N92-24418

GRAPEVINE: GRIDS ABOUT ANYTHING BY POISSON'S EQUATION IN A VISUALLY INTERACTIVE NETWORKING ENVIRONMENT*

Reese L. Sorenson
NASA Ames Research Center
Moffett Field, CA

Karen McCann
Sterling Software
Palo Alto, CA

SUMMARY

A proven three-dimensional multiple-block elliptic grid generator, designed to run in "batch mode" on a supercomputer, is improved by the creation of a modern graphical user interface (GUI) running on a workstation. The two parts are connected in real time by a network. The resultant system offers a significant speedup in the process of preparing and formatting input data and the ability to watch the grid solution converge by re-plotting the grid at each iteration step. The result is a reduction in user time and CPU time required to generate the grid and an enhanced understanding of the elliptic solution process. This software system, called GRAPEVINE, is described, and certain observations are made concerning the creation of such software.

INTRODUCTION

Mathematical algorithms for grid generation, especially structured grid generation, are well advanced. But even so it can take six man-months to generate a useable multiple-block grid about a real airplane or any other complex real-world shape, with that effort split between surface modeling, surface gridding, and volume gridding. A major reduction of that discrepancy between algorithms and performance, in the area of volume grid generation, is the subject of this paper.

3DGRAPE[1,2] is an example of grid generation software which is algorithmically mature, but the use of which can account for a generous share of those six man-months. The user's time goes to collecting and formatting input data and to actually running the grid generation program. Input data for such a grid generator (EAGLE[3,4] is another example of the genre) can extend to thousands (or even tens of thousands) of lines for a grid about a realistic shape. A reasonable approach to reducing that burden is to use the graphical power of a workstation to speed-up the processes of collecting and formatting the input data, and of running the program. GRAPEVINE is a software system which attempts to do that.

* Ms. McCann's participation in this project was under contract to Sterling Software, NAS2-13210.

GRAPEVINE consists of the five code modules illustrated in Figure 1. First is a newly-written user-friendly GUI, running on a workstation, which controls the process. Second is 3DECANT, a code module which displays the grid. The third code module is 3DGRAPE, the elliptic multi-block grid generator program. The fourth code module is 3DPREP[5], a recently-written workstation program to speed the process of collecting and formatting the input data for 3DGRAPE. 3DPREP is not yet fully integrated with the other code modules, and so is shown as separated from them in the Figure.

While the GUI, 3DECANT, and 3DPREP are written in C and naturally reside on a workstation, the elliptic solver is written in FORTRAN and must run on a supercomputer due to the large amount of calculation required. Hence the GRAPEVINE system operates across a network, with the C modules running on a workstation and the elliptic solver running concurrently on a supercomputer. The code to effect that communication and control across the network is the fifth module making up GRAPEVINE.

The result is a system which will work as shown in the flowchart in Figure 2. The user first prepares input data in a graphically enhanced and accelerated fashion. The user then decides just what grid surfaces should be viewed during the iteration process and how they should be viewed, and then chooses to do the numerical processing on either the workstation or a supercomputer. He sends that data to the numerical processor and starts it running. The numerical processor notes what very small subset of all the grid data is required each iteration to update the grid plotting, and sends that data back to the workstation each iteration. Thus the grid plot is updated every iteration, allowing the user to watch it as it converges. By using these features the user can allow the solution process to continue, or stop it and change some data, or abort it and start anew. The result is a greatly enhanced understanding of the elliptic grid generation process, and significant savings in CPU time and the user's time. At present the step labeled "Create control scalars," shown in the upper left-hand corner of the Figure, is done in the separate program 3DPREP, or in an editor.

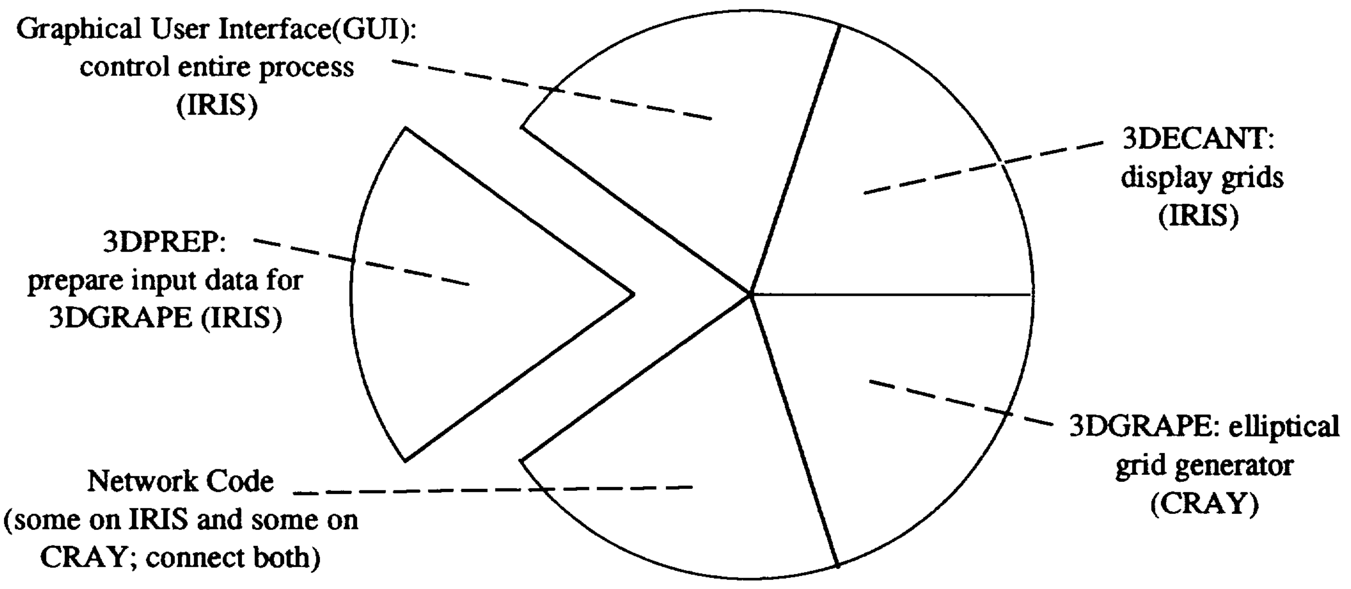

Graphical User Interface(GUI): control entire process (IRIS)
3DPREP: prepare input data for 3DGRAPE (IRIS)
Network Code (some on IRIS and some on CRAY; connect both)
3DECANT: display grids (IRIS)
3DGRAPE: elliptical grid generator (CRAY)

Figure 1. Modules comprising GRAPEVINE.

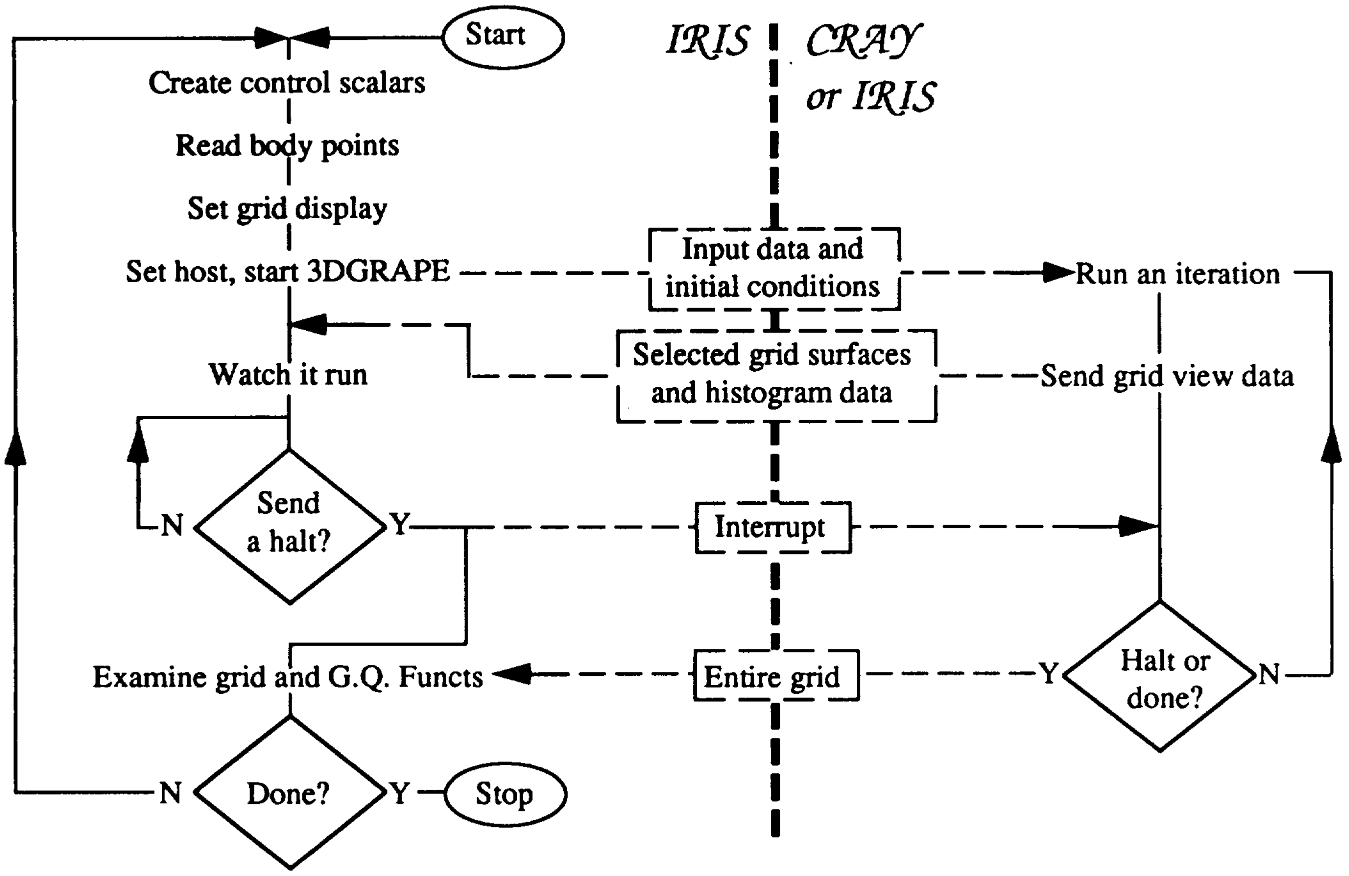


Figure 2. Flow chart summarizing GRAPEVINE operation.

CODE MODULES

GRAPEVINE is a combination of several code modules, some new and some old.

The 3DPREP Module

This module is currently a separate program, running on a workstation, which helps the user collect and format input data for the 3DGRAPE program. It consists of a multiple-screen graphical user interface. It obtains its input data by asking the user questions, or allowing the user to enter input data in a random-access fashion, or allowing the user to read in an old case and then selectively modify it. The module outputs two files (called file10 and file11, file names and file structures which are familiar to users of 3DGRAPE). These files are then used as input data for the 3DGRAPE grid generator. This code module, 3DPREP, is discussed in detail in Ref. 5. Future plans call for integrating its capabilities into GRAPEVINE.

The 3DECANT Module

This module has been fully integrated into GRAPEVINE. It consists of graphics code and transformation algorithms. It allows the user to select the grid surfaces, taken from any block or blocks, which are to be plotted. The user can color the surfaces arbitrarily or by grid quality functions, and do any combination of translation, rotation, and zooming, about either the body's axes or the screen's axes, with the center of rotation being the centroid of that which is being viewed. Various other display options, such as hidden-surface-removal, are included. This module is also discussed in Ref. 5.

The 3DGRAPE Module

This code is the proven three-dimensional multiple-block elliptic grid generator. Given the user's requirement for cell height on boundary surfaces, numerical values for inhomogeneous terms are found iteratively by the code. The result is local near-orthogonality and controlled cell height on boundary surfaces, with those controlling influences decaying exponentially toward the interior of the blocks. It is a multiple-block volume grid generator wherein the block-to-block boundary surfaces are found automatically by the code, alleviating the need for the user to define them before starting. This module is discussed in detail in Refs. 1 and 2. As used in GRAPEVINE the code is improved by better initial conditions, and by an improved treatment of sharp corners in the middle of block faces.

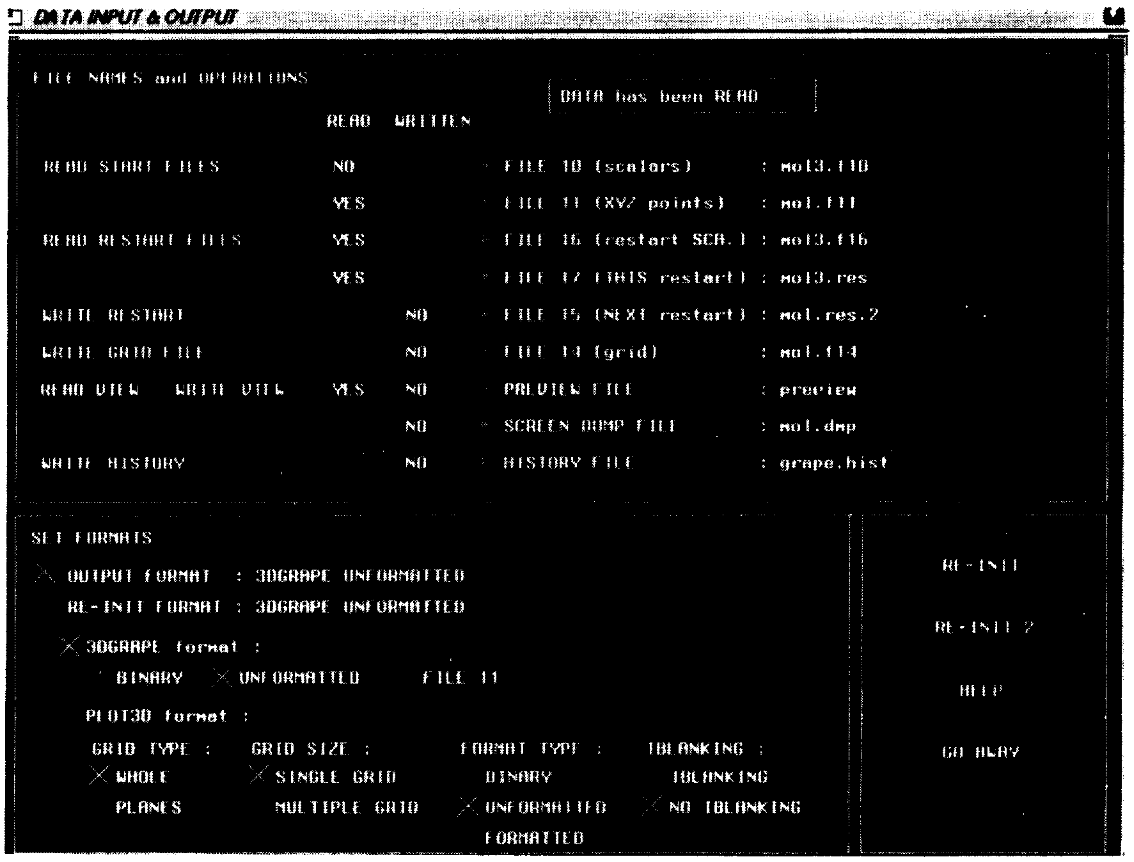
The Graphical User Interface Module

The graphical user interface and the network code are the principal contributions presented in this paper. The graphical user interface consists of several screens.

The Data Input/Output (I/O) Screen

GRAPEVINE begins in the Data I/O Panel, as shown in Figure 3 (all the screen Figures in this paper have been reduced to grayscale, but generous use of color is made in the program). Default file names for input and output are given. The user has the opportunity to overwrite those file names. Upon exit, the current file names are written into an "environment file" which is read upon the succeeding startup. Thus the user's chosen file names in the current run become the default file names for the next run, simplifying operation. The input files include the file10 and file11 input files which are created by 3DPREP. Once the user has entered the necessary file names, he clicks on either the "read start files" button or the "read restart files" button. Those simple operations give GRAPEVINE all the information it needs to generate the grid.

However, the program needs one more input data file, this one telling it how to plot the grid. The creation of this file is discussed below, but once it has been created it is typically just read in and used as



```
DATA INPUT & OUTPUT

FILE NAMES and OPERATIONS                          DATA has been READ
                         READ  WRITTEN

READ START FILES          NO          FILE 10 (scalors)        : mol3.f10
                          YES         FILE 11 (XYZ points)     : mol.f11
READ RESTART FILES        YES         FILE 16 (restart SCR.)   : mol3.f16
                          YES         FILE 17 (THIS restart)   : mol3.res
WRITE RESTART                   NO    FILE 15 (NEXT restart)   : mol.res.2
WRITE GRID FILE                 NO    FILE 14 (grid)           : mol.f14
READ VIEW   WRITE VIEW    YES   NO    PREVIEW FILE             : preview
                                NO    SCREEN DUMP FILE         : mol.dmp
WRITE HISTORY                   NO    HISTORY FILE             : grape.hist

SET FORMATS                                                            RE-INIT
[✓ OUTPUT FORMAT  : 3DGRAPE UNFORMATTED]
   RE-INIT FORMAT : 3DGRAPE UNFORMATTED                                RE-INIT 2
 [✓ 3DGRAPE format :]
    BINARY  [✓ UNFORMATTED]     FILE 11                                HELP
   PLOT3D format :
   GRID TYPE :    GRID SIZE :        FORMAT TYPE :    IBLANKING :      GO AWAY
   [✓ WHOLE]      [✓ SINGLE GRID]     BINARY           IBLANKING
    PLANES         MULTIPLE GRID     [✓ UNFORMATTED]  [✓ NO IBLANKING]
                                      FORMATTED
```

Figure 3. Data I/O panel.

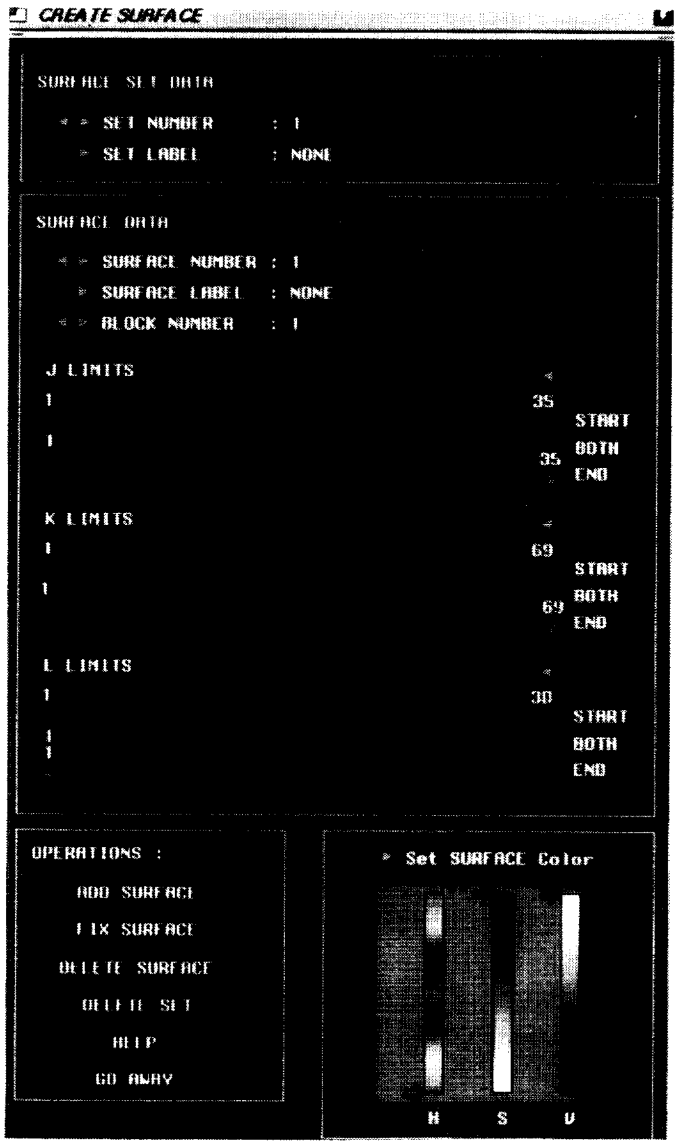
in the previous run. Doing this requires clicking on the "read view" button. Thus after clicking on two buttons, the program is in most cases ready to begin calculating.

The lower part of the Data I/O Panel shows the different file type options available for outputting the grid. It can write in the 3DGRAPE or PLOT3D formats, with several options for each format. 3DGRAPE has the ability to do both new starts and restarts; this capability is preserved in GRAPEVINE.

Upon exiting the program this panel comes up again. It shows the user not only what files have been read but also what files have been written, in the small boxes containing the words "yes" or "no." If the user desires to write a file which was not requested in the input data, such as a restart file, it may be done from this panel.

The Create Surface Panel

This panel, shown in Figure 4, is where the user specifies what surfaces are to be plotted and how they are to be colored. Any collection of surfaces or portions thereof viewed together make a "surface set." The user may specify any number of surface sets. When viewing the user may cycle through the



```
CREATE SURFACE

SURFACE SET DATA
  SET NUMBER      : 1
  SET LABEL       : NONE

SURFACE DATA
  SURFACE NUMBER  : 1
  SURFACE LABEL   : NONE
  BLOCK NUMBER    : 1

J LIMITS
1                                35   START
1                                35   BOTH
                                      END
K LIMITS
1                                69   START
1                                69   BOTH
                                      END
L LIMITS
1                                30   START
1                                     BOTH
1                                     END

OPERATIONS :            Set SURFACE Color
  ADD SURFACE
  FIX SURFACE
  DELETE SURFACE
  DELETE SET
  HELP
  GO AWAY                 H    S    V
```

Figure 4. Create surface panel

given surface sets, viewing each separately.

Sliders shown in the center of the Create Surface Panel are used to specify the starting and ending index values which, together with a block number, specify each surface or portion thereof (the user could even specify a solid region for a surface, such as an entire wing including both upper- and lower-surfaces). The lower-right-hand corner of the Create Surface Panel gives three vertical sliders which are used to specify the color in which the surface is to be plotted. The color is specified by the hue-saturation-value method.

Plotting transformations, discussed above, are used to translate, rotate, and zoom the plots of the surface-sets. At the user's option a different orientation for each surface set may be used, or the same orientation may apply to all surface sets.

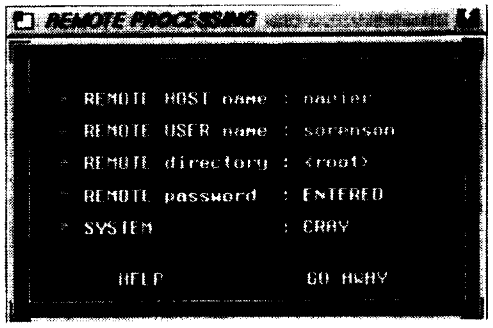
The data specified in this panel — block numbers, index limits, color codes — together with the orientation matrices which are the result of operating the transformations, are written at the conclusion of the run and are read in the next time as the "view file." This file is simple text, and can be viewed and modified with an editor. This is an example of an attempt, also made in 3DGRAPE, to keep the input data file types accessible to the user. This is much appreciated if, for whatever reason, things go wrong.

The Main Iteration Screen

There is one task which the user will wish to do before beginning the iteration process, and that is to specify upon which host processor the numerical calculations for solving the Poisson equations will be done. For small cases the user may wish to perform the calculations in "standalone" mode on the workstation. In most cases, however, a supercomputer is required. The "remote host" panel, shown in Fig. 5, effects this choice. The computers available are, of course, installation-dependent. At the Numerical Aerodynamic Simulation (NAS) facility at NASA Ames Research Center the choice is between the user's

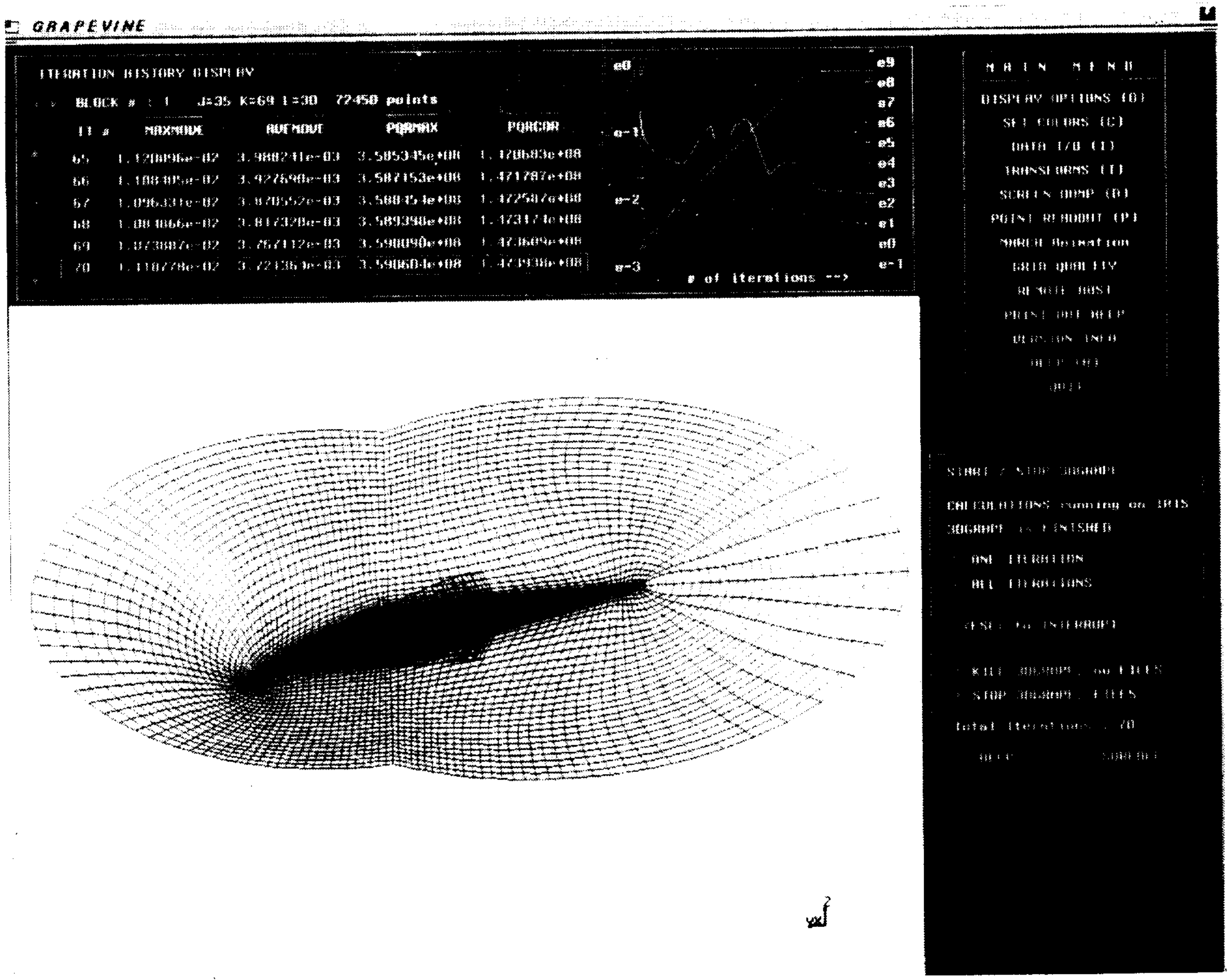
IRIS workstation (a product of Silicon Graphics, Inc., also referred to as SGI), a CRAY-2 called Navier, and a CRAY-YMP called Reynolds (both CRAYs are products of CRAY Research, Inc.).

The main iteration screen, which the user watches during the iteration process, is shown in Fig. 6. The large plotting region on the lower left shows the grid at each time step. Before starting the iteration the user cycles through the given surface sets, choosing the one to be viewed during the iteration, and positions it as



REMOTE HOST name	: navier
REMOTE USER name	: sorenson
REMOTE directory	: <root>
REMOTE password	: ENTERED
SYSTEM	: CRAY

Figure 5. Remote host panel



BLOCK # : 1   J=35 K=69 L=30   72450 points

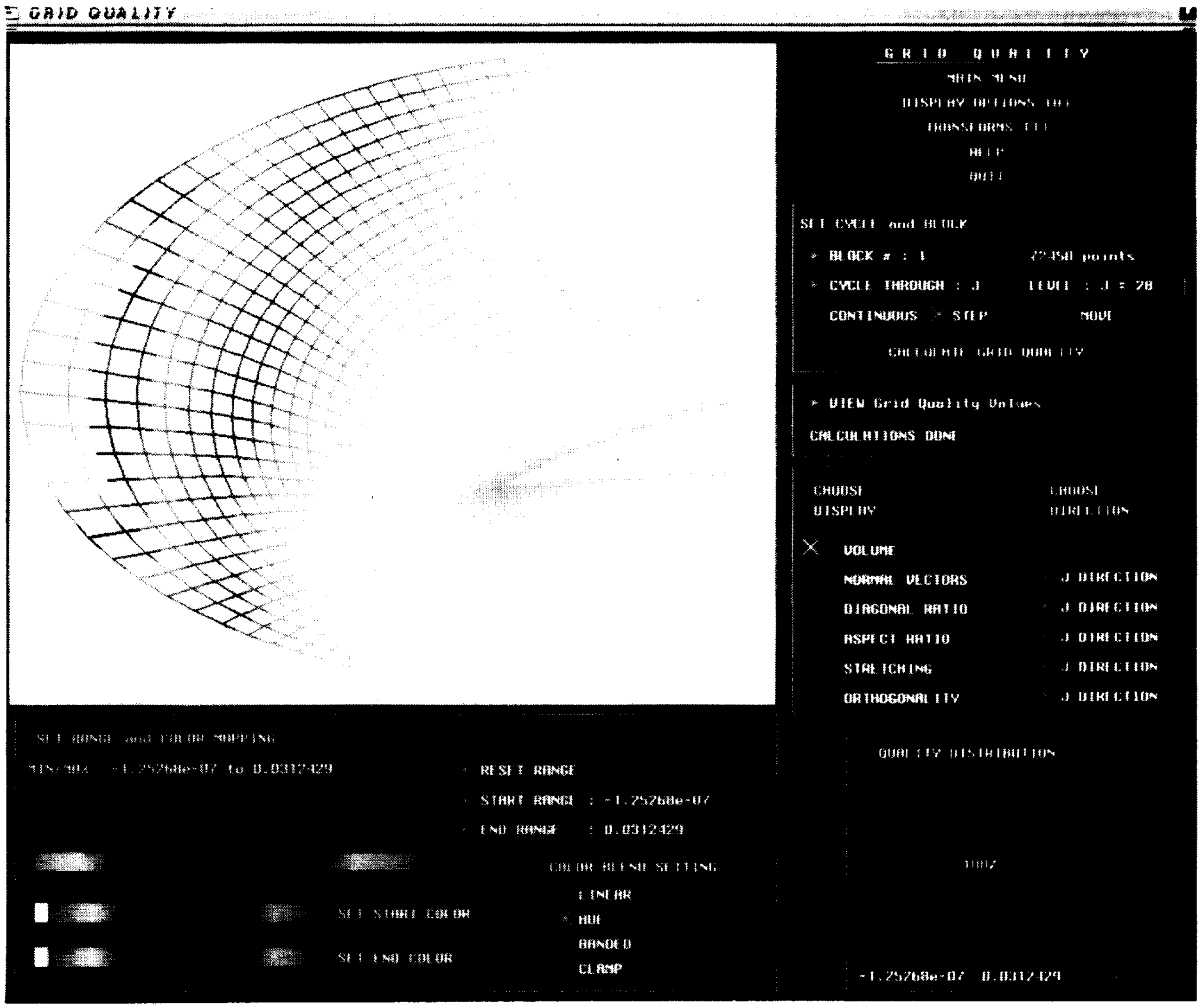
IT #	MAXMOVE	AVEMOVE	PQRMAX	PQRCOR
65	1.120896e-02	3.988241e-03	3.585345e+08	1.470683e+08
66	1.108805e-02	3.927690e-03	3.587153e+08	1.471787e+08
67	1.096331e-02	3.870552e-03	3.588454e+08	1.472587e+08
68	1.084866e-02	3.817320e-03	3.589398e+08	1.473174e+08
69	1.073887e-02	3.767112e-03	3.590090e+08	1.473609e+08
70	1.110778e-02	3.721364e-03	3.590604e+08	1.473936e+08

CALCULATIONS running on IRIS
3DGRAPE is FINISHED
total iterations : 70

Figure 6. Main iteration screen

desired by using the transformations. At any time during the iteration process the user may interrupt, reposition the viewed set, or choose another surface set, and continue.

Above the plotting region is the iteration history display. For a chosen block four parameters are given in numerical form at each iteration. These parameters, familiar to users of 3DGRAPE, are MAXMOVE, the distance moved by the point which moved the farthest, AVEMOVE, the average of the distances moved by all interior points, PQRMAX, the maximum absolute value of the forcing functions at each point, and PQRCOR, the maximum absolute value of the corrections applied to the forcing functions. These same four parameters, all functions of iteration count, are also shown as plots. In the actual program color is used to differentiate between the different function plots.



```
* BLOCK # : 1                22450 points
* CYCLE THROUGH : J          LEVEL : J = 28
  CONTINUOUS  * STEP
* VIEW Grid Quality Values
CALCULATIONS DONE
X VOLUME
  NORMAL VECTORS       J DIRECTION
  DIAGONAL RATIO       J DIRECTION
  ASPECT RATIO         J DIRECTION
  STRETCHING           J DIRECTION
  ORTHOGONALITY        J DIRECTION
MIN/MAX  -1.25268e-07 to 0.0312429
START RANGE : -1.25268e-07
END RANGE   : 0.0312429
* HUE
100%
-1.25268e-07  0.0312429
```

Figure 7. Grid quality screen.

The main menu is given on the upper right, and the panel on the lower right controls starting and stopping the iteration process.

The Grid Quality Screen

The last remaining screen, the grid quality screen, shown in Figure 7, is a graphical implementation of a grid quality program originally written for batch operation in FORTRAN by K. Chawla. For a chosen block it gives the user the option of calculating the cell volume, and up to five other grid quality functions, with each of the other five functions in any of the three coordinate directions. Those grid quality functions are: (1) the angle between neighboring normal vectors, (2) the ratio between the lengths of the two diagonals in a cell face, (3) the aspect ratio of the cells, (4) the ratio between the heights of adjacent cells, called the stretching ratio, and (5) the orthogonality given by dot-products. Volume or any one of the other functions can be shown at any time. The chosen function is used to color grid surfaces as the program marches through the grid by incrementing or decrementing a chosen index. The marching may be manual or automatic. Various options are given to map the calculated function values into colorations. A distribution display shows how many of the points have various values of the chosen function. The user may rotate, translate, and zoom the displayed surfaces.

The Network Code Module

As mentioned above, the user may choose to do the mathematical calculations (the numerical solution of the Poisson equations) on either the IRIS upon which the remainder of the program is running, or on some remote host processor (typically a supercomputer such as a CRAY). If the choice is the IRIS, no networking is needed; the 3DGRAPE program is simply linked with the other code and called.

But if a remote host is chosen, a modified version of 3DGRAPE is run on the CRAY. That operation requires several steps:

1. The input data files (file10 in the case of a new start or file16 in the case of a restart, and file11) are sent from the IRIS to the CRAY. This is done by using the C function "system" to call the UNIX function "rcp."

2. The UNIX function "rexec" is used to start the remote process on the CRAY.

3. The C header code on the CRAY reads the input files to determine what dimension sizes are required, and allocates storage on the CRAY accordingly.

4. The IRIS sends to the CRAY a list of the surfaces which the user has chosen to display during the iterative process.

5. 3DGRAPE in FORTRAN on the CRAY re-reads the input data, initializes its variables, and begins iterating.

6. After each iteration the x,y,z values for the displayed surfaces are sent to the IRIS, along with convergence history information which is to be plotted as a function of iteration count. The IRIS display is then updated.

7. When the iteration procedure is completed, or when an exit interrupt is received by the CRAY, the entire grid is sent to the IRIS.

Three different IRIS-to-CRAY interrupts were found to be necessary: *kill, exit,* and *pause-continue*. A *kill* interrupt causes the CRAY process to halt, possibly in the middle of an iteration, and not send the grid back. This is useful when the grid being generated is obviously unsatisfactory, and the most expedient halt is desired. An *exit* interrupt causes the CRAY to complete the current iteration, send the grid back, and stop. This is used when the grid appears to be good enough to use without further iteration. A *pause* interrupt causes the CRAY to finish the current iteration, send the entire grid back to the IRIS, and then halt without releasing CRAY memory or terminating the run. The user can then examine the grid graphically on the IRIS by cycling through other surface sets; do translation, rotation, and zoom operations on those plots; and plot the grid with coloration by grid quality functions with marching through all grid points. Once development of the program is finished, and the functionality of 3DPREP is fully integrated into the code, the user will be able at this point to modify the input parameters. After performing these tasks during the pause, the user can issue the *continue* command and re-start the CRAY process.

Network functions “socket” and “bind” were used to open sockets on both the IRIS and CRAY sides of the network. All communication is done over the sockets, with the exception of the *kill* interrupt which is sent at the system level. Routines from the CRAY library “binconv” were used to change floating point and integer numbers between CRAY and IRIS formats. Algorithms used were partially modeled on the sample code “chat” provided by SGI on IRIS systems, and the networking library “dlib” written by M. Yamasaki. All network functions are error-trapped; this is necessary because hardware interruptions from both the CRAY and the network may occur at any time.

The purpose of distributing the code over a network is to accelerate the iterative solution of the Poisson equations, relative to the speed at which they would be solved running on the IRIS alone. The speed at which the code runs when distributed is dependent upon many factors. First is the IRIS, which slows due to a lack of free memory or too many users. Second is the network, which slows under heavy use. Third is the CRAY which, when many users are logged on, can cause a particular task to wait for as much as sixty seconds before getting another time slice. For these reasons it is difficult to give a definitive statistic concerning the degree to which the code speeds up when distributed, but experience indicates a speedup by a factor of ten to fifteen.

LESSONS LEARNED

It should be recognized that distributed processing, at least in the CFD context, is in its relative infancy. There have been networks for many years, over which data and news are shared. But there are few examples of systems wherein the *processing* of data is shared over more than one node of a network.

Doing so allows specialization wherein individual processors are assigned those tasks at which they excel. The present work is one example of truly distributed processing. The workstation excels at graphical interfacing, while the supercomputer excels at "number crunching."

When this project was begun approximately two years ago CRAY computers (together, possibly, with the emerging Japanese supercomputers), held a unique position in the computational scheme of things, widely accepted as the only real practical supercomputers. And so at that time, due to their position in the industry and the fact of the availability of several of them at NASA Ames Research Center, CRAYs were the obvious choice for the numerical processor for this project. It is doubtful that networking was high on the priority scheme during their design (one might argue that CRAYs are good at number crunching *because* they do *not* excel at networking). But that lack of emphasis has revealed itself in certain difficulties faced by this design team. CRAYs tend, it would seem, to favor long time slices per user, causing the interactive user to have to wait as long as sixty seconds for the processing of an interrupt. And it was the experience of this design team that the CRAYs crashed several times per day (the IRIS workstations are up for weeks at a time). And because they are so attractive for number crunching, CRAYs tend to be heavily loaded, further slowing things. Also, it was found that certain networking functions worked differently on different host computers. CRAYs and IRISes have different internal data formats, giving rise to the need for conversion before transfer. These difficulties took their toll on this development effort. CRAY's position in the scheme of things is now threatened by ultra-high-end multiple-processor workstations, which have a clear price/performance advantage, and by massively parallel machines, which have the potential for much higher mflops rates. Ultra-high-end workstations and massively parallel machines should be considered as candidates for the numerical processor in the design of any such distributed system in the future.

At the outset of this project there was debate over how much of the pre-existing 3DGRAPE code should be translated into C. Opinion ranged from "all" to "none." As it turned out, a header program was written in C which reads the input data to determine array dimension requirements, allocates arrays accordingly, processes iterrupts, and does data output for transfer back to the IRIS. This header calls various parts of 3DGRAPE in FORTRAN to re-read the input and initialize variables. And the C header contains the "main iteration loop" in which other parts of 3DGRAPE in FORTRAN are called to actually perform an iteration. If these authors had it to do over again, more of the FORTRAN would be translated into C, principally the remainder of the data I/O, and the variable initialization, leaving only the contents of the main iteration loop in FORTRAN.

It is obvious that the graphical capabilities of modern workstations can have a great and beneficial effect upon productivity of scientists and engineers in general, and on practitioners of CFD in particular. But they bring about a situation with respect to programming languages which is analogous to that which argues for distributed computing. Just as distributed computing allows different computers to do that part of the job at which they excel, so does multi-linguistic programming allow different languages to do what they do best. C (or C++) is the language of choice for graphical interfaces for many reasons including versatility in data structures, versatility in performing I/O, and compatibility with graphical libraries.

FORTRAN (especially as implemented on CRAY computers) vectorizes best, and is still the language of choice among the scientists and engineers who develop the technology implemented by these systems. And so the paradigm suggested above, wherein the graphical user interface, the networking, array allocation, initialization of variables, and I/O are done in C (or C++), and the contents of the "main iteration loop" remains in FORTRAN, seems at present to be a workable compromise.

CONCLUSIONS

A working software system for generating structured multiple-block grids, distributed over a network between a supercomputer and a workstation, has been developed and tested. Because the code is distributed its user interface is interactive on a workstation, and its numerical processing is performed on a supercomputer. Thus a great reduction is seen in user time required to collect and format the input data and to run the code, and the numerical processing is performed much faster than if it were done on a workstation.

It was the experience of these developers that CRAY supercomputers, though powerful at numerical processing, are not ideally suited for interactive networking applications. Developers of similar distributed software systems in the future should instead consider the use of ultra-high-end workstations or massively parallel computers for role of numerical processor. It is further the conclusion of these developers that when coding a graphical user interface in C for a pre-existing numerical application in FORTRAN, everything except the contents of the "main iteration loop" should be translated into C.

ACKNOWLEDGEMENTS

The authors are grateful to Michael J. Yamasaki and Matthew W. Blake of NASA Ames Research Center, and John R. Campbell of Computer Science Corp. for their help in writing the networking code. The authors are grateful to Kalpana Chawla of the MCAT Institute for supplying a grid quality code. And the authors thank the NAS Applied Research Branch, the NAS Systems Development Branch, and the Office of Technology Utilization at NASA Ames Research Center for partial financial support.

REFERENCES

1. Sorenson, R. L.: Three-Dimensional Zonal Grids About Arbitrary Shapes by Poisson's Equation. Appears in Sengupta, S., Häuser, J., Eiseman, P. R., and Taylor, C., eds.: *Numerical Grid Generation in Computational Fluid Mechanics*. Pineridge Press Ltd., 1988.

2. Sorenson, R. L.: *The 3DGRAPE Book: Theory, Users' Manual, Examples,* NASA TM-10224, 1989.

3. Thompson, J.F.: A Composite Grid Generation Code for General 3D Regions — the EAGLE Code. *AIAA Journal,* vol. 26, no. 3, March, 1988, p. 271.

4. Thompson, J.F. and Lijewski, L.E.: Composite Grid Generation for Aircraft Configurations with the EAGLE Code. Appears in Thompson, J.F. and Steger, J. L., eds.: Three Dimensional Grid Gen*eration for Complex Configurations — Recent Progress,* AGARD-AG-309, 1988.

5. Sorenson, R. L., and McCann, K. M.: A Method for Interactive Specification of Multiple-Block Topologies. AIAA 91-0147, Jan., 1991.