components that surfaced during the design and development phases as well. For example, one generic CCTV model was created and referenced for each of the shuttle, RMS, and MRMS CCTV locations; one grapple fixture model was created and referenced for the two space telescope grapple fixtures, the shuttle grapple fixture, and the space station grapple fixture; one v-guide model was created and referenced for each of the three locations in the payload bay; one set of visual approach slope indicator (VASI) lights, ball bar lights, and precision approach path indicator (PAPI) lights were created and referenced for each applicable runway at Edwards Air Force Base.

Each of these data base components, along with many others, are available on the IMSGS modeling system to allow modification of existing data bases or construct new data bases in order to meet new or expanded requirements. SES has already begun utilizing many of these components to implement the Infrared Background Signature Survey (IBSS) and Orbital Maneuvering Vehicle (OMV) simulations not specified in the IMSGS contract.

CONCLUSION

Throughout the project life cycle there have been several key items identified as reconfigurable in nature. Many of these items were identified as specific requirements in the contract, some of the items were already embodied in various combinations of hardware, software, or data base, and some of the items surfaced while in the design or development phase. In all cases it was evident that if a particular item had been anticipated and identified during either the requirement or system definition phase, it was cheaper in terms of raw cost and schedule to implement than if it was identified later in the life cycle. Even when items were identified late in the contract, it was still beneficial in the long run to either include them as part of the contract, or recommend them for inclusion at a later date. Also, in all cases, once a given item was implemented, the savings in exercising the feature in terms of time, fidelity, maintainability, and development cost was obvious. The IMSGS is providing SES with a truly reconfigurable scene generation system that can grow and adapt with their new and changing requirements.

In an industry where change and redefinition are the norm, reconfigurability provides an important implementation and budget control strategy to assist in large scale space simulations. In order to be most effective, the reconfigurability strategy requires significant forethought and planning at the earliest phases of definition. Anticipation of expanded capabilities in performance, fidelity, and implementations can greatly enhance the systems potential. The results are clear, the strategy of reconfigurability applied to space simulation requirements provide a viable path to supporting diverse applications with an adaptable computer image generation system.

ACKNOWLEDGEMENTS

The author would like to thank the following persons for their help and encouragement in producing this paper: James R. Smith and David C. Christianson of NASA/JSC, and Mercedes Delugo, Ralph Howes, Michael Jackson, Lance Moss, Janice Poulson, and the remainder of the Evans & Sutherland NASA/JSC project team.

REFERENCES AND SUGGESTED READING

(1) Bondzeit, Fred and Edwards, Robert E., "Image Generation For Rotary Wing Applications," PROCEEDINGS OF THE 10TH INTERSERVICE/INDUSTRY TRAINING SYSTEMS CONFERENCE, December 1988.

(2) Bruce, Robert C., "Simulation Fidelity: A Rational Process For Its Identification And Implementation," PROCEEDINGS OF THE 9TH INTERSERVICE/INDUSTRY TRAINING SYSTEMS CONFERENCE, December 1987.

(3) O'Neal, Lt. Col. Maston E. and Brown, James E., "F-15 Limited Field Of View Visual System Training Effectiveness Evaluation," PROCEEDINGS OF THE 6TH INTERSERVICE/INDUSTRY TRAINING SYSTEMS CONFERENCE, October 1984.

(4) St. John, Robert H. and Moorman, Gerard J. and Brown, Blaine W., "Real-Time Simulation For Space Station," PROCEEDINGS OF THE IEEE, Vol. 75, No. 3, March 1987.

(5) Hajare, Ankur R., "Planning the Space Station Training Facility," PROCEEDINGS OF THE 10TH INTERSERVICE/INDUSTRY TRAINING SYSTEMS CONFERENCE, December 1988.

(6) Pressman, Roger S., "System Planning," SOFTWARE ENGINEERING: A PRACTITIONER'S APPROACH, First Edition, McGraw-Hill Book Company, New York, New York, 1982, p. 32.

(7) Hearn, Donald and Baker, M. Pauline, COMPUTER GRAPHICS, First Edition, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

(8) Newman, William E. and Sproull, Robert F., PRINCIPLES OF INTERACTIVE COMPUTER GRAPHICS, Second Edition, McGraw-Hill Book Company, New York, New York, 1979.

(9) Townsend, Barbara and Stovall, Beth and Colbert, Cheryl, "Correlation Of Sensor Data Bases In The Full Mission Training Simulator," PROCEEDINGS OF THE 7TH INTERSERVICE/INDUSTRY TRAINING SYSTEMS CONFERENCE, November 1985.

(10) SYSTEMS DEVELOPMENT AND SIMULATION DIVISION FY-87 ANNUAL SUMMARY, NASA Johnson Space Center, Houston, Texas, January 1988.

(11) CONTRACT NAS 9-17696, IMPROVED MULTI-FUNCTIONAL ELECTRONIC SCENE GENERATOR VISUAL SYSTEM, NASA Johnson Space Center, Houston, Texas, September 1986.

REAL-TIME GRAPHICS FOR THE SPACE STATION FREEDOM CUPOLA, DEVELOPED IN THE SYSTEMS ENGINEERING SIMULATOR

Michael T. Red
National Aeronautics and Space Administration
Lyndon B. Johnson Space Center
and
Philip W. Hess
Lockheed Engineering and Sciences Company

ABSTRACT

Among the Lyndon B. Johnson Space Center's responsibilities for Space Station Freedom is the cupola. Attached to the resource node, the cupola is a windowed structure that will serve as the space station's secondary control center. viewing. From the cupola, operations involving the mobile service center and orbital maneuvering vehicle will be conducted.

The Systems Engineering Simulator (SES), located in building 16, activated a real-time man-in-the-loop cupola simulator in November 1987. The SES cupola is an engineering tool with the flexibility to evolve in both hardware and software as the final cupola design matures. Two workstations are simulated with closed-circuit television monitors, rotational and translational hand controllers, programmable display pushbuttons, and graphics display with trackball and keyboard.

The displays and controls of the SES cupola are driven by a Silicon Graphics Integrated Raster Imaging System (IRIS) 4D/70 GT computer. Through the use of an interactive display builder program SES cupola display pages consisting of two dimensional and three dimensional graphics are constructed. These display pages interact with the SES via the IRIS real-time graphics interface. This paper focuses on the real-time graphics interface applications software developed on the IRIS.

LIST OF ACRONYMS AND ABBREVIATIONS

3D	3 dimensional
CCTV	closed-circuit television
CDB	changed data block
CDBRD	CDB read (processor)
CDBWR	CDB write (processor)
CPU	central processing unit
CRT	cathode-ray tube
CVM	current value memory
DLM	display list memory
DLRD	downlink read (processor)
DU	display update (processor)
EXEC	executive (task)
HSD	high speed data
HSDRD	HSD read (processor)
HSDWR	HSD write (processor)
IND	indicator (processor)
INTF	interface (task)
IP	input processor
IP/DU	IP and DU (task)
IRIS	Integrated Raster Imaging System
MSC	mobile service center
OMV	orbital maneuvering vehicle
OTW	out-the-window
PDP	programmable display pushbutton
RISC	reduced instruction set CPU
SES	Systems Engineering Simulator
SW	switch processor
SW/IND	SW and IND (task)
SYSID	system identification

INTRODUCTION

The purpose of simulation is to provide an accurate, economical, and most importantly safe means of testing a product. The product may range from a crewperson's expertise in performing a particular procedure to the procedure itself. Real-time simulation implies that if an event in the real world takes five seconds to transpire, the same simulated event would also take five seconds. Man-in-the-loop simulation places a human in the simulation loop, reacting to the simulation computers. For example, a crewperson initiates a command to a system. The simulation computers receive the command and perform the appropriate response. The crewperson recognizes the response and continues with a new command, completing the simulation loop. Real-time man-in-the-loop simulation provides an individual with the means of performing a task in real time.

The Systems Engineering Simulator (SES) is located in building 16 of the Lyndon B. Johnson Space Center. The SES, depicted in Figure 1, is a real-time man-in-the-loop simulation facility dedicated to providing engineering support for the Space Shuttle and Space Station Programs. SES support covers a wide spectrum ranging from engineering studies to procedures development and crew training.

The SES is composed of a computation facility, scene generation computers, and four crew stations. The computation facility consists of simu-

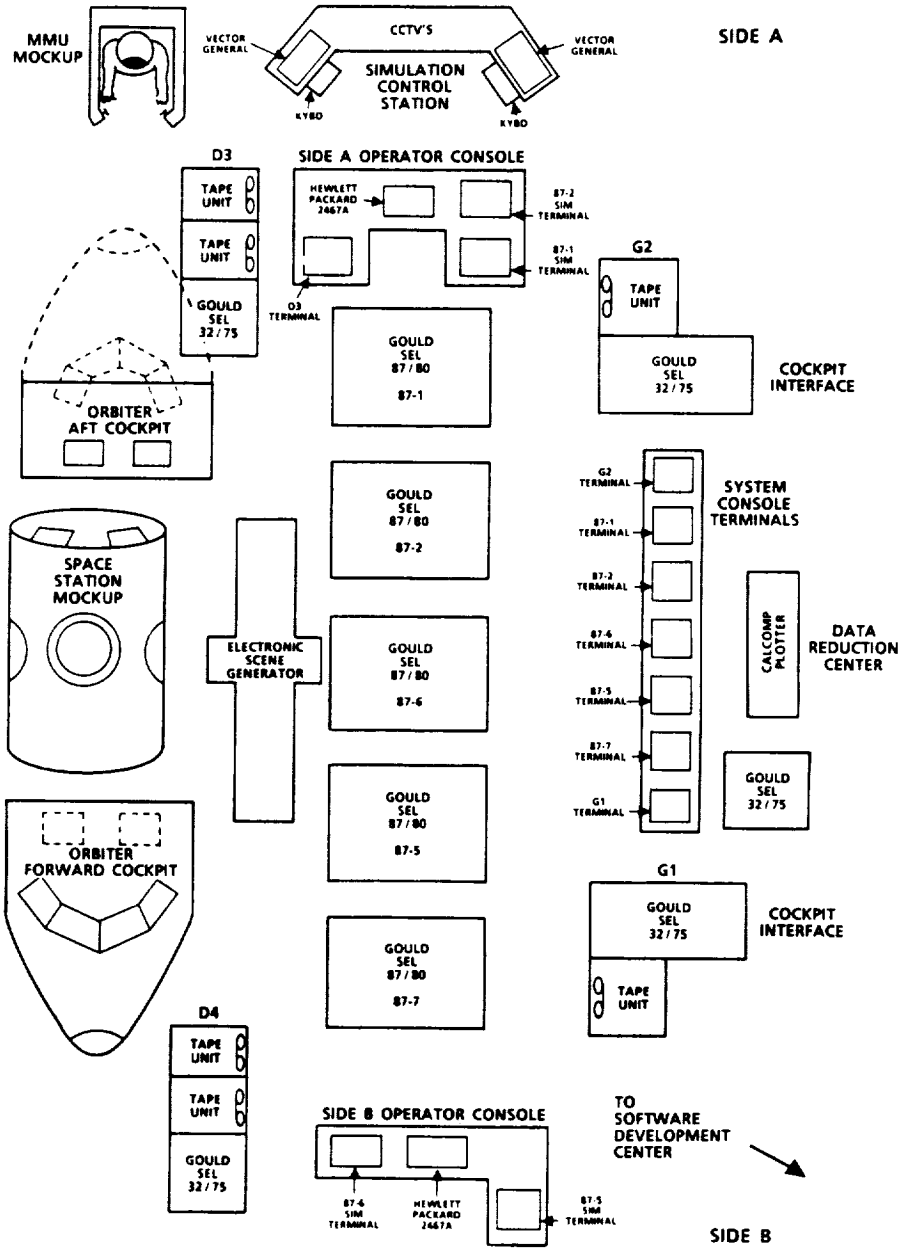


Figure 1
System Engineering Simulator

lation computers, mass storage units, data recording, and development facilities. Three real-time scene generation computers provide a combination of up to eleven out-the-window (OTW) and closed-circuit television (CCTV) views. The four crew stations supported by the SES are the forward shuttle cockpit, aft shuttle cockpit, manned maneuvering unit, and space station cupola.

The space station cupola is the only windowed structure to provide direct line of sight viewing from the space station. In its final phase I configuration the space station will have two cupolas attached to two of the space station nodes. The cupola will serve as the secondary command control station where much of the latter portion of phase I and most of phase II space station assembly will be conducted. Operations of the space station mobile service center (MSC) and orbital maneuvering vehicle (OMV) will also be conducted from the cupola.

The cupola crew station in the SES, referred to as the SES cupola, is designed to be an engineering

tool. With the final configuration not yet established the SES cupola is designed to evolve in both hardware and software configuration. The wooden mockup models half of the cupola with a viewing area for visitors or training personnel in the rear. The crew station portion consists of six OTW views and two side by side crew stations. The next phase of the SES cupola includes many hardware updates driven by McDonnell Douglas, the primary contractor for the space station cupola. This phase III SES cupola, due to be operational in April 1989, will not only include a new physical shell, but also a reconfigured interior and a new OTW optics system. As the design of the space station cupola evolves into a final state, the SES cupola configuration will evolve to match. It is planned that the SES cupola will eventually evolve into a real-time man-in-the-loop simulator with actual flight hardware.

SES CUPOLA HARDWARE

The SES cupola instrumentation and controls consist of several integrated components used to simulate possible flight hardware for two crew stations (Figure 2). The heart of each crew station is a Silicon Graphics Integrated Raster Imaging System (IRIS) 4D/70 GT workstation class computer. The IRIS has a reduced instruction set CPU (RISC) architecture, and therefore processes approximately twelve million instructions per second. This high speed provides quality graphics rendering on top of applications software adequate for a real-time simulation environment.

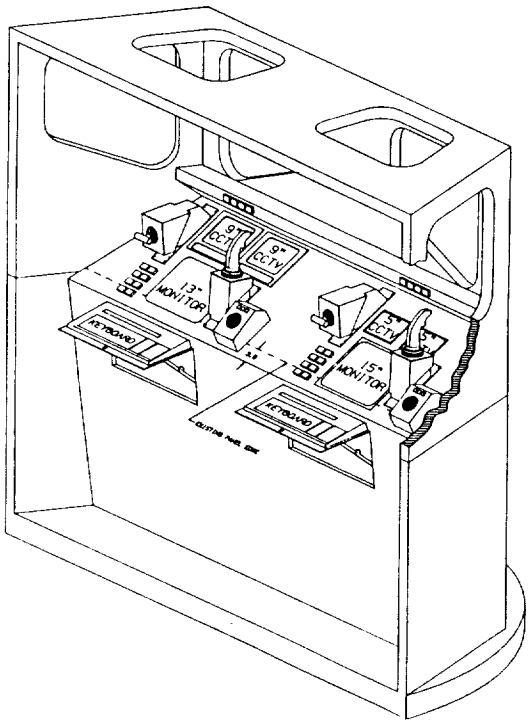


Figure 2
System Engineering Simulator Cupola

The IRIS drives and receives input commands from four user interface components in the crew station. First, the IRIS displays its graphics information on a 1024 raster line by 1280 pixel resolution 15" color monitor. The IRIS receives command input from a keyboard and three button trackball. Finally the IRIS drives the displays and receives command input from three sets of four programmable display pushbuttons (PDP). Rotational and translational hand controllers are currently interfaced to a separate general purpose computer supporting the SES cupola simulator rather than to the IRIS.

A total of three IRIS units are used in the SES. As stated previously, two are used for operations inside the cupola simulator. The third IRIS is used for development. All three IRIS units are connected together through an ethernet interface. One of the IRIS units inside the cupola, referred to as the master IRIS, sends and receives data to and from the SES simulation computers via a high speed data (HSD) interface. The other two IRIS units are referred to as slaves, but only because they receive information from the SES simulation computers via the master IRIS and ethernet. All three IRIS units operate asynchronously from any other computer.

CREW STATION DISPLAYS AND CONTROLS

The displays and controls in the SES cupola provide the user interface to the system a crewperson wishes to access. The focus here is on how information from some probable space station based computer system is displayed to the crewperson as well as how he can input commands to such a system. Therefore, four devices in the crew station will be discussed in detail: the display monitor, three button trackball, keyboard, and PDPs.

As is common with many personal computers and workstations today, the IRIS offers a window management system for flexibility and ease in displaying information. Windowing systems allow the user to display information in a specific portion of the physical CRT screen space. The "window" of information can then be moved from one position on the screen to another. In fact, numerous windows can be displayed on the screen at one time in an overlapping fashion. A user can "pop" a window to the foreground thus allowing all the information in the window to be visible or "push" the window behind all other currently visible windows. A cursor on the screen is usually used to target a specific window for one of the functions mentioned above. The cursor can be moved about the screen by a number of devices including the arrow keys on a keyboard, a mouse unit, and a trackball.

The SES cupola crew station employs a three button trackball instead of a mouse unit to position the cursor on the 15" monitor. Response from astronauts' use of the crew station dictated a preference for the trackball. Restrictions were applied to the IRIS window management system to simplify the operation of the crew station. For example, windows can be popped to the foreground but cannot be pushed to the background, and windows cannot be

reshaped. Furthermore, specific functions were tied to the three buttons on the trackball. The right button only pops windows. The middle button only moves windows. The left button is used only to select functions on the screen such as a switch. By limiting the complexity of the window management system through dedicated trackball buttons, the crewperson interfaces with an extremely user friendly system with little chance of error on the user's part.

The IRIS real-time applications software recognizes three types of display windows: data, banner, and pop-up windows. Data windows are by far the most common. They can be moved with the middle trackball button and popped with the right trackball button. As the name implies data windows display data, but they can also be used to call up new windows as well as receive inputs. Pop-up windows cannot be moved or popped to the foreground. They are designed to emulate pull down menus and thus are used only to call up new windows. When a pop-up window is called up, the next depression of the left trackball button is expected to be inside the pop-up window. Otherwise, the window is deleted. The banner window is the most unique display window. The banner window covers the entire screen and contains simulation status and time information as well as the ability to call up other windows. It cannot be moved, deleted, or popped to the foreground. The banner window is brought up when the IRIS real-time applications software is initialized and remains present during the entire simulation run with all other display superimposed.

The keyboard in the current SES cupola crew station has a very limited function. During simulation operations there is a keyboard display type available on some display windows. This display type requires keyboard input from the crewperson in the form of a floating point number. McDonnell Douglas, as a future user of the SES cupola, has requested increased use of the keyboard.

One of the thrusts behind the design of the space station cupola is a reduction in the number of hardware switches due to the lack of available space. The use of twelve PDPs per crew station in the SES cupola is one method of reaching this design goal. As the name implies PDPs can be programmed for numerous functions at different points in time. For example a specific PDP may be programmed to pan a CCTV camera to the right when depressed. Later the same PDP may be programmed to trigger the snares in the MSC end effector to capture a target. In this way the total number of hardware switches in the cupola can be significantly reduced. Currently, in the SES cupola PDPs are used to pan, tilt, and zoom CCTV cameras, as well as control several MSC functions including: turning off the master alarm, turning MSC brakes on or off, driving individual MSC joints positive or negative, and triggering the capture or release of a target.

SES CUPOLA DISPLAY AND CONTROL MONITOR

One of the primary concepts in development of the IRIS real-time applications software for the SES cupola was flexibility. Past experience had proven that hard coded display windows were difficult to modify and maintain. Because the SES cupola is an engineering simulator, the ability to modify display windows with minimal turnaround time is very important to many potential customers. It is not unreasonable that they may wish to try several different display window layouts. The IRIS real-time applications software must be flexible enough to change the display window layout as quickly as possible. Therefore, all display windows, regardless of the type (data, pop-up, or banner), are read from display data files.

The concept of the display file (Figure 3) is very straight-forward. Each display file is constructed with an off-line display builder program and contains all the information needed to produce a display window in the real-time applications software. The information in the display file is organized into units referred to as display types. Therefore, when a window is called up by the real-time system, such as the banner window upon initialization, a specific display file is read, and the display types in that file are used to draw the window during operations.

There is a finite number of defined display types that are recognizable to both the display builder and the real-time applications software. However, one of the major advantages to this method of constructing display windows is that new display types can be added with minimal impact. Once a display type is defined, construction and modification of display files becomes almost a trivial process due to the user friendly nature of the display builder. Display types in general contain the following information: an opcode to designate the type, the number of words used to define the display type, a system identification number (SYSID) used for variable data related to the display type, and (x,y) coordinates to position the display type in the display window. Beyond this preliminary data, display type information becomes more specific to the actual display type. A list

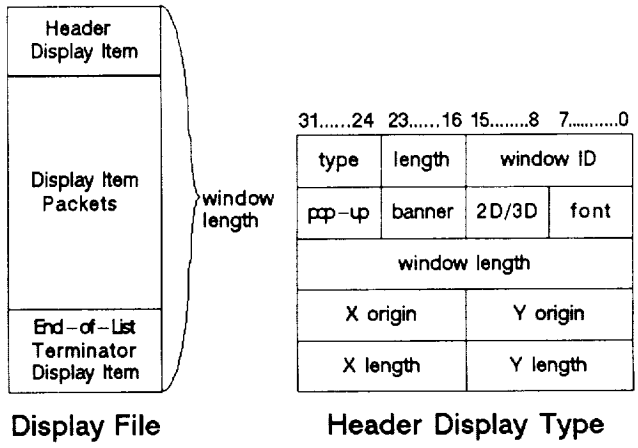


Figure 3
Display File and Header Display Type

of currently available display types is provided in Table I and Table II, and a more detailed description of each display type can be found in Appendix A.

Table I
2 Dimensional Display Types

Header Entry	End of List Terminator
Integer	Real
Long Float	Hexadecimal
ASCII Message	Static Text
Page Call	Keyboard Input
Delete	Default
Circular Gauge	Meter Bar
Indicator	Switch
Momentary Switch	Line
Rectangle	Circle
Polygon	Dynamic Position Indicator

Table II
3 Dimensional Display Types

Header Entry	End of List Terminator
Begin Coordinate Frame	End Coordinate Frame
Rotation	Translation
Scale	Box
Cylinder	Sphere
Line	

Another concern related to display windows is color. Programs prior to space station have implemented monochrome display systems and have not had to necessarily deal with the potential excessive use of color in a display system. Astronaut response to displays in the SES cupola has indicated that a wide variety in color is distracting. The number of available colors in the SES cupola has been limited to sixteen, including white and black. The complete list of available colors is provided in Table III. An attempt has been made to reduce the amount of color actually used in display windows. For instance, switches in the off position and indicators in the false state are colored gray by convention. In general green is used to indicate an active or true state, yellow indicates caution, and red indicates a warning. The other colors are used discriminatingly always attempting to reduce the amount of color on the display.

Most displays use only 2 dimensional display types. However, the graphics capability of the IRIS supports 3 dimensional (3D) graphics. While the cupola is a structure with several windows, a large percentage of the potential field of view is obstructed. CCTV cameras help, but it is very simple to lose your orientation. Radar and telemetry data from vehicles in the proximity of the space station could be used to develop a 3D situation display. The 3D situation display simulates this capability. It contains 3D wire frame drawings of the space station with MSC and vehicles in proximity of the space station (e.g., orbiter and OMV) in the correct orientation and position relative to each other. The display can be rotated, translated, or zoomed, giving the crewperson an excellent omniscient view of vehicles in proximity of the space station.

Table III
SES Cupola Graphics Monitor Colors

Black	Bright Red
Bright Green	Bright Yellow
Dark Blue	Magenta
Cyan	White
Dark Red	Dark Green
Dark Yellow	Blue
Orange	Purple
Gray	Blinking Red

REAL-TIME SYSTEM DATABASE

The SES cupola applications software utilizes an indexed shared memory concept that allows for rapid modification of shared memory (Figure 4). The allocated shared memory, called current value memory (CVM) is divided into two sections: the pointer section and the data section. The pointer section is in the lower address portion of CVM. As the name implies the pointer section contains pointers to the higher address portion of CVM or data section. The offset from the CVM base address corresponds to the SYSID of the variable. Therefore, the length of the pointer section is defined by the largest SYSID. The data section of shared memory contains data packets described below.

An off-line database program is used to maintain the SES cupola CVM. The Informix relational database program is used to keep track of all SYSID variables. Information such as the variable name and description, where it resides in the uplink or downlink buffer, as well as the variable type and initial value are kept in the database.

Applications programs were written to access the Informix database and extract information needed to build three data files: the memory image file, the uplink parameter file, and the downlink parameter file. The memory image file is a replica of the SES cupola CVM during operations (Figure 5). It contains all of the data packets that will reside in CVM. Each packet has the variable type (ie. floating point, double precision floating point, signed or unsigned 32-bit integer, 16-bit integer, bit, or character string), the length in words, the SYSID of the variable, and the current value of the variable. The uplink and downlink parameter files represent a mapping from CVM to the buffer of data sent to (uplink) and from

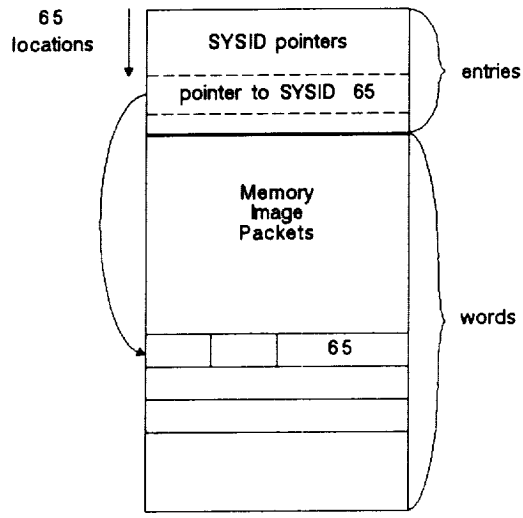


Figure 4
Current Value Memory Format

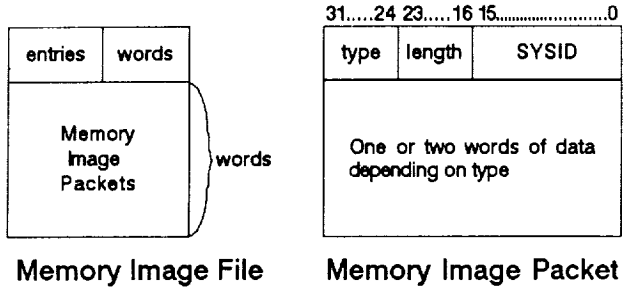


Figure 5
Database Memory Image File

(downlink) the simulation computers by the master IRIS via the HSD interface. The parameter file is ordered by buffer word first and if necessary by bit location second. Each entry in the parameter file contains the SYSID of the variable, word location in the buffer, and start bit.

Upon initialization of the real-time applications software, the executive task reads the memory image file, extracting the size of CVM, and dynamically allocates enough shared memory for CVM. As the data packets are read from the memory image file and placed into the data section of CVM, the pointer in the pointer section is resolved for the appropriate SYSID. This process continues until the memory image file is completely read.

There are some important advantages to this concept of accessing and maintaining data. First, the database is maintained off-line, and is always current to the real-time applications software because it is always read in each time the SES cupola is initialized. Second, all SYSIDs within a particular range do not have to used. In other words, there can be "holes" in the database. Third, holes in the database effect only the pointer section of CVM; the data section is always compressed with no wasted memory. Fourth, the database is used to generate reports that document the uplink and downlink buffer definitions; list SYSID variables associated with particular subsystems; list SYSID variables not in use; list the database sorted by variable type, name, SYSID, subsystem, and other criteria.

SES CUPOLA REAL-TIME SYSTEM SOFTWARE

The SES cupola real-time applications software for the IRIS was developed in-house by NASA and support contractor personnel (Figure 6). Several guidelines were adhered to in development of the real-time system to facilitate maintenance. First, the real-time system would be machine independent. Only one version of the real-time system would exist and be run on both the master and slave IRIS units. Second, the real-time system would be broken down into major functions. These major functions would reside in separate tasks so that if changes were made to a specific task and the real-time system failed, then that task would be suspect. Third, an executive task would be used to initiate and schedule the real-time system. Along with the executive (EXEC) task the real-time system is made up of the input processor and display update (IP/DU) task, the switch processor and indicator processor (SW/IND) task, PDP task, HSD task, and interface (INTF) task.

The EXEC task is the heart of the real-time system. It is responsible for allocating and initializing CVM, the changed data block (CDB) which will be detailed later, and executive shared memory which contains variables needed by other tasks in the real-time system. EXEC is also responsible for initiating the other five tasks in the real-time system as well as establishing communications between itself and the other tasks. Finally, EXEC is responsible for scheduling the other tasks in the real-time system. Considerable attention was given to the problem of homogeneous data in CVM. The order of scheduling shown in Figure 7 insures that by the time display related processing is begun in the SW/IND, CVM is updated.

Each of the five subordinate tasks contain a task executive and the processes that actually perform the function of the task. The task executive (not to be confused with EXEC) is essentially generic from subordinate task to subordinate task. Its purpose is to initialize the task in terms of access to CVM, executive shared memory, and CDB if necessary. The task executive also allows its processes to initialize if necessary. Finally, the task executive completes establishment of communications with EXEC. Once the task executive has finished initialization, it enters an infinite run loop and is put to sleep until EXEC signals it to go.

The IP/DU task contains two separate processors: the input processor (IP) and the display update

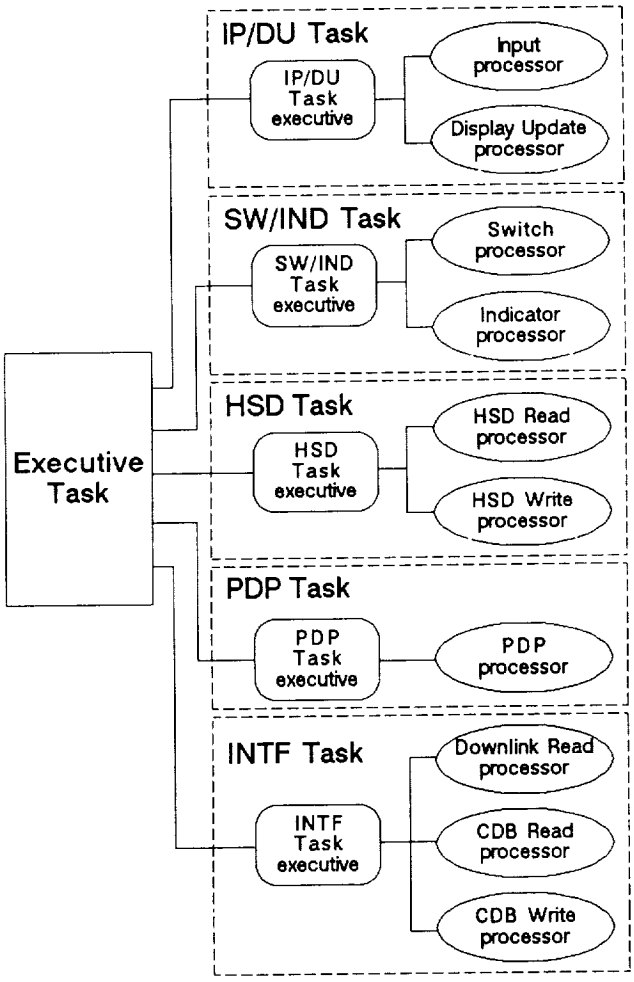


Figure 6
SES Cupola Real time Applications Software

processor (DU). The IP/DU task executive uses information in executive shared memory to determine which process to execute when it is signaled to go by EXEC.

The IP, as the name implies, processes input information from the keyboard and three button trackball. Upon initialization the IP reads the display file for the banner window and places the display information in dynamically allocated memory, called display list memory (DLM). The IP resolves overlapping display windows, as well as allocation and deletion of display windows. When a position in a display window is selected with the left button of the trackball, the IP determines the cursor position and compares it to display item positions in the currently allocated DLM. Once the proper display item is found, the IP executes the appropriate function based on the type of the display item. For example, if a switch was selected, the appropriate switch SYSID is set true, or if a page call was selected the appropriate display file is read and placed in DLM. When a display window is selected with the middle trackball button, the IP is responsible for moving that window. Finally, when a display window is selected with the right trackball button, the IP is responsible for popping that window to the foreground. Currently, the IP processes keyboard input only if the keyboard display type is selected with the left trackball button.

The DU is responsible for update of all display windows. The DU begins a trace of DLM at the appropriate starting point for a particular display window. As it encounters each display type, the DU executes the graphics commands to draw that display type. The DU must also perform some calculations to correctly draw the display type. For example, the gauge display type has a needle that must be positioned correctly based on the limits of the gauge and the current value of the gauge in CVM. The DU must perform the appropriate calculations to correctly position the gauge needle. In this manner the DU uses CVM to correctly display gauges, meter bars, switches, indicators, and any other display type that changes based on its associated SYSID value in CVM.

The SW/IND task contains two separate processors: the switch processor (SW) and the indicator processor (IND). The SW/IND task executive uses information in executive shared memory to determine which process to execute when it is signaled to go by EXEC. SW/IND is the only task in the real-time system that must be hard coded with SES cupola specific functions.

The SW is responsible for resolving switch selection in the SES cupola. It is through the SW that mechanical devices such as rotary switches are duplicated in software. For example, a bank of switches on a display window may have the implied function that no two switches may be selected at any one time (a rotary switch). The SW resolves which switch has been selected and deselects all of the other switches in the bank.

The IND works closely with the SW to perform hardware functions in software. Like the SW, the IND must resolve some banks of indicators where only one indicator in the bank may be active at one time. However, the IND also deciphers the time data from the simulation computers to correctly display mission elapsed time and Greenwich Mean time.

The PDP task is responsible for processing input from the PDPs and updating the PDP displays. Upon initialization the PDP task reads a data file designed to establish the PDP configuration. As with the display files, it is the PDP data file that defines the PDP configuration for the SES cupola; the real-time PDP software is generic and simply responds to the data file. The PDP data file defines which switches are momentary (active only when depressed) and which change state on each depression. The data file also defines the PDP tree structure. For example, one PDP may be used to reconfigure an entire bank of PDPs.

The HSD task is responsible for communications

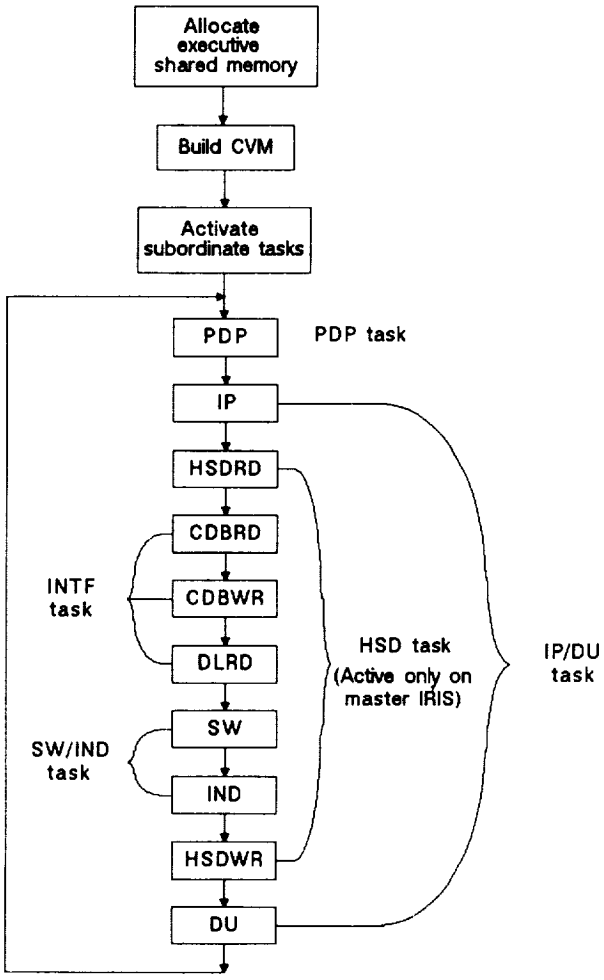


Figure 7
Executive Task Flow
and Process Scheduling

with the HSD interface. This task is active only on the master IRIS and has two separate processors: the HSD read processor (HSDRD) and the HSD write processor (HSDWR). The HSD task executive uses information in executive shared memory to determine which process to execute when it is signaled to go by EXEC.

As stated previously the IRIS computers operate asynchronously from each other and all other computers. Therefore, information from the simulation computers is used only when the master IRIS asks for it. Timing data indicates that in general the master IRIS requests information more often than the simulation computers are prepared to offer it. The HSDRD retrieves a buffer of downlinked information from the simulation computers if available. If a buffer of data is received then it is immediately broadcasted to all IRIS units on the ethernet so that each unit may properly update CVM.

Upon initialization the HSDWR reads into dynamically allocated memory the uplink parameter file built from the Informix database explained earlier. The HSDWR uses the uplink parameter table to map information from the master IRIS CVM into a data buffer that the simulation computers will understand. This data buffer is then sent to the simulation computers. It is important to note that only the master IRIS CVM is used as the source to build the uplink buffer. The INTF task is responsible for making the CVM on each IRIS machine identical.

The INTF task works closely with the HSD task in the area of communication. However, while the HSD task is most concerned with the HSD interface, the INTF task is involved solely with the ethernet interface. The INTF task contains three separate processors: the downlink read processor (DLRD), the CDB write processor (CDBWR), and the CDB read processor (CDBRD). The INTF task executive uses information in executive shared memory to determine which process to execute when it is signaled to go by EXEC.

Upon initialization the DLRD reads into dynamically allocated memory the downlink parameter file built from the Informix database explained earlier. When the HSDRD routine broadcasts the buffer of downlinked data from the simulation computers, the DLRD retrieves that buffer of data. The DLRD then uses the downlink parameter table to map the data from the downlink buffer into CVM.

The CDB is the real-time systems method of porting changes made to one IRIS' CVM to all other IRIS units on the ethernet. For example, during SES cupola operations, a crew person at one station selects a switch. The crew person at the other station expects his display to reflect that switch selection. This is accomplished through the CDB. The CDBWR transmits the CDB across the ethernet to other IRIS units, and the CDBRD receives the CDB from other IRIS units and updates CVM.

FUTURE SES CUPOLA CONSIDERATIONS

As stated earlier, one of the objectives of the SES cupola is to provide to the engineering community a tool for development of the space station cupola. As the hardware design of the cupola changes, the SES cupola hardware will undergo incremental changes. Also a dome visual system is planned for the SES cupola in the future adding another dimension of realism to man-in-the-loop simulation.

By its very nature, software is more flexible than hardware. As this paper has demonstrated, the SES cupola real-time system was designed for flexibility and change. The offline software tools (display builder and relational database) are integral parts of the real-time systems built in flexibility.

With these concepts in mind the future of the SES cupola is bright. Currently OMV simulation in the SES is undergoing validation. The SES cupola crew station will be used as both a ground based con-

trol station for OMV operations and a space based control station. Likewise, the MSC is currently being implemented in the SES. Many of the MSC operations will be developed and analyzed from the SES cupola. Beginning in April 1989, McDonnell Douglas will utilize the SES cupola for displays and controls development. A number of requests concerning software modification have been made by McDonnell Douglas, and those changes are currently in work.

It is through its flexibility and ability to adapt to the needs of the sponsor that the SES in general becomes an excellent engineering tool. The SES has been directed to be the primary real-time man-in-the-loop engineering simulation facility for support of the Space Station Program. The SES cupola is a precise and visible attempt to meet that directive.

REFERENCES

1. St. John, Robert H.; Moorman, Gerard J.; and Brown, Blaine W., "Real-time Simulation for Space Stations", Proceedings of the IEEE, Vol. 75, No. 3, March, 1987, pp. 383-398.

2. "Systems Engineering Simulator (SES) Laboratory Description Document: Simulator Foundation", Vol. 1, LEMSCO-23181, December, 1987.

3. "Systems Engineering Simulator (SES) Laboratory Description Document: On-Orbit Element Simulator", Vol. 3, LEMSCO-23183, April, 1988.

Appendix A

SES Cupola Display Type Descriptions

The IRIS 4D supports the SES Cupola through display windows on the multi-purpose applications console (MPAC) as well as keyboard and trackball (or mouse) input. The display windows respond to the SES via a data buffer through M1. Variables downlinked from the SES to the IRIS are displayed in various formats depending on the display type used. Likewise, inputs from the crewperson are interpreted by the IRIS and uplinked to the SES.

Everything shown on a display window is a display type. All variable data as well as static data is defined in terms of display types. So a display type is simply a functional unit on the display. Variable data display types are used to display data from the SES. These display types tag a unique number termed a system identification (sysid) to their variable so that the real-time SES Cupola software can keep track of variables passed to and from the SES as well as internal variables. The following is a description of display types required for the SES Cupola MPAC.

2D Header Entry - The header entry is at the beginning of every display file that describes a MPAC display window. It contains data that describes the window as a popup window, banner window, or data window. A banner window is a unique window in the real-time system that covers the entire display screen and cannot be popped to the foreground, moved, or deleted. A popup window usually contains a number of page calls (described later) to bring up data windows of related data. Data windows present data to the crewperson. The font used to display text is determined by the font flag. The header entry also contains the length of the window data display file in bytes, the X and Y coordinates of the window origin, and the X and Y length of the window in pixels.

3D Header Entry - The 3D header entry is at the beginning of every display file that describes a MPAC 3D display window. All 3D display windows are data windows. The font used to display text is determined by the font flag. The 3D header entry contains the length of the window data display file in bytes, the X and Y coordinates of the window origin, and the X and Y length of the window in pixels. The distance of the eyepoint from the origin, near clip plane, and far clip plane distances are specified. Finally, the perspective angle and z-buffer flag are specified.

Begin Coordinate Frame - The begin coordinate frame display type is associated with 3D display windows. This display type causes all subsequent 3D objects to be drawn relative to the relocated local origin as specified by the six data variables. Six sysids tag data variables for X, Y, and Z position and X, Y, and Z rotation. A display type name is also specified.

End Coordinate Frame - The end coordinate frame display type is associated with 3D display windows. This display type cancels the coordinate frame display type and the local origin is returned to the previous global origin. All coordinate frames must be terminated by an end coordinate frame. A display type name is specified.

Rotation - The rotation display type is associated with 3D display windows. This display type allows the user to rotate the eyepoint about the global origin. The X, Y, and Z rotations are specified deltas. A sysid tags the data variable that is used to determine if the eyepoint is to be rotated. A display type name is also specified.

Translation - The translation display type is associated with 3D display windows. This display type allows the user to translate the eyepoint along the X, Y, and Z axis a specified distance. The X, Y, and Z translations are specified deltas. A sysid tags the data variable that is used to determine if the eyepoint is to be translated. A display type name is also specified.

Scale - The scale display type is associated with 3D display windows. This display type allows the user to scale a pre-defined amount about the origin along any or all axes. The X, Y, and Z scale factors are specified deltas. A sysid tags the data variable that is used to determine if the display is to be scaled. A display type name is also specified.

End of List Terminator - The end of list terminator is at the end of every display

file that describes a SES Cupola MPAC display window. The sole purpose of the end of list terminator is to flag the end of the display list.

Integer - The integer display type is used to display integer data on the display window. A sysid tags the data variable. The X and Y window coordinates of the display type and field width are specified. A yellow (warning) threshold and red (critical) threshold are available. The integer is also described as increasing or decreasing so that the thresholds reside at the upper or lower end of the expected range. Normally the data is displayed in white. If the value crosses the warning threshold the data is displayed in yellow. Likewise, if the data crosses the critical threshold the data is displayed in red. Thresholds are optional and both thresholds do not have to be used.

Real - The real display type is used to display floating point data on the display window. A sysid tags the data variable. The X and Y window coordinates of the display type, total field width, and number of digits to the right of the decimal point are specified. A yellow (warning) threshold and red (critical) threshold are available. The real number is also described as increasing or decreasing so that the thresholds reside at the upper or lower end of the expected range. Normally the data is displayed in white. If the value crosses the warning threshold the data is displayed in yellow. Likewise, if the data crosses the critical threshold the data is displayed in red. Thresholds are optional and both thresholds do not have to be used.

Long Float - The long float display type is used to display double precision floating point data on the display window. A sysid tags the data variable. The X and Y window coordinates of the display type, total field width, and number of digits to the right of the decimal point are specified. A yellow (warning) threshold and red (critical) threshold are available. The double precision number is also described as increasing or decreasing so that the thresholds reside at the upper or lower end of the expected range. Normally the data is displayed in white. If the value crosses the warning threshold the data is displayed in yellow. Likewise, if the data crosses the critical threshold the data is displayed in red. Thresholds are optional and both thresholds do not have to be used.

Hexidecimal - The hexidecimal display type is used to display a 32-bit word in memory on the display window in the form of a hexidecimal number. A sysid tags the data variable. The X and Y window coordinates of the display type and color are specified.

Ascii Message - The ascii message display type is used to display ascii text that is variable such as error messages on the display window. A sysid tags the ascii data. The X and Y coordinates of the display type as well as color are specified.

Static Text - The static text display type is used to display ascii text that is static such as labels on the display window. The X and Y coordinates of the display type as well as color are specified. The character string is a maximum of 8 characters in length.

Page Call - The page call display type is used to "call up" new display windows for the MPAC. The display type bounds (left X, right X, bottom Y, top Y), color, and text are specified. A filename is specified to indicate the display file to be read.

Keyboard Input - The keyboard display type allows user input from the keyboard. The display type bounds (left X, right X, bottom Y, top Y), color, and text are specified. A filename is specified to indicate the display file to be read.

Delete - The delete display type allows the user to delete a display window in real time. The display type bounds (left X, right X, bottom Y, top Y) are specified.

Default - The default display type allows the user to save a screen configuration of several display windows and recall that particular configuration at some later time. The display type bounds (left X, right X, bottom Y, top Y) as well as the middle Y position and color are specified. Text labels for the default and save default portions of the display type are specified. Finally, the name of the save file is specified.

Dynamic Position Indicator - The dynamic position indicator is a cursor on a bar. The position of the cursor is determined by the associated data variable and the specified upper and lower limits of the indicator. A sysid tags the data variable. Another sysid tags a visibility flag which is used to determine if this display type is drawn. The cursor may take the following forms: empty square with "X", filled square, empty circle with cross-hair, filled circle, empty triangle, filled triangle, caret, or cross-hair. The bar may be vertical or horizontal. The display type bounds (left X, right X, bottom Y, top Y), bar color, and cursor color are specified. The upper and lower

limits are specified.

Circular Gauge - The circular gauge display type is used to display floating point data in the form of a gauge on the display window. A sysid tags the data variable. The X and Y window coordinates of the display type, total field width, and number of digits to the right of the decimal point are specified. The upper and lower limits of the gauge are specified. A yellow (warning) threshold and red (critical) threshold are available. The gauge is also described as increasing or decreasing so that the thresholds reside at the upper or lower limits of the gauge. Normally the data is displayed in white. If the value crosses the warning threshold the data is displayed in yellow. Likewise, if the data crosses the critical threshold the data is displayed in red. Thresholds are optional and both thresholds do not have to be used.

Meter Bar - The meter bar display type is used to display floating point data in the form of a meter bar on the display window. A sysid tags the data variable. The display type bounds (left X, right X, bottom Y, top Y), total field width, and number of digits to the right of the decimal point are specified. The upper and lower limits of the meter bar are specified. A yellow (warning) threshold and red (critical) threshold are available. The meter bar is defined as horizontal or vertical and with or without threshold and limit labels. The meter bar is also described as increasing or decreasing so that the thresholds reside at the upper or lower limits of the meter bar. Normally the data is displayed in white. If the value crosses the warning threshold the data is displayed in yellow. Likewise, if the data crosses the critical threshold the data is displayed in red. Thresholds are optional and both thresholds do not have to be used.

Indicator and Rounded Indicator - The indicator display type represents a mechanical light indicator on the display window. A sysid tags the data variable. The display type bounds (left X, right X, bottom Y, top Y) are specified. Also, the "true" state text, text color, and background as well as the "false" state text, text color, and background are specified. If the variable associated with the indicator is 0 (False) the "false" text, text color, and background are displayed. Any other value is considered true, and the "true" text, text color, and background are displayed. The rounded indicator display type has rounded ends.

Switch - The switch display type represents a mechanical 2-way toggle switch on the display window and is drawn to create the illusion of a 3-D push button. An additional feature of the switch display type is that an indicator can be incorporated into the switch. A sysid tags the switch data variable, and another sysid tags the indicator data variable. The display type bounds (left X, right X, bottom Y, top Y) are specified. Also, the "true" state text, text color, and background as well as the "false" state text, text color, and background are specified. A transition color is specified if the indicator option is used. The truth table below indicates the state of the switch based on the value of the data variables. Note that if the switch sysid (ss) and indicator sysid (is) are identical the switch acts as a 2-way toggle. If the two sysids are different then the switch has a transition state.

switch sysid = indicator sysid

ss	is	position	color
0	0	up	false
1	1	down	true

switch sysid <> indicator sysid

ss	is	position	color
0	0	up	false
0	1	up	true
1	0	down	transition
1	1	down	true

Momentary Switch - The momentary switch display type represents a mechanical 2-way toggle momentary switch on the display window and is drawn to create the illusion of a 3-D push button. An additional feature of the display type is that an indicator can be incorporated into the momentary switch. A sysid tags the switch data variable, and another sysid tags the indicator data variable. The display type bounds (left X, right X, bottom Y, top Y) are specified. Also, the "true" state text, text color, and background as well as the "false" state text, text color, and background are specified. A transition color is specified if the indicator option is used. The truth table below indicates the state of the momentary based on the value of the data variables. Note that if the switch sysid (ss) and indicator sysid (is) are identical the momentary acts as a 2-way toggle. If the two sysids are different then the momentary has a

switch sysid = indicator sysid

ss	is	position	color
0	0	up	false
1	1	down	true

switch sysid <> indicator sysid

ss	is	position	color
0	0	up	false
0	1	up	true
1	0	down	transition
1	1	down	true

Line - The line display type draws a line on the display window. The starting X and Y coordinates, ending X and Y coordinates, and color are specified.

Rectangle - The rectangle display type draws a rectangle on the display window. The display type bounds (left X, right X, bottom Y, top Y) and color are specified. The rectangle can be filled or empty.

Circle - The circle display type draws a circle on the display window. The display type X and Y coordinates, radius, and color are specified. The circle can be filled or empty.

Polygon - The polygon display type draws a polygon on the display window. The polygon can have up to and including 10 vertices. The number of vertices, all (X, Y) coordinate pairs, and color are specified. The polygon can be filled or empty.

3D Line - The 3D line display type draws a line in three dimensional space and is used on 3D display windows. Starting X, Y, Z and ending X, Y, Z coordinates are specified. A display type name and color are specified also.

Box - The box display type draws a box on the 3D display window. The center X, Y, and Z coordinates; height and width on the -X and +X ends of the box; and length are specified. An offset along the Y or Z axis may be specified to shift the front face of the box. Rotations about the three axes may be specified to orient the box. A display type name and color are specified also.

Cylinder - The cylinder display type draws a cylinder on the 3D display window. The X, Y, and Z coordinates; diameter at the -X and +X ends of the cylinder; and length are specified. The number of sides and angle of rotation (full cylinder = 360, half cylinder = 180, etc.) are specified. Rotations about the three axes may be specified to orient the cylinder. A display type name and color are specified also.

Sphere - The sphere display type draws a sphere on the 3D display window. The X, Y, and Z coordinates of the center, radius, and number of sides are specified. A display type name and color are specified also.

THE ORBITAL MANEUVERING VEHICLE TRAINING FACILITY VISUAL SYSTEM CONCEPT

KeithWilliams
CAE-Link Corporation
Link Flight Simulation Division
2222 Bay Area Blvd.
Houston, Texas 77058

ABSTRACT

The purpose of the Orbital Maneuvering Vehicle (OMV) Training Facility (OTF) is to provide effective training for OMV pilots. A critical part of the training environment is the Visual System, which will simulate the video scenes produced by the OMV Closed-Circuit Television (CCTV) system. The simulation will include camera models, dynamic target models, moving appendages, and scene degradation due to the compression/decompression of video signal. Video system malfunctions will also be provided to ensure that the pilot is ready to meet all challenges the real-world might provide. This paper describes one possible visual system configuration for the training facility that will meet existing requirements. This paper reflects work performed for NASA by CAE-Link Corporation.

INTRODUCTION

The OTF visual system must provide the CCTV capabilities at a cost-effective price. The scene content update rate is only 5 times per second with a low-resolution requirement. This enables the use of a high-end super-graphics workstation as the medium for the CCTV simulation. Combining the CCTV simulation with the full-feature OTF simulation maximizes pilot training. To further enhance training capabilities, stand-alone and integrated modes will challenge the pilot with limited and full-mission scenarios.

The stand-alone mode provides the pilot with a partial-task, one-on-one training environment that guides the pilot's progress in a systematic manner. Integrated mode allows the linking of the OTF with several simulators at NASA, Johnson Space Center (JSC). Integrated mode will challenge the pilot to apply the lessons learned from the stand-alone sessions with new and more difficult mission objectives. Both modes enable the pilot to handle any situation that could possibly occur in an actual mission. The functional design diagram for the OTF Visual System (figure 1) shows the relationship of the host computer with the Image Generation system and also depicts the two configurations available for the integration with respect to the Visual System.

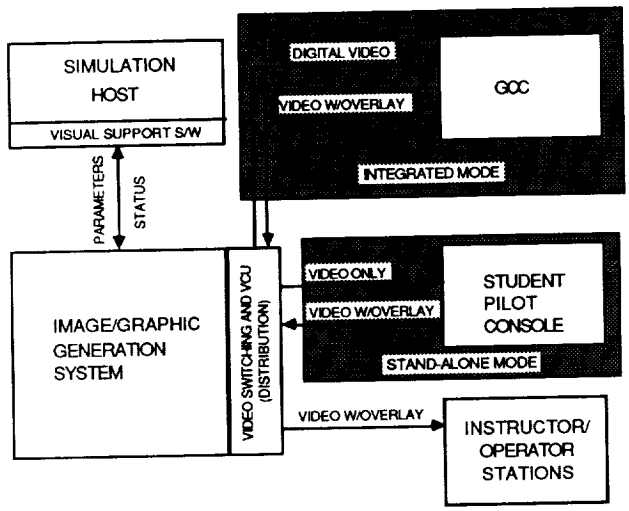


FIGURE 1 OTF Visual System Functional Design Diagram

The OMV is a remotely-piloted spacecraft. Currently defined mission scenarios include rendezvous and docking with satellites, the Orbiter, and the Space Station. To provide training for these missions, a simulation environment is being developed to train the

pilot to interact with the OMV. Before detailing the OTF Visual System, an explanation of the real-world OMV is in order. Familiarity with the actual system is essential to understand the training requirements and how our functional simulation of these systems will provide effective training for OMV pilots. A basic functional diagram of the OMV CCTV system is presented in Figure 2. It provides the data flow from the time an image is captured by the CCTV camera, to when the image appears on the pilot's display.

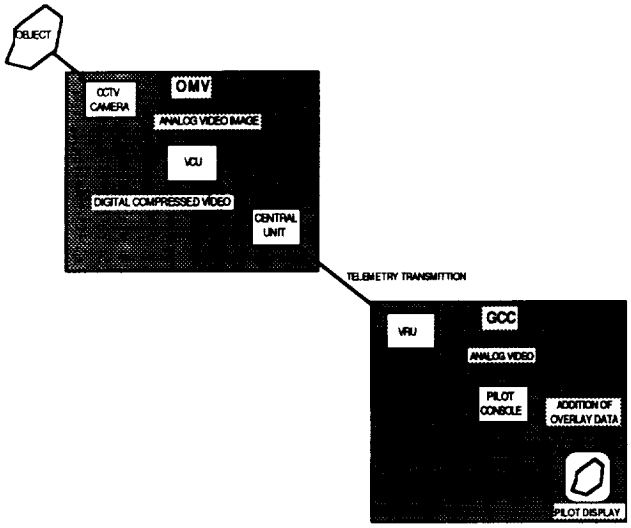


FIGURE 2 OMV CCTV Functional Diagram

OMV VIDEO SYSTEM

The OMV has two independent and redundant camera systems: docking and pan/tilt/zoom. Lighting equipment is associated with each camera system. Four additional cameras can be attached to OMV payloads, making a maximum of eight cameras allowed. Camera images are sent to the pilot at the Ground Control Console (GCC) in the JSC Mission Control Center (MCC) via the Tracking and Data Relay Satellite System (TDRSS). The camera images are compressed and placed into the OMV telemetry stream. The bandwidth available for the video telemetry is only 972 kbps, or 5 video frames per second. The 972 kbps telemetry data rate can be allocated to two cameras at 486 kbps each or can be dedicated to a single camera. At MCC, the GCC decompresses the telemetry and performs error checking. The video image is then combined with an overlay of flight-critical data and displayed to the pilot.

Camera System

The pan/tilt/zoom camera is a redundant system with a 6:1 zoom ratio and is typically used by the pilot for initial acquisition of the target vehicle and for the initial stages of OMV docking. The docking camera is also a redundant system and is mounted on the OMV docking axis. This permits the pilot a boresight view of the target docking mechanism alignment with the OMV grapple mechanism. Two redundant docking lights are included with each camera system for a total of four lights.

Video Compression Unit

The Video Compression Unit (VCU) compresses the video images. The VCU utilizes a frame-grabbing technique to acquire 5 frames of RS-170 video data per second in normal operational modes. Compression and Huffman and Reed-Solomon encoding are performed on each frame of data. In the event that a single docking light is the only available light source, the pilot can select an extended imaging mode that extends the usable camera range to 200 feet. The extended imaging mode increases the VCU video sample rate and combines multiple frames of data into a single enhanced image. This is analogous to increasing the exposure time of a photograph in a camera, allowing the film to receive multiple images that combine for a single photograph. The VCU also provides the capability to send memory dumps from the OMV Command and Data Management systems to the GCC.

Video Reconstruction Unit

The Video Reconstruction Unit (VRU) decompresses the digitized video images. Huffman and Reed-Solomon decoding is also performed. The VRU has an additional unit attached called the Bit Error Rate Monitor (BEM), which provides verification of pixel count, line count, and correct subframe sequence in the video frame. The BEM replaces sections of corrupted data with data from the previous frame. The pilot can increase the number of reference pixels, which lowers the resolution of the image being transferred and reduces the amount of corrupted data. Not only will the granularity of the picture increase but also the validity of the image.

Ground Control Console

All commanding of the OMV is done via the GCC. The pilot has a redundant station with a keyboard, two cathode-ray tubes (CRTs), and rotational and translational hand controllers. Preprogrammed commands are entered via the operator's workstation. The GCC workstation receives OMV telemetry and displays it to the pilot. The pilot console receives the RS-170 output from the VRU and adds the overlays that contain flight-critical data. This output is then displayed on the CRTs. The video image has a resolution of 510 by 244 pixels for a single camera image or 255 by 244 for two cameras.

OTF VISUAL SYSTEM CONCEPT

The OTF Visual System must provide two separate modes of operation: stand-alone and integrated. In each of these modes, the camera system of the OMV must be simulated. From these CCTV models and from target control data, the Image Generation System will generate the representative scene to be displayed for the OMV pilot.

Basic requirements for the Visual System include simulation of the CCTV system, transport of visual data to MCC, decoding of the data, and addition of flight-critical parameters to the display for the pilot. The OTF Visual System will use a combination of hardware and software on two different computer systems. One computer will provide modeling information while the other will transform that information into a graphic representation. An image generation computer will produce the CCTV camera scenes for the pilot with a host computer controlling the simulation models of the vehicle camera systems. The host computer also provides all commanding of the Image Generation System. All database modeling will be performed on the Image Generation System.

Training Configurations

Providing both stand-alone and integrated simulation capabilities is required to supply various levels of training. During initial pilot training, the stand-alone mode allows instructors to remain in close proximity to the student pilot. Instruction on basic system operation and scenarios is given. Partial task/mission training is also possible. The OTF Visual System will provide all nominal system capabilities. In this mode, no effects caused by telemetry degradation or compression/decompression of the video signal will be simulated.

The integrated mode connects the OTF with the Shuttle Mission Simulator (SMS), Network Simulation System (NSS), and MCC. This mode provides full mission scenario training and refines the pilot's proficiency and skills. The OMV pilot is placed in situations as close as possible to an actual flight, from prelaunch to Orbiter retrieval. All interactions with MCC and Shuttle personnel occur as they would in an actual flight.

Both modes must provide image generation, CCTV, and target control. The differences lie in the distribution of the video image to the pilot, the GCC, or the student pilot console.

Video Distribution

The OTF Visual System produces the raw video image as directed by the host simulation. This video is in RS-170 format to remain compatible with the pilot console/student pilot console hardware.

Integrated Mode: A VCU is used on the OMV to convert and compress the video image into digital telemetry. A non-flight-rated version of the VCU is used in the integrated mode to perform Reed-Solomon encoding and compression of the Image Generation System video. The VCU outputs this data to the Data Acquisition System (DAS), which places the video into the OMV downlink telemetry.

The real-world pilot console is used in integrated training. The pilot's visual hardware consists of a VRU, a frame grabber and graphics generator, and two graphics CRTs. The VRU accepts digital video from the telemetry network and converts the digital compressed signal to RS-170 format. The frame grabber on the GCC acquires the image and adds the overlay of flight-critical data. This composite image is then displayed for the pilot.

By using the real-world VCU and VRU, additional capabilities are available for training. This includes command and data handling memory dumps, BEM effects, and telemetry degradation effects.

Stand-alone Mode: In the stand-alone mode the output of the Image Generation System is directly routed to the student pilot console; only

the addition of switching capabilities and signal amplifiers is necessary. No telemetry degradation and no compression/decompression effects are simulated in the stand-alone mode. This approach reduces the complexity and cost of the OTF simulator.

OMV Host Computer

The OMV host computer has several resident math models. These models include simulation of the OMV environment and onboard systems. The OTF Visual System encompasses the following areas:

- CCTV
- Visual Real-Time Support
- Visual Mode and Control
- Visual Special Effects

Closed-Circuit Television

The CCTV models dynamically simulate the CCTV camera system on the OMV. All telemetry data is passed from the camera systems to the central unit. The central unit then places the data into the telemetry stream that is sent to MCC. The central unit also passes this information to the redundancy management unit for wellness checks and appropriate self-reconfiguration in event of correctable malfunctions. The CCTV camera system consists of the pan/tilt/zoom camera, docking camera, docking lights, and the VCU.

Pan/Tilt/Zoom Camera Model Functions

1) Thermal effects are modeled to provide telemetry data to the central unit.

2) The electrical system is modeled to provide status information to the central unit. Electrical power consumption data is provided to the OMV onboard systems electrical system model. This model provides all power-available data for the CCTV model.

3) The camera gimbal control respond to command data from the GCC. These commands generate data that is sent to the image generator. The mechanical and electrical dynamics are simulated. The net effect is the movement of the simulated camera.

4) Gamma, focus, and iris control from the GCC are simulated as near to the real-world as is possible with the image generation hardware.

Docking Camera Model Functions

1) Thermal effects are modeled to provide telemetry data to the central unit.

2) The electrical system is modeled to provide status information to the central unit. Electrical power consumption data is provided to the OMV onboard systems electrical system model. This model provides all power-available data for the CCTV model.

3) Gamma, focus, and iris control from the GCC are simulated as near to the real-world as is possible with the image generation hardware.

Docking Lights

1) The luminosity control commands for the docking lights will be sent to the Image Generation System.

2) Thermal effects are modeled to provide heat transfer information to the camera thermal models.

3) The electrical system is modeled to provide status information to the central unit. Electrical power consumption data is provided to the OMV onboard systems electrical system model. This model provides all power-available data for the CCTV model.

Telemetry outputs deemed necessary for training but not previously defined will be provided.

Processing of malfunctions will be provided at the level of detail specified in the Level B requirements.

The host computer (Concurrent 3280) contains mathematical models for the OMV and its environment. These models include the CCTV system and the control of free-flying targets. The host computer models propagate all state vectors for the OMV and the free-flying targets. When the CCTV system is in view of a free-flying object, commands are given to the Image Generation System to place the target at the

given state vectors. All camera parameters (field of view, focus, iris, gamma correction, and lighting control) are sent with the state vector data to the Image Generation System.

Camera focus is required for OMV training. For the systems investigated, no focus or blur commands were available in an off-the-shelf product. A focus algorithm will be derived using pixel pairing or filter algorithms.

Visual Real-Time Support Software

The Visual Real-Time Support software provides commands for the image generator. OMV target and Earth/Moon/Sun information and data are processed. This process includes the conversion of all Concurrent floating point numbers to industry standard Institute of Electrical and Electronics Engineers (IEEE) floating point format.

a) Earth/Moon/Sun

   1) State vector IEEE conversions must be performed for the Image Generation System

   2) The luminosity of the Sun must be set to provide correct representation of the day/night terminator and shading of objects.

b) OMV Visual Model and Target Control

   1) State vector IEEE conversions must be performed for the Image Generation System

   2) Appendages are commandable with representative visual cues reflecting the actions.

   3) Navigation lights are represented as polygons and do not add any shading or luminosity effects.

Visual Mode and Control

The visual mode and control software provides the functions necessary to maintain the simulation modes (run, freeze, data store, and return to data store). This software commands the following subsystems:

a) Image Generation System - The image generation mode and control also includes the model selection and image generation initialization.

b) Video Distribution System - The video distribution mode and control configures the distribution hardware with predefined parameters for the mode selected. It also allows system reconfiguration as needed when the simulation is in freeze mode. Freeze mode allows the simulator to halt and suspend all integrations, as if time has stopped inside the simulator.

c) Video Compression Unit - The VCU mode and control software initializes and modes the VCU hardware (only in integrated simulation mode). A representative model of the VCU is used in the stand-alone mode.

Visual Special Effects

All special effects (such as focus and radio frequency interference (RFI) noise) hardware will be controlled by the visual special effects software. Only the use of step attenuators to degrade the OMV telemetry stream when sent to the GCC in MCC is planned.

Image Generation

The image generation software is required to produce a new scene 5 times per second. With this low scene content update rate, it is possible to use a high-end super-graphics workstation. We estimate that 35,000 four sided polygons per second are required for a 5 hertz update rate. This estimate was produced by using existing SMS and space model databases. The scene content must include the Sun/Earth/Moon and the possibility of four-free flying targets.

Image Generation Software

The image generation software provides the following capabilities:

- Network connectivity
- Initialization mode processing
- Message processing
- Sequencing
- Screen-application processing

The workstation also provides all capabilities for database model generation.

CONCLUSIONS

The OTF provides an effective training environment for the OMV pilots. Training flexibility is achieved using the stand-alone and integrated modes. In stand-alone mode the pilot is introduced to the basic handling capabilities of the OMV. The pilot can then proceed to basic procedures and scenarios and be challenged by instructor-inserted malfunctions. In the integrated mode, the pilot is integrated into the NASA team and learns to work with all other MCC ground controllers and Shuttle personnel. Enhanced capabilities are added to the VCU and VRU within the command and data handling simulation. The capability to degrade the video image proportionally to the amount of telemetry degradation is inherent in the system and is supported by the visual cues the pilot receives as a result of his commands. These reactions in combination with the capabilities described above provide a realistic and effective training environment for the OTF.

REFERENCES

OMV Level A Training Requirements, NASA Lyndon B. Johnson Space Center, Houston, Texas, April 4, 1988

OMV Preliminary Design Review Volume 4 Avionics Part 3 Communications and Data Management, NAS8-36800, NASA, George C. Marshall Space Flight Center, Alabama, August, 1988

System Functional Requirements for the Orbital Maneuvering Vehicle Training Facility, JSC-22976, NASA Lyndon B. Johnson Space Center, Houston, Texas, December 19, 1988

THE SEARCH FOR REPLACEMENT VISUAL SYSTEMS FOR THE SHUTTLE MISSION TRAINING FACILITY (SMTF)

Mark Teigler
CAE LINK FLIGHT SIMULATIONS

(Paper not provided by publication date.)

AUTHOR INDEX

NASA
National Aeronautics and Space Administration

REPORT DOCUMENTATION PAGE

1 Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CP-3045		

4 Title and Subtitle	5. Report Date
Graphics Technology in Space Applications (GTSA 1989)	August 1989
	6. Performing Organization Code
7 Author(s)	8. Performing Organization Report No.
Sandy Griffin, Editor	S-594
9 Performing Organization Name and Address	10. Work Unit No.
Lyndon B. Johnson Space Center Houston, Texas 77058	11. Contract or Grant No.
12 Sponsoring Agency Name and Address	13. Type of Report and Period Covered
National Aeronautics and Space Administration Washington, D.C. 20546 University of Houston-Clear Lake	Conference Publication
	14. Sponsoring Agency Code

15 Supplementary Notes

16. Abstract

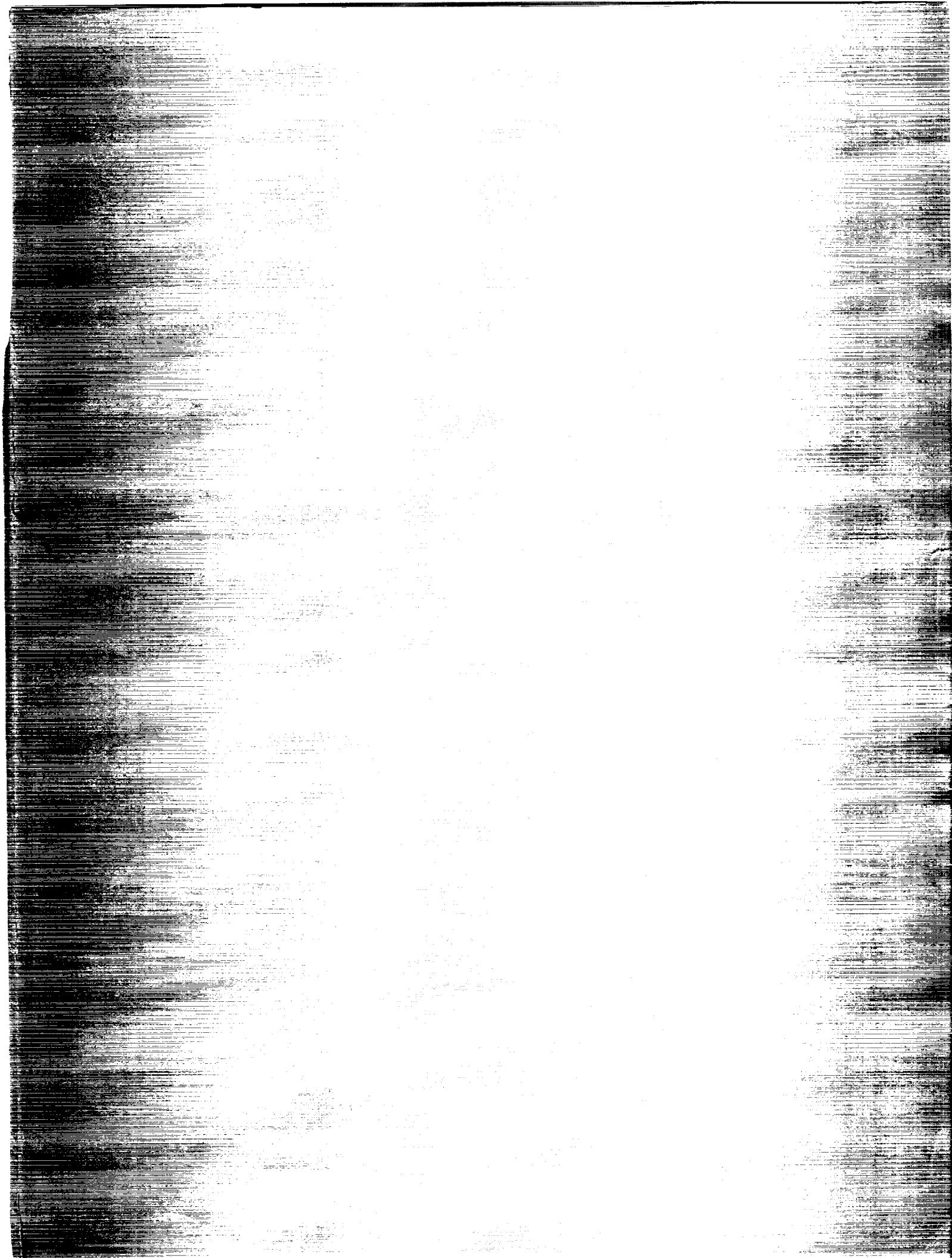
Papers presented at the Graphics Technology in Space Applications hosted by University of Houston-Clear Lake at Johnson Space Center on April 12, 13, and 14, 1989 are documented herein. During the three days, approximately 35 papers were presented. Technical topics addressed included Graphics Standards, Graphics Applications, and Tools, Merging of Graphics and Video Display Technology, Partial Task and Stand Alone Simulations, Space Station Freedom Graphics and Large Scale Simulations.

17 Key Words (Suggested by Author(s))	18. Distribution Statement
3-D Scene Description, photorealistic, display building tool, Binary Space Partitioning, image generator, real-time simulation, computational proceedings, AUTOPS, high-fidelity, solid modeled images	Unclassified - Unlimited Subject Category: 60

19 Security Classification (of this report)	20 Security Classification (of this page)	21. No. of pages	22 Price
Unclassified	Unclassified	248	A11

For sale by the National Technical Information Service, Springfield, VA 22161-2171

This publication was printed at the UNICOR Print Plant,
Federal Correctional Institution, Petersburg, VA



National Aeronautics and
Space Administration
Code NTT-4

Washington, D.C.
20546-0001

Official Business
Penalty for Private Use $300

NASA

National Aeronautics and
Space Administration

Washington, D.C.
20546

SPECIAL FOURTH CLASS MAIL
BOOK

Postage and Fees Paid
National Aeronautics and
Space Administration
NASA-451

Official Business
Penalty for Private Use $300

U.S. MAIL

NASA
SCIEN & TECH INFO FACILITY
ACCESSIONING DEPT
P O BOX [illegible] BWI ARPRT
BALTIMORE MD 21240