BNL 18803

SECOND ANNUAL
AEC SCIENTIFIC COMPUTER INFORMATION EXCHANGE MEETING

PROCEEDINGS OF THE TECHNICAL PROGRAM
THEME: COMPUTER GRAPHICS

May 2-3, 1974
New York City

Hosted by

BROOKHAVEN NATIONAL LABORATORY
ASSOCIATED UNIVERSITIES, INC.
UPTON, NEW YORK 11973

BNL 18803

SECOND ANNUAL
AEC SCIENTIFIC COMPUTER INFORMATION EXCHANGE MEETING

PROCEEDINGS OF THE TECHNICAL PROGRAM
THEME: COMPUTER GRAPHICS

General Chairman: Y. SHIMAMOTO
Program Chairman: A.M. PESKIN

May 2-3, 1974
New York City

BROOKHAVEN NATIONAL LABORATORY
UPTON, NEW YORK 11973

CONTENTS

May 2, 1974
Thursday Morning

May 2, 1974
Thursday Afternoon

SESSION II

Physical, Engineering, and Biomedical Applications

Chairman: G. H. Campbell
Brookhaven National Laboratory

May 3, 1974
Friday Morning

SESSION III

General Graphic Facilities

Chairman: A. M. Peskin
Brookhaven National Laboratory

OVERVIEW OF SESSIONS

A. M. Peskin, BNL
Program Committee

The topic of computer graphics serves well to illustrate that AEC affilliated scientific computing installations are well represented in the forefront of computing science activities. The participant response to the technical program was overwhelming--both in number of contributions and quality of the work described.

Session I, entitled Advanced Systems, contains presentations describing systems that contain features not generally found in graphics facilities. These features can be roughly classified as extensions of standard two-dimensional monochromatic imaging to higher dimensions including color and time as well as multidimensional metrics. Session II presents seven diverse applications ranging from high energy physics to medicine. Session III describes a number of important developments in establishing facilities, techniques and enhancements in the computer graphics area.

Although an attempt was made to schedule as many of these worthwhile presentations as possible, it appeared impossible to do so given the scheduling constraints of the meeting. A number of prospective presenters "came to the rescue" by graciously withdrawing from the sessions. Some of their abstracts have been included in the Proceedings.

I wish to acknowledge the contribution of Robert M. Lee (LLL) for his assistance in the planning of this program.

SESSION I

Advanced Systems

Chairman: R. M. Lee
Lawrence Livermore Laboratory

PRIM-9

An Interactive Multidimensional Data Display and Analysis System

Mary Anne Fisherkeller, Jerome H. Friedman

Stanford Linear Accelerator Center*
Stanford, California 94305

and

John W. Tukey

Princeton University**
Princeton, New Jersey 08540

ABSTRACT

(Submitted to A.E.C. Scientific Computer Information Exchange Meeting, May 2-3, 1974)

PRIM-9 is an interactive data display and analysis system for the examination and dissection of multidimensional data. It allows the user to manipulate and view point sets in up to nine dimensions. This is accomplished by providing all 36 two-dimensional projections along the original axes at the push of a button, along with the ability to rotate the data to any desired orientation. These rotations are performed in real time and in a continuous manner under operator control. From the parallax effect, arising from the dynamic aspect of this continuous rotation, one perceives a third dimension (depth into the screen).

PRIM-9 gives the operator the ability to perform manual projection pursuit. That is, by rotation and view change he can look at his data from all possible angles in the multidimensional space and try to find

those that provide interesting structure. The system also allows interactive masking and isolation. The user can conveniently mask on any or all of the current variables, thus isolating interesting structures found along the way. These interesting structures can then be further analyzed alone, or may be subtracted from the total sample to simplify the search for still other structures. In addition to this strategy, the user may invoke an automatic projection pursuit algorithm. Starting at any projection (view), this numerical algorithm will search (in much the same manner as a human operator) for those projections that provide interesting structure. The system also incorporates the ability to save either temporarily, or permanently, any interesting view that is found. The operator can return to these views at any later time or reproduce them on a hard copy device.

*Supported by the U.S. Atomic Energy Commission under Contract AT(043)515

**Prepared in part in connection with research at Princeton University supported by the U.S. Atomic Energy Commission.

INTRODUCTION

PRIM-9 is an interactive computer graphics program for Picturing, Rotation, Isolation, and Masking - in up to 9 dimensions. It is implemented on the Graphics Interpretation Facility of the Stanford Linear Accelerator Center, Stanford University. This facility consists of an Information Display's IDIIOM refresh CRT and a Varian 620/i minicomputer, linked to an IBM 360/91.

PRIM-9 is a result of a continuing program of research into techniques for applying computer graphics to exploratory data analysis. A general introduction to its properties and uses is documented in a 25 minute sound motion picture entitled "PRIM-9"[1], produced by the Computation Research Group of the Stanford Linear Accelerator Center, Stanford University. This note details its properties with emphasis on the human engineering aspects of its implementation and on the various data analysis problems to which it can be applied.

PRIM-9 has been developed toward ends of very different breadth and distance: first, to gain insight into what can be learned by looking at the numerical aspects of data in more than two aspects at a time and, second, to implement a tool for pictorial examination and dissection of multidimensional data. The development of PRIM-9 has grown through many stages, and many of the early techniques that were implemented and then later discarded may turn out to be central to other data display systems. The resulting system as it is currently implemented is especially straightforward in concept. Its emphasis is on picturing and rotation on the one hand and masking and isolation on the other. Picturing means an ability to look at the data from several different directions in the multidimensional space. Rotation means, as a minimum, the ability to turn the data so that it can be viewed from any direction that is chosen. Picturing

and Rotation are essential abilities, valuable alone, but their usefulness is greatly enhanced when combined with Masking and Isolation. Masking is the ability to select suitable subregions of the multidimensional space for consideration. That is, only those data points that lie in the subregion are displayed. Isolation is the ability to select any subsample of the data points for consideration. That is, only those points in the selected subsample are displayed. It is important to note that masking is tied to coordinates. If we rotate the data points, different points will enter and leave the masked region. Isolation, on the other hand, is tied to the data points. Under all operations of the system (except further isolation), the isolated sample remains the same. In the absence of rotation, masking and isolation are equivalent; however, in the presence of rotation the distinction is essential.

By interactively applying picturing, rotation, isolation and masking to his data, the user can, in particular, perform projection pursuit. That is, he can look for those views that display to him interesting structure. He can isolate any structures so found and study them separately and/or remove them from the remaining sample, simplifying the search for still further structures. In this way, he may gain considerable insight into the multidimensional properties of his data. As an aid to this process, the present version of PRIM-9 also provides an automatic projection pursuit algorithm.[2] This algorithm assigns to each view a numerical index that has been found to closely correspond to the degree of data structuring in the projection. When invoked, the algorithm will search for those views of the multidimensional data that maximize this projection index. At any point in the interactive session the user may invoke the automatic projection pursuit algorithm. Starting at the current view, the algorithm will find the view corresponding to the first

maximum of the projection index, uphill from the starting view. Manual projection pursuit can then continue from this new (hopefully more structured) view. The algorithm can also be invoked at the beginning of the session starting with the various original or principal axes of the data. The resulting views can then provide useful starting points for further investigation.

The next section gives a brief description of the hardware and software configuration of the Graphic Interpretation Facility of the Stanford Linear Accelerator Center, on which PRIM-9 was developed. This is followed by several sections describing the implementation of the various features of the PRIM-9 system. The note then concludes with a section discussing various techniques for applying these features to some multidimensional data analysis problems.

THE GRAPHIC INTERPRETATION FACILITY

The Graphic Interpretation Facility of the Stanford Linear Accelerator Center (SLAC) is pictured in Figure 1 and is described in detail elsewhere.[3,4] A brief description emphasizing those properties relevant to the implementation of PRIM-9 is included here.

The primary computing resource at SLAC during the development of PRIM-9 was an IBM SYSTEM/360 Model 91, with 2048k bytes of storage, operating under OS/MVT with HASP.* The two basic components of the Graphic Interpretation Facility are a Varian Data Machines 620/i computer[5] with 8k 16-bit words of storage, and an IDIIOM Display Console[6] with a 21-inch CRT made by Information Displays, Inc. When the display is operating, the 620/i memory contains a program for the 620/i to execute and a program (display file) for

---

*the current resource includes, in addition to the S/360-91, twin IBM SYSTEM 370/168's, operating under VS/2R1.6 with ASP.

the IDIIOM to execute. These two programs run concurrently with the IDIIOM stealing cycles from the 620/i. The instructions (orders) in the display file may display characters, points, or straight line segments, perform unconditional or subroutine jumps, count and index, or interrupt the 620/i. Characters, points, and lines are positioned on a 1024 by 1024 raster unit grid on the face of the CRT. The 620/i instruction set includes, in addition to the usual set for a standard mini-computer, instructions to start and stop the IDIIOM in its execution of the display file, and instructions to read and reset registers associated with the display operation.

Interaction at the console is by means of a solid state alphameric keyboard, a light pen, and a function keyboard with 32 buttons. Under program control, portions of the display file may be designated as light pen sensitive or insensitive. When the light pen is pointed at a sensitive item on the CRT, the 620/i will be interrupted. The function buttons generate interrupts on both the closing and the opening of the switch. This means that software can produce either (1) single impulse for each depression of the button (cycling) or (2) a repeated impulse controlled by software timing, occurring so long as the button remains down. This last facility, especially when combined with automatic reversal (see below), is of great importance in allowing effective control. Plastic overlays may be placed on the function keyboard to identify the purpose of each button.

The 620/i and SYSTEM/360 are connected through an IBM 2701 Parallel Data Adapter Unit. Data may be transmitted in either direction through this link. The 620/i can interrupt the SYSTEM/360 and determine whether the SYSTEM/360 is trying to read or write; however, the SYSTEM/360 is not able to interrupt the 620/i.

Although the Facility can operate in a stand-alone fashion, most of our work (including the implementation of PRIM-9) has been done using it in conjunction with the SYSTEM/360. The SYSTEM/360 provides fast computation and mass storage, while the 620/i maintains the display.

A package of PL/1 procedures, the IDIIOM Scope Package,[4] has been provided for writing highly interactive programs on the IBM 360/91 without burdening the writer with all of the details of programming the 620/i and IDIIOM. The user of this package can, by means of procedure calls, control the display console in addition to having all of the facilities of PL/1 available to him. Procedures are supplied to construct graphic display elements which will display information on the IDIIOM CRT, and transmit elements as well as interrupts between the SYSTEM/360 and 620/i.

IMPLEMENTATION

A. Scaling and Coordinates

The data is stored (in the 360/91) in initial coordinates either as scaled before entry or as rescaled to fit into the display region. It remains in this form. Rotations to current coordinates are performed by a transformation matrix. An ongoing step of rotation involves (A) updating this transformation matrix (multiplication by a simple rotation matrix) and (B) passing the data through the modified transformation matrix and transferring the relevant coordinates to the 620/i. Thus in the PRIM-9 system, it is important to distinguish between the initial coordinates and the current coordinates. The initial coordinates are defined to be an orthogonal set of fixed axes in the multivariate space. The input data to PRIM-9 consists of a number of ordered sets of numbers. Each of these numbers is assigned to the initial axis corresponding to its position in the ordered set. These numbers then become the values of the initial coordinates and each

ordered set of numbers becomes a point in the Euclcadian space spanned by the initial axes. The current coordinate axes are another orthogonal set spanning the same space that is connected to the original set by a similarity transformation. That is, each of the current coordinates is a particular linear combination of the initial coordinates. So long as only rotations are used (see F for exceptions), these linear combinations are restricted to those that do not change the interpoint distances in the multidimensional space.

Rotation has the effect of continuously changing the linear transformation between the current and initial coordinates. At the beginning of the session, the initial and current coordinates coincide, but as soon as a rotation is performed the current coordinates become distinct from the initial ones.

B. Picturing

For picturing, PRIM-9 provides the choice of any two of the current coordinates as horizontal and vertical axes. Two buttons cycle through the choices. One button changes the coordinate displayed in the vertical direction, while the other button changes the coordinate shown along the horizontal direction. In this way, it is easy to go through the $\binom{NDIM}{2}$ (NDIM is the total number of coordinates) possible displays, as well as getting from one particular display to another. Choosing a display involves selecting two integers i and j, where i is the y (vertical) coordinate and j is the x (horizontal) one. Pushing the first button increments i by one and the second, similarly, increments j, both modulo NDIM. In order to speed up the selection process, all unnecessary combinations have

been eliminated by requiring that i be greater than j. That is, i cycles from 2 to NDIM while j cycles from 1 to i-1.**

C. Rotation

Being able to see projections on all coordinate pairs can be very useful. But it is not enough. To be able to get reasonably to any two-dimensional projection means either a way to call for the projection that we want, or a way to move about in the multi-dimensional space. Since we usually do not know just what we want, and when we do we will find it difficult to learn to call for it in a general way, we need a way to move about. Continuous controlled rotation is a natural way to move about (change projection) in a multidimensional space.

In the implementation of controlled rotation, one naturally thinks of turning a knob. However, the configuration of the Graphic Interpretation Facility does not support a knob. Thus, we are forced to implement rotation through pushing buttons. The naive approach to the control of a single rotation by buttons involves two buttons, with these responses:

- to one button: rotation "to the right" at a constant angular rate so long as the button is depressed.
- to the other button: rotation "to the left" at the same constant angular rate so long as the button is depressed.

** If we were programming this action again, we would use a constant-step version of the increasing-step-and-reversal control described under rotation.

This type of control has two serious flaws:

- if the rotation rate is slow enough for fine adjustment, the delay time for large rotations is undesirable -- so undesirable as to be nearly impractical.
- if used naively -- two buttons per rotation -- it is too easy to use up too many buttons. (PRIM-9, in its present version, makes as many as $\binom{9}{2} = 36$ different rotations available.)

Both of these disadvantages can be overcome, the first by an "increasing-step-and-reversal" control, the second by "time-sharing" the rotation drive.

The type of rotation control used in PRIM-9 has two features:

(1) rotation reversal -- rotation in one sense so long as the single button is held down -- when the same button is released and then again depressed, rotation in the opposite sense so long as the button is held down.

(2) rotation by increasing (accelerating) steps. These steps are presently taken as 1, 1, 2, 4, 8, 16, 32,... times a small (unit) angle. (Note the repetition of 1.) The largest possible step is limited by a speed limit settable at 1, 2, 4, 8, 16, 32, 64, or 128.

This combination of rotation reversal and acceleration gives the operator fast and easy approach to a desired data orientation. The rotation starts out slowly but quickly accelerates to the speed limit so long as the button remains depressed. When the operator sees an interesting data orientation on the CRT screen, he releases the button. Due to human reaction time, both in perception and in

releasing the button, the rotation usually overshoots the desired data orientation. Depressing the same button again causes the rotation to proceed in the opposite direction, starting out at the slowest speed, and again accelerating. When the desired orientation again comes into view on the screen the button is released. Now, due to the slower speed (the acceleration usually has not reached the speed limit), the overshoot is much less (in the opposite direction). Again, depressing the button causes rotation reversal at the slowest speed allowing the operator to home in on the desired data orientation. In the usual case, at most two reversals are necessary. This strategy allows rotation in both directions, minimizes the delay time for large rotations, allows a slow rotation for fine adjustment, and requires only one button to drive the rotation.

In order to specify a rotation, one needs not only to specify the sign and magnitude of the rotation angle but also the rotation axis. A general rotation axis in a multidimensional space (for example, in terms of its direction cosines) is complicated to understand and time consuming to specify. In PRIM-9, directly available rotations are confined to those associated with pairs of the current coordinates. This makes both control and understanding relatively easy. A rotation axis is specified by two integers i and j. These integers specify the current coordinate axes that participate in the rotation. That is, coordinates i and j rotate while the other NDIM-2 coordinate axes, orthogonal to i and j, remain fixed. It is possible to get from any one two-dimensional projection to any other, with relative ease, by combining these selected rotations in the correct amount and sequence (this is the multidimensional analog of the Euler angle specification of a rotation in three dimensions.) The two in-

tegers (i and j) that specify the coordinates that participate in the rotation are selected in exactly the same manner, as discussed in the previous section for choosing the current projection axes. The selection is controlled by two buttons that cycle through the $\binom{NDIM}{2}$ possibilities. One button cycles through $2 \leq i \leq NDIM$ in increments of one while the other, similarly, cycles through $1 \leq j \leq i-1$.

If the axes that define the rotation coincide with the current projection axes, then the data points will simply move in circular orbits about their relative mean in the projection. If neither axis corresponds to a current projection axis, then the display will remain unchanged because the rotation is orthogonal to the current projection axes. Both rotation axes are "invisible" to the screen. Useful rotations occur when one of the axes that participates in the rotation corresponds to a current projection axis, while the other is one of the NDIM-2 axes orthogonal to the current display plane. In this case, the operator sees the projected points change their positions on the screen as the invisible coordinate is rotated against the visible one. When an interesting pattern emerges, as a result of the rotation, the operator can home in on it as described above. The invisible coordinate can then either be rotated against the other visible one or the operator can change to another invisible coordinate. He can then rotate these new coordinates to try to sharpen the structure even further. Continuing in this manner, the operator may manually iterate to a data orientation that provides an informative view of his multidimensional point cloud.

Easily recognizable continuous rotation provides an additional advantage. Its dynamic effects let one see an additional dimension, not instantaneously, but yet at the same time -- in the best sense of those words. This is due to the relative motions of the points associated with parallax. The points that are closer in the invisible rotation coordinate move across the screen more rapidly than those that are farther away. This parallax effect gives the illusion of a third dimension (depth into the screen) and this aspect seems to be a very useful complement to the two aspects (horizontal and vertical) provided directly by the screen.

To help in the interpretation of a particular projection, a display of the initial coordinate axes, as projected onto the current projection plane, is easily switched (with a pushbutton) in and out of view. This display allows the user to see graphically the linear combinations of the original coordinates that comprise the two axes of the current projection plane. Another button (neutral) returns the display to its initial state. That is, the current coordinates are reset to the initial ones.

D. Masking

Masking is the ability to select any subregion of the multi-dimensional space, and have only those data points that lie in the subregion displayed on the screen. Masking is used in connection with isolation and also has some interesting uses of its own (these are discussed in the last section). It is important to note that masking is tied to the coordinates. Under rotation, the data points will enter and leave the masked region.

The flexibility of PRIM-9's masking, like the flexibility of its rotation, is limited so that it is easy to control and understand. PRIM-9 allows simple masking on any or all of the current coordinates. That is, those points for which $X_i < F_i$ or those for which $B_i < X_i$, or both -- for a single i or several i's -- are caused not to appear on the screen. Here X_i $(1 \leq i \leq NDIM)$ are the current coordinates and F_i, B_i are forward and backward bounds on each of these current coordinates.

Masking is controlled by five buttons. One button toggles the masking on and off. A second button cycles (one step per press) through the integer, i, $(1 \leq i \leq NDIM)$ which identifies that current coordinate to which the mask is to be applied or altered. The third identifies which edge F (front), B (back) or J (joint) is to be driven. These three options are cycled through by successive pushes of the button. The fourth button drives the selected mask in a continuous motion using the same "increasing-step-and-reversal" control technique described above for rotation. The fifth button allows the rapid selective removal of the mask for a particular coordinate (identified using the second button) by resetting both the front and back mask edges to their outside positions.

This is in contrast to the effect of the first button which, when toggled off, removes the masks on all coordinates simultaneously. The J (joint) drive moves the front and back masks together in the same direction and speed, maintaining a fixed separation between them. This allows driving an unmasked zone of chosen width back-and-forth on a particular coordinate.

If the masking coordinate is one of the current projection axes, then driving the mask away from its outside position and toward the center of the screen will cause the points to disappear along an advancing line, accelerating to the speed limit so long as the drive button is held down. When the button is released and again depressed, the masking line will reverse, starting at the slow speed. The masked points will reappear as the mask boundary retreats. As for the case of rotation, this control allows the operator to arrive at and home in on a desired mask position easily and quickly.

A masked coordinate need not be a current projection axis. For example, one can mask on a current coordinate orthogonal to the projection plane. As the mask moves from its outside position on this "invisible" coordinate, points will disappear from various regions of the screen giving insight into the relationship contained in the data between the invisible coordinate and the two visible ones. In particular, the joint drive allows driving an unmasked zone of chosen width back-and-forth on the invisible coordinate. This will cause points to appear and disappear as the mask passes through the various values along the invisible coordinate. More sophisticated techniques using moving masks are discussed in the last section.

E. Isolation

Isolation is the ability to select an arbitrary subset of the data sample at any point in the analysis, and perform upon this subset or its complement (the full sample with the subset removed), all operations that one can perform on the full sample. Experience with PRIM-9 has shown that isolation is a most essential adjunct to picturing and rotation. It extends the system from being a purely linear device to a piecewise linear device, greatly increasing its effectiveness and power. Some of its more standard applications are discussed in the last section.

As implemented in PRIM-9, isolation begins with masking. The data points to be isolated are defined by constructing a mask (in terms of the current coordinates) that just contains the points to be isolated. As the mask is applied, the points that are masked out disappear from the screen, making it relatively easy to interactively construct a mask that just includes the desired subset of points to be isolated. It might seem, at first thought, that because of the limited flexibility of PRIM-9's masking, it would be difficult to construct mask boundaries that include arbitrary point subsets. This turns out not to be the case. This is due mainly to the fact that the current coordinates, on which the mask is defined, can be interactively rotated to an arbitrary orientation with respect to the initial coordinates, and that the isolation can be applied repeatedly in defining the subset. In this way, the user can construct a piecewise linear approximation to any boundary surface in the multidimensional space to sufficient accuracy to just include the points to be isolated.

When all of the undesired points have been masked out, so that only the subsample to be isolated appears on the screen, the user presses a button to invoke the isolation. This causes a menu of options to appear. Figure 2 is a photograph of this menu. The user selects the appropriate option by touching a light pen to those places on the screen that define the option.

The numbers at the right reference the sixteen isolates that can be simultaneously defined. Isolate "0=ALL" always references the total sample and "15=RESIDUAL" is reserved for special use with the "residual after" option. This leaves fourteen isolates that can be arbitrarily defined and stored by the user.

Touching the light pen to one of the seven options to the left causes a question mark to appear either to the right of the option or at a blank space within it. Touching the pen to one of the sixteen numbers causes the selected number to replace the question mark. When all of the question marks have been replaced by numbers and the command is correct, then touching "APPROVED" initiates the action. The commands are:

FILL n:	Save currently masked subset at n.
RECALL n:	Recall isolate saved at n and replace current subset with it.
SUBSELECT ON n1 FILL n2:	Save at n2 the intersection of the current subset with the isolate stored at n1.
INTERSECT n1 AND n2 FILL n3:	Save the intersection of isolates n1 and n2, at n3.
UNION n1 AND n2 FILL n3:	Save the union of isolates n1 and n2, at n3.
NOT n1 BUT n2 FILL n3:	Save the intersection of the complement of isolate n1 with isolate n2, at n3.

RESIDUAL AFTER n ALSO: Replace isolate 15 by the intersection of the complement of isolate n with the current isolate 15.

If only a number is light-penned, "RECALL" is the default command. As each subset is isolated, the remainder may be saved with the last command. An asterisk appears to the left of those numbers that represent isolates that are currently in use. When an isolate has been saved or recalled, it becomes the current subset. If instead of a light pen hit the button invoking the isolation is simply pushed a second time, the display will alternate between the current subset and the entire data sample. The current isolate number always appears at the bottom of the screen.

F. Scale and Location Transformations

The location may be displaced in either the positive or negative direction and/or the scale can be expanded or contracted on any current coordinate axis. This is accomplished by specifying a "key" integer, i, ($1 \leq i \leq$ NDIM) representing the coordinate to be transformed. A single button cycles through the possible values of i. Another button displaces the origin of the selected coordinate while a third button scales the coordinate. These latter two buttons drive the displacement or scale change in a continuous motion, using the "increasing-step-and-reversal" control technique described earlier.

G. Saving Views (Projections)

Quite often during a session, the user finds a projected view of his data that is sufficiently interesting to warrant saving it so that he may return to it later in the session, at another session, or perhaps record it permanently on a hard copy device.

PRIM-9 provides for both temporary (within a session) and permanent (between sessions) view saves. The permanent saves can also be transferred to a hard copy device at the user's discretion. This facility allows the user to continue with an analysis after he has found an interesting view. If further analysis should not improve the structuring, or perhaps even worsens it, the user can simply recall the saved view and begin again on a different track.

Saving a view (projection) consists of storing the transformation matrix connecting the initial coordinates to the current coordinates along with the mask bounds at the time of the save. Up to six different views may be saved on a temporary basis and another eighteen may be saved in a permanent disk data set. The temporarily saved views are simply identified by their number, i ($1 \leq i \leq 6$), while the permanently saved views on the disk are given identifying names typed by the user on the keyboard. If no such name is typed, then the system assigns a default identifier which is the date and time of the save. To retrieve one of these views, the user depresses a button which presents a menu on the screen listing the names of all of his saved views. He selects a view by touching the light pen to the appropriate name.

H. Automatic Projection Pursuit

The PRIM-9 system provides the user with a sort of "automatic pilot" for rotation. That is, at the user's request, the system will invoke a numerical algorithm that automatically searches for data orientations (projection directions) that display interesting structure. This algorithm is detailed elsewhere[2] and only its general properties, as they relate to the implementation on the PRIM-9 system, are discussed here.

The automatic projection pursuit algorithm assigns to each projection a numerical index, $I(\hat{k},\hat{\ell})$, that corresponds to the degree of data structuring present in the projection. Here $\hat{k}$ and $\hat{\ell}$ are the two orthogonal unit vectors representing the particular two-dimensional projection of the NDIM-dimensional data. The more structure present in the projection, the larger $I(\hat{k},\hat{\ell})$ becomes. The essence of the algorithm is to find those projection directions ($\hat{k}$ and $\hat{\ell}$) that maximize $I(\hat{k},\hat{\ell})$, subject to the constraint $\hat{k} \cdot \hat{\ell} = 0$. This projection index, $I(\hat{k},\hat{\ell})$, is constructed to be a smooth function of its arguments so that sophisticated numerical maximization algorithms can be employed, minimizing the CPU cycles required.

The automatic projection pursuit algorithm can be invoked in two ways from the PRIM-9 system. At the simplest level the user simply depresses a button. Starting with the current projection (appearing on the screen), the algorithm finds (and displays on the screen), the projection corresponding to the first local maximum of the projection index, $I(\hat{k},\hat{\ell})$, uphill from it. In this search the horizontal coordinate, $\hat{k}$, is held fixed while the vertical coordinate is varied in the (NDIM-1)-dimensional subspace orthogonal to $\hat{k}$. Depressing the button a second time causes the search to continue, but this time the veritical coordinate is held fixed at the previous solution value, $\hat{\ell} = \hat{\ell}^*$, while the horizontal coordinate, $\hat{k}$, is varied in the (NDIM-1)-dimensional subspace orthogonal to $\hat{\ell}^*$, seeking a further maximum of $I(\hat{k},\hat{\ell}^*)$. Pressing the button a third time causes a further maximization, this time holding the horizontal coordinate $\hat{k}$ fixed at its solution value, $\hat{k} = \hat{k}^*$, and again varying the vertical one, but this time in the subspace orthogonal to the new horizontal coordinate $\hat{k}^*$. This procedure of alternately holding one coordinate

fixed while varying the other in the subspace orthogonal to the first, can be continued (by repeated pushes of the button) until either an interesting view appears or until it converges (subsequent searches produce no change in the projection). At each stage in this iterative process, the results are displayed on the screen as the current projection.

The second mode of invoking automatic projection pursuit allows starting the search at projections other than the current one displayed on the screen. This is accomplished by depressing a different button. This causes a menu to appear on the screen listing the options available. These options allow the choice of any two of the initial coordinates or principal axes of the current data set (isolate) as starting axes for the automatic projection pursuit. This menu also allows the interactive modification of some of the parameters of the automatic algorithm as well as simply displaying the data along the various principal axes without the corresponding search. The user selects from among these options by touching the light pen to the appropriate places on the screen where the options appear. After starting the search at two of these alternate axes, the user continues it by repeatedly pressing the first button as described above.

DISCUSSION

PRIM-9 has been available for production data analysis for a relatively short time so that our experience with it is necessarily limited. We have probably not yet discovered many of its most interesting and useful applications. It has, so far, been employed in the exploratory analysis of multivariate high energy particle physics data, and in both supervised and unsupervised multivariate discrimination analysis in pattern recognition problems.

In the exploratory data analysis application, the system essentially functions as a cluster detection and separation device. By repeated application of view change and rotation (both manual and automatic), the user tries to find those data orientations that reveal to him interesting structure or clustering. Using the automatic projection pursuit algorithm (started at the larger principal axes of the data) to find interesting starting projections, the user then manually iterates the rotation to try to find structure or to sharpen any structure found. This iteration process usually proceeds as follows. A visible coordinate is rotated against one of the invisible ones until the clustering is as sharp as possible. This invisible coordinate is then rotated against the other visible one toward the same end. Another invisible coordinate is then chosen to be rotated against the two visible ones, hopefully increasing the cluster formation and separation. After all of the NDIM-2 invisible coordinates have been rotated against the current visible ones, the whole process can be repeated (since in the process of these rotations the current coordinates have been considerably changed) so long as progress is being made. At any point, the automatic projection pursuit algorithm can be invoked (this usually happens when progress being made by manual rotations is slow).

If a view is found that separates the data into two or more clusters, this view can be used (via masking and isolation) to isolate the clusters into separate data subsamples. These subsamples can then be analyzed individually to see if there are different projections that reveal still further clustering within each of the subsamples. Even if no further clustering is found in an isolate, rotating it in a controlled manner in the multidimensional space can still give considerable insight as to its multivariate properties. The parallax effect of the continuous real time rotation is very useful in providing the third-dimensional effect of depth into the screen.

Moving masks can also be useful in gaining insight into the multidimensional characteristics of data point sets. The simplest of these (as mentioned above) is the sliding of an unmasked zone of chosen width back and forth on the various coordinates invisible to the screen, one at a time. Another more sophisticated approach might be called the "concealed generalized episcotister". Its operation would be roughly as follows. Let each coordinate run between -1 and +1, let a, b, c and d be small fractions, and let the data for this example be six-dimensional.

picture = coordinates 1 and 2

masks = F_3 at -a, B_4 at +b, F_5 at -c, B_6 at +d

rotation = play with (4,3), (5,3), (6,3), (5,4), (6,4), (6,5)

The unmasked hypervolume is a hexadecant in the four coordinates 3, 4, 5 and 6. Rotating these coordinates among themselves essentially rotates this hyper-conical (linearly, not spherically hyper-conical) region around among the (four-dimensional projections of the) points, thus generalizing the rotation of a conventional (sector-disk) episcotister. This approach seems often rewarding in gaining the first clues as to major structure in a point cloud.

If, as another example, we suspect that the "core" (in one or more of the invisible coordinates) of a visible concentration of points is curved, the natural mode of inquiry is to (in order):

(1) rotate until the concentration seems as strong as possible (this will tend to eliminate "tilts" of the core),

(2) mask on one invisible coordinate, coming in from each side until perhaps one-sixth to one-fourth of all points are blotted out from each side,

(3) operate mask-countermask rapidly,

(4) rotate among the invisible coordinates at whim, repeating number (3) all the while.

Appearance of a displacement oscillating with the mask-countermask frequency is an indication of existence, direction, and amount of curvature.

In the pattern recognition applications, the system functions as both a linear and piecewise linear device in supervised and unsupervised discrimination analysis.

As an example of its use as a linear device in a supervised application, consider the problem of finding the best linear discriminant direction in the multivariate space for separating two known data classes. Here, the points corresponding to each class are correctly identified from some external source and the problem is to find the best direction separating the two classes when the data are projected onto that direction. The analysis proceeds as follows. A coordinate is added which contains the information identifying the class of each data point. This extra coordinate is pictured (vertically) against each of the data coordinates (horizontally) in turn, starting with the second coordinate.

While viewing each such projection, the data is rotated among all the NDIM coordinates with the objective of achieving maximal overlap of the two classes in each horizontally pictured coordinate. This process is repeated iteratively until it is impossible to increase the overlap in each of the NDIM-1 projections (2 through NDIM). At that point the direction of the current first coordinate represents the optimal discrimination axis, and picturing the extra (class identifying) coordinate against this first one, directly displays the discrimination achieved.

It is the presence of isolation in conjunction with rotation that gives the system its piecewise linear capability. This is easily demonstrated by considering the simple example of a two-dimensional data set, consisting of two classes whose boundaries are outlined in Figure 3. Clearly, there is no single one-dimensional discriminant direction that can completely separate the two classes (A and $B = B_1 + B_2 + B_3$).

Applying rotation (both manual and automatic), the user might find a projection that achieves a good partial separation (such as P_1 in Figure 3). Using this projection, a mask is constructed at M_1 and the B_1 sample is isolated from its complement $A + B_2 + B_3$. These two isolates are then rotated separately, searching for further structure. In the case of subsample B_1, this will yield a null result. For the subsample complement to B_1, however, the user may iterate to projection P_2 which does exhibit further structure. Using this projection to apply a mask at M_2, the B_2 sample is isolated from its complement $A + B_3$. Continuing with this procedure, the user can further iterate to projection P_3. Applying a mask at M_3 in this projection completely isolates Class A from the remainder of Class B (B_3). Thus, the application of the piecewise linear mask M_1, M_3, M_2 completely separates the two data classes. For the supervised case, where it is known in advance that there are only two classes, this completes the solution.

For the unsupervised case, one can separately analyze the B_1, B_2 and B_3 isolates as well as their union $(B_1 + B_2 + B_3)$, using rotation and masking to determine whether they represent separate classes or whether they are connected, thus representing a single class.

For this simple illustration, the dimensionality of the data was two while the projection pursuit was in one dimension. In PRIM-9, the data dimensionality can be as large as nine while the projection pursuit is in two dimensions. However, the basic principles are the same. One applies rotation and view change, trying to find projections with structure. When structures are found, they are isolated and the isolates are each individually studied seeking further clustering. If found, these are further isolated and so on. When no more structuring can be found among the isolates, then they are analyzed separately and together to try to understand their multivariate properties.

CONCLUSIONS

PRIM-9 has been in use for a short time examining and dissecting multivariate data. Its direct value for this purpose will have to be learned by experience. We have learned from its development that pictorial systems to be effective must, as did PRIM-9, go through many stages of trial and error learning. We now understand that the details of control can make or break such a system. We now recognize that these details of control must be adapted to what is available and to the people who are to use the system. If we had ten knobs and six switches instead of 32 buttons and a light pen, we would have realized the same four essentials in a quite different way. We now recognize the great value of the dynamic aspects of the display, especially easily recognizable rotation. Two aspects, horizontal and vertical, are always before us. We now have a strong feeling

that the third aspect which supports these two best is this dynamic aspect of rotation, more useful than stereoscopy, color, flicker or distinctive characters. We have learned that this sort of pictorial facility can be useful in two quite different ways: first, directly in the interactive analysis of multivariate data, and second, as a source of ideas and approaches for the development of computer algorithms for multivariate analysis. The automatic projection pursuit algorithm, for example, was developed by observing the systematics of the interaction between the users and the PRIM-9 system. These systematics were encoded into a computer algorithm which has been very successful in seeking optimum projections.

Finally, and above all, we have learned that the four essentials of Picturing, Rotation, Isolation and Masking need to work together and that from them much can be learned.

REFERENCES

1. Film: "PRIM-9", produced by Stanford Linear Accelerator Center, Stanford California, S. Steppel, Ed., Bin 88 Productions, April 1973.

2. J.H. Friedman and J.W. Tukey, "A Projection Pursuit Algorithm for Exploratory Data Analysis," Stanford Linear Accelerator Center, Stanford, California, Report, SLAC PUB-1312, September 1973. (to be published in IEEE Trans. Computers)

3. R.C. Beach, M.A. Fisherkeller, and G.A. Robinson, "An On-Line System for Interactive Programming and Computer Generated Animation," Stanford Linear Accelerator Center, Stanford, California, Report, SLAC PUB-939, August 1971.

4. R.C. Beach, "The SLAC Scope Package for the IDIIOM--A Collection of PL/1 Procedures which may be used to control the IDDIOM Display Console," Stanford Linear Accelerator Center, Computation Research Group, Report No. CGTM No. 80, December 1969.

5. Varian Data 620/i Computer Manual. Bulletin No. 606-A, Varian Data Machines, 2722 Michelson Drive, Irvine, California.

6. IDIIOM Technical Description. Information Displays, Inc., 333 North Bedford Road, Mount Kisco, New York 10549.

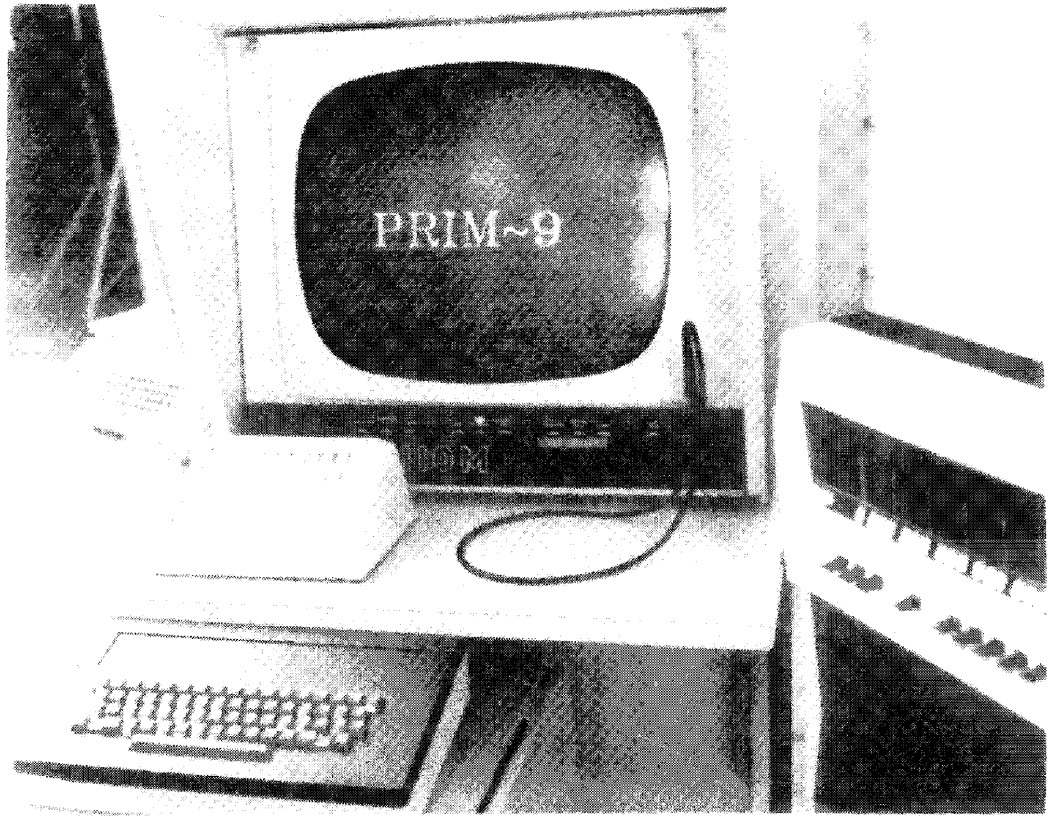


FIGURE 1

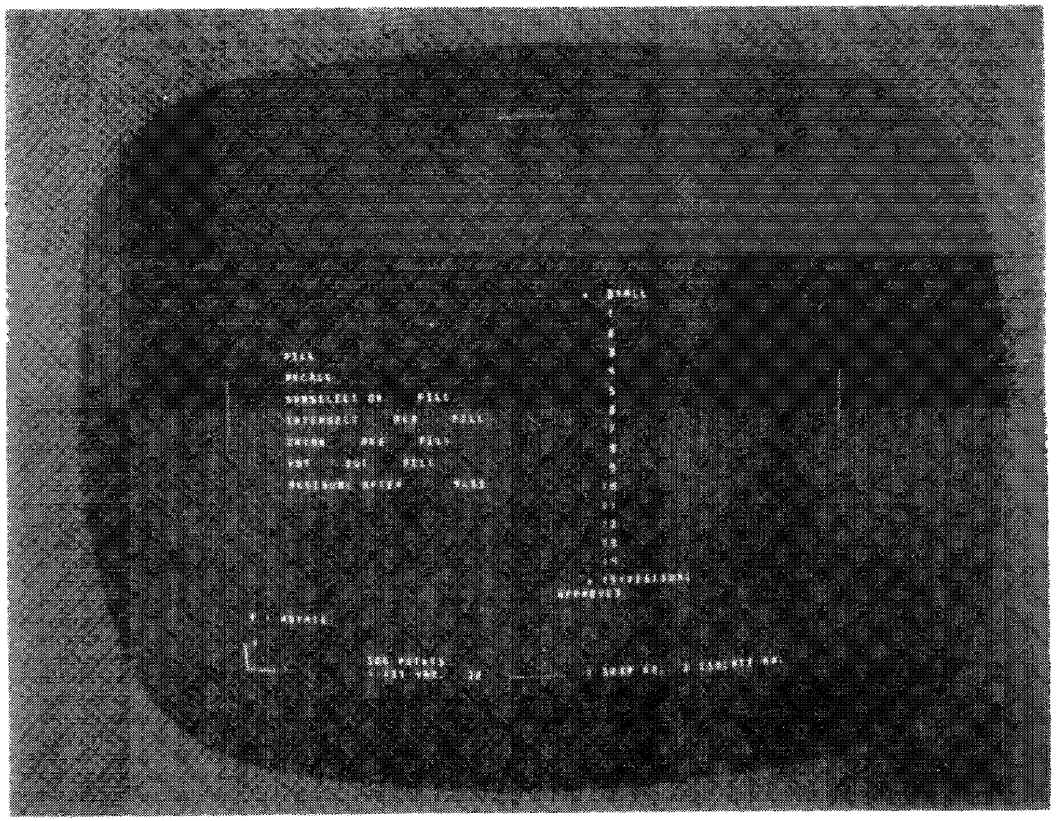


FIGURE 2

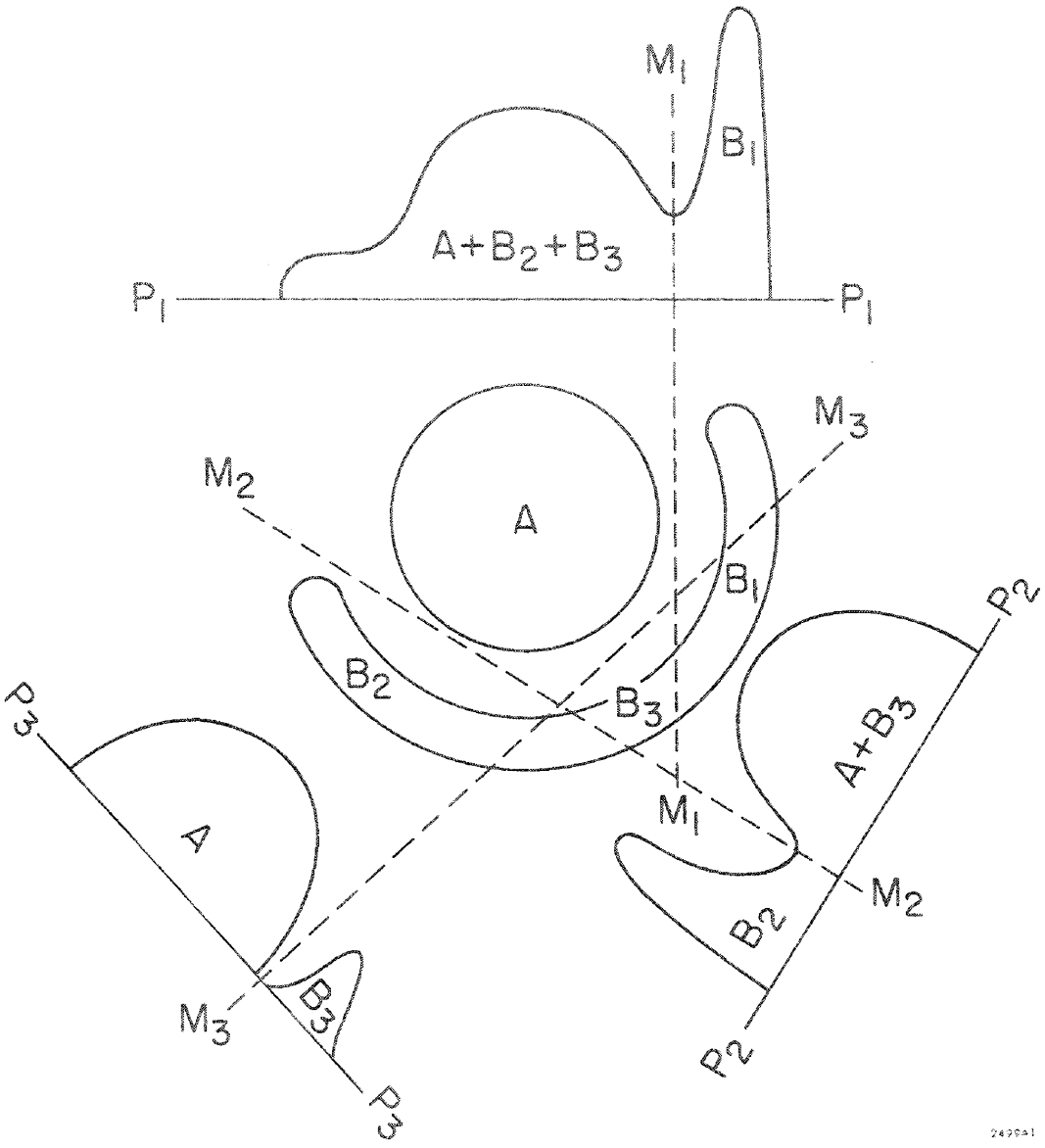


FIGURE 4

Displaying Complex Three Dimensional Objects

Michael J. Archuleta

March 20, 1974

This paper was prepared for presentation at the
Second Spring AEC Scientific Computer Information Exchange Meeting
Work performed under the auspices of the U.S. Atomic Energy Commission

DISPLAYING COMPLEX THREE DIMENSIONAL OBJECTS

by Michael J. Archuleta
Lawrence Livermore Laboratory, L-73
Livermore, California 94550
(415) 447-1100 x3361

ABSTRACT

This paper describes a powerful machine transportable visible surface algorithm. Input to the algorithm can be concave or convex polygons. The output can be a combination of line drawings, shaded raster lines, or 3D contour plots. Several new techniques for displaying multi-valued functions are described.

AN OVERVIEW

While other researchers have been pursuing the development of new visible surface algorithms[4], I have been faithfully enhancing the technique that was developed by Gary S. Watkins in 1970 at the Universtiy of Utah[6]. I had the great pleasure in being able to work with Gary Watkins as his programmer and thus became quite familiar with the algorithm. There are several reasons for using this algorithm: 1) It can produce a variety of different types of grey level pictures; 2) line drawing output is available [1]; and 3) it can handle a very broad range of polyhedra as input. In working with this algorithm, I have been able to develop a machine transportable code that incorporates all of the above features plus some new special techniques.

The Watkins' visible surface algorithm accepts as input convex or concave polygons. The coordinates for these polygons are in a left handed coordinate system. Polygon clipping is performed to a frustrum of vision which opens out along the positive z axis. It assumes that the eyepoint is located at the origin of the system. (See Figure 1) The algorithm produces as output either vectors for line drawing displays or shaded raster line segments for grey level displays.

Approximately 95% of the code is written in ANSI Fortran. The remaining 5% of the code is highly machine dependent since it can take advantage of packing data together according to the word size of your machine. Since you must create a subroutine which packs the data, you must also develop a subroutine which will unpack the data. The reason I have you make up your own packing and unpacking routines is that these routines handle up to 13 different variables which collectively require 130 bits. In addition to the packing routines, there are several other machine dependent routines which perform teletypewriter I/O, and hence are machine dependent.

There are three "new" features which I have incorporated into this algorithm: 1) Superimposing contour bands on a 3D surface; 2) creation of cap polygons in slicing; and 3) extensions to the concept of edge sharing.

2D contour plots are nothing new in computer graphics; neither are 3D contour plots[3]. However, the Watkins' algorithm was designed in such a way that implementing 3D contour plots was a simple task.

The Watkins' algorithm projects all 3 dimensional polygons onto a 2D plane. The projection plane is divided into horizontal raster lines and the algorithm computes the visibility of the polygons a raster line at a time. A segment (the portion of the raster line which intersects the polygon) is sorted and compared with other segments to see which

are visible. (See Figure 2) Each segment has x, y, z, delta x, delta y, and delta z information for both the left and right end points. The delta information is added to the current data to see what the new value will be on the next scan line. In 1971 at the University of Utah, Henri Gouraud added a shade and a delta shade to each end point of a segment. This resulted in the now popular Gouraud smooth shading technique[2].

To do the contour plots, I added a current contour value and a delta contour value to the segment information. Assume we have a visible segment whose left contour value is 10 and right contour value is 18. If we are interested in seeing contour bands at 12 and 15, we perform a simple calcualtion to determine the x location of these contour bands on the segment. When we update to the next scan line, we might get a different location for the x values. When these points are connected together, there will appear a curved contour band. (See Figure 3) This is rather unique since most contour plotting techniques must draw straight lines across the face of a polygon. The only way they can approximate curves is by increasing the number of polygons or do curve fitting techniques. (See Figure 4)

Now for a few words about cap polygons. The polygon clipper that is used in this algorithm is similar to the one developed by Sutherland and Hodgman[4] in that a polygon is clipped to the six planes of a frustrum of vision. There developed at LLL a need to be able to clip the front portion of an object away so the innards of a 3D finite element or finite difference model could be studied. Since these models represent solid objects, it was necessary to put a cap polygon over the hollow shell created by polygons. (See Figure 5) This was a natural for the polygon clipper since it would save the edges that it had to create while clipping to the front plane of the frustrum. Once the clipper had processed all the polygons of a solid object, then it would be left with a set of edges on the front plane which defined a closed cap polygon. Since contouring and shading information is passed through the clipper, the cap polygon will also have this information and contour bands can thus be plotted on the inside of an object.

The last item worth discussing is edge sharing. One drawback of the Watkins' algorithm is that adjacent polygons had to store the shared edge twice. I have done nothing more than provide simple coding to eliminate this redundant storage. However, there is an interesting application of edge sharing in doing line drawings. Take, for instance, a cylinder which is composed of many polygons. If you go through the process of eliminating polygons which face away from you, and you say don't draw edges which are shared, you will end up with a picture of a tube. Since the back facing polygons were never stored, the edges on the perimeter of the cylinder were never shared thus making them plottable. (See Figure 6) The utility of this is in doing

contour plots where it is often difficult to determine which are contour bands or polygon boundaries. This technique resolves the ambiguity.

The greatest feature of all is that this program is available for use at installations other than Livermore. There is a complete document available on how to use this algorithm and it can be obtained by calling or writing to me. A lot of work has gone into the algorithm in making it easy to implement on different computer systems. The numerous options which are available allow you to explore many new ways in displaying your data. Thus, in a few weeks time, you too can be producing sophisticated pictures with a powerful visible surface algorithm.

REFERENCES

[1] Archuleta, M.J., Hidden surface line drawing algorithm, Computer Science Department, University of Utah, UTECH-CSc-72-121, June 1972.

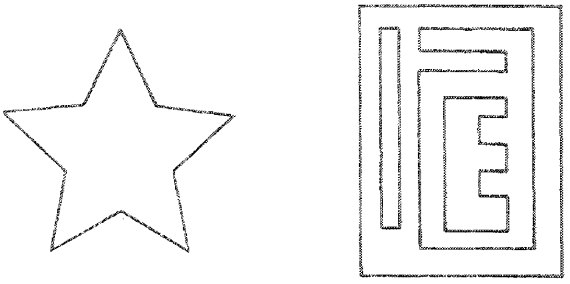
[2] Gouraud, H., Computer display of curved surfaces, Computer Science Department, University of Utah, UTECH-CSc-71-113, June 1971.

[3] Rashide, Y.R., Computer graphics in stress analysis, ACME Winter Annual Meeting, November 1970.

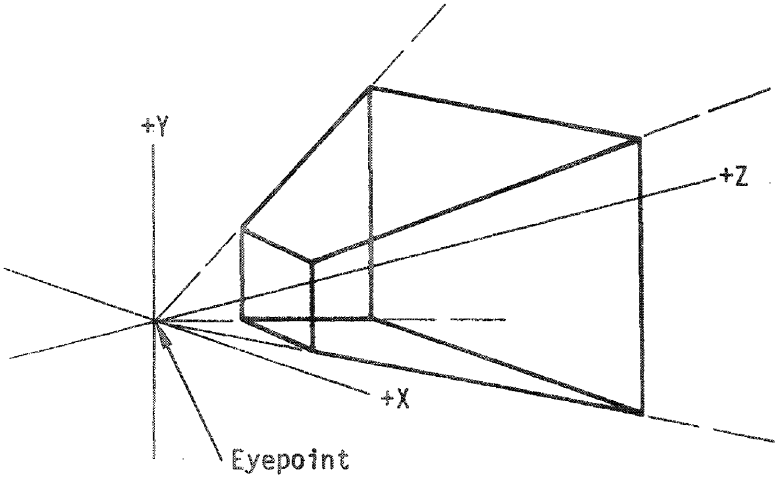
[4] Sutherland, I.E., and Hodgman, G.W., Reentrant polygon clipper, Comm. ACM 17, 1 (January 1974), pp32-42.

[5] Sutherland, I.E., Sproull, R.F., and Schumacker, R.A., Sorting and the hidden surface problem, National Computer Conference (1973), pp685-693.

[6] Watkins, G.S., A real time visible surface algorithm, Computer Science Department, University of Utah, UTECH-CSc-70-101, June 1970.



Input polygons



Frustrum of Vision

Figure 1

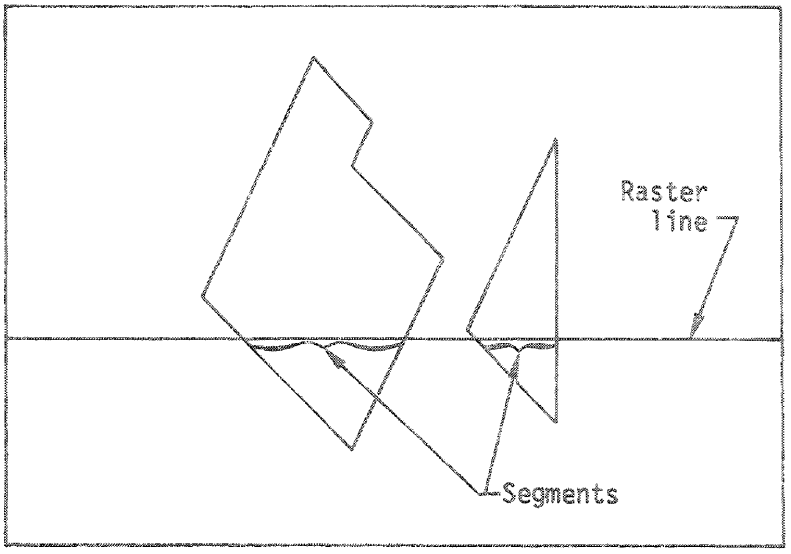


Figure 2

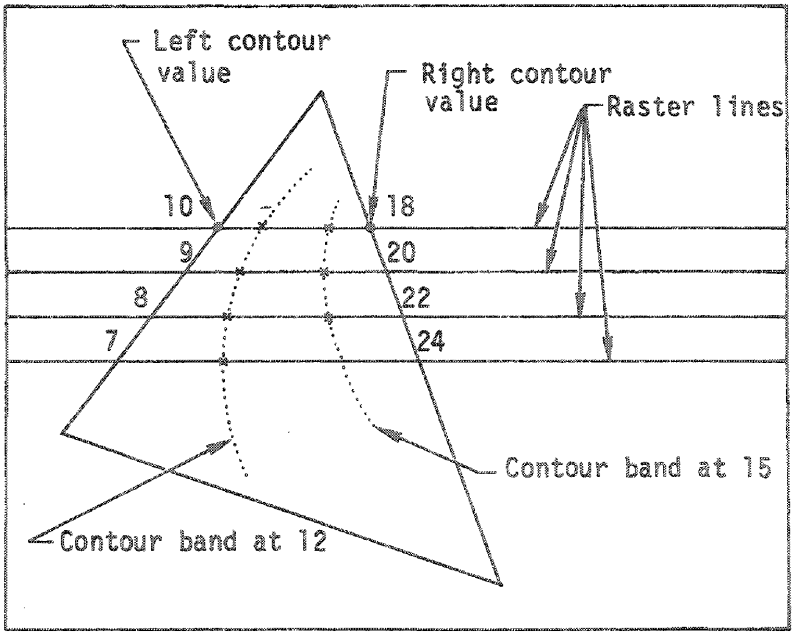


Figure 3

Cube with Superimposed Contour Bands

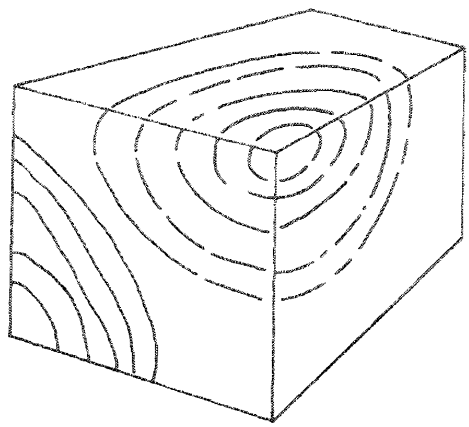
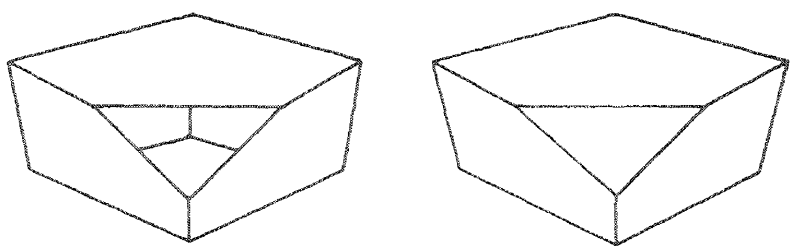


Figure 4

Shell of a Cube after Cipping
to Front Z Plane



Without cap polygon

With cap polygon

Figure 5

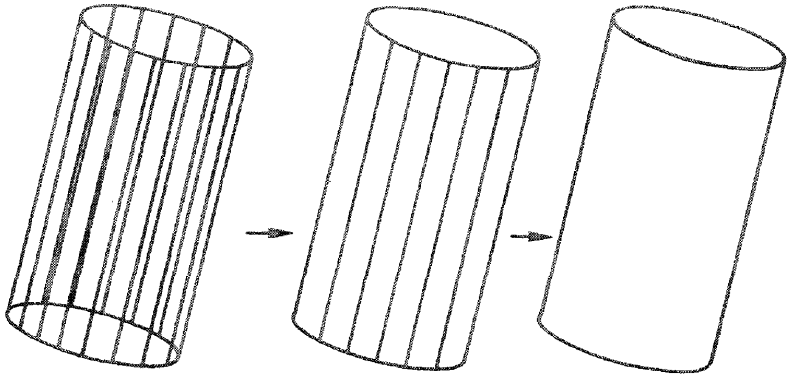

Cylinder with all polygons
Cylinder without backfacing polygons
Cylinder without shared edges

Figure 6

DISTRIBUTION

LLL Internal Distribution

Michael J. Archuleta	L-73	5
TID File	L-9	15

External Distribution

Arnold Peskin Brookhaven National Laboratory Associated Universities, Inc. Upton, L.I., NY 11973	2
Technical Information Center Oak Ridge, TN 37830	2

COMPUTER GENERATED MOVIES - ANOTHER DIMENSION IN MAN-MACHINE COMMUNICATIONS*

by

Raymond L. Elliott, S. Robert Orr, and

Eldon C. Pequette

ABSTRACT

A typical hydrodynamic physics program may run up to 20 hours on a CDC 7600 computer and generate up to a billion numbers. The biggest problem facing the users of these programs is simply comprehending what has been calculated. Stacks of computer listings are typical outputs but serve as a poor communication medium. Static pictures are much better than listings but are limited to two dimensions. Through the use of movies, one adds a third dimension to the communication process. This paper will describe the evolution of computer generated movies at Los Alamos Scientific Laboratory and describe the techniques now in use.

"What results do we have from last night's production runs?" is a question asked daily by weapons designers at Los Alamos Scientific Laboratory. It is not a question that can be answered by a single number or a small set of numbers. A typical production run will take from 20 minutes to 20 hours of CDC 7600 time and may generate billions of numbers. Through the use of computer generated movies we are able to present a very good description of what has been calculated. The movies are not only an excellent source of information, but valuable as a teaching medium.

The production codes are used to model physical events and usually calculate physical variables as a function of time. The geometry of a problem is usually described by a mesh containing from 1,000 to 15,000 nodes. These two dimensional meshes are not necessarily rectangular and are, at times, quite distorted. A number of physical variables are associated with each node and each zone. Each problem will run several thousand cycles; thus generating billions of numbers. The presentation and interpretation of these numbers is a very complex problem.

The production codes exhibit their results in a number of ways. One method is to generate hundreds of pages of printed output. This is the most visual demonstration to the casual observer that the individual is doing some work as shown by the huge piles of listings occupying his office. They are of very limited use otherwise with only a limited number of pages giving useful information. It should be pointed out that the designer is constrained to produce such long listings

*This work performed under the auspices of the U. S. Atomic Energy Commission.

since he never knows precisely which parameters will be essential for his analysis. In a given case, however, he may use only a small percentage of the information printed.

A second form of output is a set of tapes to draw pictures on paper through the use of a mechanical plotter. The pictures are complicated, taking up to an hour each to plot. If the plotter is available, the tapes are readable, and the plotter does not malfunction, the designer will have one or two excellent plots late in the day--provided he chose the right set of parameters to plot. These plots are excellent to work from (very detailed, large scale, and may be written on) but are too time-consuming and awkward to convey enough information about a production run.

The main source of information has been and will continue to be computer output microfilm (COM). This is generally 35 mm roll film with an ever-increasing percentage being in color. The Laboratory will also have a microfiche capability in the near future. The results are displayed at regular intervals during the course of a problem and in many different ways. This film provides all the information that the designer was able to request before the problem was run. In most cases, hindsight indicates that a picture of the problems at a given time and a particular view was not requested and would have been invaluable in analyzing the results of the production run. The standard COM output gives little feel for the relationship between physical variables and time. However, the data record of the run is in a compact form on COM and is saved for later reference.

Movies give the viewer another dimension in viewing the results of the calculations. It gives a qualitative indication of the relationship between physical variables and time. We generated our first movie from computer-generated output in 1960. The problem was run on an IBM 704 computer and the geometry was plotted on a mechanical plotter at intervals during the problem. These drawings were then photographed on 16 mm film with each plot being exposed 10 times. The result was a poor movie, rather jumpy, but still valuable and indicated that movies would indeed serve a useful purpose in problem analysis.

During the early development of 16 mm film by the Laboratory's Central Computing Facility, the turn-around time for 16 mm film was from two to six weeks. When this was reduced to a one day turn-around time, the use of movies became a practical tool for the designers.

Several problems become immediately evident when attempting to make movies simultaneously during the running of a production code:

1. The standard film rate for 16 mm projectors is 24 frames/second. This would require 1440 frames for a one-minute movie which is very expensive if done on a production basis.
2. The designer must decide prior to his run what kind of movie he wants, the portion of the problem he wishes to view, and the time interval for generating movie frames. It is easy to make an error.
3. It is difficult to make more than one movie on a given production run.
4. Production codes typically run at a variable time step, and it is difficult and expensive to provide equal time intervals for the movie.
5. It is difficult to make a movie and generate standard COM output on a given run.

In item one (1) we have been able to reduce the number of frames needed by the use of variable speed projectors. These projectors have viewing rates of from one to 24 frames/second, both forward and reverse. This enables one to stop the film at various times, reverse it and view a portion of the film repeatedly. This has proved invaluable, not only in computer-generated movies but for other movies also.

A separate movie code, M^2C (MAGEE Movie Code), was developed to solve some of these problems. M^2C picks up data from a file written by the production codes. This intermediate file contains all designated variables at a number of problem times. These variables are represented by 20 bit words giving about four significant figures and a data range of 0.000015 to 65,536.0, both positive and negative. In most problems, several hundred problem times will adequately describe a problem. Often data are saved at large time intervals during certain phases of a problem and at small time intervals at other phases. This intermediate file structure is well defined and is now being generated by a majority of production codes.

M^2C has the ability to generate a specified set of problem times from those given through interpolation. This enables one to make a movie with constant time increments with data that is sparse at some times and dense at others or even when given at unequal time intervals. The code will also generate new variables from those given; for example, one may convert from rectangular to polar coordinates. The modified data is then available for making movies.

At present, M^2C contains eight different plotting modules. Each module is independent of the other modules and is easily modified for special plots. New modules will be added as required. Each plot module contains a number of easily specified options; giving the user complete control of the generated movie. Some of the options available are color, plot orientation, background grid control, selection of variables, dynamic tracking of a specified node, partial mesh selection, and time-step control. The plot modules include mesh plots, interface plots, contour plots, several isometric plots, rotational plots, and several experimental plots.

M^2C has, in addition, a number of options for titles. A movie can be completely generated by M^2C with titles, any number of individual movies, and leader at both ends. The generated film is ready for public showing upon processing.

The movie code is at present used for movie generation by a number of production codes at LASL and is being adapted by others to provide all COM plotting output. We intend to use the code as a basis for providing terminal graphics. It will be extended to provide more types of plotting and to give multiple plots per frame.

This is not the only code in use at LASL for movie generation. It is a special code designed to display the results of calculations involving two dimensional meshes. It has proved itself as a valuable tool in a production environment and is unequaled in presenting the results of two dimensional time-dependent mesh calculations.

AN INTERACTIVE DIGITAL IMAGE PROCESSING AND DISPLAY SYSTEM

L. Hayes, C. Journeay, M. Wirth, Lawrence Livermore Laboratory; L. Hatfield, University of California, Davis

(Paper not received in time for inclusion in the Proceedings)

A COLOR MOVIE FACILITY

by Stephen R. Levine

March 26, 1974

This paper was prepared for submittal to
Second Spring AEC Scientific Computer Information
Exchange Meeting

A COLOR MOVIE FACILITY *

by Stephen P. Levine
Lawrence Livermore Laboratory, L-73
Livermore, California 94550

INTRODUCTION

At the Lawrence Livermore Laboratory, the present scheme for generation of computer-generated color films is to record each of the desired color segments separately on black and white film. These black and white films are then combined in an optical printer using color filters to produce the final film.

A major obstacle to wide usage of this method is the long turn-around time required for completion of the film. Consequently, color films have been used only in the final stages of a project rather then during the debug phase. With the intent to make color film a routine output medium, a DD80A Microfilm Recorder was modified by adding a white phosphor CRT and a color filter changer permitting fast, one-pass generation of color computer-generated film.

With the advent of software to produce shaded color pictures, a new problem appeared. A single color frame at full resolution contains over 3 million points. Using the point plot command (36-bits), the storage costs on disk or tape of even a film of moderate length become prohibitive.

A special interface was designed that allowed compact representations of these shaded pictures. The 36-bit shaded raster command will plot up to 4095 points, varying the intensity according to a user-specified slope. Typical shaded pictures can be described using 0.1% to 1.0% of the storage previously required. In addition, the interface allows compact representation of digitized pictures and automatic expansion of low-resolution pictures on a high-resolution grid.

The direct connection of the device to a CDC 7600 will make feasible for the first time at Lawrence Livermore Laboratory the routine generation of complex computer-generated color film.

---

* This work was performed under the auspices of the United States Atomic Energy Commission.

DESIGN GOALS

The function of the DD80A microfilm recorder is to provide a facility for direct generation of color computer-generated films. The requirments for this facility are

1. Direct recording on color film for rapid turnaround.
2. Capability of 32 grey levels.
3. Efficient handling of raster pictures.
4. Software compatibility with existing DD80C microfilm recorder.

DD80A HARDWARE

The DD80A [1] is an old (vintage 1962) electrostatic deflection microfilm recorder. It has the capability for drawing vectors, plotting characters and plotting points on an addressible raster of 1024 x 1024 points. It is quite fast in that it can plot a character or point approximently every 5 microseconds. Its instruction set contains only five 32-bit instructions.

1. Plot a point at (x,y).
2. Position beam at (x,y).
3. Draw vector from current beam position to (x,y).
4. Enter character mode at (x,y).
5. Advance film.

The DD80A has been modified to plot at 32 intensity (grey) levels. The intensity is selected from the interface via a new command. A color filter changer with 8 filters has been added along with a new cathode ray tube. This CRT has a special "white" phospher to allow recording of "color" images. The filters are selected by setting the appropriate bits in the new command.

Color computer-generated film is created by using color film in the camera and exposing it through the color filters. The user sets the color filter to the desired color and then outputs the data to be plotted in that color.

The DD80A interfaces to a portion of the LLL computer network as shown in figure 1. The DD80A looks like an on-line tape unit to the host system. The user interacts with the computer via the TTY subnetwork. A preview of graphic output can be viewed on a TV display also connected to the computers via another subnetwork.

When the user is satisfied with the output, she/he can then direct the graphics to the DD80A for recording on film. Since the device is on-line, no intermediate disk storage for pictures is required.

INTERFACE

The DD80A is connected to two CDC 7600 PPU's by means of an LLL designed interface [3]. In addition to handling the transfer of commands from the 7600 to the DD80A, the interface contains logic that allows the DD80A to appear to the user as if it had an entirely different command structure, including some complex new commands. In particular, the 32-bit, 4-character-per-word DD80A is programmed using the 36-bit, 6-character-per-word DD80C instructions. The interface translates these commands into DD80A instructions. DD80C tapes can be plotted on the DD80A with no software translation whatsoever. In addition, a new command directs the interface to set intensity and to change color filters.

NEW COMMANDS

Four new commands have been added to the DD80A via the interface. Their purposes are to provide access to the new color and grey level features and to allow a compact description for certain types of raster data.

The raster commands are designed to plot pictures which are made up of points lying on a uniform grid. Since the points appear in a fixed order (left to right, top to bottom), the raster commands are designed to provide a savings in user picture storage by requiring the user to supply only the picture information, with the interface calculating the appropiate x,y positions. In addition, the auto-increment feature allows pictures defined on a 512 x 512 or 256 x 256 grid to be plotted on the full 1024 x 1024 raster. The interface can automatically plot the picture information on every other raster position, skipping every other line, or plot on every fourth point and every fourth line. This feature is operable on all of the raster commands.

1. COLOR-INTENSITY

This command sets the intensity and/or color filters. It can appear before or after any other command.

2. SHADED RASTER LINE (SRL)

The hidden surface software [2] at LLL generates a picture as a series of raster lines. Each line is broken up into a number of segments. A segment can extend over as few as 1 or as many as 1024 points. A segment is specified by the number of points to be plotted, the starting intensity of the segment, and the intensity change per point. The interface takes this single command and generates a sequence of point-plot commands along with the linearly interpolated intensity for each point. To plot a picture, a Set Raster Origin command (SRO) is issued which sets the upper left corner of picture. This is typically x=0, y=1023. The SRO command is followed by all of the segments. After raster position 1024 of any line is plotted, the interface skips the correct number of lines (0,1,3) and repositions its x register to zero. Thus only segment data need be supplied by the user.

3. BUCKET OF INTENSITIES (BOI)

This command is designed to plot digitized pictures, and also facilitates the display of data digitized on a larger grid than the hardware's 1024 x 1024. The data is organized as a collection of lines (1024 maximum). Each line is a collection of picture elements. A picture element is 2,3,4, or 6 bits. (Picture elements of 2,3, or 4 bits are used for data that is digitized at 4,8, and 16 levels of intensity respectivly. The 5 high-order bits of the 6-bit picture element are used for 32 levels.) These 4 picture element sizes are provided to minimize storage when 32 levels are not required. The picture elements are plotted on raster positions according to the auto-increment setting.

For each picture, the user specifies 4 items.

1. The number of picture elements to be skipped at the beginning of each line. The hardware performs this skip very rapidly.

2. The number of picture elements to be plotted.

3. The number of picture elements to be skipped at the end of each line.

4. The number of bits defining a picture element.

This information, along with the number of lines of data to be plotted, is issued prior to the actual data. This command can be used for scanning data. For example, consider data that has been digitized on a 2048 x 1024 grid. Without moving the data in core, different portions of the data may be presented and/or expanded (auto-incremented) by changing only the control words of the BOI command.

4. BUCKET OF BITS (BOB)

This command is identical to the BOI command except for two distinctions. First, the picture elements are 1 bit, allowing for either a point or a blank. Second, where a 0 appears, the interface does not issue a point-plot command but rather skips to the next picture element in 50 nanoseconds. Thus, the time required to plot a BOB picture is a function of the number of 1 bits present, rather than of the total number of picture elements in the picture. Data that is formatted for the TV displays can be plotted very quickly using this command.

CONCLUSION

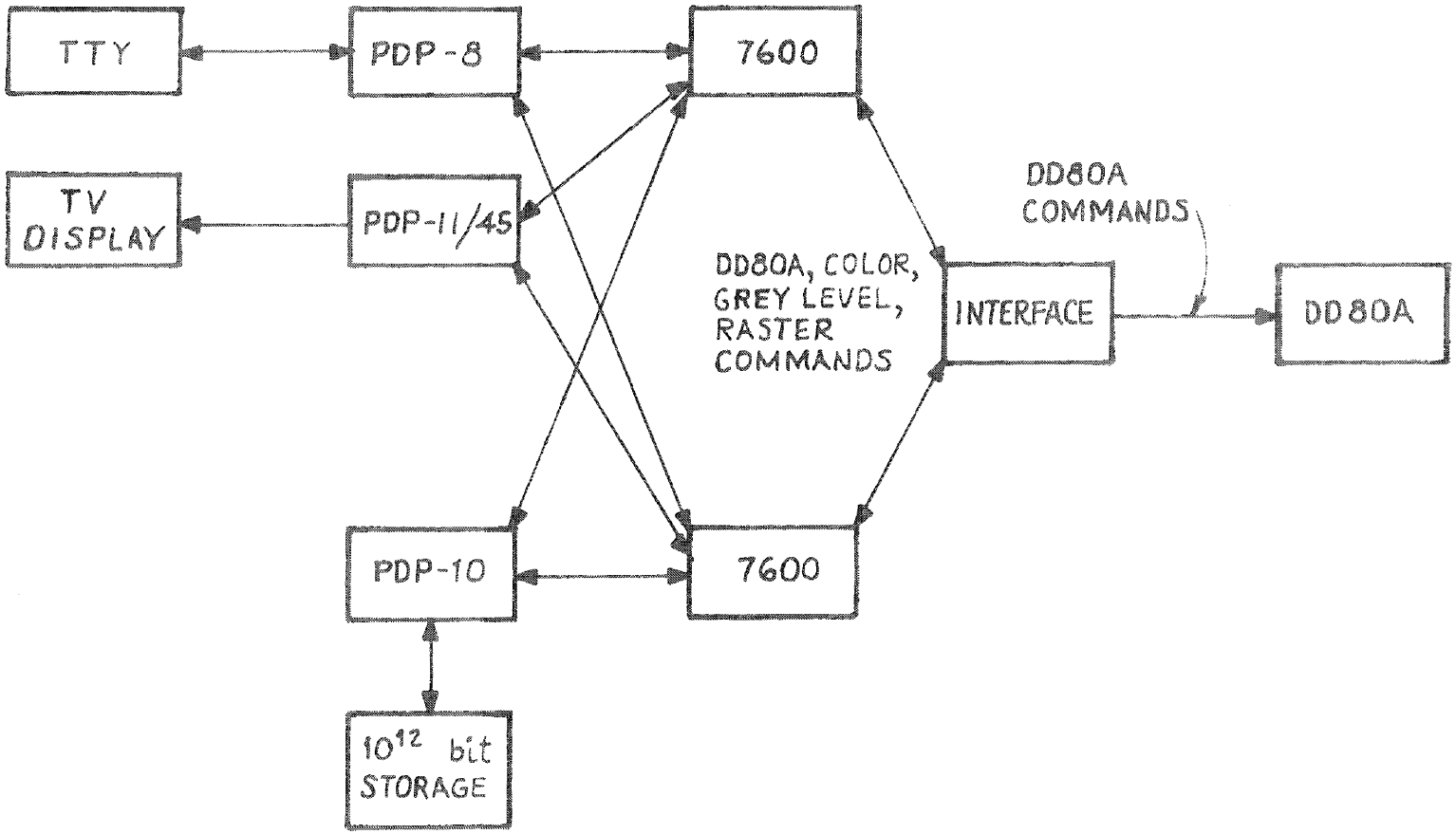
This facility has been designed to efficiently produce color computer-generated films with several new features that minimize storage requirements for picture data, thus reducing the overhead for handling large and complex files.

REFERENCES

[1] Cecil, A. and Michael, G. DD80 Programmer's Manual, Lawrence Livermore Laboratory, Report N 2.8-002 (1964).

[2] Archuleta, M. Hidden Surface Processing, Lawrence Livermore Laboratory, UCID-30057 (1973).

[3] Pryor, K. and Long, R. Lawrence Livermore Laboratory, Internal Document LEA-73-3005-99 (1973). Readers outside the Laboratory who desire further information on LLL internal documents should address their inquiries to the Technical Information Department, LLL, Livermore, California 94550.



A PORTION OF THE LLL COMPUTER NETWORK

fig. 1

DISTRIBUTION

LLL Internal Dstribution

Stephen R. Levine	L-73	1[illegible]
TID File	L-9	1[illegible]

External Distribution

Technical Information Center Oak Ridge, TN 37830	2

SESSION II

Physical, Engineering, and Biomedical Applications

Chairman: G. H. Campbell
Brookhaven National Laboratory

ADVANCED GRAPHICAL DISPLAYS USED IN THE ANALYSIS OF HIGH ENERGY PHYSICS DATA

M. F. Hodous and I. A. Pless, Massachusetts Institute of Technology

(Paper not received in time for inclusion in the Proceedings)

A Pattern Recognition Code for Curved Tracks in Cylindrical Spark Chambers*

W. N. Schreiner†, D.R. Gilbert‡, W. P. Trower

Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24061

P. Schübelin

Brookhaven National Laboratory, Upton, New York 11973

ABSTRACT

We describe and evaluate a computer code, PITRACK, which associates sparks into tracks from digitizings produced by a system of nine cylindrical wire spark chambers operating in a 10 kG magnetic field. PITRACK was written in FORTRAN IV and requires 72K octal words of CDC-6600 core storage for execution. Packing and unpacking routines required by the data tape format account for ~ 20% of this core. Track recognition time principally depends on the initial number of tracks to be recognized, N, as

$$t \sim 0.4 + 0.04N^2 \text{(seconds per event)}.$$

PITRACK identifies ~ 94% of all tracks found by a human scanner while ~ 1% of the tracks it found were spurious.

---

*Work performed under the auspices of the U.S. Atomic Energy Commission

†Present Address: Brookhaven National Laboratory, Upton, N.Y. 11973

‡Present Address: University of Toronto, Toronto, Canada

A Pattern Recognition Code for Curved Tracks in Cylindrical Spark Chambers

Introduction

The Multiparticle Argo Spectrometer System, MASS, seen in Figure 1, was capable of recording $\sim 10^4$ inelastic events per hour and recorded 4×10^6 events during its first experiment: proton-proton interactions at 28.5 GeV/c[1]. The analysis of any significant portion of these data required a highly efficient automatic procedure for associating sparks into tracks to form events. Human intervention, by visually scanning events, had to be limited to a relatively small control sample to keep the data processing tractable.

The Vertex Spectrometer[2], VS, of MASS was a system of nine cylindrical wire spark chambers operating in a 10 kG magnetic field. Charged particles produced in an event emerged from a centrally located target, followed helical trajectories, and where they intersected a chamber a spark occurred. The data from each chamber appeared as two independent sets of digitizings. The first set was equivalent to a projection of the helical tracks onto a plane perpendicular to the axis of the helix and resulted in circles. The second set was related to the dip angle of the helical tracks and resulted in straight lines. Combining the two sets of tracks produced a three-dimensional representation of the particle trajectories of an event.

Track Recognition Code

The computer code, PITRACK, was developed to provide track recognition for the VS. The global strategy used in PITRACK was to develop several algorithms which provided a few good initial track candidates. These track candidates would then be upgraded until they either fulfilled most of the conditions for acceptable tracks or failed enough to be rejected. Several

intermediate stages of tests and a final track selection determined the ultimately acceptable tracks. Permeating the code was the philosophy that when a track failed a specific test, every effort was made to modify the track until it became acceptable rather than reject it.

We found no single search algorithm which would provide satisfactory track recognition under all circumstances, thus several techniques and multiple searches were employed. The most difficult task was defining for a digital computer precisely what constituted an acceptable track. Again, no single set of conditions was found to exist, so a complex set of rules was developed to include the diverse range of tracks acceptable to human scanners. For example, in many cases it was found necessary to explicitly take into account certain idiosyncracies of the VS chambers and readout system.

Each chamber of the VS produced two sets of digitizings by means of a magnetostrictive readout system. The first set was from the high voltage wires which ran vertically and were parallel to the magnetic field. The second was from the ground wires which were rotated at an angle of $\pm$ 26.5° with respect to the vertical, the slope alternating on successive chambers. The high voltage and ground wires were separated by 3/8 inch, with the result that a spark produced two digitizings in different planes, rather than a single point in 3-dimensional space.

The projection of the digitizings from the vertical wires onto the VS median plane, shown in Figure 2, was called the S-view. The rectangle symbolizes the liquid hydrogen target. The beam enters from the bottom, and

the magnetic field of 10 kG is directed into the plane of the figure. The nine arcs are the outlines of the cylindrical chambers. The darker points are the S-sparks obtained from the vertical wires. The tracks, as found by PITRACK, are shown along with certain ancillary information. The arc distance to a spark measured along a chamber from the spectrometer center line was called S.

Spark height information was obtained from the slanted wires. It was generated on the set of lines of intersection of the chambers and a cylinder defined by the helical path of the particle and was called the Y-view. Possible spark coordinates occurred whenever a slant wire which fired crossed a vertical wire which also fired. The Y-view of a typical track is shown in Figure 3. The vertical lines represent the chambers and the dark points are the Y-sparks. The circled sparks have previously been associated with another track. The horizontal axis is the S-view arc distance along the track. In the Y-view the track is a straight line. The dark point at the rear of the target is the event vertex.

The Y-view differed from the S-view in two respects. First, the former was not defined until a possible track had been found in the S-view. Second, in contrast to the S-view, not all the coordinates defined by the many intersections corresponded to real sparks. Therefore, tracking was performed first in the S-view. The Y-view was then used to confirm a track candidate and determine its dip angle.

Strategy

An initial event vertex was obtained by projecting the trajectories of external triggering particles back into the VS and intersecting them with the known beam trajectory. For particles detected in the High Momentum Spectrometer (HMS) the digitizings nearest its extrapolated trajectory in the VS were

accepted as those from the LMS track. Particles detected in the Low Momentum Spectrometer (LMS) were not similarly treated because of inexactitudes in its trajectory introduced by multiple scattering, energy loss, and LMS spatial resolution.

Five distinct interlocking operations indicated in Figure 4, were performed to extract tracks from the sets of S and Y coordinates.

1. S-view Track Hypothesis

Initial S-spark track candidates were provided by four search techniques. The first, Smooth Track Search, emulated the ability of the human eye to distinguish smooth arcs in a collection of digitizings. The second, Forward Search, found tracks with little curvature which lay close to one another in the forward direction. The third, Brute Force Search, resorted to trial and error to sort out the more complicated tracks. The fourth, 2-spark Search, looked for steeply dipping tracks which exited the chamber volume after passing through only the first two chambers.

Once an S-spark was successfully associated with a track, it was excluded from further initial track searches. However, associated sparks would be used to fill in gaps on other tracks during their development. Typically 5% of all the S-sparks were associated with more than one track.

2. S-view Evaluation

Track evaluation took place at many stages of track development. Before the initial track evaluation, as many sparks as possible were associated with the trial tracks. Chambers were flagged if the track missed or went through an inactive region. The most notable requirement in track evaluation was that trial tracks have a minimum number of S-sparks depending on their configuration.

3. Y-view Tracking

A search was made for trial S-view tracks to find Y-sparks falling along a straight line emanating near the vertex. If no track with a sufficient number of Y-sparks was found, the S-view track was flagged.

Because slant wires alternated in direction from chamber to chamber, the ambiguity created by recording the passage of more than one particle through the chambers was removed. However, reflections due to nearby tracks often occurred on the even or odd numbered chambers. An example of such a track reflection is seen in the upper part of Figure 3 among the circled sparks. Authentic Y-spark associations must therefore include both even and odd numbered chambers.

4. S-view Track Development

The S-spark tracks provided by the initial track searches had many shortcomings. For example, the Smooth Track Search often purposely supplied incomplete tracks. Furthermore incorrect sparks were frequently associated with a track when more than one digitizing existed near the track on a given chamber, or one of the initial searches projected a track incorrectly to a neighboring chamber. These problems stemmed from difficulties such as readout noise, signal inversion, track age, lineup imprecision, delta rays, multiple scattering, nearby tracks, etc. Convergence to the best possible track in the S-view was affected by examining alternative spark combinations. The best track was defined in terms of, first, the number of S-sparks associated with the track and then its chi-squared.

The above type of track development was concerned primarily with the internal consistency of the sparks in a track. Further optimization of the tracks occurred periodically when study was made of how the tracks collectively formed an event. For example, since the vertex was not fixed during tracking whenever it moved significantly an attempt was made to bend tracks to the new

vertex by substituting S-sparks on the innermost and outermost chambers, provided this resulted in an improvement in the track quality.

5. Intertrack Comparison

At several stages in the program the established tracks were compared with one another to eliminate spurious ones. These occurred primarily in the forward direction where tracks were abundant, rigid, and closely spaced. Near the VS center line there were dead regions in each chamber which permitted the non-sparking passage of the beam particles. These dead spots introduced distortions and occasional spurious sparking. Thus, for example, forward tracks and tracks which shared sparks in the S-view or the Y-view were carefully scrutinized. In other regions of the chambers the comparison allowed replacement of an occasional incorrect or incomplete track by the correct one.

The location of the vertex was computed after each new track was found by forming a weighted average of track intersections with the beam. The weighting factor was $\sin^2 \theta$ for the Z position and $\cos^2 \theta$ for the X position, where θ was the angle between the track and the beam at their point of intersection. The largest cluster of weighted track intersections was used to compute the event vertex and tracks whose weighted intersections fell outside this cluster were excluded.

After tracking was completed, if there was an IMS trigger, its track was selected from among the tracks identified by the program and flagged. Any of the remaining tracks which passed too far from the final vertex were flagged. These tracks typically resulted from beam halo particles, secondary interactions or decays.

Flexible requirements were an important ingredient in the code. Since the chamber efficiencies had been found to be $\sim 95\%$ and independent of the number of tracks[2], a valid track would be expected to have a small number of

missing sparks in the S-view. However, before rejecting a trial track with several apparent misses, the S-spark acceptance window was enlarged to allow sparks twice as far from the projected trajectory as normal to be associated with the track, again providing they had no nearby neighboring sparks. Such poorly fitting S-sparks were flagged after tracking was completed.

Another useful technique resulted from the observation that two or more digitizings frequently occurred in near proximity to one another and that these groupings were probably associated with the passage of a single particle. Such S-spark groupings were treated as a unit, although the individual digitizings retained their identity. On the average PITRACK associated 55% of the S-view digitizings into tracks and an additional 15% were indirectly associated by this method. Most of the remaining digitizings are observed in the forward direction and form a non-random background.

Figure 2 is an S-view of an atypical event containing several difficult situations with which this code must contend: a 2-spark track, two side-going tracks with multiple sparks on several chambers, a portion of a chamber where there are no digitizings, tracks which cross one another near the vertex, a possibly ambiguous or spurious track in the forward direction, and several sparks around the dead spaces.

Initial Searches

Four separate searches constituted the Smooth Track Search, each of which began by selecting from two adjacent chambers one S-spark each whose line of connection roughly pointed toward the vertex. A circle was thus defined and used to predict the location of sparks on adjacent chambers. A spark near the predicted location was accepted only if it lay within some angular window. If accepted, the spark was used to calculate a new curvature

for the track which then predicted the location of a spark on the next chamber. Each spark of a track was required to have a minimum separation from neighboring sparks on its chamber. If the spark did not meet this separation criterion or if there was no spark in the angular acceptance window, a miss was recorded for that chamber. The search would stop if two consecutive chambers did not provide spark candidates. This procedure often resulted in partial and incomplete tracks which required routines to extend and complete them before they were evaluated.

During the Smooth Track Search the firmness in location of the vertex was continually evaluated. Occasionally for HMS triggers the vertex location had to be stepped through the length of the target until a track with a sufficiently large angle could be found to localize it. If the vertex was found to lie more than one inch outside the physical limits of the target, processing of the event was halted after the Smooth Track Search and the event was flagged.

The Forward Search operated by choosing two previously unassociated S-sparks, one from the first two chambers and the other from the last two. From these sparks a straight line was constructed which was intersected with the remaining chambers. If at least three additional sparks could be found within a window around this line, the track was sent on for development and evaluation.

With the vertex well determined, the Brute Force Search began by taking all the remaining unassociated S-sparks, connecting them two-at-a-time with the vertex to form a circle. If enough sparks were found within a window around the circle, the track was sent on for further work.

The 2-spark Track Search examined pairs of unassociated S-sparks on the first two chambers. Here the only constraint occurred in the Y-view, where

it was required that two Y-sparks be found which defined a straight line passing near the vertex. This search increased the effective solid angle of the VS by 20% to 2π sr. Care was exercised to insure that spurious tracks were not introduced by this search.

Evaluation and Performance

PITRACK was written in FORTRAN IV and required 72k octal words of CDC 6600 of core storage for execution. Packing and unpacking routines required by the format of our data tapes accounted for ~ 20% of the core used. Reconstruction time depended on many factors but we found empirically its average approximately depended upon the charge multiplicity of the event, N as

$$t \sim 0.4 + 0.04\ N^2 \text{ (seconds)}$$

(e.g. for a 6-prong 1.8s and for a 10-prong 4.4s). The time to reconstruct individual events, however, varied by a factor of three or more from this average.

The reliability of PITRACK was evaluated by scanning a sample of several hundred HMS trigger events utilizing the interactive graphics program VUE. Initially an event's digitizings were displayed on a video screen with the PITRACK solution superimposed as in Figure 2. The scanner by examining the event in various perspectives, such as the one shown in Figure 5, either verified or improved upon the solution. Interactive operations, using a track ball and teletype, allowed the association of any sparks to make new tracks, and the deletion of associated sparks on all or part of any track.

When VUE was used on the Brookhaven Sigma 7, where it occupied 18k decimal words of core, data to be inspected was stored on a disc file of the CDC 6600. A high speed data link, BROOKNET[3], permitted events to be transferred to the Sigma 7 core at a rate of ~ 200 ms/event when requested by the scanner. After scanning, the event was returned to another CDC 6600 file where it could be retrieved at a later time.

About 94% of the tracks were identified by PITRACK, of which 3% required some improvement, while another 2.5% needed an alteration of flags. Less than 1% of the identified tracks were found to be spurious in the scan. Of the unidentified tracks, a quarter were not acceptable on the basis of the code's predetermined criteria, yet the scanner accepted them on an individual basis. We found that the performance of PITRACK was independent of charge multiplicity up to 8-prongs. For example, in a sample of 6-prong events, 76% had every track identified. Thus, we concluded that there was little correlation among the unidentified tracks.

The performance and yield of the PITRACK search procedures are given in Table I. The Smooth Track Search which was meant to find the easy tracks in fact did identify 70% of all tracks found. In addition its yield was high - more than half the trial tracks it identified were finally accepted. The Brute Force Search, which was left to sort out the more difficult tracks, examined more than twice as many trial tracks as the Smooth Track Search, yet only 9% of those were eventually accepted.

We have thus far processed with PITRACK ~ 600K events recorded by MASS. For these events the VS and PITRACK successfully identified 83% of all charged particles originating from the primary vertex, the remaining 17% were accounted for by particles escaping detection by being produced outside the VS solid angle (~ 9%), in the dead spots of the chambers (~ 1%), by particles of too low a momentum (~ 1%), and by software inefficiencies (~ 6%). Most of these losses can be recovered by requiring charge balance.

Our corrected multiplicity distributions are compared in Table II with bubble chamber data[4] at the same energy, four-momentum transfer and missing mass. Our average charge multiplicity, $\bar{n}_{CH}$, is ~ 5% higher. This systematic deviation is not unexpected since we have not corrected our data for undetected

interactions of secondaries, gamma conversions and decays of neutral particles near the primary vertex, charge misidentification of fast tracks, etc. If a 5% excess of tracks is uniformly added to the bubble chamber multiplicity distributions to simulate these effects, the MASS and BC distributions are seen to be consistent.

Conclusions

We have written a pattern recognition program for a digital computer which associates digitizings from a set of nine cylindrical wire spark chambers into helical tracks in 3-dimensional space. The code successfully identifies ~ 94% of all tracks found by a human scanner while ~ 1% of the tracks it finds are spurious. Data taken with MASS and reconstructed by PITRACK are compatible within errors with processed bubble chamber results. We believe that our pioneering efforts in automatic track recognition demonstrates that the large amounts of data from a magnetic multiparticle spectrometer can be correctly and efficiently processed and analyzed.

REFERENCES

1. "Multiparticle Spectrometer System for the 10-30 GeV/c Region," J.R. Ficenec, T.S. Clifford, W.N. Schreiner, B.C. Stringfellow, W.P. Trower, E.W. Anderson, G.B. Collins, N.C. Hien, K.M. Moy, A. Ramanauskas, P. Schübelin, A.M. Thorndike, F. Turkot, and L. von Lindern, Experimental Meson Spectroscopy 1970. C. Baltay and A.H. Rosenfeld, (Columbia University Press, New York, 1970), 581, Eds. A more complete and updated description is in preparation.
2. J. R. Ficenec, B.C. Stringfellow, G.B. Collins, A. Ramanauskas, P. Schübelin and F. Turkot, Nucl. Inst. & Meth., 113 (1973) 535-540.
3. G. Campbell, K. Fuchel and L. Padwa, BNL 17054, July 1972 (unpublished).
4. Private Communication, J. Hanlon and R. Panvini.

Table I. Performance of PITRACK Search Procedure

Search	Identified	Yield
Smooth.Track	70.1%	57.2%
Forward	1.9%	16.5%
Brute Force	27.5%	8.7%
2-spark Track	0.5%	~25%

Table II

Charge Multiplicity Distribution in pp Collisions at 28.5 GeV/c: MASS vs BC

	2 prong	4 prong	6 prong	8 prong	10 prong	$\bar{n}_{CH}$
BC*	30.3% ± .9%	61.7% ± 1.2%	5.9% ± .4%	1.7% ± .2%	.4% ± .1%	3.65 ± .10
MASS†	29.6% ± 1.0%	58.4% ± 1.4%	9.8% ± .6%	1.7% ± .2%	.5% ± .1%	3.83 ± .04
BC + 5%††	27.6% ± .9%	58.9% ± 1.2%	10.8% ± .4%	2.2% ± .2%	.5% ± .1%	

*BC: Recoil proton identified and momentum $\leq$ 1.3 GeV/c, missing mass to the identified proton between 2.0 and 3.0 GeV.

†MASS: Missing mass to the fast forward proton between 2.0 and 3.0 GeV. The error on $\bar{n}_{CH}$ is statistical only.

††BC + 5%: BC Distribution with a 5% excess of tracks.

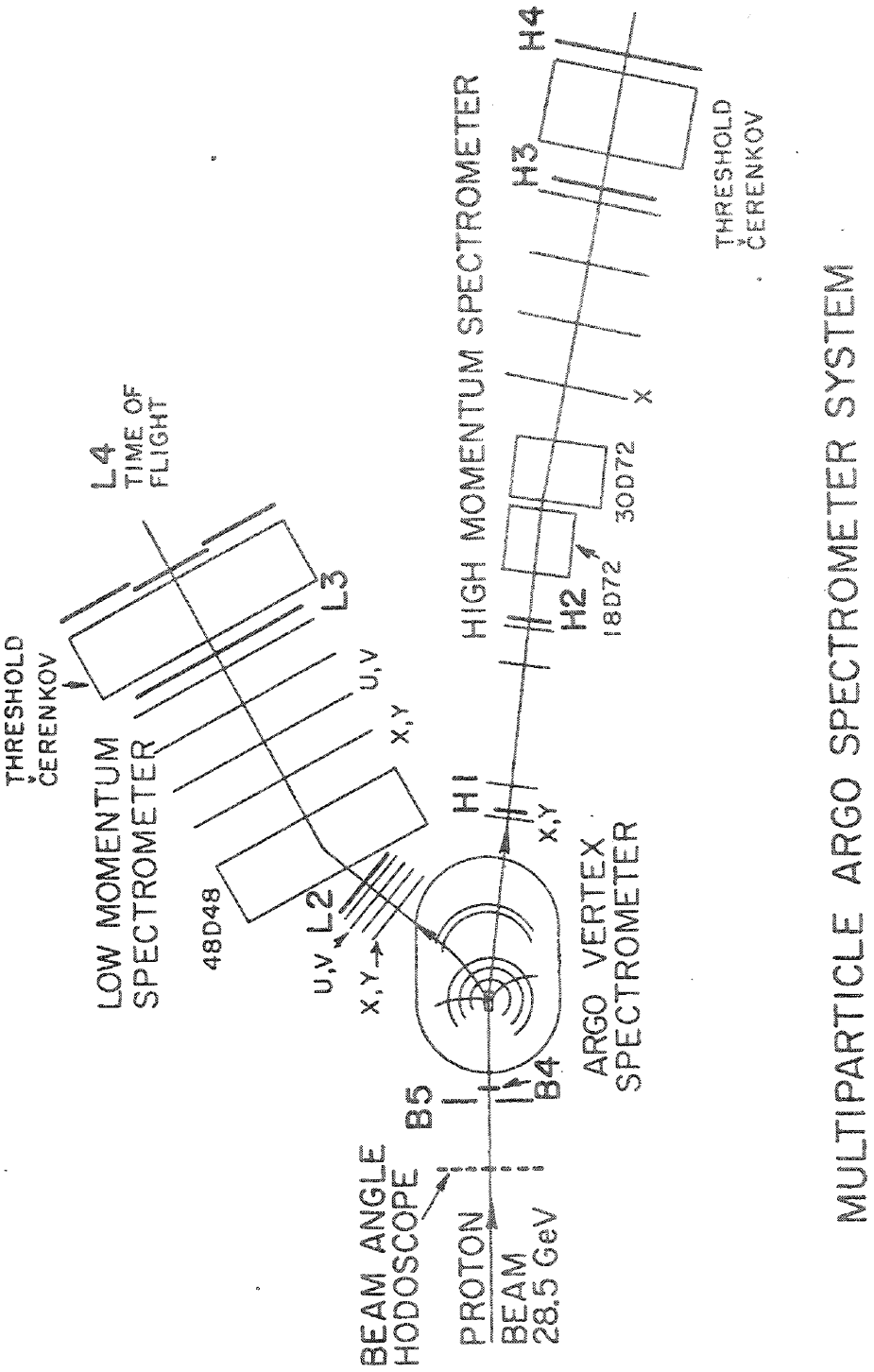


Fig. 1

Schematic floor layout of the Multiparticle Argo Spectrometer System

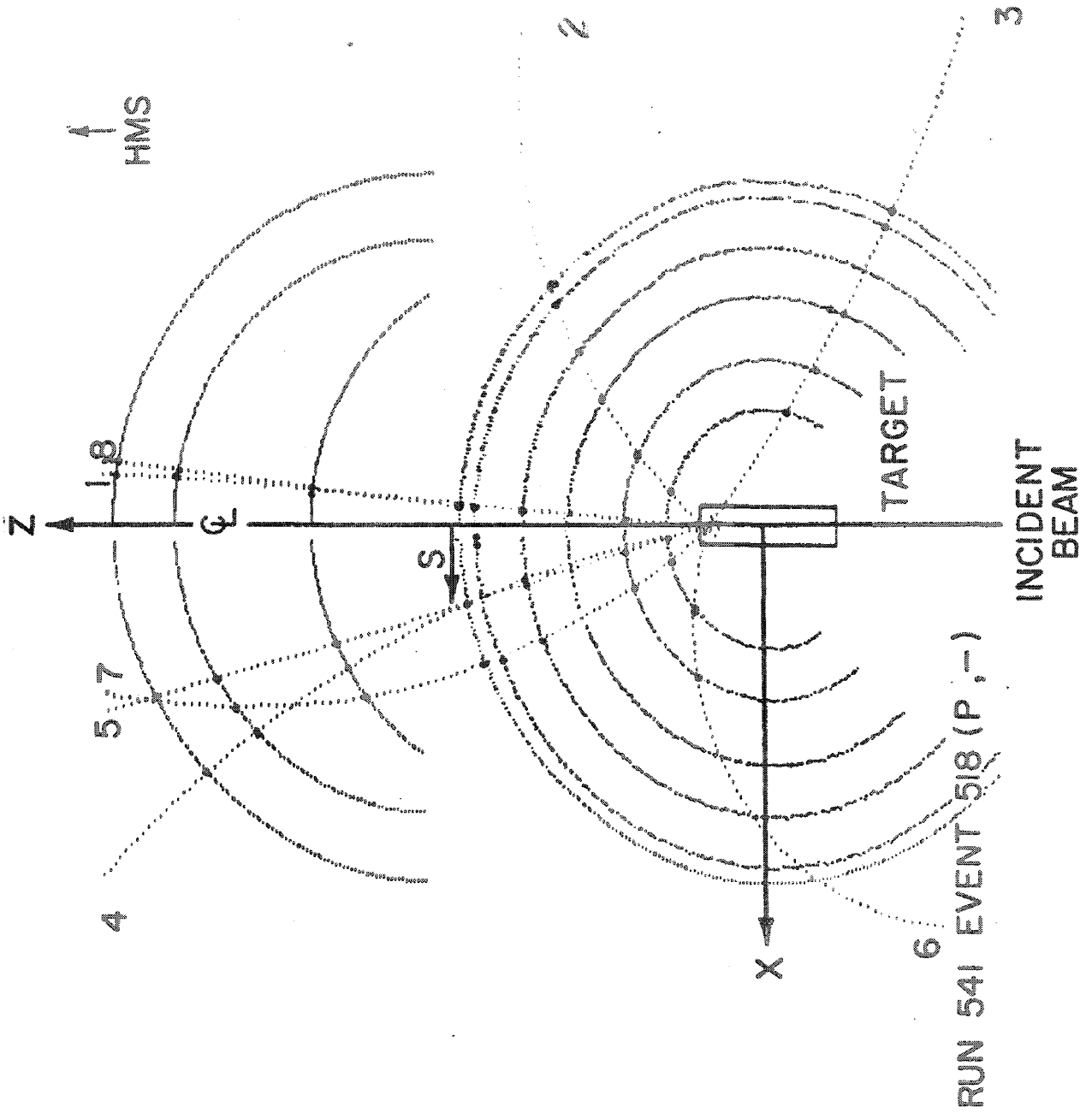


Fig. 2

An atypical event in the Vertex Spectrometer

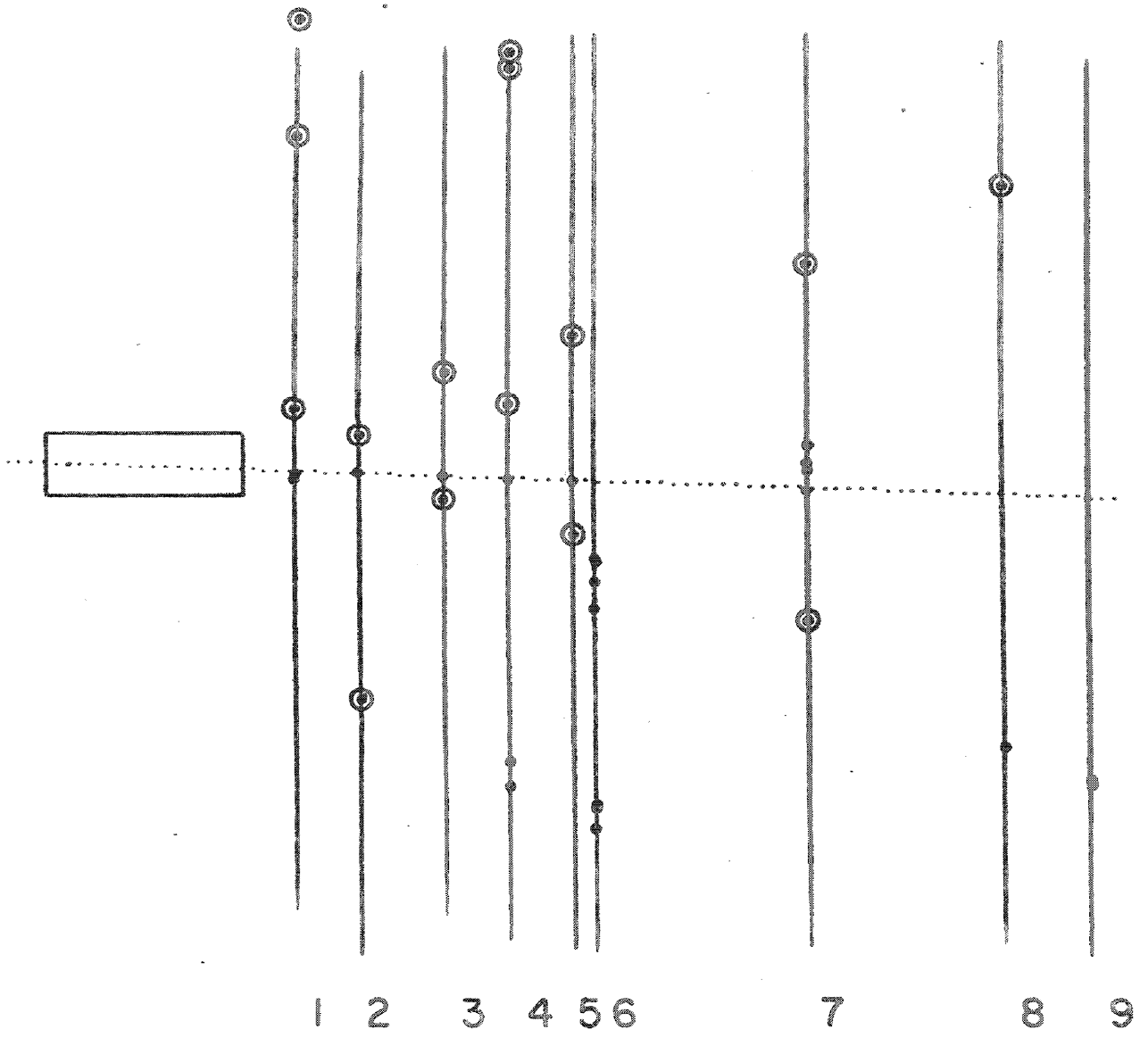


Fig. 3

Y-view of a typical track. The circled points are reflections of sparks which have been associated with other tracks.

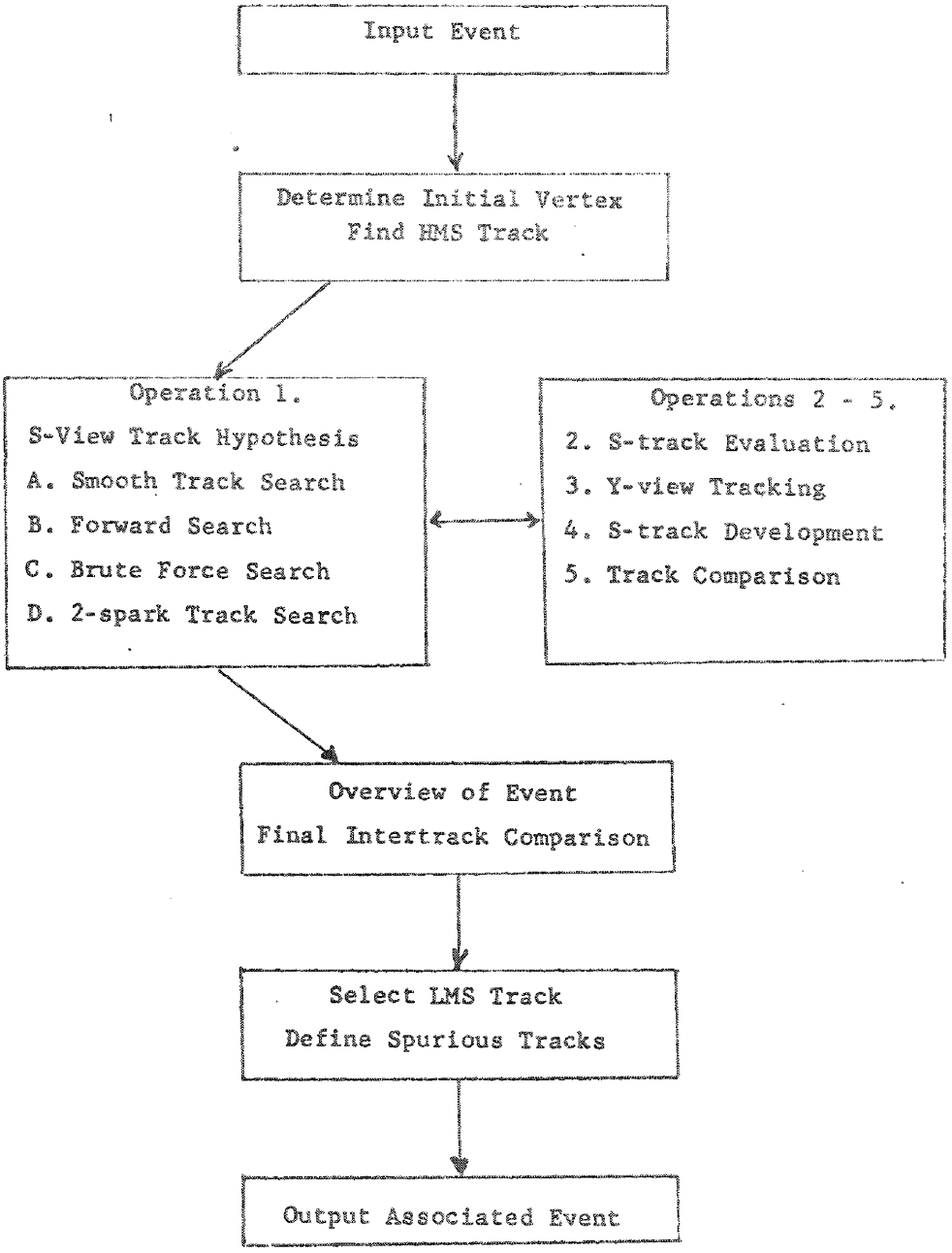


Fig. 4

Logic diagram of the Vertex Spectrometer track recognition code, PITRACK.

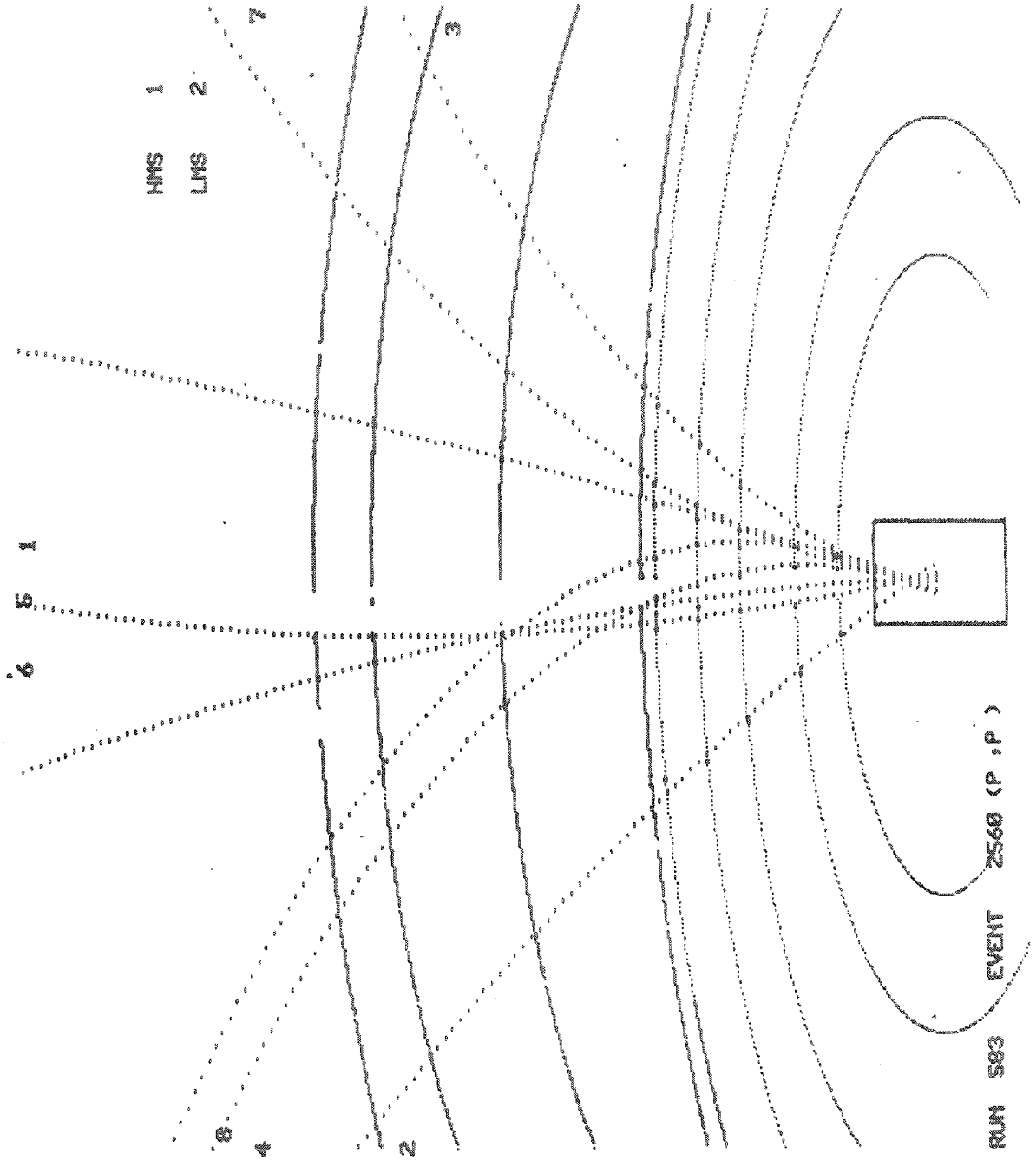


Fig. 5

S-view of an 8-prong event in another perspective as displayed by VUE.

COMPUTER GENERATED VISUAL DOCUMENTATION OF THEORETICAL STORE SEPARATION ANALYSES*

by

Harold R. Spahr
Sandia Laboratories
Albuquerque, New Mexico 87115

A paper presented at the Scientific Computer Information Exchange Meeting on topics in computer graphics (sponsored by AEC organizations), New York City, New York, May 2-3, 1974.

*This work was supported by the United States Atomic Energy Commission.

COMPUTER GENERATED VISUAL DOCUMENTATION OF THEORETICAL STORE SEPARATION ANALYSES*

by

Harold R. Spahr**
Sandia Laboratories
Albuquerque, New Mexico 87115

ABSTRACT

Recently, a computer code was developed which computes the theoretical trajectory of a store (i.e., bomb, fuel tank, etc.) in the complex aircraft flow field after it is released from an aircraft flying at subsonic speeds. However, the engineer was still faced with the problem of documenting the results of the store separation analysis in a concise, clear manner.

This paper describes a visual documentation system being used by Sandia Laboratories to document the results of theoretical store separation analyses. The documentation system uses a new Sandia Laboratories computer program, MOVIE1, with a CDC 6600 computer to generate a magnetic tape of plotting commands for a DatagraphiX 4020 plotter. Techniques are discussed which reduce the computer time required for one theoretical store separation to a few seconds to generate a magnetic tape for drawings and to a few minutes to generate the magnetic tape for movies.

---

*Work supported by U.S. Atomic Energy Commission.

**Member of Technical Staff, Aeroballistics Division, Aerodynamics Projects Department.

To illustrate the visual documentation provided, the paper contains computer generated black and white drawings of the side and bottom views of two theoretical store separation analyses. The paper presentation uses color slides and color movies (with real time and slow motion sequences) of the same two theoretical store separation analyses.

Possible future refinements to and future extensions of the MOVIE1 computer program are discussed.

COMPUTER GENERATED VISUAL DOCUMENTATION OF THEORETICAL STORE SEPARATION ANALYSES

INTRODUCTION

The word store, as used in this paper, is defined as any object (bomb, weapon, rocket, fuel tank, instrumentation pod, or container) which is carried on an aircraft. Thus, store separation analysis is defined as the determination of the position and attitude histories of a store after it is deliberately separated or ejected from the aircraft while the store is still in the complex nonuniform flow field near the aircraft.

Store separation problems and their analysis continue to be important in determining the effectiveness of any aircraft-delivered weapons system. Store separation problems can result in reductions in the allowable delivery speed of the weapon, increased dispersion of the impact point or target intercept point, and even, in rare cases, lead to loss of the aircraft (References 1 and 2).

Final store separation studies are usually based on extensive and expensive wind tunnel tests or full-scale flight drop tests. However, the need has been recognized recently for theoretical store separation analyses for use in preliminary design and to supplement and, hopefully, reduce the number and magnitude of wind tunnel and flight drop tests. To meet this need, a computer program (References 3, 4, and 5) was developed by Nielsen Engineering and Research, Inc., under contract from the Air Force Flight Dynamics Laboratory, to compute the theoretical separation trajectory of an external store released from an aircraft flying at subsonic speeds. This computer program currently is being used by several agencies.

However, the theoretical store separation problem does not end with the computation of the store separation trajectory. The engineer is still faced with the problem of presenting the results of the store separation analysis in a concise, clear manner. The engineer would like to replace the tabulated computer output of the relative location of the store and aircraft with a graphic presentation. What is needed is either a "theoretical chase aircraft" or a "theoretical camera pod" which would provide pictures or movies of the theoretical store separation process similar to those taken during experimental wind tunnel or full-scale drop test programs.

The first need for a visual presentation of theoretical store separation analysis results is while the engineer is examining the effects of different ejection conditions or different flight conditions on the store separation trajectory. The engineer would like to "instantly" see the results of one theoretical store separation trajectory to select the input conditions for the next analysis. Interactive computer graphics store separation computer codes, such as the one described in References 6 and 7, provide the best means of meeting this need.

The second need for a visual presentation of theoretical store separation analysis results arises when the engineer must present the results of the analysis in a briefing, letter, or report. This requires generating permanent visual documentation of the store separation process. To provide permanent visual documentation on short time scales and at low cost, the visual documentation must be computer generated using off-line plotters. A visual documentation system which generates permanent visual documentation of the theoretical store separation process can also be used to meet the first need, described in the preceding paragraph, with longer time scales. Thus, for those organizations which do not have access to an interactive graphics terminal on a large computer, a system which generates permanent documentation of the store separation process can be used to meet both needs.

This paper describes a permanent visual documentation system being used by Sandia Laboratories to provide visual documentation of theoretical store separation analyses. The documentation system uses a new Sandia Laboratories computer program, MOVIE1, with a CDC 6600 computer to generate a magnetic tape of plotting commands for an off-line DatagraphiX 4020 plotter.

This paper first defines the desirable characteristics of a permanent visual documentation system for the theoretical store separation process. Then, the computer program MOVIE1 is described. Techniques are discussed which reduce the computer time required for one theoretical store separation to a few seconds to generate the magnetic tape for drawings and to a few minutes to generate a magentic tape for movies.

Next, the Sandia Laboratories modified DatagraphiX 4020 plotter and the output media provided from it are described. To illustrate the visual documentation provided, the paper contains black and white drawings of the results of two theoretical store separation analyses. Color and black and white 35mm slides and color and black and white movies (with real time and slow motion sequences) of theoretical store separation analyses results also can be generated.

The paper also describes possible future refinements to and future extensions of the MOVIE1 computer program. The final section of the paper defines how a copy of the source deck of computer program MOVIE1 can be requested.

DESIRABLE CHARACTERISTICS

Before one examines the visual documentation system for theoretical store separation results currently being used at Sandia Laboratories, the major desirable characteristics of such a system should be identified. These characteristics are:

1. A wide selection of output media;
2. Minimum use of computer time;
3. Provide at least two orthogonal views of the store separation process;
4. Show the store shape, including fins, in both orthogonal views;
5. Require a minimum of input data;
6. Rapid availability of visual documentation;
7. Easy conversion from computer to computer and plotter to plotter; and
8. Available to government agencies and their contractors.

Each of these major desirable characteristics is discussed in detail in subsequent paragraphs.

Different output media are required for different purposes. Black and white drawings of the theoretical store separation results are needed for informal briefings where slide or movie projection equipment is not available, and for documentation in reports or letters. Color slides and color movies are desirable for more formal briefings and presentations where the longer lead time required for color film processing is available.

The computer code used to generate the plotter commands should use a minimum of computer time. The computer time required per frame of output documentation should be very small to permit generating theoretical store separation movies corresponding to the high frame rates of several hundred frames per second used for experimental store separation movies. This permits direct side-by-side projection and comparison of the theoretical and experimental store separation results.

Total computer time required for visual documentation should be kept to a relatively small percentage of the computer time required to compute the theoretical store separation trajectory. Otherwise, the cost of the complete store separation analysis will be significantly increased.

The visual documentation provided should contain at least two orthogonal views of the store separation results to show the relative position of the store and aircraft to determine whether store and aircraft contact occurs. One of these views should be a side view for comparison with pictures or movies obtained later from camera pods or a chase plane in full-scale flight drop tests or pictures or movies obtained later through the side walls of wind tunnels during drop tests.

The store shape, including fins, should be shown in both orthogonal views. By showing the store shape, including fins, store and aircraft contact can be determined which would not be apparent if only the store centerline were shown.

The computer program used to generate the plotter commands should require a minimum of input data. Input data defining the relative position of the store and aircraft should be automatically generated by the computer program which computes the theoretical store separation trajectory. A visual documentation system will be used the most if the time required to prepare the input data is a relatively small fraction of the time required to prepare the input data for the store separation trajectory computer program.

The visual documentation system should rapidly provide the desired output media. Frequently, the mass properties of a prototype store are measured a few days before the full-scale drop test. Thus, a desirable goal of a visual documentation system is to provide black and white output media prior to 8:00 a.m. from visual documentation computer program runs submitted prior to 5:00 p.m. the preceding day.

The visual documentation system should be easy to convert from computer to computer and plotter to plotter to increase its use by the store separation analysis community. Thus, the computer program used to generate the magnetic tape of plotter commands should be written in a widely used scientific programming language available on most computers.

The computer program used to generate the magnetic tape of plotter commands should be usable with the same plotter used at different computer facilities. Thus, a "standard," readily available plotter language should be used. Nonstandard plotting language subroutines should be used only when their use provides significant benefits.

Since different computer facilities will have different plotters, the computer program used to generate magnetic tapes of plotter commands should be easily converted from plotter to plotter. This conversion will be aided by using the smallest possible subset of plotting commands.

Obviously, a visual documentation system for theoretical store separation analyses is useful only if it is readily available. Thus, the computer program which generates the magnetic tape of plot commands should be available to government agencies and their contractors at no cost.

COMPUTER PROGRAM MOVIE1

Computer program MOVIE1 was written at Sandia Laboratories to prepare a magnetic tape of plot commands to generate visual documentation of the results of theoretical store separation analyses. To facilitate conversion of the computer program from computer to computer, the program was written in FORTRAN (Reference 8), the most widely used engineering and scientific programming language.

The first step in the development of the program was to select the plotter to be used. The modified DatagraphiX 4020 plotter, described in the next section of this paper, was selected because it is the high-speed plotter with the widest selection of output media available at the Sandia Laboratories computer facility.

The next step in the development of the computer code was to select a plotter programming language for the DatagraphiX 4020 plotter. The SCORS plotting language (Reference 9) was selected because it is currently in use by Sandia Laboratories for the DatagraphiX 4020 and also is used by a number of other agencies.

Next, the visual documentation to be provided was defined. The theoretical store separation results were to be documented by both side and bottom views. These views were to be generated using any of the output media available for the DatagraphiX 4020.

Most aircraft-delivered stores have small low aspect ratio fins which do not impart significant roll rates or roll angles to the store during the store separation process. Thus, to simplify the input data required and to minimize the computer time required, it was decided to not show any change in roll angle of the separated store. The side and bottom views of the separated store show an apparent change in length as the store oscillates in pitch and yaw, but do not show any change in roll orientation.

Slides or black and white drawings were to be generated for each 0.1 second during the store separation process. Movies (to be projected at 16 frames per second) were to present both real time and slow motion (one-twentieth as fast as real time) sequences for each of the two views.

Next, the input data required to generate the visual documentation of the theoretical store separation results were defined. The form of the input data used was selected to make the input data as flexible as possible but still simple to prepare.

The side view of the aircraft is defined by up to 10 geometry files, with up to 100 X and Y coordinate pairs in each file. The points in a geometry file are sequentially connected by straight line segments to draw part of the aircraft side view.

Figure 1 shows the X and Y coordinate system used in defining the input data for the aircraft side view. The F-4D aircraft side view shown in Figure 1 is generated using 3 geometry files with a total of 100 points.

The side view of the separated store is defined by up to 10 geometry files with up to 100 X and Y coordinate pairs in each file. The points in a geometry file are sequentially connected by straight line segments to draw part of the store side view.

Figure 2 shows the X and Y coordinate system used in defining the input data for the separated store side view. The B57 nuclear weapon side view shown in Figure 2 is generated using 6 geometry files with a total of 84 points.

The bottom view of the aircraft is defined by up to 10 geometry files with up to 100 X and Y coordinate pairs in each file. The points in a geometry file are sequentially connected by straight line segments to draw part of the aircraft bottom view.

Figure 3 shows the X and Y coordinate system used in defining the input data for the aircraft bottom view. The F-4D aircraft bottom view shown in Figure 3 is generated using 6 geometry files with a total of 122 points.

The bottom view of the separated store is defined by up to 10 geometry files with up to 100 X and Y coordinate pairs in each file. The points in a geometry file are sequentially connected by straight line segments to draw part of the store bottom view.

Figure 4 shows the X and Y coordinate system used in defining the input data for the separated store bottom view. The B57 nuclear weapon bottom view shown in Figure 4 is generated using 6 geometry files with a total of 74 points.

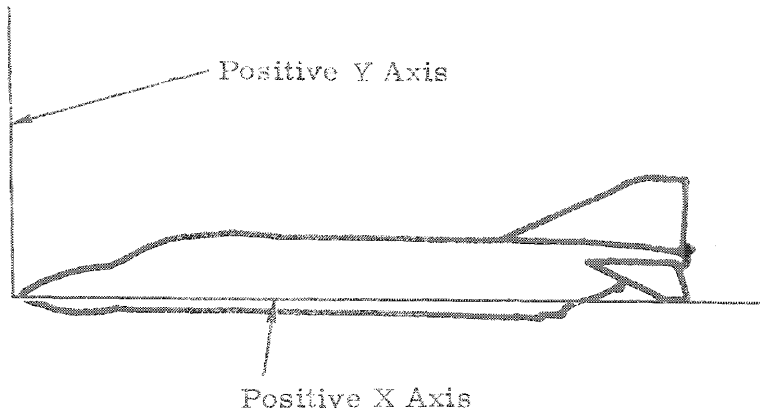


Figure 1. Coordinate System Used For Input Data For Aircraft Side View (F-4D Aircraft Shown)

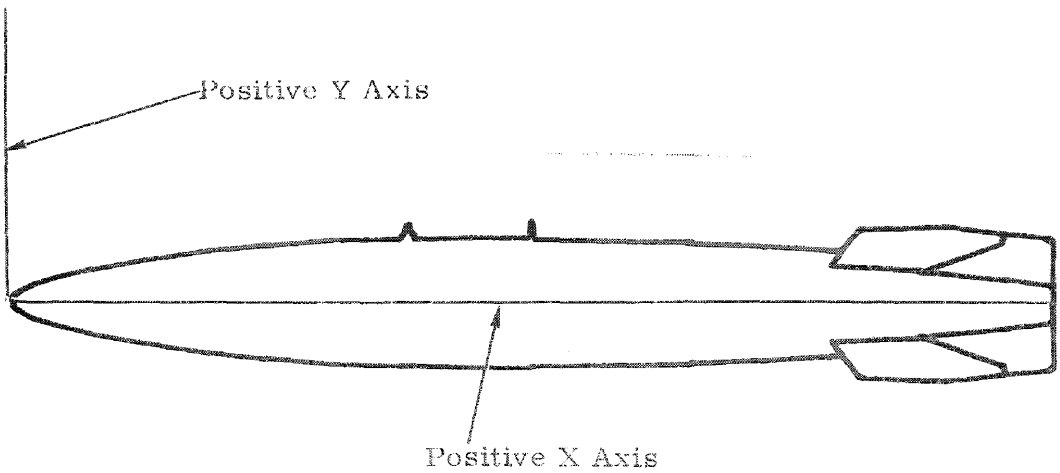


Figure 2. Coordinate System Used For Input Data For Separated Store Side View (B57 Nuclear Weapon Shown)

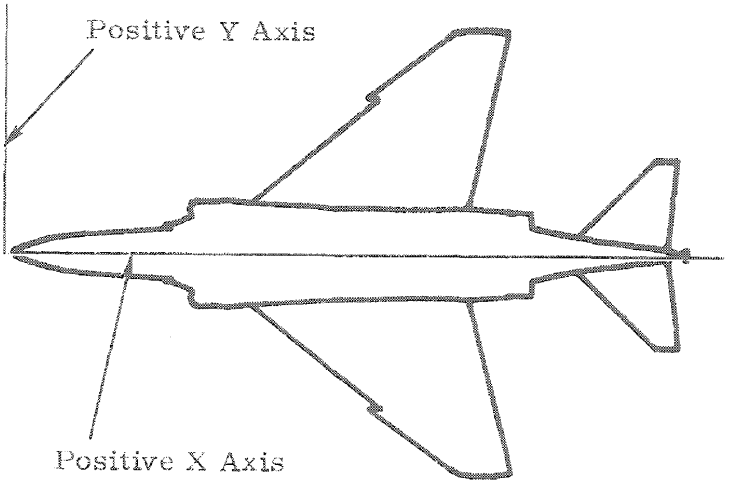


Figure 3. Coordinate System Used For Input Data For Aircraft Bottom View (F-4D Aircraft Shown)

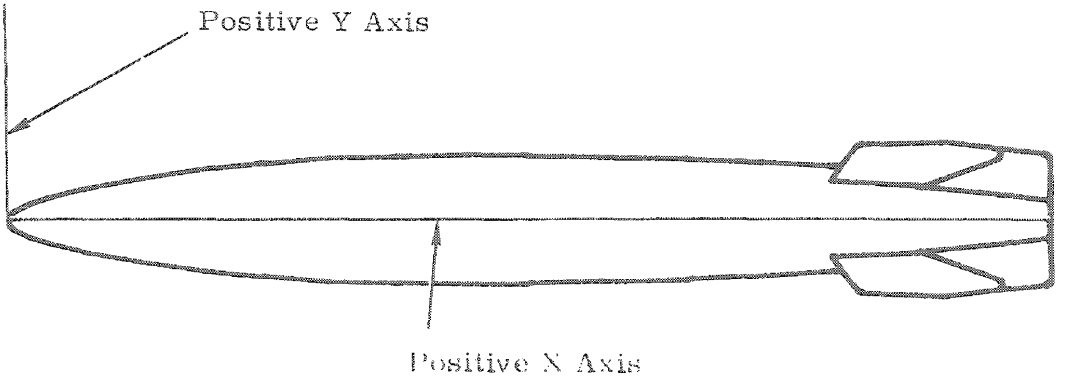


Figure 4. Coordinate System Used For Input Data For Separated Store Bottom View (B57 Nuclear Weapon Shown)

The input data in the four sets of geometry files described previously can be in any units. Scale factors are provided for each set of geometry files to scale the input dimensions to full-scale dimensions in feet.

The four sets of geometry files for the aircraft and separated store side and bottom views would be time-consuming to prepare using manual means. At Sandia Laboratories, a Bendix digitizer is used to digitize the points manually marked on the side and bottom views of a drawing of the aircraft or the separated store. When the button on the digitizer cursor is pushed, the X and Y coordinates of the point under the cursor cross hairs are automatically written into a disc file on a PDP-10 computer. After all the points are entered, the points on the disc file are automatically punched into cards. Less than one hour is required to digitize both the side and bottom views of an aircraft or a separated store.

The next set of data required is the data defining the position of the store relative to the aircraft in the side and bottom views for each time step in the store separation trajectory calculation. This is defined by giving the X, Y, and Z coordinates of the nose and tail of the store in an aircraft fuselage coordinate system. Figure 5 shows a side view of this aircraft fuselage coordinate system, while Figure 6 shows a bottom view of the same coordinate system.

The X, Y, and Z coordinates of the nose and tail of the separated store are in units of feet. The time is the time from the start of the store separation trajectory in seconds.

The punched card input data deck defining the location of the nose and tail of the separated store and the time is generated automatically when the theoretical store separation trajectory program (References 3, 4, and 5) is used. Four cards were added to the OUTPUT subroutine in the theoretical store separation trajectory program to generate this input data deck.

The aircraft flight path angle and the fuselage angle of attack must be supplied in degrees. These angles are used to rotate the aircraft and the store in the side and bottom views to provide the correct perspective.

Control parameters must be entered to determine the size of the aircraft and separated store, and position them in the side and bottom view. The time in seconds in the theoretical store separation process when the visual documentation is to be stopped must also be supplied as input data.

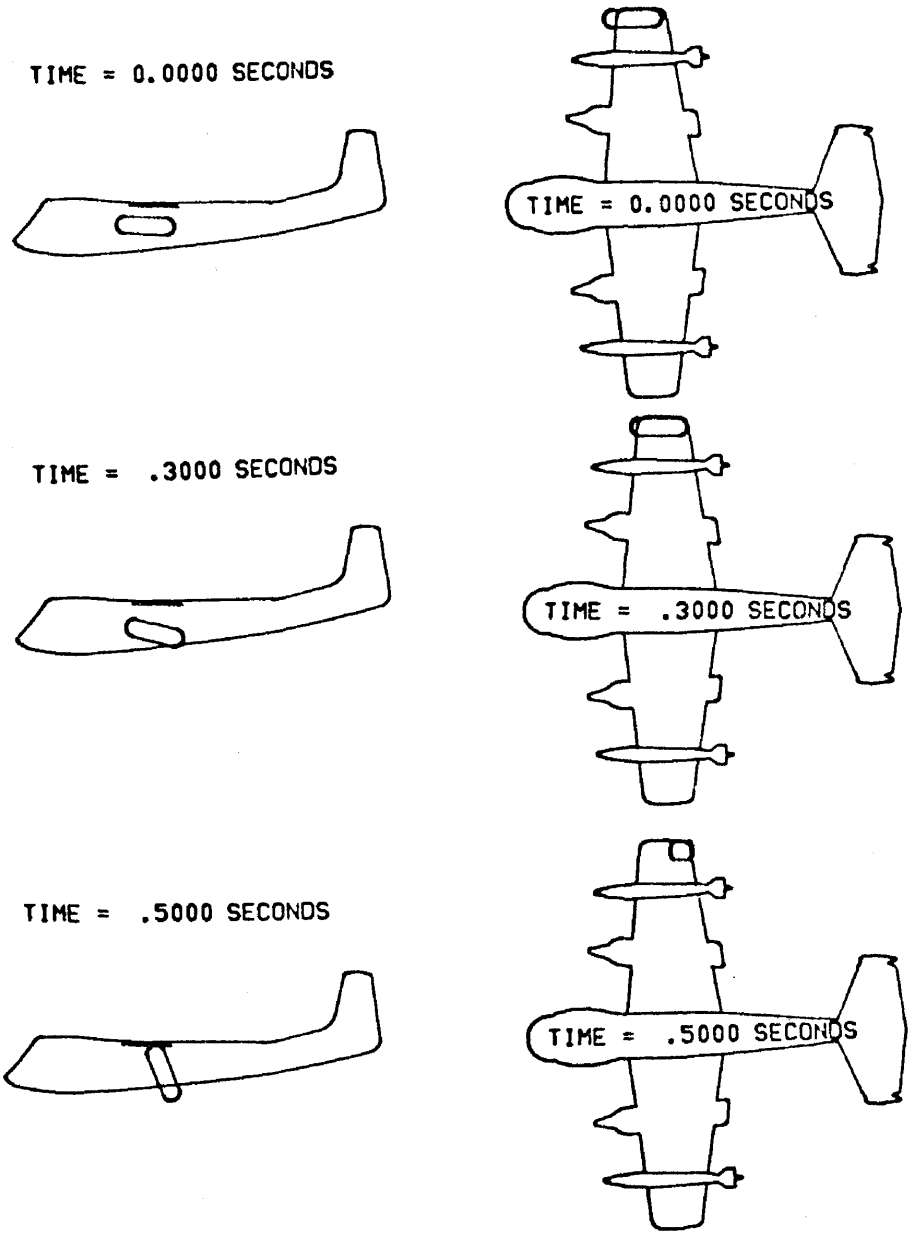


Figure 7. Selected Frames of Visual Documentation For An EG&G Pod Released From An OV-1C Aircraft At 250 Knots True Airspeed At Sea Level

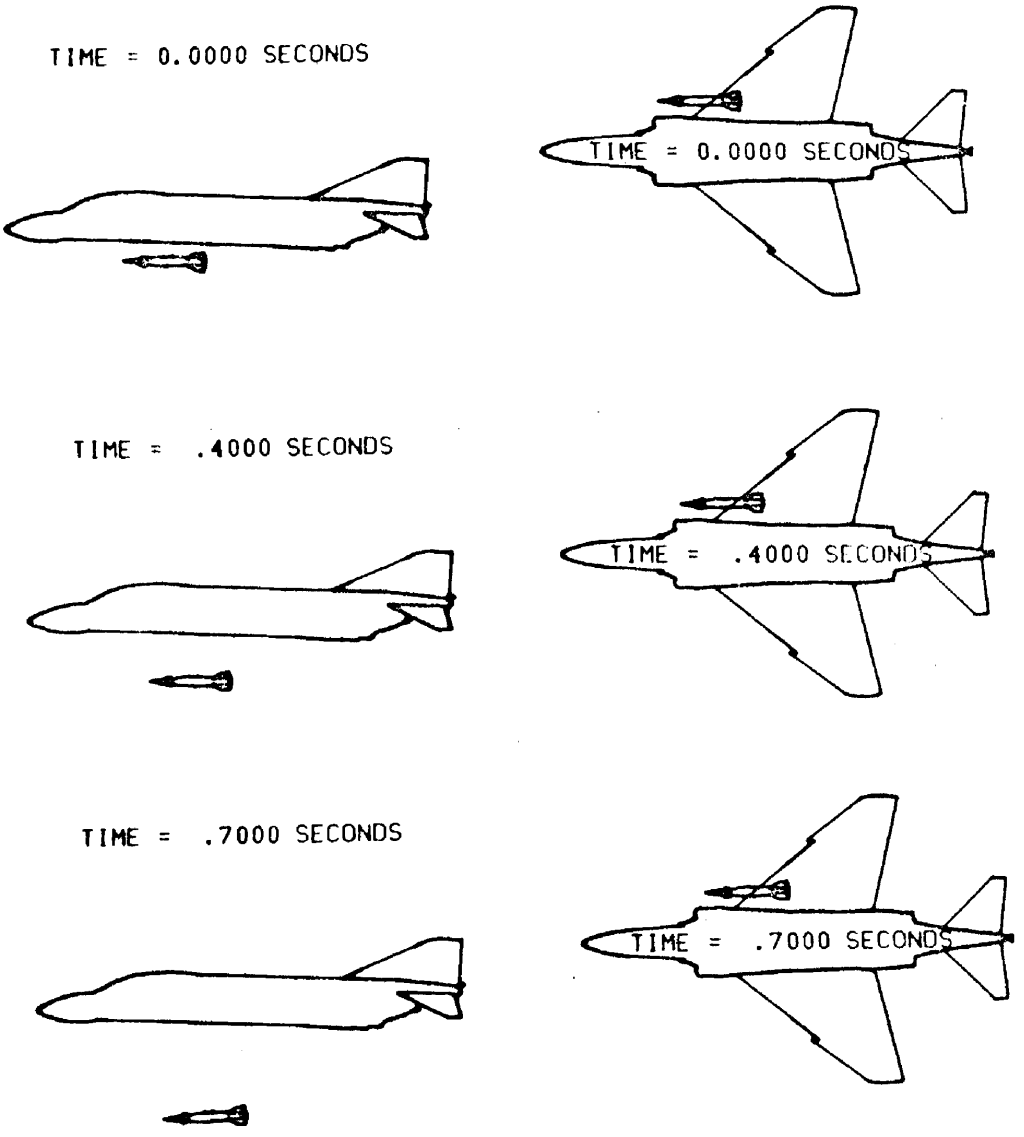


Figure 8. Selected Frames Of Visual Documentation For A Sandia Laboratories Prototype Store Ejected From A F-4D Aircraft At A Mach Number Of 0.7 At 15,000 Feet Altitude Above Mean Sea Level

2. Define the speed, altitude, and flight path angle of the drop aircraft at the time of store separation;

3. Define the separated store; and

4. Define the ejection velocity and ejection angular rates of the separated store at separation.

This refinement can be done easily, but would require numerous time-consuming changes to computer program MOVIE1.

An additional desirable refinement would be to add the capability to generate close-up views of the separated store where some parts of the aircraft and perhaps store are outside the field of view. This can be easily done, but would require adding a "scissoring" or "clipping" subroutine with logic to eliminate the points outside the field of view.

A highly desirable refinement, which will be completed in the near future, is to add the capability to present experimental data from wind tunnel tests or full-scale drop tests directly on the output media for comparison with the theoretical calculations. This will require reading in a second set of store position data from the experimental test. Then, the store will be drawn twice per frame of output media.

One store drawing, in one color, will show the position of the store from the theoretical calculations. The second store drawing, in a second color, will show the position of the store from the experimental test at the same time. This capability is being developed to compare the experimental results from Reference 16 with the results of planned theoretical store separation analysis for the same store and aircraft at the same flight conditions.

An additional highly desirable refinement, which will be completed in the near future, is to add the capability to present "strobe" pictures of the theoretical store separation. Pictures made of experimental store separation tests in wind tunnels sometimes use a "strobe" light which is used to repetatively illuminate the aircraft and store several times during the store separation process. This results in several pictures of the separated store, all on the same frame of film, which correspond to its location at the times the "strobe" was pulsed.

A desirable long-term refinement of the visual documentation system would be to add three-dimensional representations of the separated store and aircraft to permit generating visual documentation as seen from any arbitrary angle. The three-dimensional shapes could be represented by quadrilateral surface elements (figure on Page 49, Reference 17), half-tone shading techniques (Figure I-6, Page xxii,

Reference 18), station lines (Figure 3.9, Page 34, Reference 19), or detailed station lines and selected longitudinal lines (Figure 4.0, Page 34, Reference 19).

While the three-dimensional representations would add additional realism, several problems arise. The input data needed to define the separated store and, especially, the aircraft would increase significantly in magnitude. Also, the computer time required to compute the location of all the points in the three-dimensional representation of the separated store would increase very significantly over the current two-dimensional visual documentation. The final problem is that "hidden line" subroutines would be required to most effectively use three-dimensional representations of the separated store and aircraft. The computer time required to do the "hidden line" computations for each frame of a movie might be prohibitive with current subroutines and computers.

The visual documentation system defined in this paper could be extended in at least three areas. The first area would be to extend the MOVIE1 computer program to prepare slides and movies showing the separated store, its orientation, and the corresponding body normal force and side force distribution along the store body during the store separation process. The theoretical store separation trajectory computer program (References 3, 4, and 5) currently computes the necessary store body loading data as part of the calculation of the theoretical store separation trajectory.

A second area where the visual documentation system could be extended would be to extend the MOVIE1 computer program to draw flow streamlines around the store to show the flow angularities in the flow field. This would require a significant modification to the theoretical store separation trajectory computer program to compute the required flow streamline data. The additional computer time required would be significant. Thus, the flow streamlines might be shown only for the store in the carriage position rather than being recomputed and redrawn for each frame of a theoretical store separation movie.

The final area where the visual documentation system could be extended would be to add the capability to draw shock waves. This would prepare the MOVIE1 computer program for use with any supersonic theoretical store separation trajectory computer program which might be developed in the future (perhaps based on References 20 through 22). The shock wave presentation would probably be limited to defining the shock waves from the aircraft and the separated store in a plane containing the nose of the separated store in both the side and bottom views.

CONCLUDING REMARKS

This paper has described a system used by Sandia Laboratories to provide permanent visual documentation of the results of theoretical store separation analyses. This visual documentation system is routinely used to rapidly and economically generate visual documentation of theoretical store separation analyses using a variety of output media.

To date, the only application of computer program MOVIE1 has been to generate visual documentation of the theoretical store separation trajectory analyses, as documented in this paper. However, since computer program MOVIE1 documents the motion of one object relative to another object, this program should be useful for other applications involving relative motion. Since this paper is being presented to the AEC computer community, computer groups in other agencies may want a copy of computer program MOVIE1.

A preliminary version of the FORTRAN computer program MOVIE1, which generates DatagraphiX 4020 plot commands in the SCORS plotting language, can be made available to requestors with a need for the program. Atomic Energy Commission computer program dissemination policy requires that each request be treated as a separate case, and that signed authorization be obtained from several levels of management at Sandia Laboratories. While this policy prevents an exact definition of the availability of the computer program, it should be available to almost all government agencies and most of their contractors.

Requests for the FORTRAN source card deck, sample input data, and sample output visual documentation should be made by a letter to:

H. R. Spahr
Division 5625
Sandia Laboratories
P. O. Box 5800
Albuquerque, New Mexico 87115

The letter should briefly define the need for the computer program, describe projects it would be used on, and describe briefly any planned use of the computer program to support contracts from government agencies. The letter should also briefly describe the computer and plotter that the computer code will be used with.

REFERENCES

1. "F-14A Crash," Aviation Week and Space Technology, June 25, 1973, Page 25.

2. "Production AIM-7F Enters Test," Aviation Week and Space Technology, August 27, 1973, Page 38.

3. Frederick K. Goodwin, Marnix F. E. Dillenius, and Jack N. Nielsen, "Method of Predicting Loading and Trajectories of Single or TER or MER Mounted Stores on Swept-Wing Aircraft," Volume 2, Aircraft/Stores Compatibility Symposium Proceedings, August 1972, sponsored by JTCG/ALNNO, held at Dayton, Ohio on December 7-9, 1971

4. Frederick K. Goodwin, Marnix F. E. Dillenius, and Jack N. Nielsen, "Prediction of Six-Degree-of-Freedom Store Separation Trajectories at Speeds Up to the Critical Speed - Volume I - Theoretical Methods and Comparisons with Experiment," Air Force Flight Dynamics Laboratory Technical Report AFFDL-TR-72-83, Volume I, October 1972.

5. Frederick K. Goodwin, Marnix F. E. Dillenius, and Jack N. Nielsen, "Prediction of Six-Degree-of-Freedom Store Separation Trajectories at Speeds Up to the Critical Speed - Volume II - User's Manual for the Computer Programs," Air Force Flight Dynamics Laboratory Technical Report AFFDL-TR-72-83, Volume II, October 1972.

6. Calvin L. Dyer, "An Interactive Graphics Program for Predicting Six-Degree-of-Freedom Store Separation at Speeds Up to the Critical Speed," Air Force Flight Dynamics Laboratory Report AFFDL/FGC-TM-73-58, July 1973.

7. Marnix F. E. Dillenius, Frederick K. Goodwin, Jack N. Nielsen, and Calvin L. Dyer, "Extensions to the Method for Prediction of Six-Degree-of-Freedom Store Separation Trajectories at Speeds Up to the Critical Speed, Including Interactive Graphics Applications and Bodies of Arbitrary Cross Section," Aircraft/Stores Compatibility Symposium Proceedings, Volume 2, sponsored by JTCG/ALNNO, held at Sacramento, California on September 18-20, 1973.

8. "Control Data 6600 Computer Systems, FORTRAN Extended Reference Manual, 6600 Version 3," Publication No. 601 76600, Revision K, Control Data Corporation, February 22, 1973.

9. "Section III - Programmer's Reference Manual," SC-4020 Usage With IBM-7090/7094, CDC-3600, Univac-1107/1108, CDC-6600, SC-M-70-68, Sandia Laboratories, Albuquerque, New Mexico, March 1970.

10. "Routines to Generate and Store SD-4020 Commands," Sandia Computing Newsletter SN 0012/1971, Sandia Laboratories, Albuquerque, New Mexico, August 9, 1971.

11. "Section II - Introduction and Basic SC-4020 Description," SC-4020 Usage with IBM-7090/7094, CDC-3600, Univac-1107/1108, CDC-6600, SC-M-70-68, Sandia Laboratories, Albuquerque, New Mexico, March 1970.

12. C. J. Fisk, "Cathode Ray Tube Color Plotting," SC-RR-68-546, Sandia Laboratories, Albuquerque, New Mexico, January 1969.

13. "35mm Computer Generated Color Slides of an EG&G Pod Released From an OV-1C Aircraft," available on loan from H. R. Spahr, Sandia Laboratories, Albuquerque, New Mexico.

14. "35mm Computer Generated Color Slides of a Sandia Laboratories Prototype Store Ejected From a F-4D Aircraft," available on loan from H. R. Spahr, Sandia Laboratories, Albuquerque, New Mexico.

15. "16mm Computer Generated Color Movie of an EG&G Pod Released From an OV-1C Aircraft and a Sandia Laboratories Prototype Store Ejected From a F-4D Aircraft," available on loan from H. R. Spahr, Sandia Laboratories, Albuquerque, New Mexico.

16. James R. Myers, "Separation Characteristics of the B-57 Bomb From the F-4C Aircraft Equipped with ECM Pods at Mach Numbers from 0.605 to 1.30," AEDC-TR-72-93, June 1972, Arnold Engineering Development Center.

17. D. S. Warren, "Tomorrow's Structural Engineering," Astronautics and Aeronautics, July 1973.

18. William M. Newman and Robert F. Sproull, Principles of Interactive Computer Graphics, McGraw-Hill Book Company, New York, 1973.

19. William A. Fetter, Computer Graphics in Communication, McGraw-Hill Book Company, New York, 1973.

20. F. Dan Fernandes, "Theoretical Prediction of Interference Loading on Aircraft Stores - Part I - Subsonic Speeds," NASA CR-112065-1, June 1972, General Dynamics.

21. F. Dan Fernandes, "Theoretical Prediction of Interference Loading on Aircraft Stores - Part II - Supersonic Speeds," NASA CR-112065-2, June 1972, General Dynamics.

22. F. Dan Fernandes, "Theoretical Prediction of Interference Loading on Aircraft Stores - Part III - Programmer's Manual," NASA CR-112065-3, June 1972, General Dynamics.

ABSTRACT

C. H. Turnbull, 8442
Sandia Laboratories Livermore

TWO APPLICATIONS OF DATA ANALYSIS BY INTERACTIVE GRAPHICS

Analysis of Data from Lagrangian Codes

Contour plots have proven very useful in analyzing the data from two-dimensional hydro codes. However, it is difficult to know in advance how to specify the plot limits to show the most interesting data to best advantage. This problem has been met by giving the analyst an interactive code by which he can easily adjust limit data and get a look at the resulting plot.

Data from the hydro code is organized first by time step. The interactive code gives the analyst the ability to select the set of data he wishes to study, to "zoom" to the area of interest, to select which parameter he wishes to observe, and to select the levels at which the contours are to be drawn to provide him with the most meaningful plots.

Since the CDC 250 is used as the interactive graphics device, we try to keep central memory requirements as low as possible. By using ECS (Extended Core Storage) to store the data for any particular time step, the contours can be plotted by having the data for only nine zones in central memory at any instant. The use of overlays aids in reducing central memory requirements. This code has been in "production" status for approximately two years.

Analyzing Spectral Data

A recurring problem is the analysis of data in which the data curves are in fact the sum of a family of similar curves. In one application, the family of curves is Gaussian. In another application, each curve in the family involves a double exponential function.

Since there is no one answer to this type of problem, the analyst must apply some knowledge to the selection of parameters to approximate the fit. When the analyst has arrived at a close approximation, a nonlinear least-squares fit algorithm will make final adjustment of the parameters. The operator may apply constraints on the least squares algorithm by allowing only selected parameters to change.

This code has been in production for approximately one year and takes advantage of ECS and overlay procedures to reduce the amount of central memory required.

C. H. Turnbull, 8442
Sandia Laboratories Livermore

Analysis of Data from Lagrangian Codes

Large two-dimension hydro codes are used to study the effects of external forces or internal energy on materials. The high volume of data associated with these codes makes graphic representation an expedient way of analyzing the data. For example, Figure 1 shows the initial condition of a small problem run on the TOODY code. This example has 895 zones and the TOODY code retains 20 values for each zone as it steps from one time step to another (called cycles in the TOODY code).

The engineer using these hydro codes is interested in variations in effect with respect to time. Figure 2 shows contours of axial stress in the same configuration at 8.02 microseconds of problem time.

Without graphics it would be necessary to read a listing of nearly 18,000 numbers at each time step of interest to the engineer. This example problem is relatively small. Some jobs contain over 20,000 zones or 360,000 numbers per cycle.

Plotting routines have been used for data analysis for as long as the TOODY code has been in use. The code prepares a plot data file and the engineer provides data to the plotting routines to get the type of plot needed to make the desired analysis.

When running a TOODY problem, the engineer will request much more output information than he really needs because it is more economical to discard the excess output than to rerun the problem had he not requested output at the

proper time steps. Using batch processing to obtain these plots would necessitate either getting all the plots on one run or iterate with several runs to get the desired information.

Considerable savings is achieved by giving the engineer the capability of interacting with the plot routines. The same iterations are used at the console as would be used by requesting one plot at a time in batch mode. The difference is that the picture is observed in a few seconds instead of from two hours to one day for each iteration.

Sandia Laboratories Livermore has had a CDC 250 system attached to a 6600 for about three years. The 1024 x 1024 raster picture is maintained on the face of a 19-inch CRT by the 252 controller. This controller has 8,000 24-bit words with a limited instruction set for generating characters at 4 different sizes; and these, along with vectors and/or dots, can be generated at 2 different intensity levels. The characters have the added capability of being in either Roman or Italic form.

When interacting with the 6600 from the 250 console, the job ties up one control point and the code remains in central memory at all times. All changes in the picture require some interaction with the 6600. However, when the program awaits some activity from the operator, the 6600 will not use CPU time except to check for an interrupt request from the 250.

Because the interactive job has a tremendously low CPU utility factor, it behooves a programmer to use all the resources available to reduce the amount of central memory required to do the job.

At Sandia Laboratories Livermore, we accomplish this by using three levels of overlay available to the SCOPE system and the use of ECS (Extended Core Storage).

The use of overlays is the obvious technique of programming in function modules where the selection of a function switch or light pen pick, etc., would call in an overlay to diagnose the request so that more than one function will not be in memory at one time.

The sequential address, random access, and buffer I/O features of ECS, are extremely useful.

The plot file from TOODY is a buffered file. Each time frame on a plot file contains one record of general information identifying that frame plus one record for each I-line necessary to define the problem.

When the console user defines the frame he wishes to study, the program searches the plot for the record that identifies the desired frame. At this point, the code takes advantage of the consecutive addressing of ECS. Knowing the number of J-zones on this I-line, the ECS address of the adjacent zone on the next I-line can be computed. Since one of the 20 parameters of each zone is an integer (material number) which will not be larger than 20, two pointers are packed into 44 bits of that word. If we consider for a moment that I-line 1 is above I-line 2 and J-zone 1 is to the left of J-zone 2, each zone has a pointer to the ECS address of the zone directly above it and the zone directly below it. Special values are used to indicate no zone is in the respective direction. Adjacent zones on an I-line are adjacent data in ECS. With the pointers added, each I-line is written to ECS. The 4040 word array necessary for the transfer of the data from mass storage (or tape) to ECS is part of an overlay and the central memory space is available for other use in later steps.

To draw the contour curves, we must have one zone and its eight immediate adjacent zones in memory at one time. The random access feature of ECS serves here. The pointer system gives us the ECS address of the zone above and below and the buffer feature provides easy access to the data for three consecutive zones.

In actuality, only one zone is needed at a time to draw the boundaries and grids. The I/O time for this would be excessive and we have discovered that we can keep two I-lines in memory in the grid drawing overlay with the same CM requirements as the contour module with only nine zones in CM at one time.

Some examples of the types of plots we can observe are: Figure 3 shows the material code numbers displayed for each zone. This information is helpful in setting up a problem to assure the problem definition is proper before submitting the problem to a long computer run. Figure 4 shows integer values of axial stress with a divisor of 10^{10}. This is used as an aid in selecting the contour values for best representation. Figure 5 shows some of the integer values removed where too much display can cause confusion. Figure 6 shows the Zoom capability. We have enlarged the area of axial stress in tension and displayed selected values. Figure 7 shows the ability to superimpose the grid on the display of Figure 6. Almost any combination of the mentioned displays can be superimposed on each other. We can <u>not</u> display contours from more than one vector at a time.

Analyzing Spectral Data

Some of our physicists at Sandia Laboratories Livermore are studying gamma ray induced radioluminesence spectrum from various materials. An example of data obtained from laboratory equipment is shown in Figure 8. It is assumed the data curve is the sum of a series of Gaussian curves, each being a function of three independent parameters, X, Y (the coordinates of the peak of the curve), and S (the standard deviation). To date, we have not come up with a good analytical solution to this problem so we have attempted to give the physicist a tool to help him find a solution.

The engineer may have a data file in the computer containing coordinate sets for several curves. At the 250 console, the engineer may select any set of data from that file. The code will compute scale limits from the data and display the data on the screen as a series of points.

At the top of the screen, we find a display of the values of the parameters for the Gaussian curves. Six columns (for six curves) and three rows (three parameters per curve). If the Y or S parameter is equal to zero, that curve has not been defined.

At this point, the user, with the light pen, will pick one of the parameters in the first column. The display for that parameter will start blinking. There will also be a prompting message, "To use the T-cross for parameters, pick here," (Figure 9). The operator may type in a value for that parameter or he may pick the prompting message. If he chooses to type in the value, the value will appear on the screen and he may repeat the process until all three parameters in that column have been defined. If he picks the prompting message, that message disappears, the T-cross is

turned on at the center of the screen and a new prompting message, "When cross in desired position, pick here," appears (Figure 10). With the light pen, the user moves the T-cross to where he believes the peak of a curve should be (Figure 11). When he then picks the word HERE, the X parameter takes on the value of the T-cross and, when the average of three consecutive points becomes less than $Y/\sqrt{2}$, the distance from X to the horizontal location of the middle point becomes the half width of the curve. The standard deviation is computed and a curve is drawn on the screen (Figure 12). The user now has control of that curve (more on this later). He may change any of the three parameters by typing the symbol followed by the percent of the value to which he wishes it changed. For example, typing "s-3.5" would reduce the standard deviation by 3.5%.

When the first curve is in the desired shape, the user can pick a parameter in the second column and repeat the process. Up to six curves can be generated and, when more than one curve is defined, a TOTAL curve is computed and displayed. Figure 13 shows the use of four Gaussian curves to fit the data. The high sweep on the left is considered background noise and not included in the fit.

In order to properly fit the data, it is obvious that the user must be able to adjust any curve he so desires. It was indicated previously that the user had control of the last curve defined. In order to change any curve previously defined, the user must get control of the curve he wants to modify. Two methods are available:

1. Pick any parameter of the curve to be modified. This will make the display value of the parameter blink and also make the curve

blink. (This is important because it is easy to forget which set of parameters is associated with a particular curve.) The user may then type in a new value for that parameter or he may type any of the three parameter symbols followed by a percentage change desired.

2. Pick the curve he wishes to change. The curve will blink and the curve can be modified by the percentage change method described above.

Should the user want to delete any curve, he may do so by getting control of that curve by the method described above and selecting a prescribed function switch.

At Sandia Laboratories Livermore, we have in our user's library a non-linear least squares fit routine which will adjust the variable parameters in search of the best fit to given data. In our problem, the number of parameters provided are three times the number of curves.

It is interesting to note that at the same time the code described up to this point was being developed, the present principal user of the code was attempting to use the least squares fit routine in batch mode. The starting parameters were approximated by visual scanning of the data curve. It was usual for the batch program to take four to six minutes of 6600 CPU time to iterate to a solution. Quite often these would take more than 15 minutes of CPU time. It became obvious that the time to find a solution was dependent upon the quality of the approximation of the input parameters.

We incorporated the least squares routines into the interactive program. The user can select a function switch which allows the use of the least squares

routines. A vertical line will appear at left-hand and right-hand data points. These indicate the boundary limits of the data the routine will attempt to fit. The operator has the capability of moving these limits to the area of curve in which he is interested. Figure 14 shows the limits have been moved.

When the user signals he has selected the desired limits, he defines an iteration limit which was included to prevent the long runs described above. He then has the opportunity to "freeze" any parameter he wants to be held constant.

When the code arrives at the best fit it can or has reached an iteration limit, the parameter value display is revised and the curves redrawn. The standard deviation of the data points from the TOTAL curve is computed and displayed (Figure 14). The process may be repeated from this point. A curve may be added or deleted in an attempt to find a better fit.

This method of arriving at a set of parameters for the least squares curves seldom takes more than 30 seconds to arrive at a solution and most of the time the result is in less than 8 seconds. The final results are much better using this combined method. The first estimate of the parameters is better. The user has the opportunity to interject some of his knowledge of the sample into the solution. Computer usage time is way down.

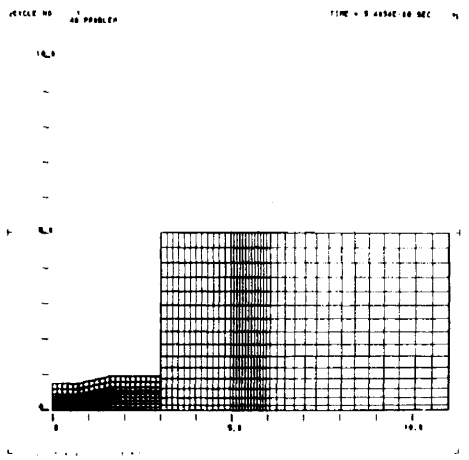


FIGURE 1

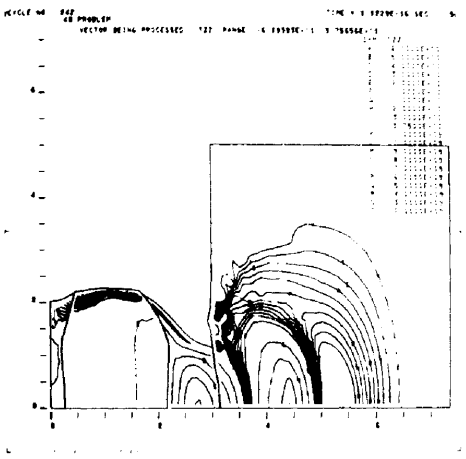


FIGURE 2

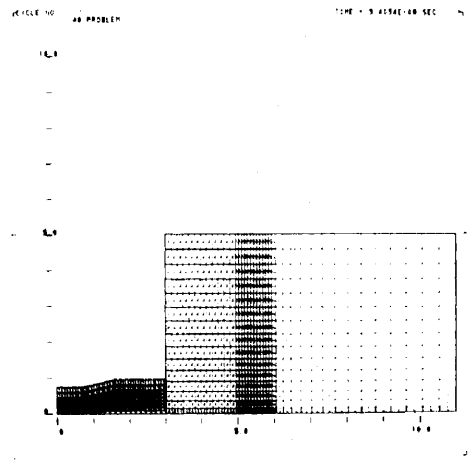


FIGURE 3

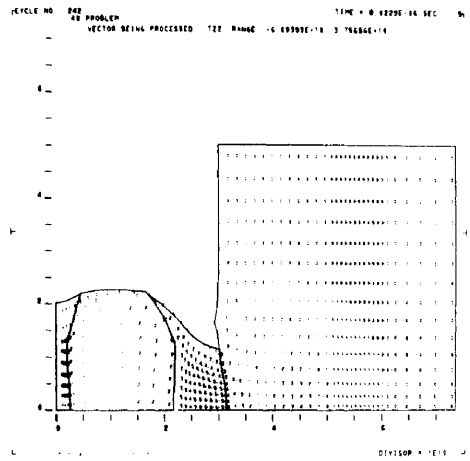


FIGURE 4

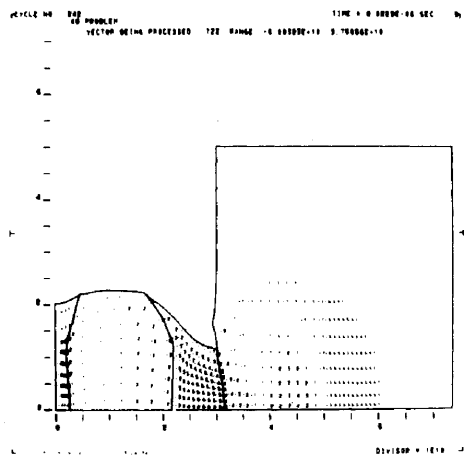


FIGURE 5

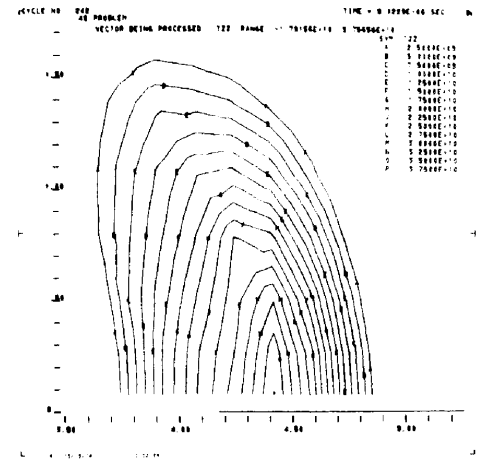


FIGURE 6

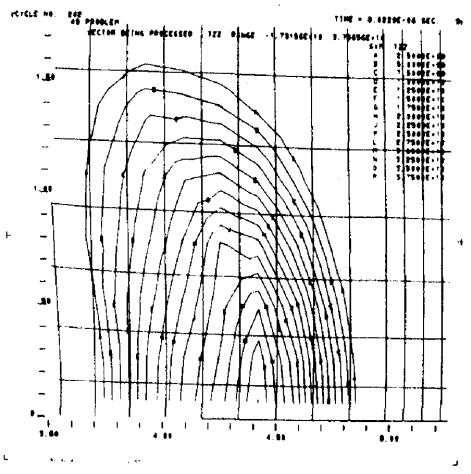


FIGURE 7

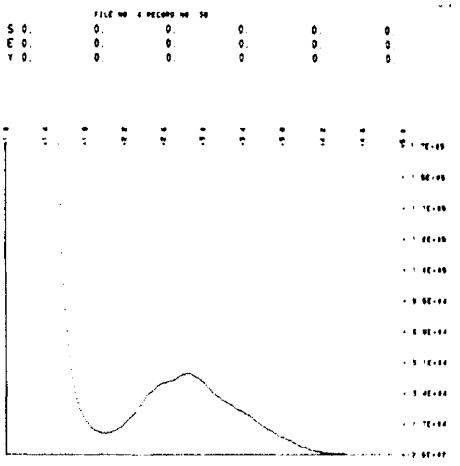


FIGURE 8

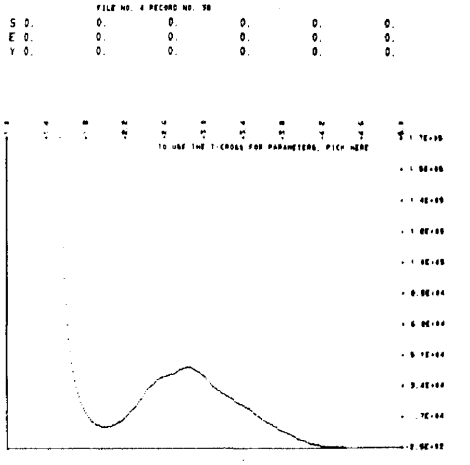


FIGURE 9

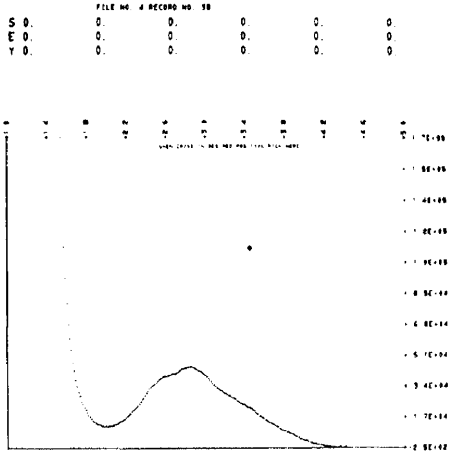


FIGURE 10

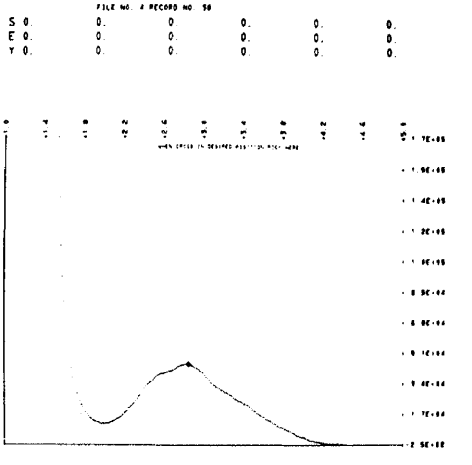


FIGURE 11

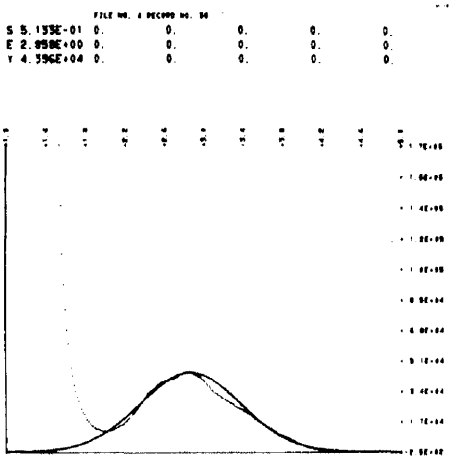


FIGURE 12

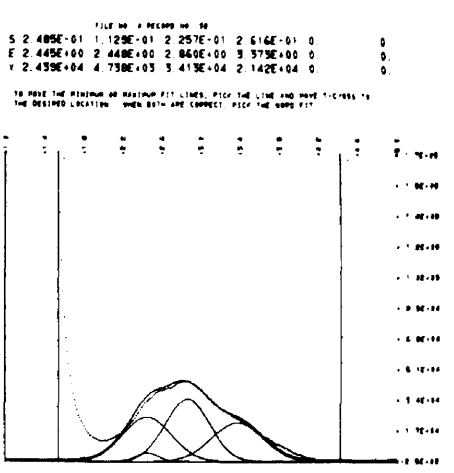


FIGURE 13

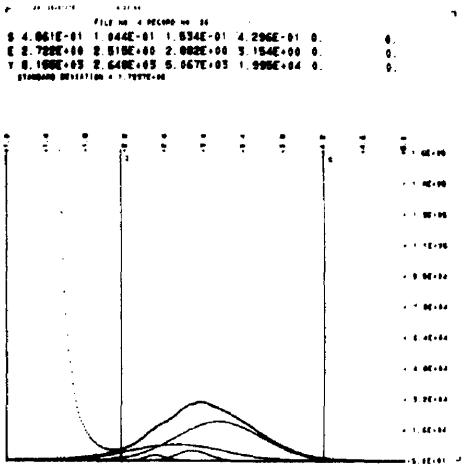


FIGURE 14

STABAN
AN INTERACTIVE GRAPHIC COMPUTERIZED STABILITY ANALYSIS PROGRAM*

B. J. Wimber**
Sandia Laboratories
Albuquerque, New Mexico 87115

ABSTRACT

This document describes an interactive graphic computerized stability analysis program (STABAN). Control system root locus plots, amplitude response characteristics, and attendant stability parameters are made available for interactive computer manipulation. Examples of the use of STABAN are included.

---

* Work supported by the U. S. Atomic Energy Commission

** Member of Technical Staff, Guidance and Control Division, Electro-Mechanical Subsystems Department.

STABAN
AN INTERACTIVE GRAPHIC COMPUTERIZED STABILITY ANALYSIS PROGRAM

I. Introduction

A computerized program for stability analysis and design of linear control systems with the use of the Sandia interactive graphics system was developed. This program is entitled STABAN (STABility ANalysis).

Control system stability analysis is computational in nature. However, the graphic methods associated with control system analysis, such as the root-locus plot, amplitude response, Bode plot, Nyquist plot, etc., convey far more information than several reams of detailed computer listings. However, to obtain these graphs for all but the simplest systems required a great deal of effort. Once the graphs were obtained, design changes could be made. But to determine the effect of these design changes, the graphs had to be redrawn with the new input information and the process repeated several times to obtain an optimum design. Thus, much of the control system engineer's time was used in laborious calculations and careful point by point graphing.

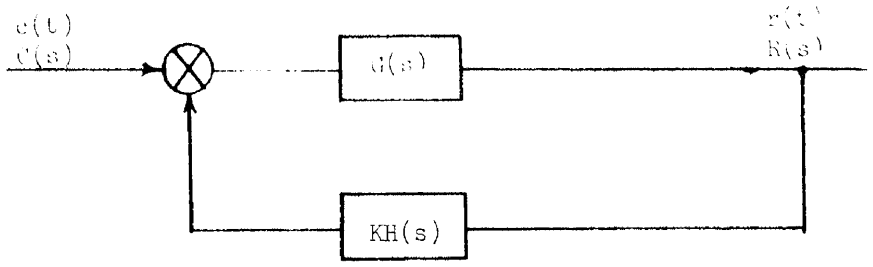
The emerging interactive graphics technology appeared to be the solution to the control system designer's dilemma; however, before a useful program could be made available, an interactive generalized graph plotting package had to be developed. This has been done, and the resulting easy to use, fast, and efficient interactive graphic computer program STABAN is now available. This paper introduces STABAN and shows how it is used. A design example is included for a better understanding of the program's utility and flexibility.

II. Basic Control System Fundamentals

This section presents a brief discussion of control system fundamentals as they relate to the use of STABAN. Reader familiarity with the Laplace transformation is assumed.

Generally, any control system can be represented by a block diagram which shows the transfer function of each component in the system as well

as the interrelationship of each component and its role in system operation. Conventional methods[1] are used to reduce a complex block diagram system to the following basic form:



where

G(s) is the Laplace transform of the feed-forward control system elements

H(s) is the Laplace transform of the feed-back control system elements

K is the open-loop servo gain

C(s) is the Laplace transform of the control function

R(s) is the Laplace transform of the response

The response is related to the control by the usual mathematical relationship (also referred to as the closed-loop transfer function $G_{c\ell}(s)$):

$$G_{c\ell}(s) = \frac{R(s)}{C(s)} = \frac{G(s)}{1 + KG(s)H(s)}$$

The concept of "stability" of the closed-loop system can be summed up as follows:

> For a system characterized by linear differential equations with constant coefficients, if for some control function, c(t), the response, r(t), does not grow without bound, the system is said to be stable.

The characteristic equation (CE) of the system contains the parameters that are needed to determine the degree of stability of the system. The CE is simply the denominator of the closed-loop transfer function and can be written in expanded form as follows:

$$CE = 1 + K \frac{\prod_{}^{n} \left(\frac{s}{z_i} + 1 \right)}{s^{\ell} \prod_{}^{m} \left(\frac{s}{p_j} + 1 \right)}$$

where

z_i represents the i^{th} open-loop zero

p_j represents the j^{th} open-loop pole

ℓ is the integration exponent

The roots (s_i) of the CE may be calculated as a function of gain, K, and their location can be plotted on the s-plane. Each point, s_i, has a real value, σ_i, and an imaginary value, $j\omega_i$. This plot is the well known root locus plot and is one of the displays provided by STABAN.

The roots of the CE are obtained by setting the above equation equal to zero. Hence, the equation becomes

$$s^{\ell} \prod_{}^{m} \left(\frac{s}{p_j} + 1\right) + K \prod_{}^{n} \left(\frac{s}{z_i} + 1\right) = 0$$

From this equation, one cannot resist the temptation to point out that:

(1) For K = 0, the roots of the CE are simply the poles of the open-loop transfer function.

(2) For K = ∞, the roots of the CE are simply the zeros of the open-loop transfer function. Thus the beginning and end of the root-locus trajectory are easily determined.

If the above equation is expanded into polynominal form, the result is:

$$\sum_{i=0}^{N} (A_i + KB_i)\, S^{N-i} = 0 \qquad N = m + \ell$$

The A_i and B_i are coefficients of the CE and are made up of the control system parameters. This form of the CE is used in the root solving subroutine. For convenience, the calculated coefficients, A_i and B_i, are available in one of the STABAN displays.

The open-loop transfer function ($G_{o\ell}(s)$) is written as follows:

$$G_{o\ell}(s) = KG(s)\, H(s)$$

If s is allowed to take on imaginary values only, i.e., let

$$s = j\omega ,$$

the open-loop transfer function becomes:

$$G_{o\ell}\ (j\omega) = KG(j\omega)\ H(j\omega)$$

Substitution of values of ω into $G_{o\ell}(j\omega)$ will result in the open-loop amplitude response where both the amplitude ratio and the phase angle for each value of ω are provided. This amplitude response and the important indicators of stability, i.e., phase margin (PM) and gain margin (GM), are readily presented using STABAN. This display also provides the servo designer with the capability of being able to "shape" the open-loop amplitude response to his satisfaction by simply selecting open-loop poles and zeros as his judgement dictates.

If the open-loop transfer function $G_{o\ell}\ (j\omega)$ is plotted on polar coordinates, the degree of stability is shown by the plot's avoidance of the $-1\ (1/\underline{180°})$ point of the plot. This important element of the Nyquist criterion can be easily investigated with the polar plots generated by STABAN.

The closed-loop transfer function may also be written as follows:

$$G_{c\ell}\ (s) = \frac{K_{fw}\ \dfrac{Z_{fw}(s)}{P_{fw}(s)}}{1 + K_{fw}\ \dfrac{Z_{fw}(s)}{P_{fw}(s)}\ K_{fb}\ \dfrac{Z_{fb}(s)}{P_{fb}(s)}}$$

where

K_{fw} feed forward gain

$Z_{fw}(s)$ feed forward zeroes

$P_{fw}(s)$ feed forward poles

K_{fb} feedback gain

$Z_{fb}(s)$ feedback zeroes

$P_{fb}(s)$ feedback poles

or

$$G_{c\ell}(s) = \frac{K_{fw}\ Z_{fw}(s)\ P_{fb}(s)}{CE}\ .$$

Thus, if the poles of the closed-loop transfer function (roots of the CE at the preselected open-loop gain, K) are available, the amplitude response of the closed-loop transfer function, $G_{c\ell}(j\omega)$, can be obtained easily. STABAN provides this capability. The STABAN program consists of the interactive graphic computer language developed at Sandia Laboratories Albuquerque[2] for use with the Control Data Corporation's 6600 computer, Digital Equipment Corporation's PDP-9, and the Vector General display equipment.

The program is ready to receive interactive commands and data when Menu A (Figure 1) is displayed. The symbols shown on the Menu are those associated with the open-loop transfer function of the control system being analyzed.

All of the variables in the Menu are capable of being interactively varied in accordance with specific design requirements. Changes of the variables and selection of the desired display are accomplished using the light pen and keyboard attached to the terminal.

To change a variable:

(1) Hold light pen directly over variable to be changed and touch ring near end of pen with finger. The value of the variable will disappear. (This procedure is referred to as a "hit".)

(2) Using the terminal keyboard, enter the new value, observing the proper format. The number will appear on the screen as it is typed.

(3) When new value has been properly entered, depress ESC (Escape) key.

After the appropriate variables are inserted, one can select one or more of the following displays:

COEF

ROOT-LOCUS

AMP RESP

LOG MOD

MODULUS

The display (or displays) is selected by a light pen "hit" of the desired code word. An asterisk preceding the code word indicates program acknowledgement of the selection.

Note: The values that appear on the screen as shown in this figure as well as in Figures 2 through 6 are from the open-loop transfer function:

$$G_{ol}(s) = \frac{2.5 \times 10^{6}\left(\frac{s}{40} + 1\right)^{2}}{s\left(\frac{s}{120 + j248} + 1\right)\left(\frac{s}{120 - j248} + 1\right)\left(\frac{s}{0.25} + 1\right)^{2}}$$

Amplitude response range: 0.01 to 10,000 radians/sec. Root locus plot in upper left quadrant bounded at 250 rad/sec.

This example is integrated with STABAN to provide a means of testing the program. Any of the variables may be changed by using the light pen.

```
NO INTEGS= 1
NO ZEROES= 2
NO POLES = 4

WMN(EXP) =-2
WMX(EXP) = 4
UMN=    -2.50E 2
UMX=     0.00E 0
VMN=     0.00E 0
VMX=     2.50E 2
K GAIN=  2.50E 6
ZEROES
   40.00      .01
   40.00      .01
    0.00     0.00
    0.00     0.00
    0.00     0.00
    0.00     0.00
    0.00     0.00
    0.00     0.00
    0.00     0.00
    0.00     0.00
POLES
 -120.00   248.00
 -120.00  -248.00
    -.25      .00
    -.25     -.00
    0.00     0.00
    0.00     0.00
    0.00     0.00
    0.00     0.00
    0.00     0.00
    0.00     0.00

COEFS
ROOT-LOCUS
AMP RESP
LOG MOD
MODULUS
COMPUTE

END                    *HARD COPY
```

Figure 1. MENU A

To run the program, hit COMPUTE. The graphic display will indicate that the 6600 computer is operating on the data by showing a message:

6600 WORKING

When the desired display is on the graphic monitor, wait for the above message to disappear before attempting to input more data.

III. The STABAN Displays

A description of the displays follows.

Menu A

The variables in the Menu are associated with the open-loop transfer function as follows.

NO INTEGS = Number of integrations, ℓ

NO ZEROS = Number of simple zeros, n ($n \leq 10$)

NO POLES = Number of simple poles, m ($m \leq 10 - \ell$)

WMN (EXP) = Minimum angular frequency exponent (power of 10) used in the calculation of amplitude response (angular frequency in radians per second)

WMX (EXP) = Maximum angular frequency exponent used in the calculation of amplitude response

UMN = Left bound (abscissa) of root-locus plot (radians per second)

UMX = Right bound (abscissa) of root-locus plot (radians per second)

VMN = Lower bound (ordinate) of root-locus plot (radians per second)

VMX = Upper bound (ordinate) of root-locus plot (radians per second)

K GAIN = Open-loop gain corresponding with design requirements and used in the calculation of amplitude ratio

ZEROS = Actual complex value of zeros entered as

real part imaginary part

POLES	- Actual complex value of poles entered as real part imaginary part
COEFS	- Code word for coefficient display
ROOT-LOCUS	- Code word for root-locus display
AMP RESP	- Code word for amplitude response characteristic display
LOG MOD MODULUS	- Code words for polar plot of amplitude response
COMPUTE	- Code word used to execute the program

The values shown in Figure 1 as well as the values and graphs in Figures 2 through 6 are from the following open-loop transfer function:

$$G_{o\ell}(s) = \frac{2 \cdot 5 \times 10^{6}\left(\frac{s}{40} + 1\right)^{2}}{s\left(\frac{s}{120 + j248} + 1\right)\left(\frac{s}{120 - j248} + 1\right)\left(\frac{s}{0.25} + 1\right)^{2}}$$

The amplitude response ranges from 0.01 to 10,000 radians per second.

The root-locus plot desired is the portion in the upper left quadrant of the s-plane bounded at 250 radians per second.

COEFS

Selection of this code word will result in the display shown in Figure 2. The main difference between this display and that of Menu A is the appearance of the following terms:

NO COEFS =

KL =

KU =

NDK =

A-COEFS

B-COEFS

Z-S+P-S

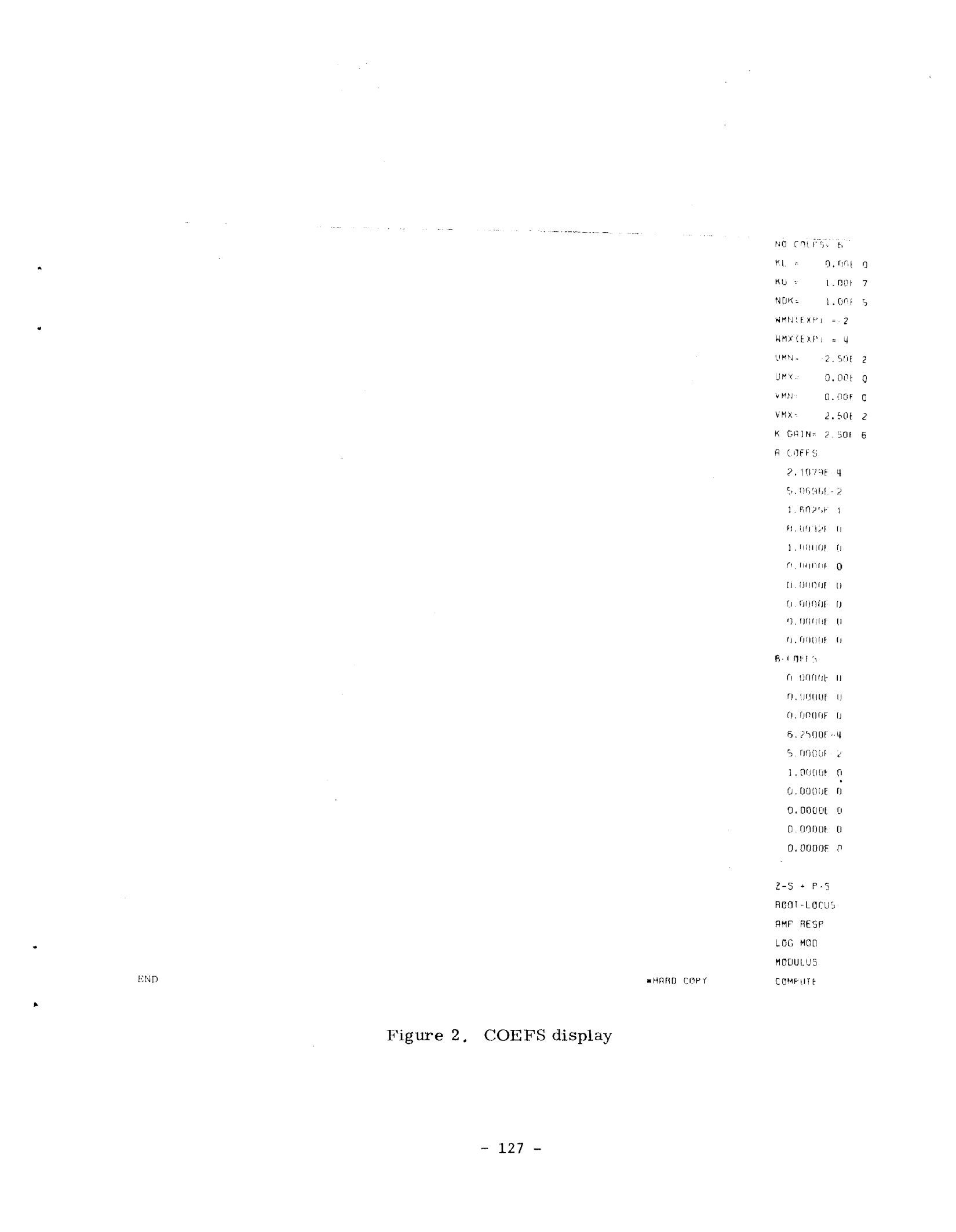


Figure 2. COEFS display

The terms NO COEFS, A-COEFS and B-COEFS are associated with the polynomial form of the CE (see page 5). Since poles, zeros, and integrators are the input variables, as entered in Menu A, the corresponding coefficients of the polynomial form of the CE are calculated and displayed. NO COEFS is the degree of the polynomial plus 1 (used in the root solving subroutine). To observe the original values of the poles and zeros and/or change their values the user must select

Z-S+P-S

and Menu A will again be displayed with the original pole and zero values.

The terms KL, KU, and NDK are the lower, upper, and incremental values of the open-loop gain K. As explained in Reference 3, the values of KL, KU, and NDK must be chosen so that the 6600 computer is not required to calculate a large number of roots. If this happens, the graphic terminal display will respond with a message "DISPLAY TOO LARGE". Adjustment of KL, KU, and NDK should be made so that the above message does not appear.

ROOT-LOCUS

Selection of this display (see Figure 3) alone will provide the s-plane bounded by preselected values UMN, UMX, VMN, VMX, and a point by point plot of the root locus beginning at the lower gain, KL, and ending at the maximum gain, KU, incrementing by the amount NDK. Also shown on the display are radial lines of constant damping ratio.

From this one display, the control system design engineer can obtain the following information about his design.

(1) The value of open-loop gain, K, which corresponds with the desired time response of the closed-loop system (i.e., damping ratio). Although this is shown in the s-plane as an intersection of root locus plot with lines of constant damping ratio (ZETA), the exact gain values versus damping ratio are tabulated below the s-plane plot. More than one value of gain may be displayed for a given damping ratio depending on which segment of the root locus plot intersected that damping ratio line.

(2) The resonant frequency of the closed-loop system.

(3) The value of gain where the root locus crosses over into the right half s-plane (system becomes unstable).

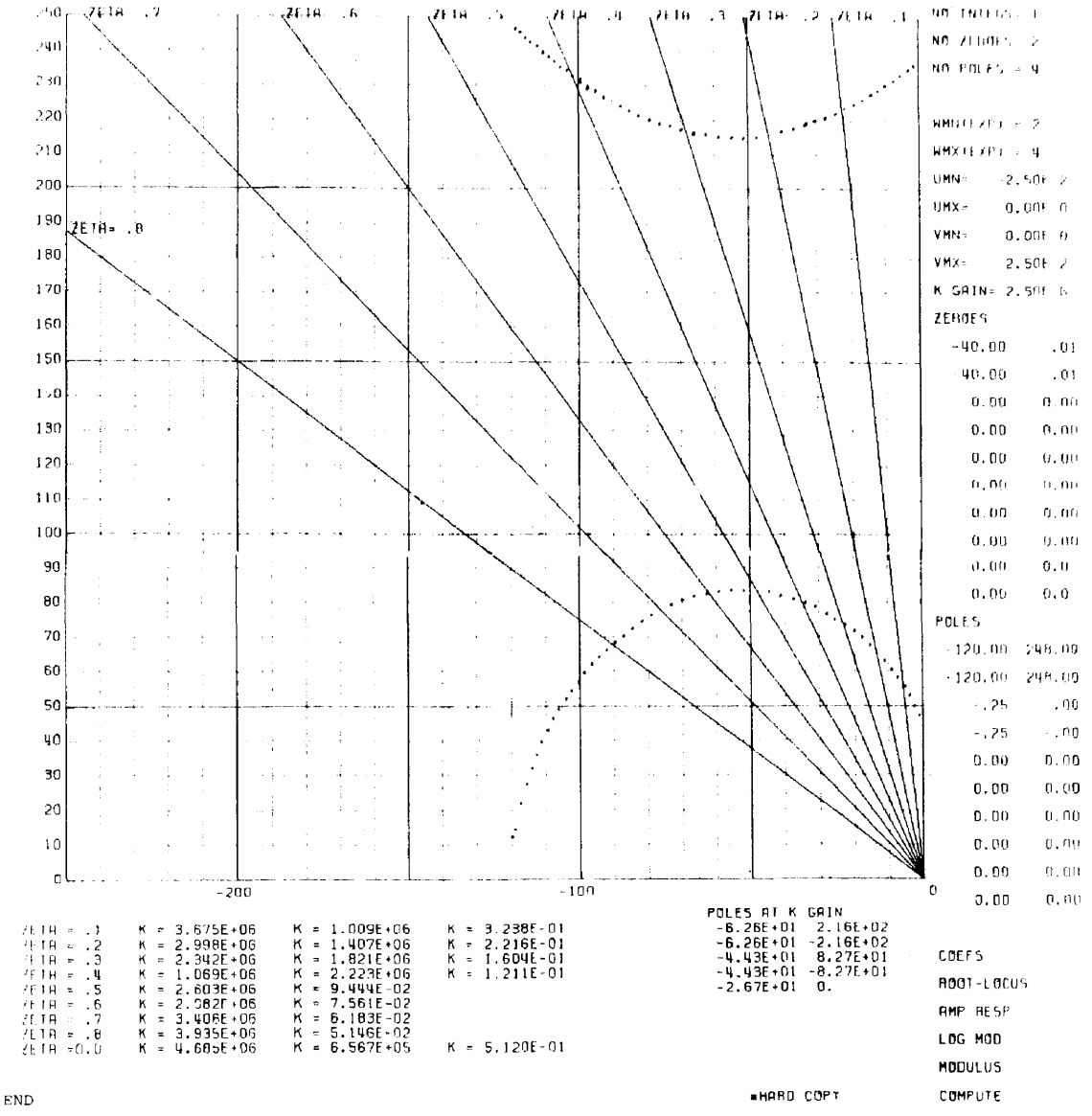


Figure 3. ROOT-LOCUS display

(4) The exact value of all roots of the CE (poles of the closed-loop transfer function) as a function of open-loop gain K. The closed-loop poles are calculated for each increment, NDK, of gain K. To obtain the poles for any open-loop gain value simply locate the symbol: K GAIN = (Menu A), enter the value of open loop gain, hit ROOT-LOCUS, hit COMPUTE, and the value of the poles will appear near the lower right corner of the display with the heading: POLES AT K GAIN.

Along the right side of this display, as well as with the other displays, is either Menu A (poles and zeroes are shown in the figure) or the COEFS display. Thus, the designer has the capability to interactively change values of variables and examine the effects of those changes. Thus, a design that may have taken many days using previous methods can be accomplished in a matter of minutes using STABAN.

The root locus plot is an excellent analysis tool. However, in some cases design efficiency is enhanced with the use of the amplitude response of the open-loop transfer function.

AMP RESP

Selection of this display (see Figure 4) alone will provide the open-loop amplitude ratio and phase angle response as a function of frequency. This response is plotted as the frequency increases from the preselected lower bound (WMN exponent) to the maximum bound (WMX exponent). A very important feature of this display is the scales of amplitude ratio (db) and the phase angle on the left side of the graph. Note that the 0 db amplitude ratio and the 180° phase angle are the same ordinate position. Thus, one can determine at a glance two very important indicators of stability, i.e., phase margin (PM), which is the phase angle avoidance of the 180° phase shift condition, and gain margin (GM), which is the factor by which the open-loop gain is less than the 0 db amplitude ratio condition when the phase angle equals 180°. This display relates directly to the "Bode" diagram, which has proven to be an excellent design tool for servo designers. A brief discussion of basic principles will facilitate a better understanding of the use of the amplitude response in control system design.[4]

Design Using the Bode Diagram

The Bode diagram is simply an asymptotic approximation to the actual amplitude ratio response. One needs only to determine the "break" frequencies of the open-loop transfer function and using

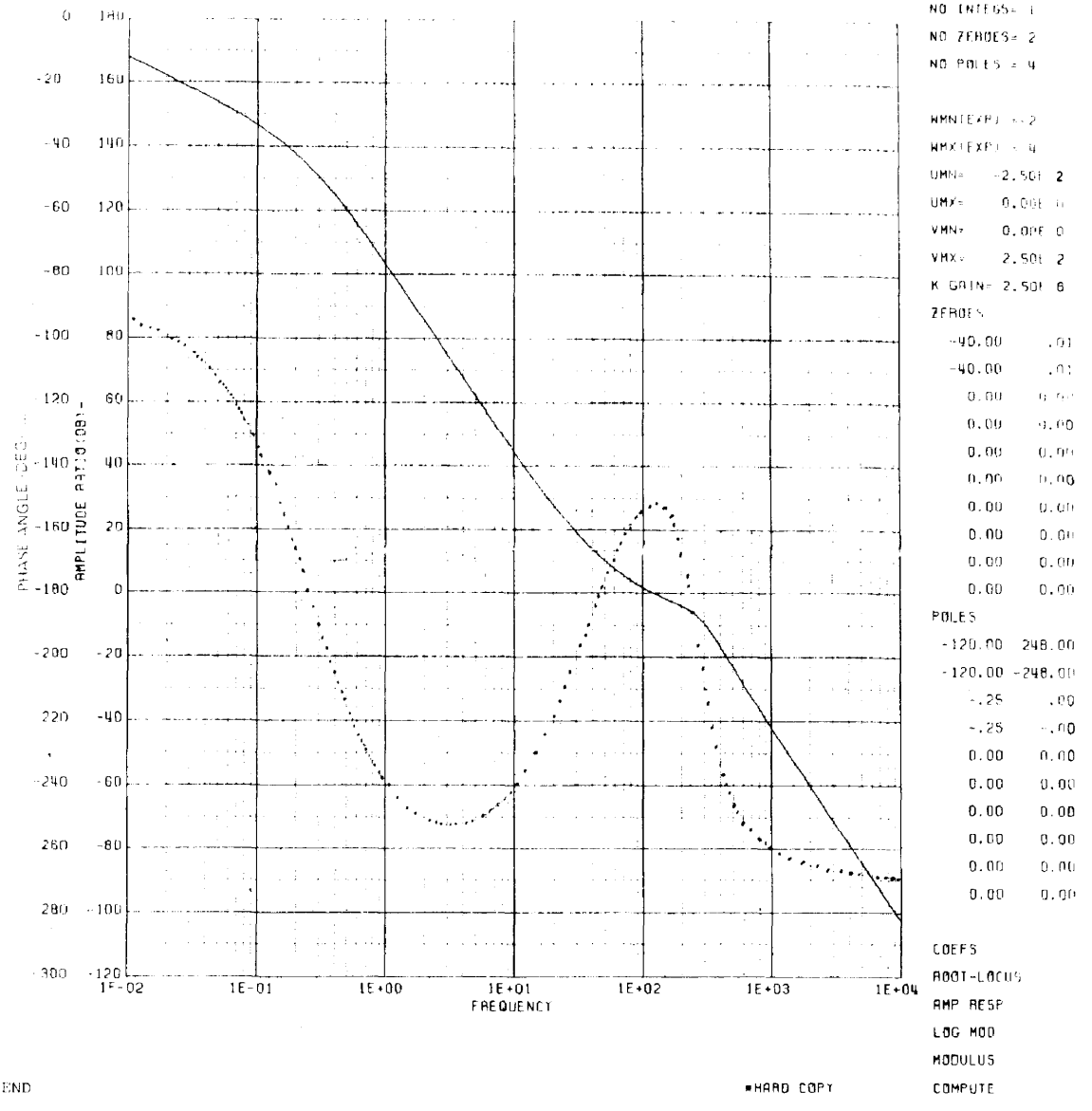


Figure 4. AMP RESP display

semi-log paper draw an asymptotic approximation to the amplitude ratio. The slope of the asymptotic approximation would correspond with the order of the term being approximated, i.e.,

1st order - 6 db/octave

2nd order - 12 db/octave

3rd order - 18 db octave

"

Nth order - 6Ndb/octave

The following principles govern this design technique:

(1) Crossover at the 0 db of amplitude ratio should be with a slope of 6 db per octave.

(2) The ratio of the break frequency at the end of the 6 db/octave crossover segment to the break frequency at the beginning of the segment should be on the order of 7 to 1 for adequate stability.

(3) The 0 db crossover point should be approximately at the center of the 6 db/octave segment.

By using the above principles, the servo designer can "shape" his amplitude response to achieve gain requirements and provide for adequate phase and gain margins. Once the amplitude response has been shaped to the designer's satisfaction, the root locus plot of the new system design can be displayed simply by selecting ROOT-LOCUS. Some adjustment of KU, KL, and NDK may be necessary to obtain a well defined root locus plot.

Another indicator of servo system performance is the LOG MODULUS or MODULUS (Nyquist) plots.

LOG MODULUS

This display (Figure 5) is a polar plot of the amplitude response. Since the radius is the logarithm of the amplitude ratio, the behavior of the control system in the mid-frequency range of interest can be easily observed. This behavior is illustrated by the amount of phase angle "roll-off" in the low to mid-frequency range. Also displayed is the calculated value of the PM, the GM, and the frequency, W(∅ DB), at which 0 db occurs.

MODULUS

This display (Figure 6) is also a polar plot of the amplitude response. However, the critical area of interest in this case is the region of the

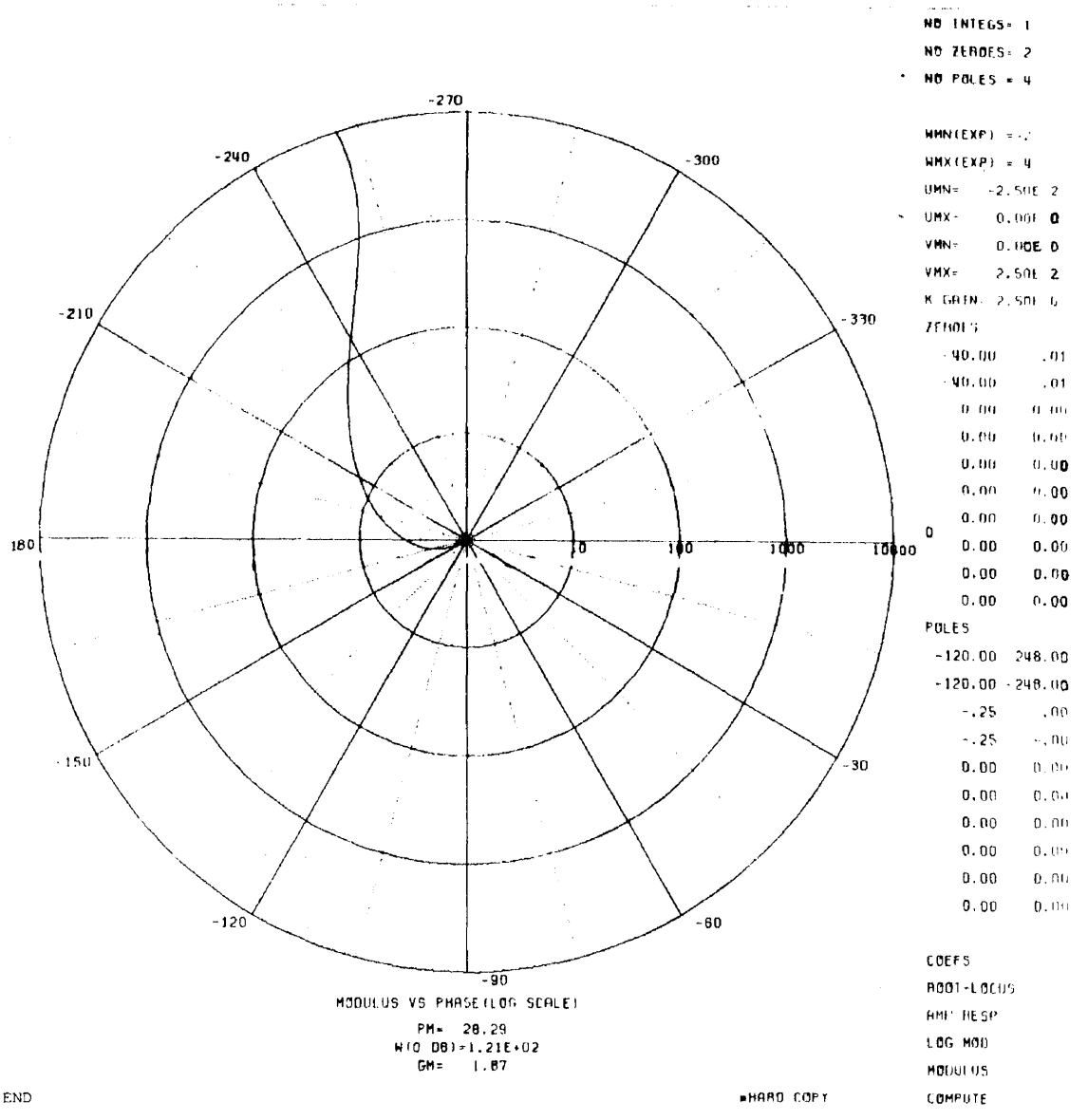


Figure 5. LOG MODULUS display

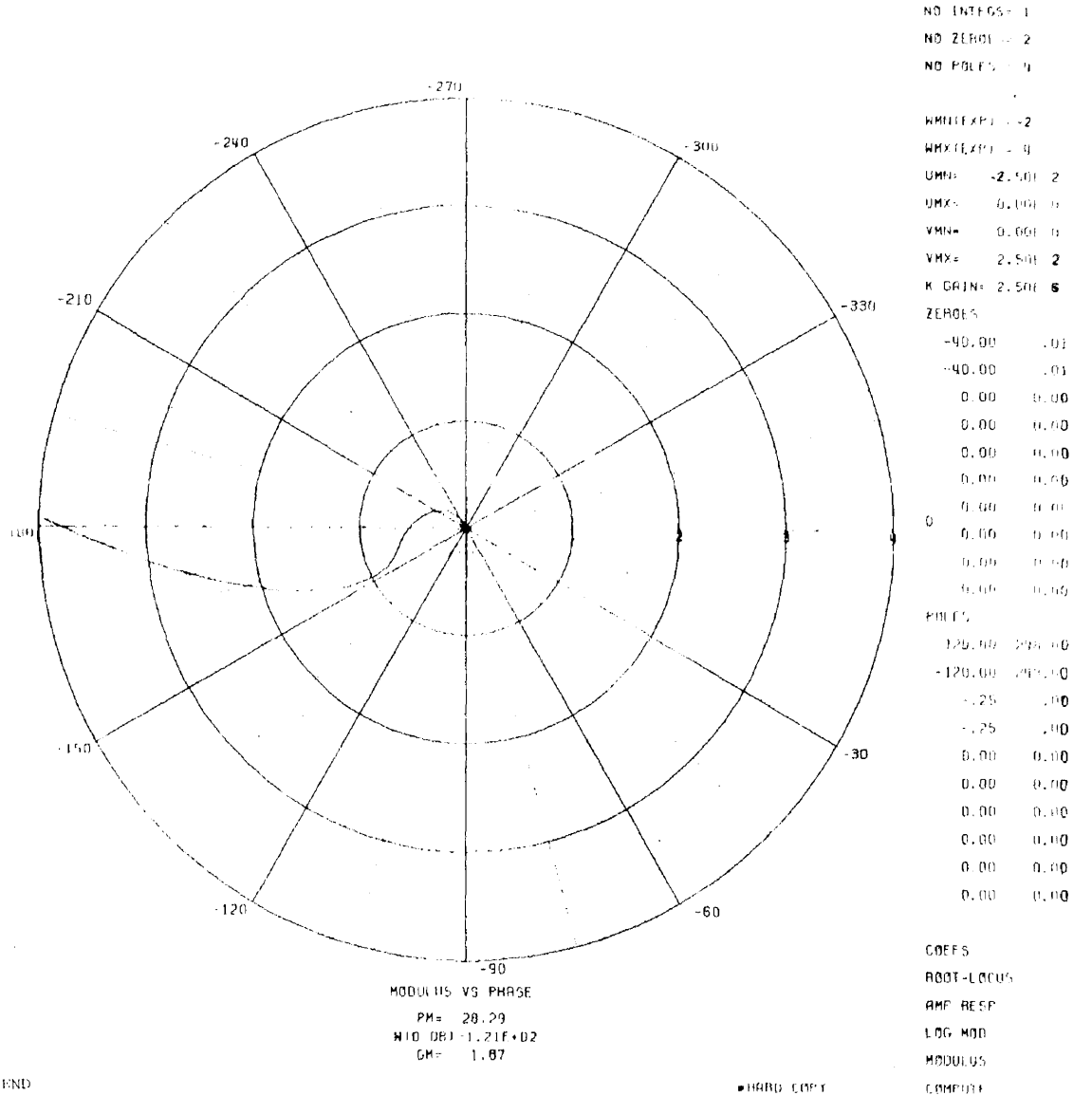


Figure 6. MODULUS display

-1(0 DB at 180°) point. Clearly shown is the PM, the GM, and the trajectory with which the -1 point is crossed. The calculated values of PM, GM and W(ϕ DB) are also displayed.

Incorporated within the STABAN program is the capability of obtaining a hard copy of any display desired. One needs only to "hit" HARD COPY and the display is stored on a memory disk within the PDP-9 for later processing using an on-line printer such as the Gould Model 4800 plotter. Another useful feature afforded by the graphic display technique is that pictures of the displays can be taken with a Polaroid camera and developed in seconds for use in log books, etc.

IV. Example of Use of STABAN

The following example is presented in detail to illustrate the use of STABAN in solving an actual control problem. The design is for an attitude control system stabilized with the use of a conventional rate sensor.[5] The simplified single axis servo block diagram is shown in Figure 7, where C(s) is the Laplace transform of the input torque to the platform and R(s) is the Laplace transform of the platform motion.

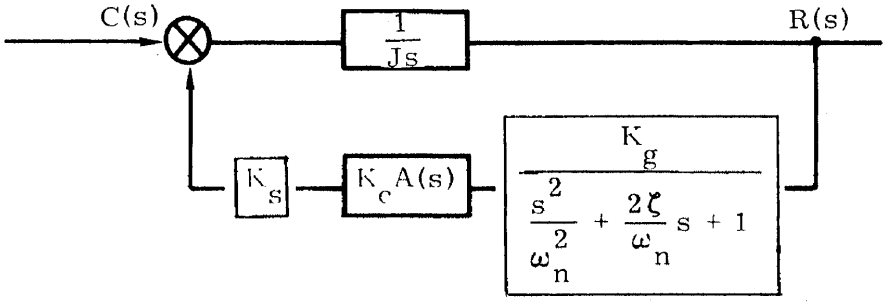


Figure 7. Simplified single axis servo block diagram.

The CE of the uncompensated ($K_c A(s) = 1$) system is

$$CE = s\left(\frac{s^2}{\omega_n^2} + \frac{2\zeta}{\omega_n} s + 1\right) + KA(s)$$

where

$\omega_n = 276$	natural frequency
$\zeta = 0.5$	damping ratio
$K = 1/J\ K_c = \frac{K_s K_c K_g}{J}$	open-loop servo gain
$K_s = 2.7 \times 10^4$ dyne-cm/volt	servo amp gain
$K_c = 1$	
$K_g = 14.7$ volts/rad/sec	rate sensor gain
$j = 515$ gm-cm^2	platform inertia

The STABAN input data and attendant root-locus plot is shown in Figure 8. As shown in the figure, the system will go unstable at a gain $K = 276\ sec^{-1}$. The appropriate compensation is not immediately obvious from this display. However, if the amplitude response is selected, as shown in Figure 9, some idea of compensation required is clear if the rules for design given on page 14 are followed.

Since there is a second-order break due to the rate sensor at a frequency of 276 rad/sec, we can use this as the break frequency at the end of the 6 db/octave crossover segment. Then for adequate stability the beginning of the 6 db/crossover segment should occur at $\frac{276}{7} \cong 40$ rad/sec, ω_b. Therefore, between some low frequency ω_a and ω_b we need to "stretch" the amplitude response to obtain the required open loop gain before crossover.

We desire the amplitude response of the lag compensator shown in Figure 10. The transfer function of the lag compensator is

$$\frac{E_2(s)}{E_1(s)} = \frac{\tau s + 1}{a\tau s + 1}$$

where

$$\tau = R_2 C$$

$$a = \frac{R_1 + R_2}{R_2} \geq 1$$

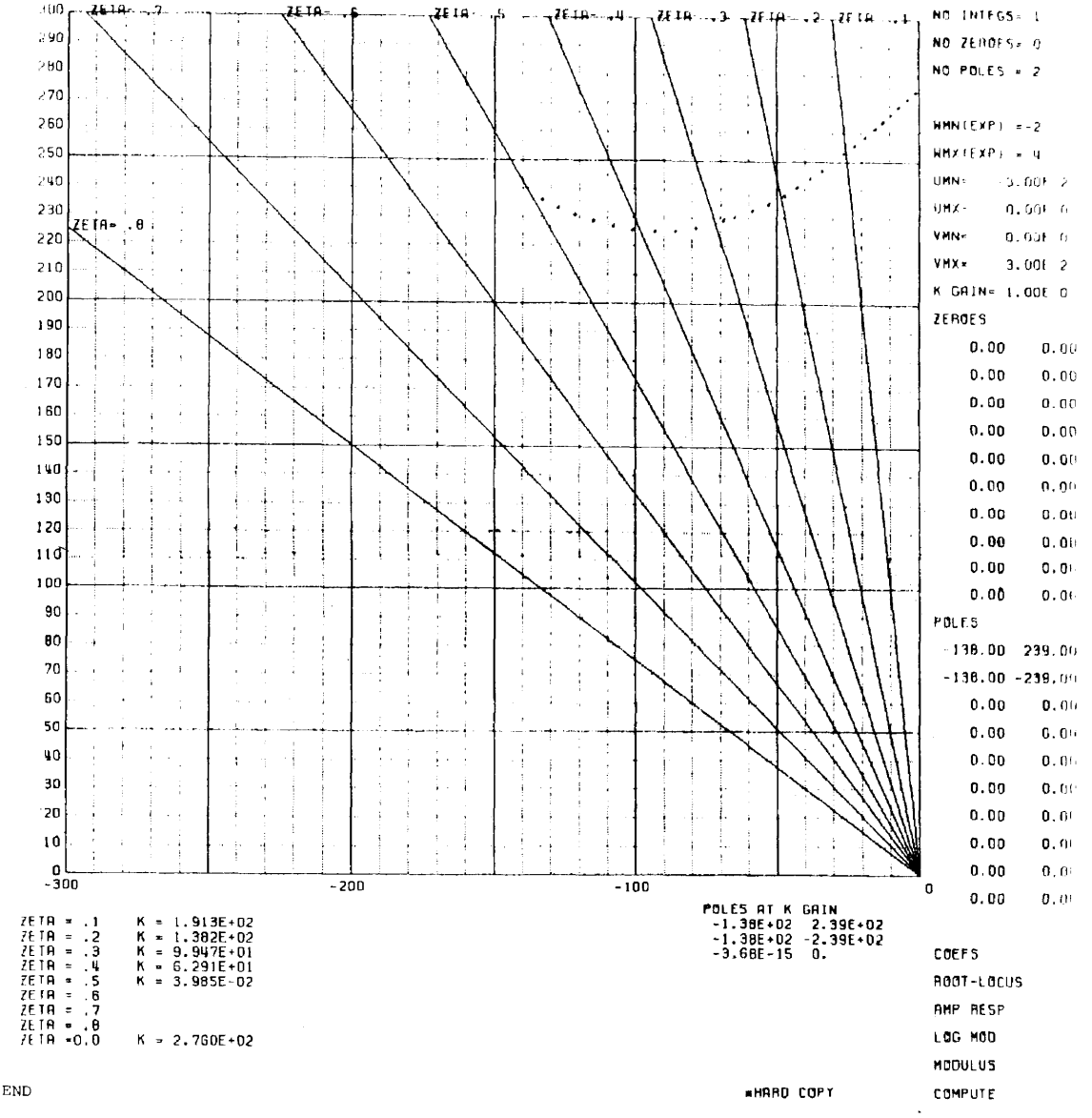


Figure 8. Root locus plot of uncompensated system

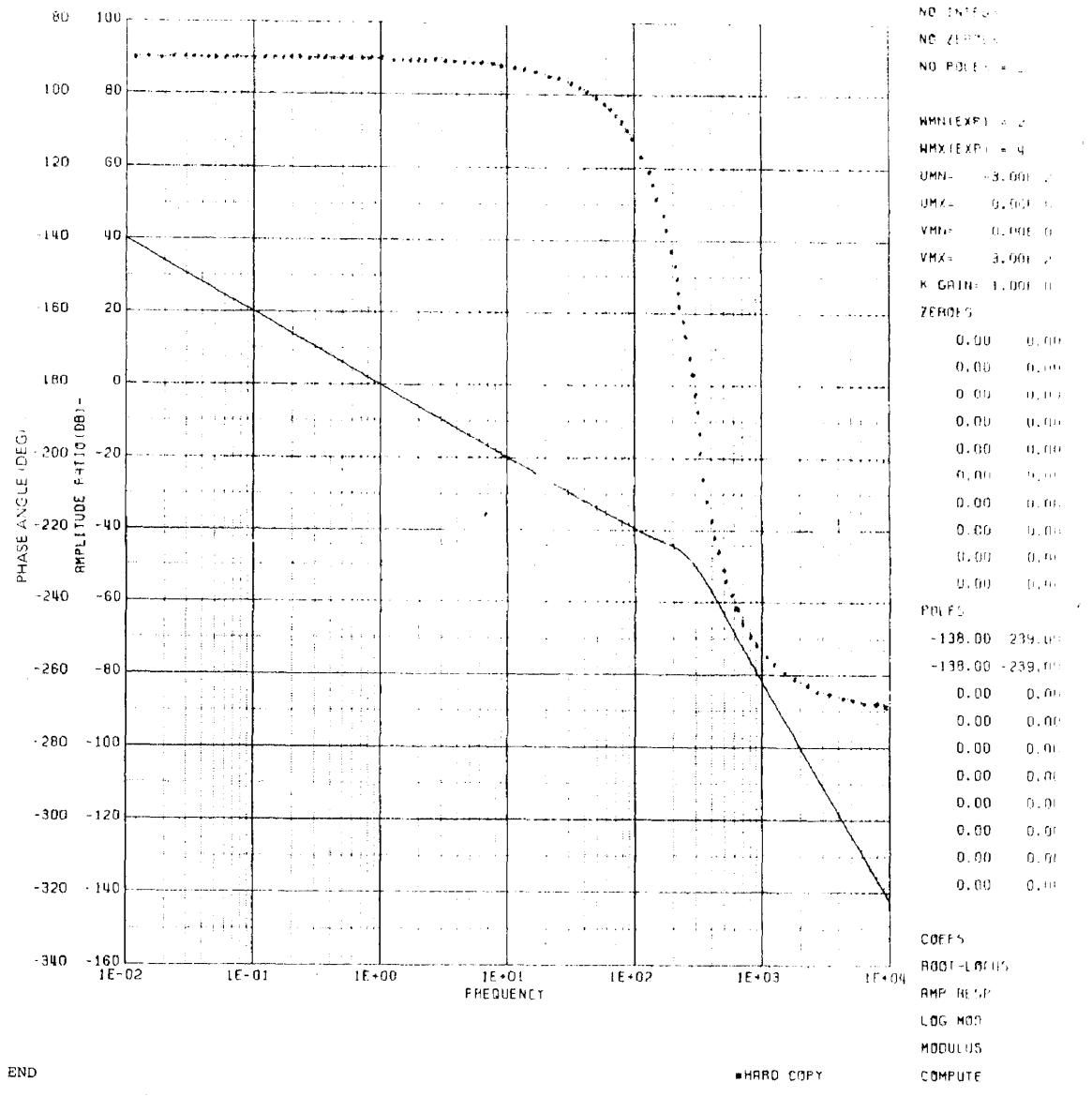


Figure 9. Amplitude response of uncompensated system

We will let $\tau = 1/\omega_b = 1/40 = 0.025$ sec. Let us assign $\omega_a = 0.25$ rad/sec. The transfer function becomes:

$$\frac{E_2(s)}{E_1(s)} = \frac{\left(\frac{s}{40} + 1\right)}{\left(\frac{s}{0.25} + 1\right)} .$$

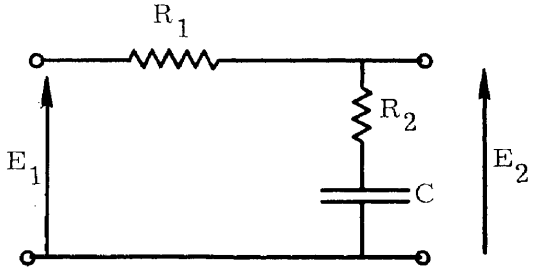


Figure 10. Lag compensator

The amplitude response of the lag compensator and the input variable values are shown in Figure 11. We can see from Figure 11 that we can stretch the open-loop amplitude response characteristic of our system by 43 db. Thus, if the lag compensator is used

$$A(s) = \frac{\left(\frac{s}{40} + 1\right)}{\left(\frac{s}{0.25} + 1\right)} .$$

We can see its effect on closed-loop stability by the root-locus display shown in Figure 12. As is shown in the figure, the maximum gain before the system goes unstable is 37,790 sec^{-1} compared with 276 sec^{-1} previously, or an increase of open-loop gain by a factor of 137 (43 db).

Thus, the system has been made more stable than before. It still is not clear as to how stable the system is until one selects a particular open-loop gain commensurate with design requirements. Let us assume for the purpose of this example that our design requirements can be met with a gain $K = 1.2 \times 10^4$ sec^{-1}.

We now desire the degree of stability afforded by our selection of compensation and open-loop gain. By imputing the desired K value into the computer and selecting AMP-RESP, LOG-MOD, and MODULUS, the displays shown in Figures 13 and 14 appear. As shown in both figures,

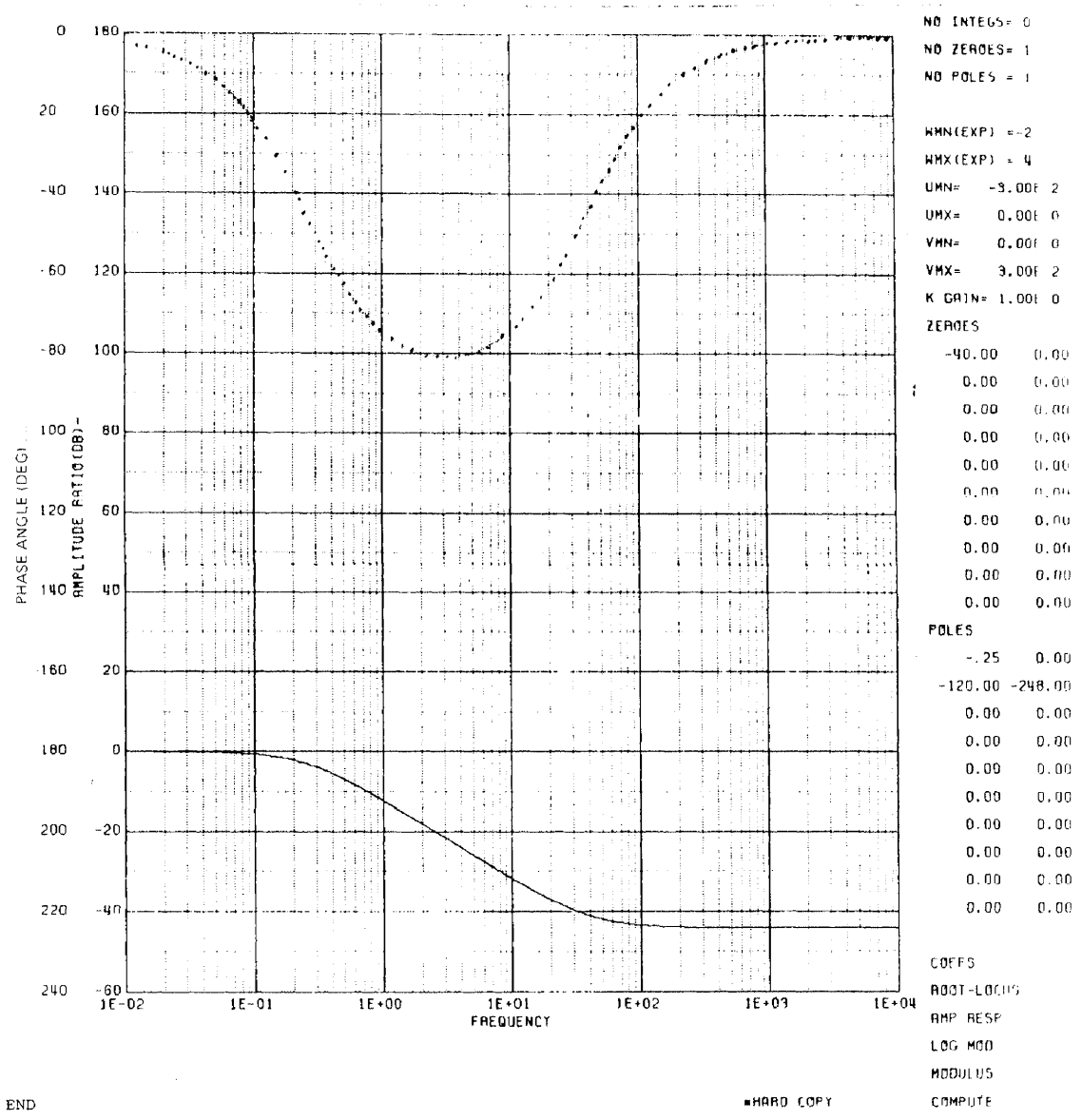


Figure 11. Lag compensator amplitude response and input variable values

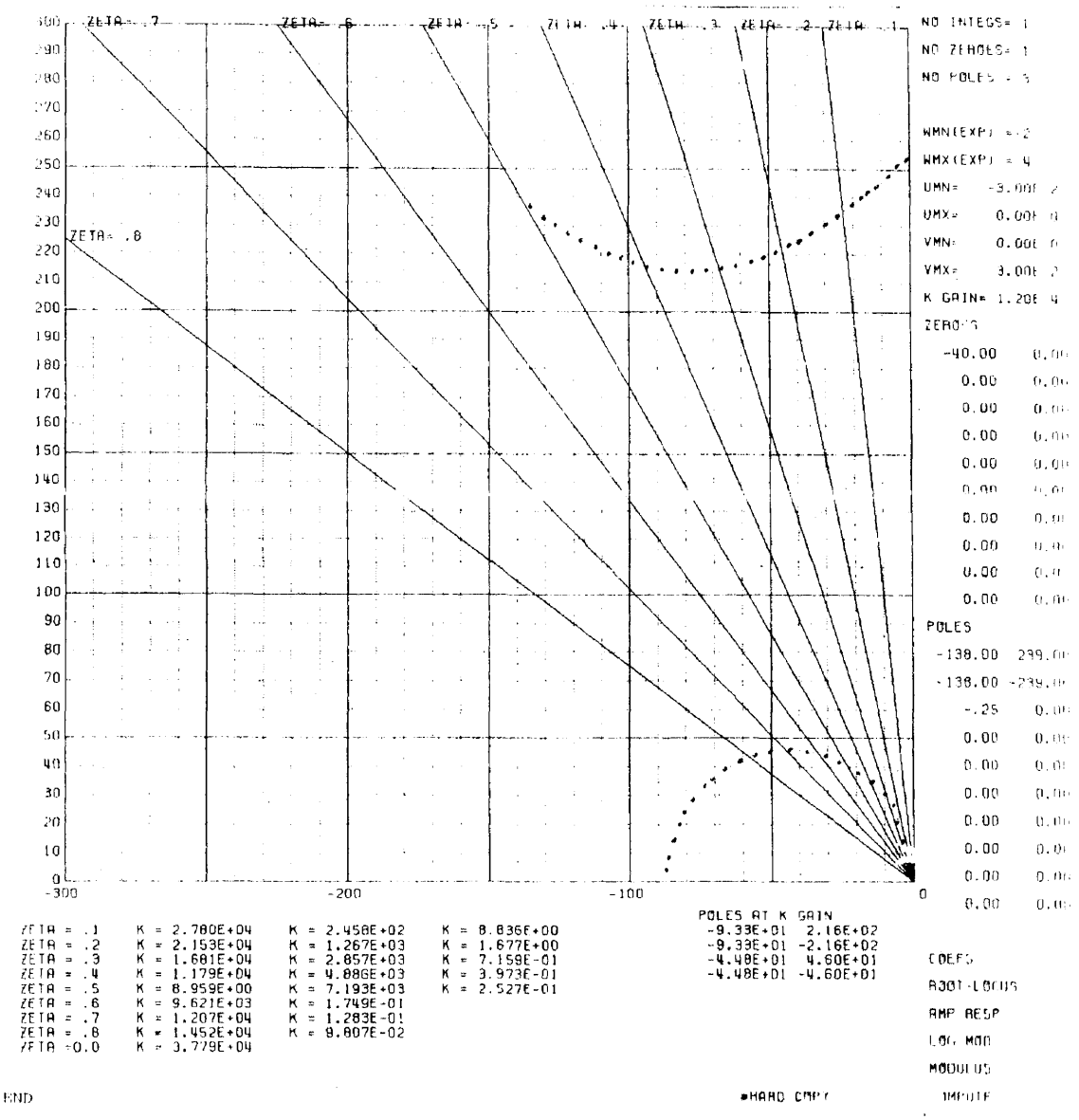

ZETA= .7
ZETA= .6
ZETA= .5
ZETA= .4
ZETA= .3
ZETA= .2
ZETA= .1
ZETA= .8
NO INTEGS= 1
NO ZEROES= 1
NO POLES = 3
WMN(EXP) = -2
WMX(EXP) = 4
UMN= -3.00E 2
UMX= 0.00E 0
VMN= 0.00E 0
VMX= 3.00E 2
K GAIN= 1.20E 4
ZEROES
-40.00 0.00
0.00 0.00
0.00 0.00
0.00 0.00
0.00 0.00
0.00 0.00
0.00 0.00
0.00 0.00
0.00 0.00
0.00 0.00
POLES
-138.00 299.00
-138.00 -299.00
-.25 0.00
0.00 0.00
0.00 0.00
0.00 0.00
0.00 0.00
0.00 0.00
0.00 0.00
0.00 0.00
-300
-200
-100
0
ZETA = .1 K = 2.780E+04 K = 2.458E+02 K = 8.836E+00
ZETA = .2 K = 2.153E+04 K = 1.267E+03 K = 1.677E+00
ZETA = .3 K = 1.681E+04 K = 2.857E+03 K = 7.159E-01
ZETA = .4 K = 1.179E+04 K = 4.886E+03 K = 3.973E-01
ZETA = .5 K = 8.959E+00 K = 7.193E+03 K = 2.527E-01
ZETA = .6 K = 9.621E+03 K = 1.749E-01
ZETA = .7 K = 1.207E+04 K = 1.283E-01
ZETA = .8 K = 1.452E+04 K = 9.807E-02
ZETA =0.0 K = 3.779E+04
POLES AT K GAIN
-9.33E+01 2.16E+02
-9.33E+01 -2.16E+02
-4.48E+01 4.60E+01
-4.48E+01 -4.60E+01
COEFS
ROOT-LOCUS
AMP RESP
LOG MOD
MODULUS
IMPUTE
END
*HARD COPY

Figure 12.

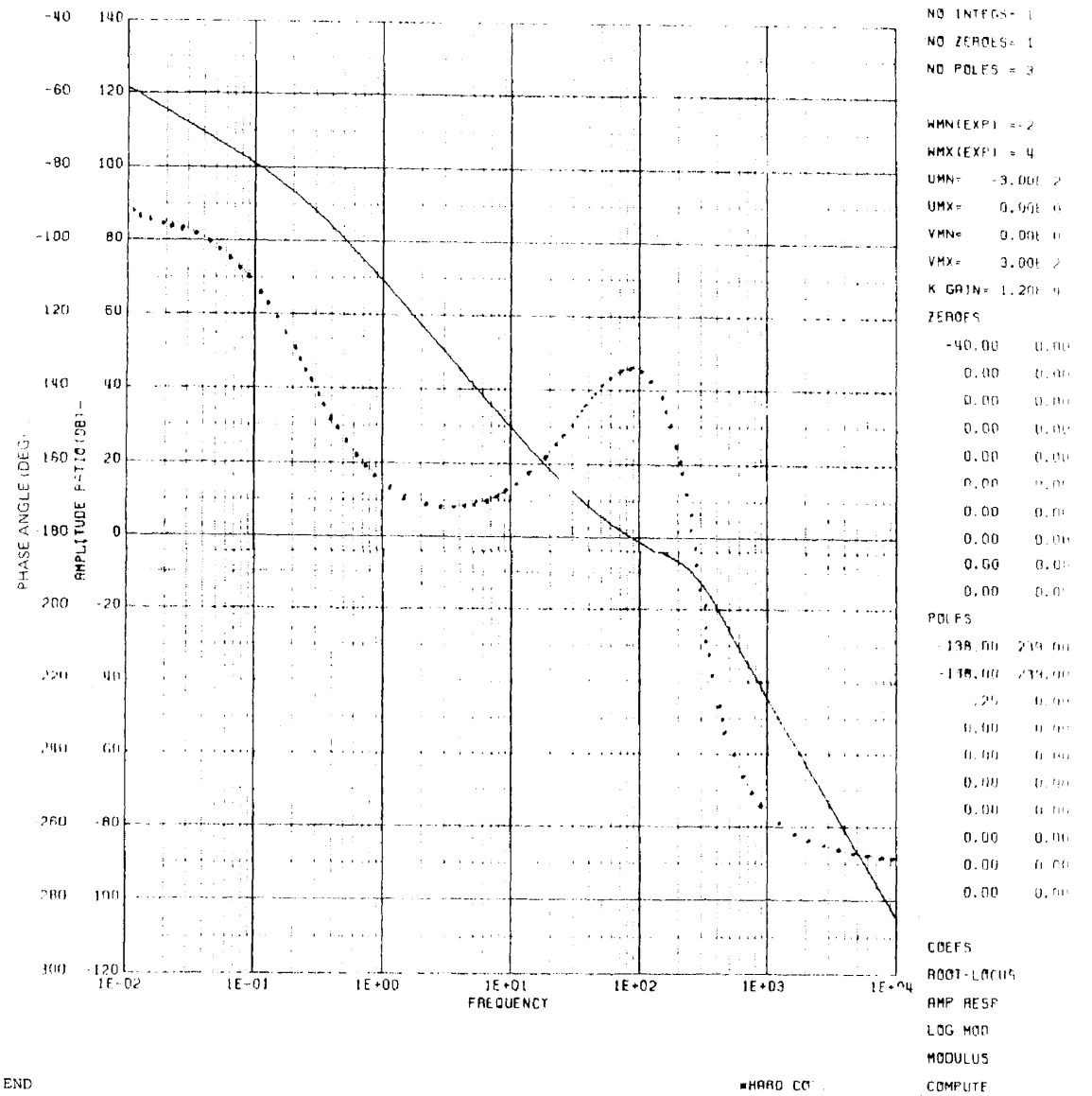


Figure 13. Amplitude response of compensated system

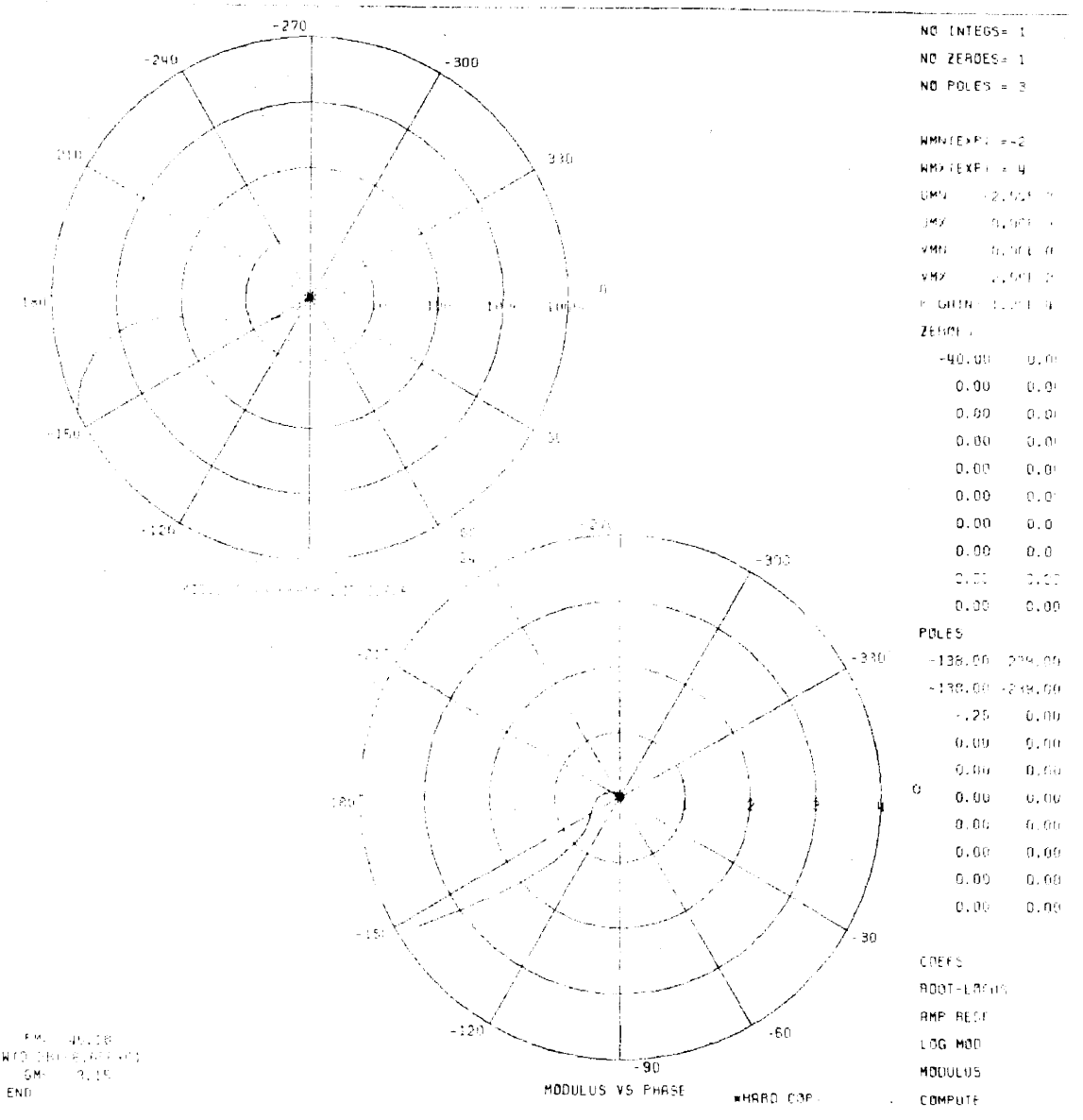

NO INTEGS= 1
NO ZEROES= 1
NO POLES = 3
ZEROS
-40.00 0.0
0.00 0.0
0.00 0.0
0.00 0.0
0.00 0.0
0.00 0.0
0.00 0.0
0.00 0.0
0.00 0.00
POLES
-.25 0.00
COEFS
AMP REQ
LOG MOD
MODULUS
COMPUTE
END
MODULUS VS PHASE

Figure 14.

PM = 46° and GM = 3.15 at crossover frequency of 86.5 rad/sec. Thus the system is adequately stable for the value of gain and compensation we have selected.

It may be of interest to view the calculated closed-loop amplitude response of the system. With reference to the closed-loop expression:

$$G_{c\ell}(s) = \frac{\frac{1}{Js}}{1 + \frac{K\left(\frac{s}{40}\right) + 1}{s\left(\frac{s}{138+j239} + 1\right)\left(\frac{s}{138-j239} + 1\right)\left(\frac{s}{0.25} - 1\right)}}$$

$$= \frac{\frac{1}{J}\left(\frac{s}{138+j239} + 1\right)\left(\frac{s}{138-j239} + 1\right)\left(\frac{s}{0.25} + 1\right)}{CE}$$

We see that all we need is the poles of $G_{c\ell}(s)$, which are the roots of the CE at the selected open-loop gain value. These roots are listed on the root locus display near the heading: POLES AT K GAIN. Furthermore, we see that the zeroes of $G_{c\ell}(s)$ are simply the open-loop feedback poles.

Therefore all that is required is to input these values of poles and zeroes and select AMP-RESP, and the closed loop amplitude response as shown in Figure 15 will be displayed.

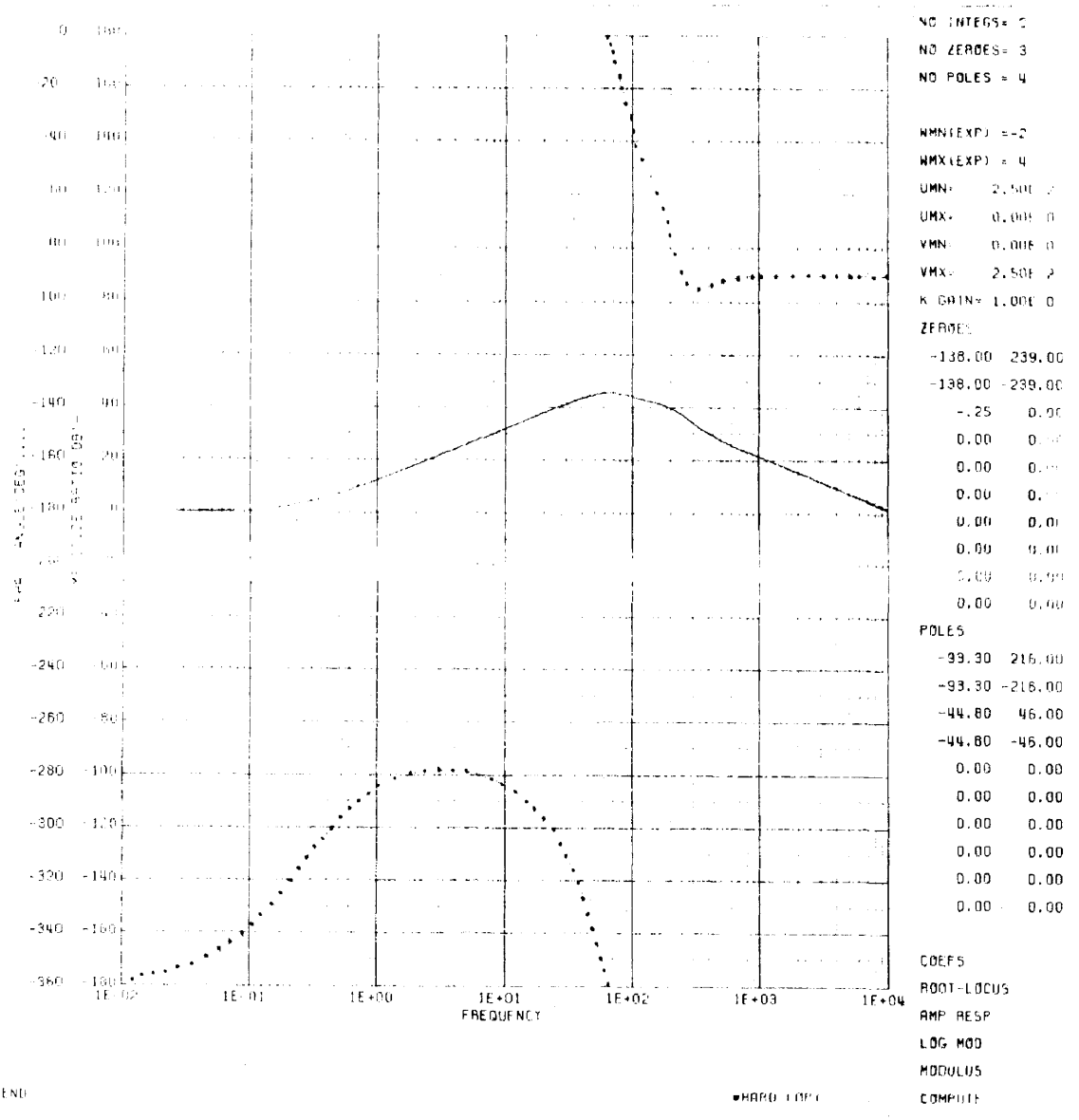

NO INTEGS= 2
NO ZEROES= 3
NO POLES = 4
WMN(EXP) =-2
WMX(EXP) = 4
UMN= 2.500E-2
UMX= 0.000E 0
VMN= 0.000E 0
VMX= 2.500E 2
K GAIN= 1.000E 0
ZEROES
-138.00 239.00
-138.00 -239.00
-.25 0.00
0.00 0.00
0.00 0.00
0.00 0.00
0.00 0.00
0.00 0.00
0.00 0.00
POLES
-99.30 216.00
-99.30 -216.00
-44.80 46.00
-44.80 -46.00
0.00 0.00
0.00 0.00
0.00 0.00
0.00 0.00
0.00 0.00
0.00 0.00
1E-02
1E-01
1E+00
1E+01
1E+02
1E+03
1E+04
FREQUENCY
END
COEFS
ROOT-LOCUS
AMP RESP
LOG MOD
MODULUS
COMPUTE

Figure 15.

References

1. Swartz and Friedland, Linear Systems, McGraw-Hill, N. Y. (1965).

2. N. Horton, J. Long, H. Sumlin, and R. Young, Sandia Interactive Graphics System Applications Manual, SLA-73-0953, Sandia Laboratories, Albuquerque, New Mexico.

3. D. C. Jones, Memorandum of Record, dated
   February 21, 1973 General Stability Analyses Program
   June 20, 1972 Root-Locus Plot Program
   December 7, 1972 Bode Plot Program

4. Eveleigh, Virgil W. Introduction to Control Systems Design, McGraw-Hill, N. Y. (1972) Chapter 9.

5. B. J. Wimber, Development of a Two-Axis Inertial Altitude Reference Assembly (TIARA), SC-DR-72 0779, Sandia Laboratories, Albuquerque, New Mexico.

6. N. Horton, DGRAPH - 6600 Graph Plotting Package for the Vector General Display, SLA-73-0952, Sandia Laboratories, Albuquerque, New Mexico.

APPLICATION OF PEPR
IN MEDICAL RESEARCH

I. A. Pless, B. Wadsworth, D. Zahniser,
Massachusetts Institute of Technology

(Paper not received in time for inclusion in the Proceedings)

Crysnet - A Network of Intelligent Remote Graphics Terminals*

by H. J. Bernstein, L. C. Andrews, H. M. Berman†, F. C. Bernstein, G. H. Campbell, H. L. Carrell†, H. B. Chiang, W. C. Hamilton‡, D. D. Jones, D. Klunk§, T. F. Koetzle, E. F. Meyer§, C. N. Morimoto§, S. S. Sevian, R. K. Stodola†, M. M. Strongson, and T. V. Willoughby**

Brookhaven National Laboratory, Upton, New York 11973

---

*Work performed under the auspices of the U. S. Atomic Energy Commission and supported by the National Science Foundation under contract AG-370 and GJ-33248X, and in part supported by U. S. Public Health Service Grants CA10925 and RR05539 from the National Institutes of Health.

†Institute for Cancer Research, Philadelphia, Pennsylvania 19111.

‡Deceased.

§Department of Biochemistry and Biophysics, Texas A & M University, College Station, Texas 77843.

**Department of Biophysics, University of Leeds, England.

1. Introduction

We will describe a group of intelligent remote graphics terminals forming a network for crystallographic computing. The terminals provide high resolution interactive graphics with batch mode access to the central facility via dial-up voice grade lines. They are used for the determination of molecular structures from X-ray and neutron diffraction data. There are three such terminals at present, one at Brookhaven National Laboratory, one at the Institute for Cancer Research in Philadelphia, and one at Texas A & M University. Each consists of a DEC PDP11/40 with card reader, line printer/plotter, disk, magnetic tape and Vector General 3D display. Communications are via 2000 baud synchronous, half duplex lines using a CDC mode 4c protocol. The preferred central site is the Brookhaven National Laboratory Central Scientific Computing Facility with two CDC 6600s.

By providing enough computing capability at the remote site, interactive graphics are supported without interactive use of the central site and without high speed dedicated lines. Off the shelf logic costs are now sufficiently low that such a terminal is moderate in cost (well below $100K, $70K in this particular case). As a bonus, the terminal is powerful enough to do a significant portion of the computing usually done at the central site.

Though some new approaches are being tried in the development of these terminals, the main thrust is not to create something at the frontiers of graphics. Rather we seek to apply the fine basic work and applications designs of others to build a useful tool for the working crystallographer, to conserve his time, and help make him more effective. We have used only

commonly available hardware, have kept the software as portable as possible, and have concentrated on working down from the needs of crystallographers, rather than up from all possible features. We have borrowed freely from the ideas of Levinthal (1) and Katz and Levinthal (2), but hope to provide similar facilities at much lower cost. For information on other efforts in this area, we would suggest the content and bibliography of Van Dam and Stabler (3), Newton (4) and Raub (5). Newman and Sproull (6) should be consulted for more basic detail.

This effort was started by W. C. Hamilton and E. F. Meyer and is being carried forward by T. F. Koetzle and E. F. Meyer under NSF contracts now entering their third year. There are many facets to the project, but here we will consider it from the graphics terminal point of view.

2. The Nature of Crystallographic Computing

Crystallographers attempt to determine molecular structures from diffraction patterns produced by X-rays or neutrons scattered by crystallized forms of the molecules under study (7). The raw data consists of diffraction intensities, and reasonably accurate information on the number and types of atoms involved in the scattering. One tries to take this data and infer a conformation of atoms which is correct. This is a non-trivial task for several reasons.

1. The data does not, in general, determine the structure, since only intensities are collected and phases would be required to analytically obtain scattering densities.
2. The crystals are imperfect, and often deteriorate in the course of data collection.

Thus the crystallographer makes an iterative use of computers, fitting models to his data, looking at the nature of the errors, and correcting until he has a chemically sensible structure. This is called refinement.

The basic needs are for

1. A means of submitting programs and data to a reasonably large computing facility to reduce raw data and fit models.
2. Facilities to obtain information about known structures to use in piecing together models.
3. Graphic facilities to look at tentative and final models, or look at forms of the data which have physical significance, and to prepare final results for publication.

These needs can be met by an in-house computer or standard remote batch facilities, by a good library, and by ball and stick models and paper plotters. Data tapes can be mailed.

A person in computing is tempted to say that all one need do is add graphics capabilities and information retrieval programs to some remote system, hand it to the crystallographer and leave. As E. F. Meyer has pointed out (8), it is not so simple.

Half the problem is psychological, and the rest is financial. One must provide a system that the crystallographer is willing to use. One must provide a system he can afford to use.

Let us consider the financial question first. An ordinary remote batch terminal costs between $20K and $60K. Graphics capabilities add between $10K and $50K to this price. The money will be there only if adding those capabilities reduces some other costs by about the same amount.

In our case the savings are there. Perhaps the same system in another field would not have such an effect, but for crystallographic work we benefit from the absolute time savings of being able to look at a tentative configuration in many different orientations in a few minutes rather than having to spend hours with ball and stick or to spend days waiting for plot output to be mailed, from the ability to compare images and density maps visually, which is both cheaper and faster than using a computer, and from the ability to use the computing capabilities of the terminal which were mandated by the graphics requirement for general problems not requiring a powerful cpu.

This last is likely to be a sore point with many in the field, especially those devoted to timesharing or running large central sites. Van Dam and Stabler (3) seem to think such use strewn with pitfalls of software incompatibilities and misuse of hardware. Our experience has been that most small I/Ø bound FØRTRAN programs can be usefully run on the terminal rather than at the central site, saving time, phone bills, and aggravation when the central site is down or overloaded. Thus, given a set of coordinates, such data manipulations as computing bond distances, angles, best fit planes, etc. are well done on the terminal.

The psychological problems are tougher to meet. Crystallographers use computers, but few of them like having to read manuals while they lunch. The terminal must be easy to use by an unsophisticated user and forgiving of mistakes. The terminal must be stable and reliable, so that the crystallographer can spend his time doing crystallography, not soldering, or debugging. For this reason it is important that the components of a terminal be standard, easily maintained, and likely to have spare parts available.

3. The Crysnet Terminals and Software

The terminals are as in Figure 1, with sufficient hardware to run a disk operating system. The display has 3D rotation and depth queuing capabilities, so that a model may be seen in three dimensions, either by showing a stereo pair, or by using motion and dimming to give the impression of depth. Further the hardware is well suited to providing independent motion for different images so that we will be able to dock molecular fragments.

The terminal software matches the task. There is a communications package to send jobs to the central site and to return listings and files for display. There is a display program which accepts coordinate information in any of the formats common to crystallography and which displays the corresponding model with facilities to introduce new bonds and inquire about the geometry. A base of data on protein structures is maintained at the central site with retrieval programs. The existing central site crystallographic programs have been modified to produce outputs better suited to remote use. These three components, communications, molecular display, and remote oriented central programs, were the basic need.

In addition we are providing utilities to move display images and paper plotter plot files to the printer. This last is being done from incremental plot files, which are converted to vector oriented commands at the central site and then shipped over the phone lines. At present, a plot of the complexity of Figure 2 takes about 3 minutes to transmit. This speed is not great, but is a significant improvement over the 24 hours needed to obtain a plot by mail. The algorithm used in reducing the many small vectors to larger vectors, which we term resolution reduction, is expected also to be of value in achieving flicker free display of complex

proteins. Experiments conducted by P. A. Wilson (9) have indicated that for this display angles flatter than 176° may always be removed.

In the remaining year of the project we will be providing display capabilities for electron density maps, other 3-D contour display capabilities, and more flexibility in the molecular display. Efforts will be made to switch to a multi-programming mode to allow overlap between communications and display work.

Most of the code is written in FØRTRAN, calling assembly language subroutines for special features. The lack of a special display language has not been felt and this pedestrian approach has allowed crystallographers, who tend to be FØRTRAN programmers, to write programs for the display themselves. Further it is reasonably likely that the same programs can be adapted for use with other display systems, and we are able to borrow static and dynamic display programs from other systems.

4. Experience and Conclusions

We have been able to use the terminals as they were intended for only a few months as of this writing, but it has become clear that a reasonably priced intelligent remote graphics terminal with good stand-alone computing capabilities can conserve a working scientist's time and dollars. On several occassions, the ability to take arbitrarily formatted coordinate information, display it, and manipulate it, has saved hours or days of time. The stand-alone computing capability has shown its value in converting what would normally be idle time on the terminal to a reduction in central site computing use.

Our experience with the communications portions of the system has indicated some need for faster speeds than 2000 baud, and lower overhead protocols. Coordinate information is not a major problem in most cases, but display buffer loads,and contour data can be painfully slow. Fortunately, the terminal computing capabilities reduce the need to transmit such data often, but designers of future systems would do well to consider 4800 baud modems and/or binary protocols with small core requirements, as well as data compression techniques.

References

1. Levinthal, C., "Molecular Model-Building by Computer", Scientific American 214, 42-52 (1966).
2. Katz, L. and Levinthal, C., "CHEMGRAF - A Computer System for Three-Dimensional Molecular Structure Studies", draft publication by Graffidi Laboratories, Dept. of Biological Sciences, Columbia Univ., N. Y., 1971.
3. Van Dam, A. and Stabler, G. M., "Intelligent Satellites for Interactive Graphics", AFIPS Conference Proceedings, 42, NCC 1973, pp 229-238.
4. Newton, C. M., "Graphics in Medicine and Biology", AFIPS Conference Proceedings, 42, NCC 1973, pp 639-642.
5. Raub, W. F., "Automated Information-Handling in Pharmacology Research", AFIPS Conference Proceedings, 40, SJCC 1972, pp 1157-1165.
6. Newman, W. M. and Sproull, R. F., "Principles of Interactive Computer Graphics", McGraw-Hill, N. Y., 1973, 607 p.
7. Stout, G. H. and Jensen, L. H., "X-ray Structure Determination--A Practical Guide", Macmillan, London, 1968, 467 p.
8. Meyer, E. F., "Interactive Graphics and Remote Computing" in Computational Needs and Resources in Crystallography--Proceedings of a Symposium, Albuquerque, New Mexico, April 8, 1972, National Academy of Sciences, Washington, D. C. 1973, pp 105-108.
9. Wilson, P. A., "Resolution Reduction on the PDP 11", BNL semester student report, fall 1973.

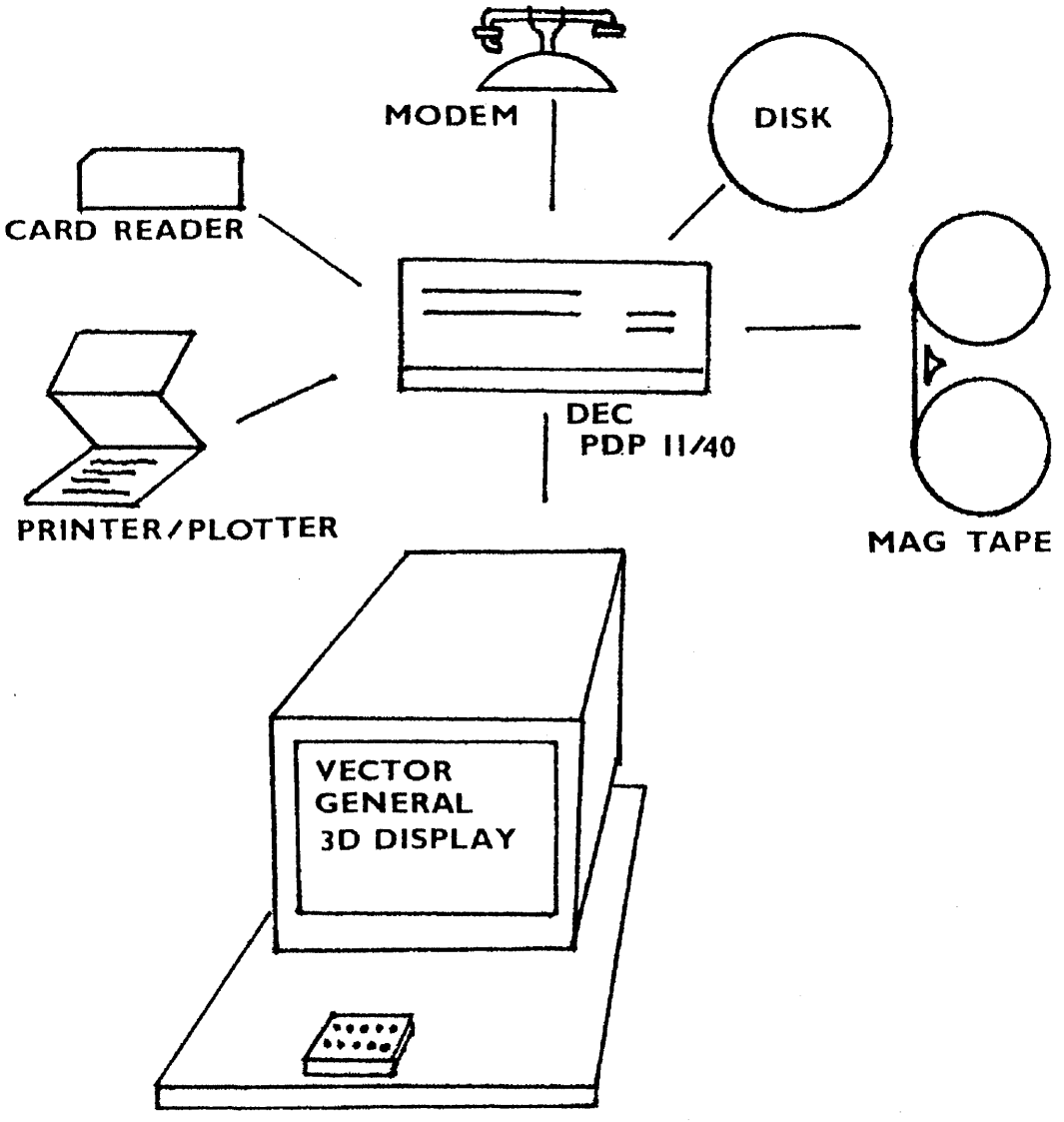


Fig. 1

CRYSNET terminal configuration

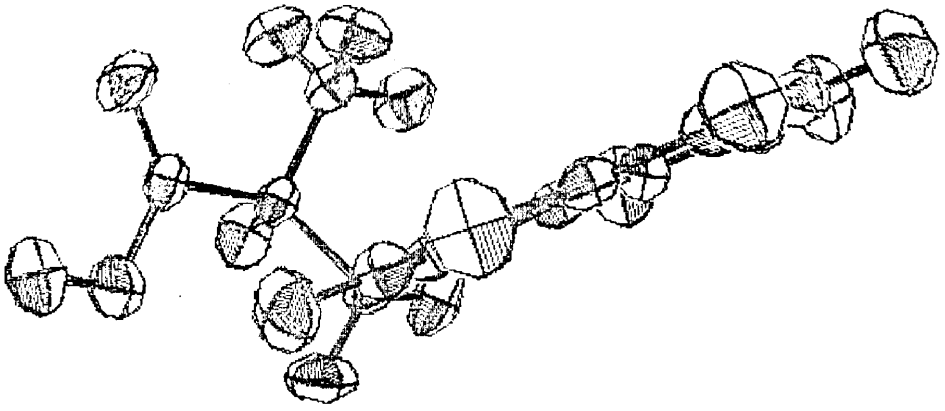


Fig. 2

Converted Calcomp plot transmitted to terminal

SESSION III

General Graphics Facilities

Chairman: A. M. Peskin
Brookhaven National Laboratory

A SET OF DEVICE-INDEPENDENT FIRST LEVEL GRAPHICS ROUTINES *

by Nancy A. Storch
Lawrence Livermore Laboratory, University of California
Livermore, California 94550

ABSTRACT

This paper describes TV80, a set of graphics routines used as a language to plot points, lines and characters by Fortran applications programs. It is a device-independent version of a set of routines long used at Lawrence Livermore Laboratory. Whereas the old routines were limited to one device, TV80 allows access to a number of devices through independent software processors. Features include scaling, clipping, scan conversion and character generation for raster devices, identification of output, handling of on-line display hardware, and generation of display commands.

BACKGROUND

Since 1964, the major portion of the computer graphics produced by scientific programs at Lawrence Livermore Laboratory has been done with a single set of Fortran subroutines. These were originally designed to generate display code for two Data Display Inc. high speed CRT's, called DD80's [1][2]. The CRT's image was recorded on 35mm film. The major use of the film was to obtain hard copies from the Xerox Copyflo. Some of it was used for movies. One DD80 was driven by a batch system IBM 7094 and the other by a time-shared CDC 6600. The Fortran subroutines were used to produce graphs, contours, three-dimensional isoplots, histograms, flowcharts and reports. Although they were simple routines which accepted data in the user's coordinate system, drew a grid, and plotted series of lines, points or characters, they were versatile enough for most user's needs. Files containing DD80 drawing commands were put on tapes and later plotted at certain times of the day by the operating system. This mode of operation is still in use with output being plotted on either a DD80 or an Information International Inc. FR80.

During the past ten years, our entire work environment has changed considerably and will probably continue to do so at an even faster pace. Most of the Laboratory's work is still done by large hydrodynamic codes which are written primarily in Fortran or assembly languages for faster program execution. Although numeric

---

* This work was performed under the auspices of the United States Atomic Energy Commission.

output continues to be standard procedure, most users want graphical output. Thus we face new demands for interactive graphics editors, color movie capabilities, shaded-tone pictures and command languages for specific graphic applications. Today we have twice as many users (approximately 1500), and our compute power is twenty-one times as great. We operate in an elaborate system of networks and computers to support time-sharing in which most users have both a teletypewriter and a television monitor in their office (which they use in combination as an interactive terminal), and a nearby remote job entry/exit terminal. A number of different graphics output devices are also available, some more recently acquired, some have been around a while; among these is an upgraded DD80 with color, grey level and raster capabilities. Figure 1 shows the different graphics choices available to a code running in one of our network "host" computers, a CDC 7600.

When raster devices were acquired, programs were written which scan converted previously made DD80 files [3][4]. The FR80 was made to simulate the DD80 and accepted DD80 tapes. We did not have the capability to use features of the FR80 which were not compatable with the DD80; these included a larger address space, control over intensity, spot size, character size, character rotation and arc drawing. Also a number of routines similar to the DD80 routines but different, had come about to produce plotting tapes for CalComp and Gerber plotters [5][6]; these forced users desiring a choice of outputs to include repetitious coding for each plotting package.

TV80 DEVELOPED

Therefore to satisfy the users requirement of easily using different devices in a like manner, and to more fully utilize the capabilities of the devices, we produced a totally new set of device-independent routines, called TV80 [7]. These routines could easily be integrated into existing codes because TV80 would contain a subset of routines which were identical in name and argument list to the old routines. TV80 would also serve as a foundation for higher level graphics packages being developed simultaneously [8][9]. In general, TV80 routines are not only as fast as the routines they are intended to replace but also they are easy to maintain and modify and can be expanded to control additional devices. TV80 consists of (1) device-independent routines that define display windows, establish mappings, draw lines, characters and points, and (2) software processors for different graphics output devices and data formats. TV80 can also be used to make a general-purpose picture description file.

HOW TV80 WORKS

Figures 2 and 3 show the relationship of TV80 to the user's code and graphics devices. A processor is initialized when a call is made from the user's program to a special "ID" routine, e. g. DD80ID. (The inclusion of this call in a program causes the entire DD80 processor to be loaded with the user's code from a graphics library.) If the initialization process for this device was successful, the processor sets its status to "active" and stores pointers to itself in the jump table used by TVSWCH, a routine in TV80. Initialization for an offline device includes creating output files and constructing an identification page which contains the time, date, machine, security labeling and distribution of hardcopy or film. Online devices are reserved by the ID routine for the current user by sending a request to either a remote system or the host operating system. For classified data, a procedure is followed whereby the system verifies that the user is at a certain monitor by having the user type in a code which has been displayed on the monitor. Processors also create temporary working files. Error recovery techniques are used depending on the output device selected.

Except for initialization, release, and special device or data base features, the running of the separate processors is controlled by the routine TVSWCH. The value of a specified task identification number is used by TVSWCH as an index into a jump table which locates those portions of each processor which handle the designated task. Tasks currently being used are: position, draw points, draw lines between pairs of points, draw lines connecting points, draw characters, output picture or buffer, and advance frame. Most of these tasks have their counterparts in the device-independent section of TV80. Individual processors are called in turn to handle the task and when finished to return control to TVSWCH. Those processors not currently "active" are skipped. No intermediate data base is constructed. Integer arrays of coordinate values are passed in common blocks. The arrays have already been scaled and clipped by TV80's device-independent routines. Character strings and information about size, orientation and intensity are also passed in common blocks.

While a code is running, a call to GSTAT, a device-independent routine in TV80, may change the status of a particular processor from active to inactive or disconnected; or from inactive to active or disconnected. Once disconnected, a processor can only be activated again by a call to its initialization routine. Disconnection severs all program ties with that processor's device and/or data files. At that time files may be given to the operating system for later plotting or printing. Inactive status is useful when you wish only to temporarily turn off output to some processor.

Although almost any number of device processors may be active at a time, users are generally limited in that the choice of where pictures are to be displayed or printed must be made before the picture is generated. An exception is a hardcopy feature which will send a television bit buffer after it has been viewed to a raster printer. We are currently working on a processor to make a general-purpose picture description file. This will make possible the saving of a device-independent representation for later use by either movie or text editors, at which time the decision may be made to obtain hardcopy from some device.

TV80 is flexible because program coordinate data may be floating-point variables in a program-defined coordinate system or they may be integers on an imaginary square display recognizable by TV80. In the latter case, all clipping and transformation operations are bypassed when the data is sent to the processors. The program may turn clipping on or off. Although each processor has its own buffers, these can be reassigned by the user's program. Depending upon the program, more than one processor may be made to use the same buffer; or buffers may be dynamically allocated and deallocated. A program can increase the size of a processor's data file when necessary.

Conclusion: A situation existed where we found we had many different types of graphics devices supported by different software. To better utilize what we had, we developed a package of routines similar to those already in use with the flexibility of having independent software processors. We are continuing our work on TV80.

REFERENCES

[1] Cecil, A. and Michael, G. DD80 Programmer's Manual, Lawrence Livermore Laboratory, Rept. N 2.8-002 (1964).

[2] Ford, J. and Welsh, M. CRT Plotting Routines in Use at LRL-Livermore, Lawrence Livermore Laboratory, UCRL-14427-T (1965).

[3] Keller, P. DDTV-to selectively scan frames of DD80 files via TMDS, Lawrence Livermore Laboratory, UR-412 (1971).

[4] Storch, N. and Fuss, D. New Routines PCNTLR and DCNTLR on G-Machine for Quick Hardcopy of Printer and DD80 Files, Lawrence Livermore Laboratory, Octopus Communique-605 (1973).

[5] Michael, G., Van Deweker, H. and Hunt, C. Use of CalComp Plotters, Lawrence Livermore Laboratory, UCRL-14834 (1966).

[6] Schwarz, R. A New Gerber Package for the 6600, Lawrence Livermore Laboratory (1970).

[7] Storch, N. TV80: Device-Independent Graphics Routines for the CDC 7600 Computer, Lawrence Livermore Laboratory, Working Paper UCIR-748 (1974).

[8] Archuleta, M. Hidden Surface Processing, Lawrence Livermore Laboratory, UCID-30057 (1973).

[9] Archuleta, M. Interactive Surface Plotting, Lawrence Livermore Laboratory, UCID-30058 (1973).

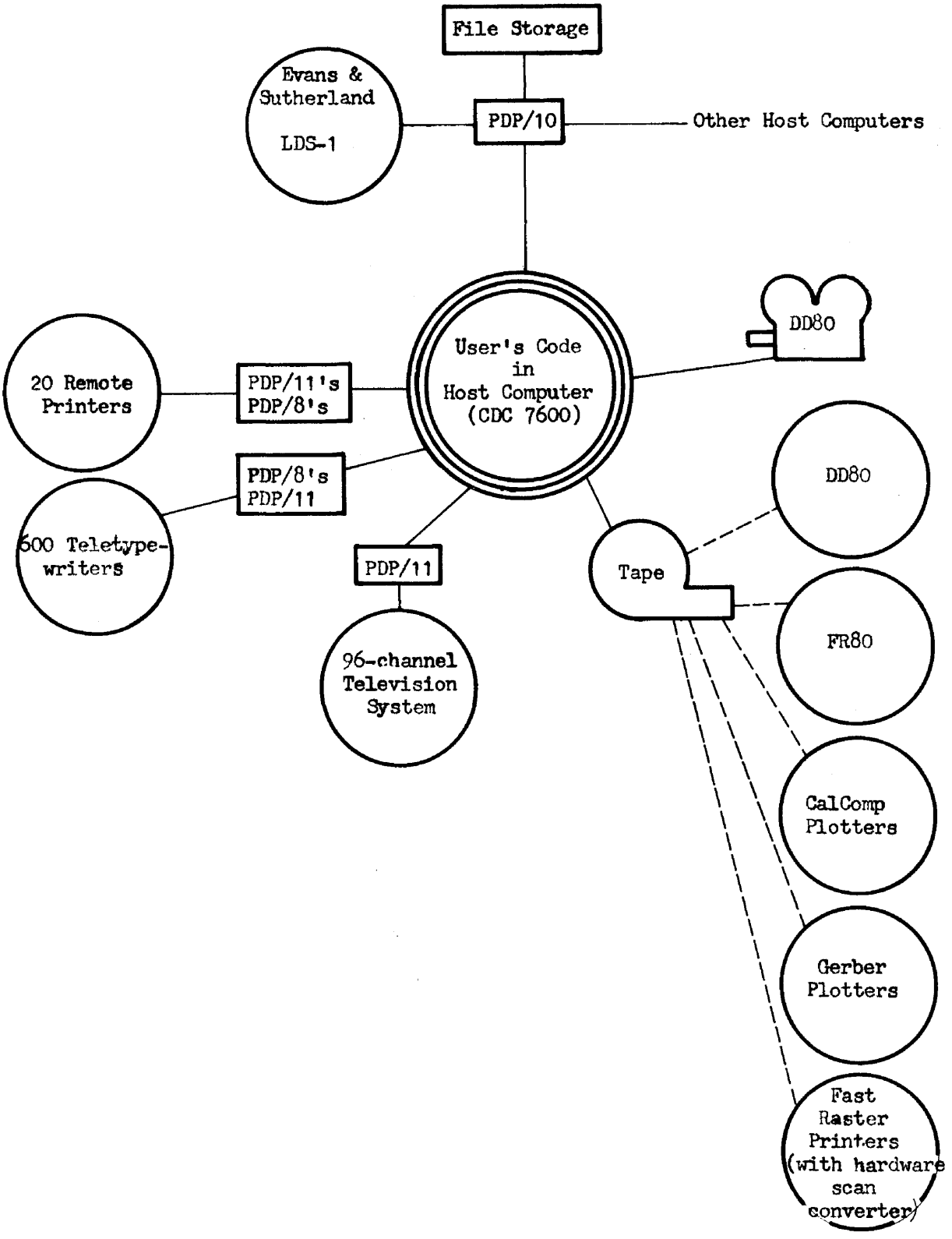


Fig. 1. Graphics devices available to user's code.

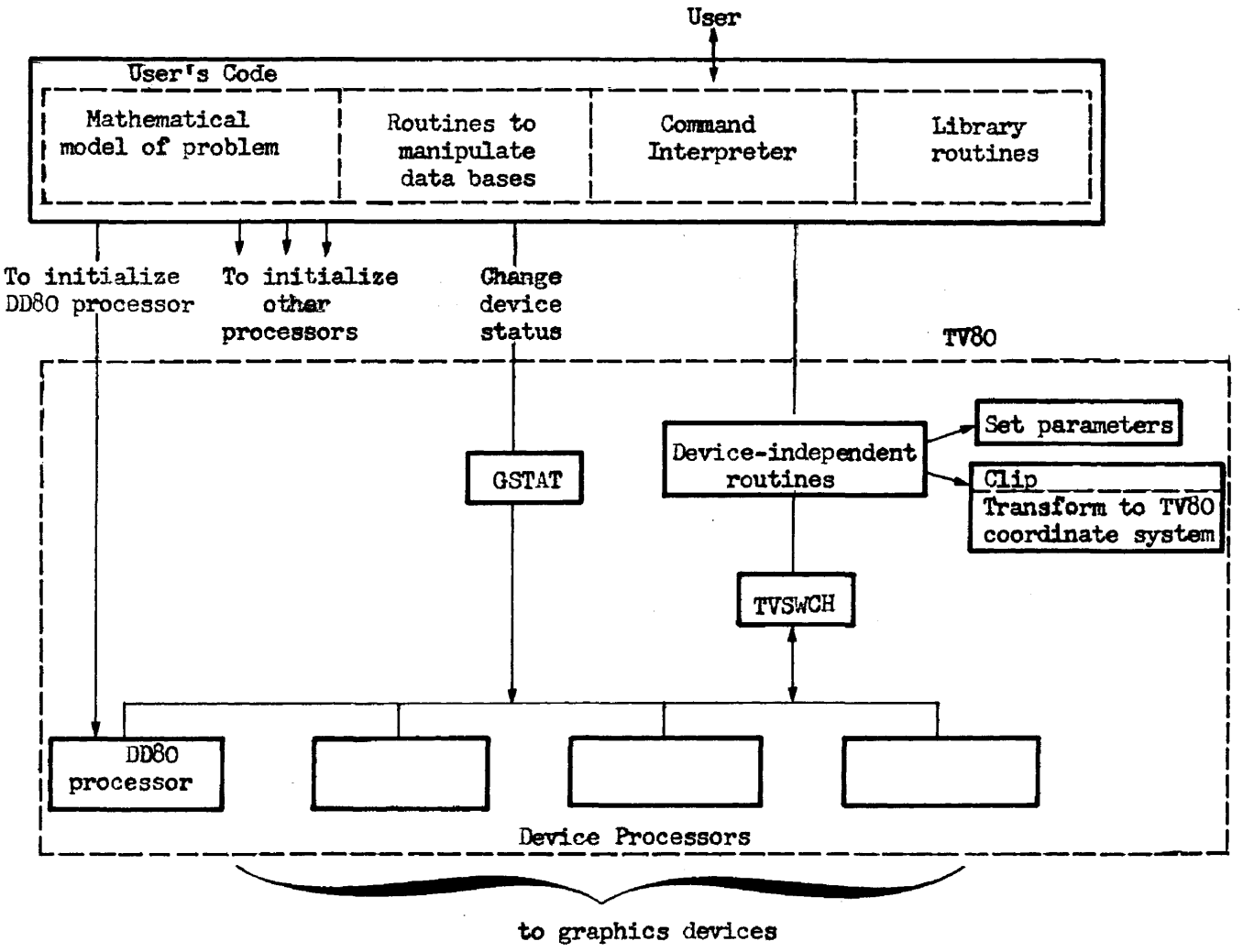


Fig. 2. Relationship of TV80 to the user's code and graphics devices.

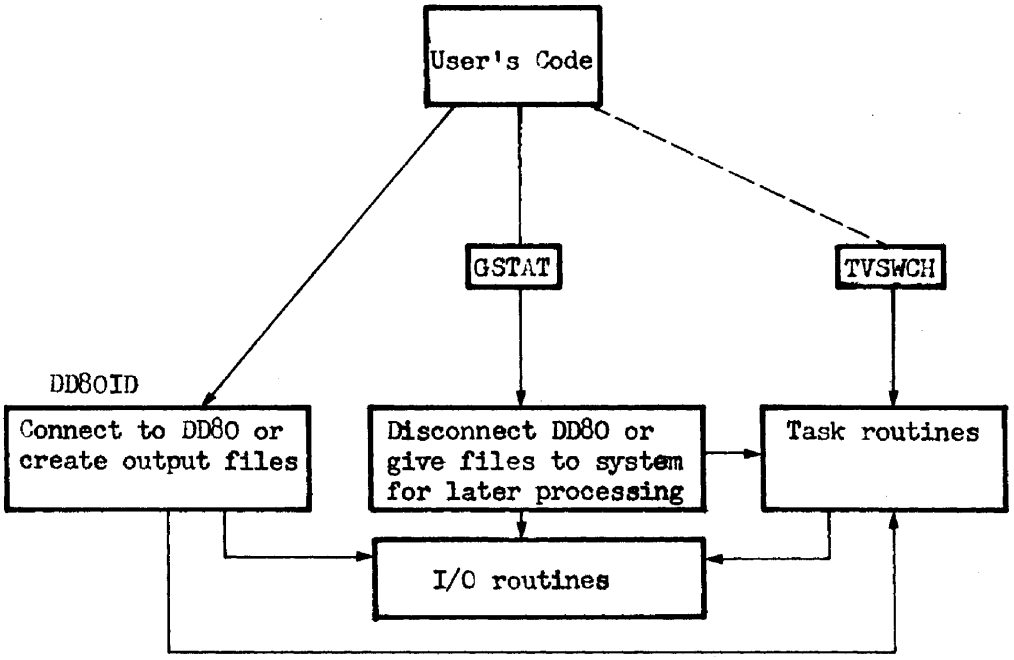


Fig. 3. The DD80 processor.

GRAIL - A Graphical Device-Independent Picture Description System

J. A. Brooking

February 8, 1974

Introduction

The absence of a definitive, accepted standard for graphical data structures imposes severe restrictions and penalties on installations which deal in any substantial way with data which represents pictures, and with programs to generate these picture-defining data.

A restriction appears when the installation considers upgrading existing graphical output facilities. Consider the case of an installation with a heavy investment in application software which generates graphical data to be realized on an existing plotter or Computer Output Microfilm (COM) device. When the capacity of this existing equipment becomes exceeded by the workload (either through increased workload or decreased capacity caused by equipment deterioration), the installation is restricted (by reprogramming costs) to consider only those vendors which offer, at a minimum, graphical subroutines with calling sequences essentially identical to those which are available for the obsolete equipment. Ideally, the picture describing code generated by the existing programs should also be compatible (upward at least) with that of the newer device. The minimum requirement above is generally satisfied by the same vendor (which is to say a given vendor will most likely supply compatible subroutines for his entire product line). The ideal is almost never satisfied. The consequence of this restriction is that those who have invested in a large body of graphical-oriented code are prevented from any meaningful competitive bidding procedure and so must either make a sole-source procurement or commit themselves to a reprogramming effort, the cost of which obliterates any differences in prices quoted for competitive equipment.

A penalty is imposed on an installation which is unable to service its graphical needs by only one type of device. An example of this is the installation which requires COM equipment for high volume archival output, a high precision automatic drafting machine for quality engineering drawings, a fast incremental plotter for general utility graphics, and a low-precision, "quick-look" device such as an electrostatic printer/plotter. Of course, an installation with such a heavy requirement for graphical output will probably have considered interactive graphics, and these will typically also have varieties of subroutine calling sequences and internal picture-description codes.

What penalties are imposed on an installation with such a wealth of graphics devices? No vendor today markets a set of devices to fill all the needs of this installation. (Vendors may, of course, subcontract to original equipment manufacturers for some devices, but this really does not solve the problem, as will be seen). The result of this situation is that the installation must be content to suffer one of the following situations:

1. (More likely). The installation is forced to clutter its subroutine libraries with a different set of routines for every different device it installs. In turn, the customers of the installations are forced to learn a different set of conventions and calling sequences for every device they use.
2. (Less likely). The installation undertakes to maintain subroutines for all devices which are installed, which subroutines appear the same for all devices. When new devices are installed, a new subroutine set will be provided.

Unfortunately, both situations have rather obvious defects, as well as advantages. The advantage of the first is, of course, that it minimizes local maintenance; one

simply installs the vendors' software packages and gives the users the vendors' manuals. Disadvantages: A plethora of subroutines of similar function occupying library space, a group of users who can never remember whether to use XCPLOT or SYMBOL to draw character data on the COM (and if SYMBOL, which calling sequence is correct), and of course, all those routines with the local modification to identify the output as belonging to individual customers must be maintained.

The second alternative affords the users an apparently compatible set of subroutines at a cost of a heavy maintenance load for the installation. The compatibility is only apparent, however, since the subroutine set implemented for a 48" x 96" flatbed drafting machine will rarely be very meaningful when implemented for an 11" incremental plotter. (For example, how does one interpret an 11" move on the pen on the plotter as a 48" move on the drafting machine? Is a 48" move on the drafting machine to be interpreted as an 11" move on the plotter?)

A disadvantage of both conditions is that the selection of graphical output device must be done prior to the time the graphics program is executed. In the first case, selection is done at the time the program is written while in the second case selections may be deferred to load time. In neither case, though, can selections of device be made dependent upon conditions which arise during program execution.

Another disadvantage of both situations is the lack of control on user programs. This disadvantage is particularly evident during debugging when lines of "semi-infinite" length often appear along with 3800 coincidental points, and character strings which wrap around the scope 2^{28} times ("I forgot the number of characters should be an integer, not a real!")

This paper will propose a solution to the graphics problem which will offer the following advantages:

1. A subroutine library which is identical for all devices and suitable for implementation on a variety of computers.
2. A picture-description code which is suitable for realization on any graphical output device.
3. Maximum ease of conversions to new graphics devices, either add-on or replacement.
4. Selection of device at any stage: before, during or after program execution.
5. "Entry-level" programmer knowledge at a minimum, but highly sophisticated techniques available.
6. Abundant error diagnostic features to facilitate program debugging.

The GRAIL System at KAPL

GRAIL is the collective term which is used to describe the conventions for describing pictures, subroutines which implement these conventions, and post-processor programs which transform GRAIL picture description files into device-dependent picture description files. We shall discuss the conventions used to describe pictures in GRAIL, and show how these conventions contribute to realizing GRAIL design objectives.

The elementary building block in GRAIL is the byte of twelve bits length. The basic structural unit in GRAIL picture description is the instruction segment, which consists of two or more contiguous bytes. The first byte defines the type of instruction segment, and determines the syntax of the remainder of the segment.

Structural entities which can be included in instruction segments are parameters, values and characters. Parameters are one byte long, and give environmental and parametic information such as required security markings, line flavor (solid, dotted, etc.) and character encoding mode (display code, ASCII, etc.). Values on the other hand are numbers and represent coordinates on the display surface. There is one distinguishing feature of values: they may (but need not) be comprised of more than one byte. This enables a high degree of coordinate precision to be attained where necessary without the requirement that all coordinates be specified with the higher precision. An instruction segment is defined to specify the number of bytes in a value. Value strings are formed by concatenating more than one value, and are terminated by a value of 7777_8 (with don't care values used as required to pad the terminating value to the required precision).

Characters occupy one byte each, and may be concatenated to form character strings. A character string is defined as a contiguous set of non-null characters, terminated by a null character. (A null character is a character with byte value 000_8.)

The graphic associated with character encoding is determined by a parameter. Graphics so specified are independent of device, and if not specified otherwise, are defaulted to the normal character set of the computer on which the GRAIL file is generated.

The overall structure of a GRAIL file consists of a set of initializing parameter segments, followed by the main body of the file: the picture description, which consists of one or more frames. Initializing parameters include, for example, the identification of the computer on which the file is generated, the identity of the user who is running the job, the date and time the job is run, the frame size and the precision of values. The order of these parameters is important, being used by the realization routines to identify the GRAIL file as such, and so no initialization parameters are written on the GRAIL file until the user performs the first

non-parametric picture-describing subroutine call. This enables him to specify any non-default parameter settings in any order he chooses.

GRAIL subroutines include capabilities for generating the usual graphical primitives of move, draw, draw symbol, draw broken line polygon, etc. These are described in Appendix A: GRAIL System Description. It may be worthwhile to note that some options are available in GRAIL which are not found in the usual run-of-the-mill graphical subroutine library. Among these are variety of character-drawing parameters such as roman or italic (slanted) font, aspect ratio and inter-character spacing. Line drawing options include line "flavor" (solid, dashed, dotted and end-point). The selection of data representations for line drawing includes vector mode ($X_1,Y_1,X_2,Y_2,\ldots,X_nY_n$), incremental mode ($X_o,Y_o,DX_1,DY_1,\ldots,DX_n,DY_n$), and auto-incremental mode ($X_o,DX_o,Y_o,Y_1,\ldots,Y_n$). Values may be scaled or unscaled for all coordinate representations.

Realization programs are programs which read GRAIL picture description files and generate files which can drive particular devices. At KAPL realization routines are available for CalComp 565 and 763 incremental plotters, a CDC 280 Computer Output Microfilm unit and a Versatec Matrix 1100A electrostatic printer-plotter.

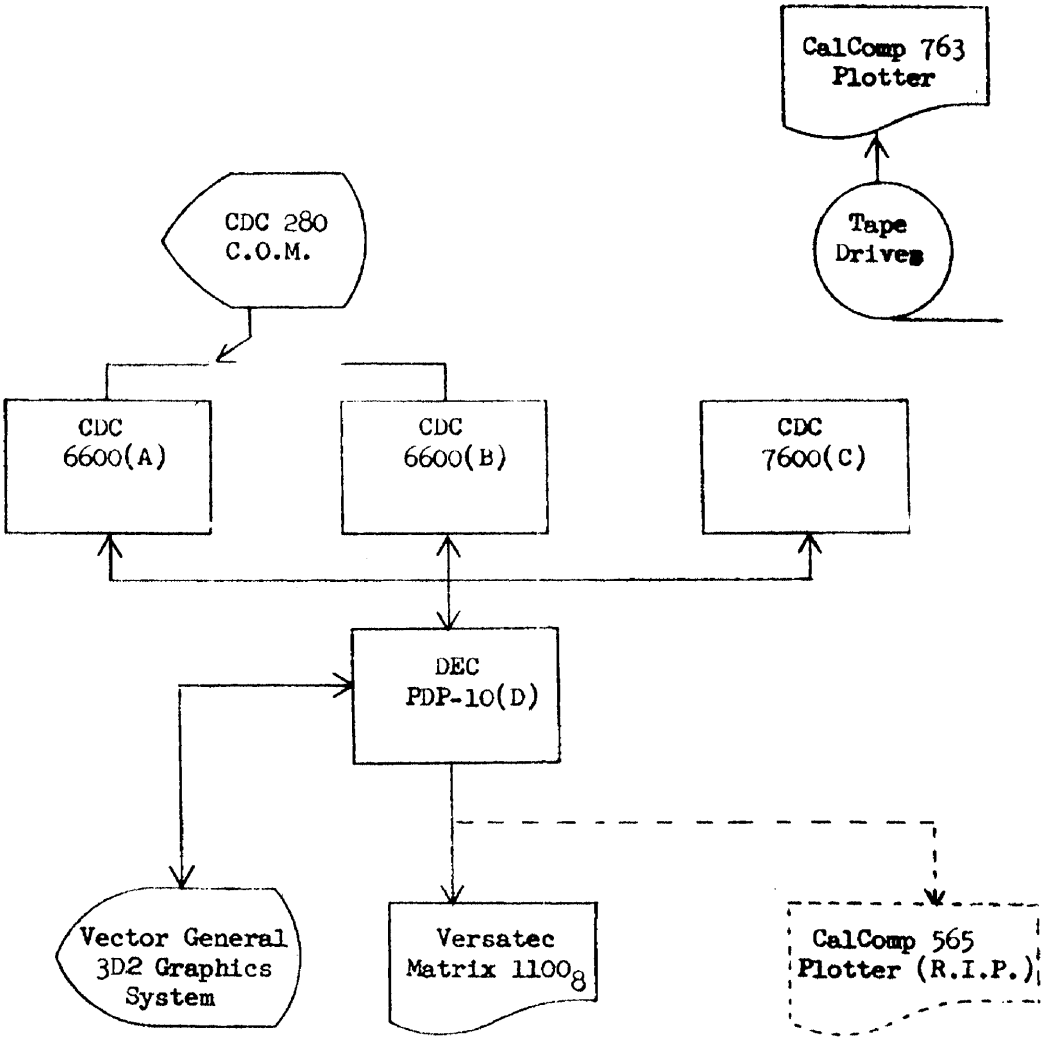
As a parenthetical note, the KAPL computer systems are linked together in various ways. In particular, as seen in Figure 1, KAPL graphics devices are driven directly or indirectly by both CDC 6600's and by the PDP-10. (The 7600 can drive only the off-line CalComp through magnetic tape). Accordingly, jobs executing on any computer in the network are given access to any graphics device in the shop. For example, execution of the control card

GRAVER.

on the 7600 will cause a GRAIL file to be forwarded to the PDP-10 and plotted on the Versatec plotter.

FIGURE 1

KAPL Computer Network and Graphics Devices



In the nature of a conversion note, GRAIL went into production in July of 1973, at the time our Versatec plotter was installed. The facts that (a) GRAIL was the only language by which the Versatec could be used, and (b) turnaround on the Versatec from all four computers was almost instantaneous, gave KAPL users strong motivation for using GRAIL subroutines.

GRAIL Features which Contribute to Design Objectives

Given this brief introduction to GRAIL, what features in GRAIL contribute to the realization of GRAIL design objectives? First of all, device independence is yielded by three primary attributes of GRAIL: (1) the inclusion in all GRAIL files of "environmental data" which provide realization routines with information about the intention of the programmer as he was writing his picture descriptions. (2) The ability to specify multiple precision values as coordinates enable GRAIL files to be realized on any device, while precision is maintained for the most exacting device on which the file will be realized. Thus, for example, if one is creating high precision engineering drawings, he can debug his program using low-precision, quick look devices such as the Versatec plotter. Then, without changing his program (and in fact, without regenerating the GRAIL file) plot it on an automatic drafting machine which recognizes GRAIL files. (3) The last primary attribute of GRAIL which contributes to device independence is the acceptance by GRAIL realization programs of instruction segments which have no meaning for the device on which the picture is being realized. For example, an instruction segment exists to specify intensity of displayed information. If this segment is encountered by the CalComp realization routine, it is simply ignored, since there is only one intensity for CalComp lines: On.

Machine independence is obtained by the use of a standard unit of information: the 12-bit byte, and by the definition of a uniform set of graphics associated with internal character codes. The selection of 12 bits per byte was made because 12 divides both 60 (the CDC word length) and 36 (the PDP-10 word length) evenly. Additionally, the expression of up to 4000 units as a value can represent up to 40 inches while maintaining 0.01 inch precision.

Our third objective of ease of use is served by several features in GRAIL. Probably the most important of these is the inclusion of multitudes of default options. These defaults enable the programmer to avoid the tedious task of specifying everything about his picture before he draws it. For example, the engineer who wants nothing more than to draw a graph of a simple function can do this by calling six subroutines: two for x- and y- scaling, two for x- and y- axes, one to draw the function which has been generated by his program and stored in an array, and last, an end-of-frame subroutine. Since he did not elect to override default options he will generate an eleven-inch square graph, with 0.21 inch character heights, 7:6 character aspect ratios, solid line mode, etc. The "price of admission" to GRAIL in terms of knowledge is, then, small. Should the engineer later decide to plot other curves, or give labels to his graph, or in other ways improve it (in what is called the Creeping Elegance Syndrome) he can do so, simply by reading more sections of the manual.

Another facet of the GRAIL implementation is that error conditions are, for the most part, handled with grace and forgiveness. A labeled common block is used to convey error conditions to programs calling GRAIL subroutines. When an error condition is noted by a GRAIL subroutine (such as a coordinate value which exceeds the frame boundary) a number is placed in one variable of this common block and

the other variable is incremented by one. Careful programmers may declare this common block in their programs and test these variables to determine if something went awry in their GRAIL subroutine calls. Alternatively the programmer may call a subroutine which causes full diagnostic print-outs when errors occur. In any case, if errors in specifications or coordinates do occur, the erroneous value is ignored and the default value used instead.

Interactive Use of GRAIL

At the present time we are formalizing the plan for extending GRAIL for interactive use. This plan is based on a number of premises:

1. The main application programs at KAPL are modular in nature; some modules process input, some do one kind of calculations, some do other kinds of calculations. Still others generate pictures based on input data or on computational results.

2. The computational modules use substantial amounts of 6600 and 7600 computer time, and can not be practically performed on the PDP-10, to which the Vector General graphics system is attached.

3. Interactive Graphics at KAPL will be used for the creation and modification of complex geometrical constructions composed of a variable initial library of simple shapes, any member of which may be translated, rotated, scaled and/or reflected, and added to the construction. Intermediate constructs may be added to the basic library, and any member of the library may be changed at any time, causing all instances of that member in the construction to assume the changes of the library figure.

4. The description of interactive graphics in (3) above is sufficiently broad and general that an interactive package which performs those functions for one application can perform those functions for <u>any</u> application.

With these premises in mind, we are building an application-independent interactive graphics package for our network which shares the workload as appropriate among the various members of the network, and which uses GRAIL files (with enhancements as described below) as the common picture and description media.

The implementation plan we have devised will be to impose an Interactive Module into application programs as they presently exist. Figure 2 illustrates a 6600 or 7600 non-interactive, graphic-oriented application, with several computational modules, each followed by graphical modules. The graphical modules generate GRAIL files which may be the results of the preceding calculation or geometrical representation of input. At job termination the GRAIL files, in a non-interactive environment, will typically be plotted or microfilmed.

Figure 3, shows the imposition of our interactive module which is entered just after the last graphical module has completed. This module will be responsible for getting the GRAIL files previously generated to the PDP-10 and awaiting the results of whatever modification is performed by the user on the Vector General terminal. The response from the PDP-10 will be either a revision of the input to the problem or a termination signal. If a revision is received, the problem input is modified accordingly and the problem is re-entered at the beginning. If a termination is received by the interactive module, the last revision of input is saved and the program exits.

FIGURE 2

Typical Graphic-Oriented Application Program

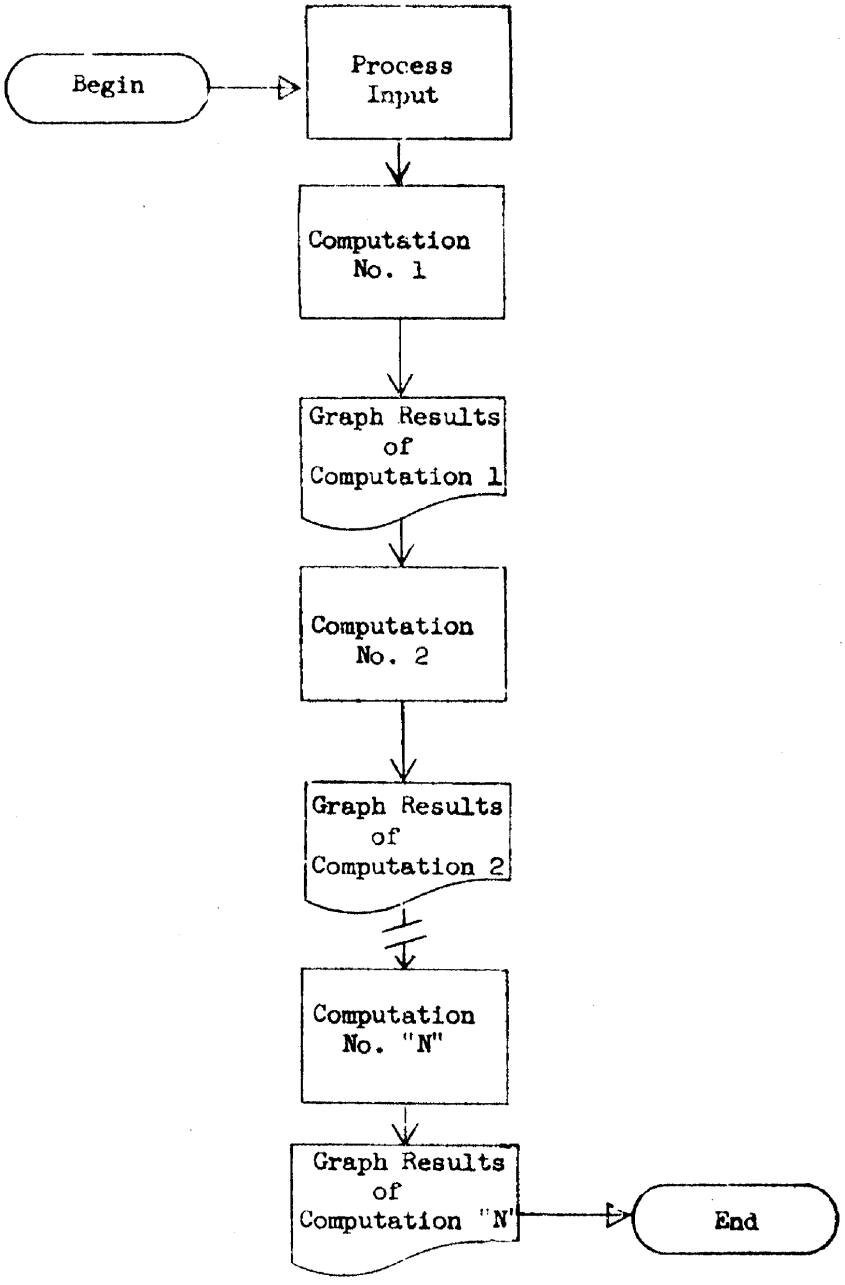
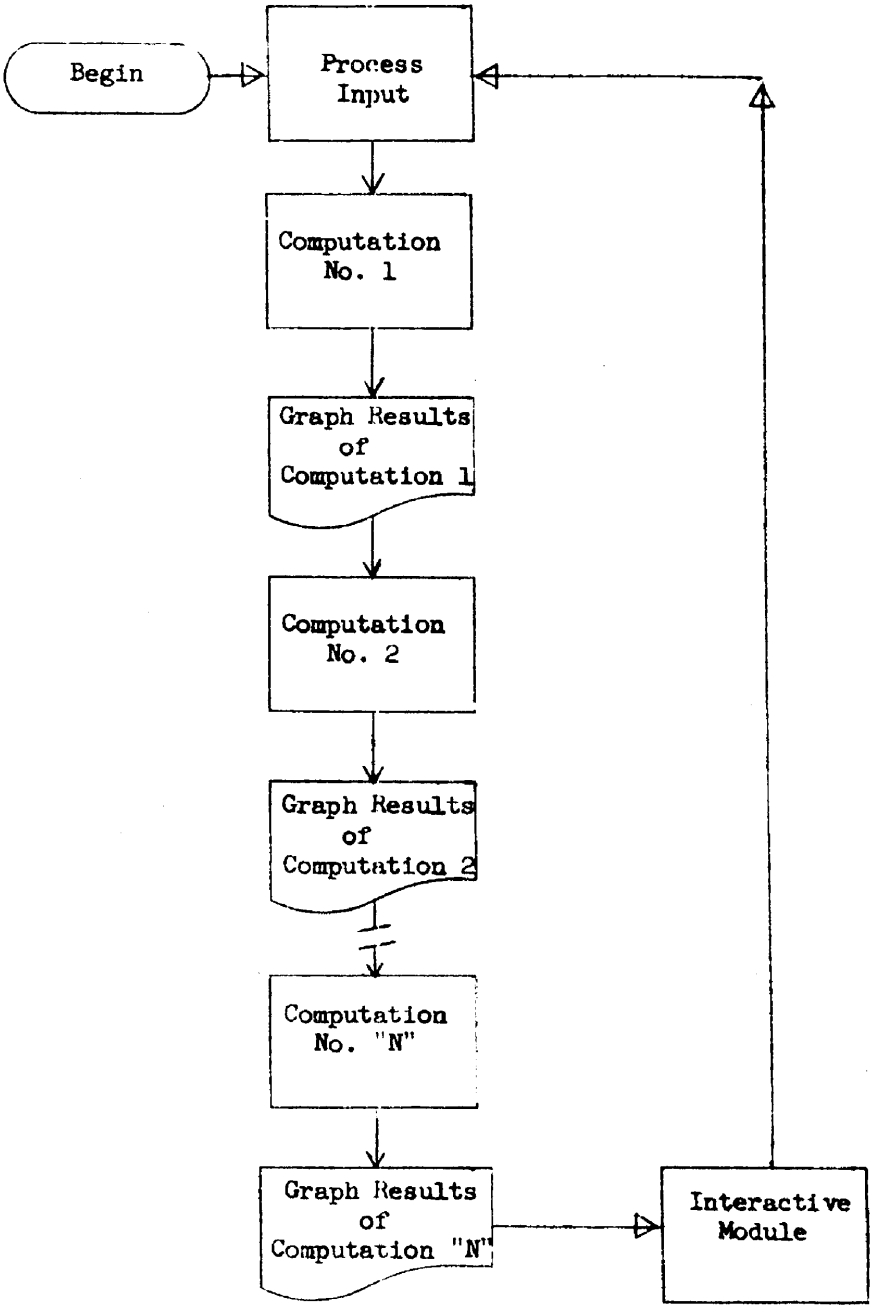


FIGURE 3

Graphic-Oriented Application with Interactive Module



On the PDP-10, a program is executing which implements interactive interpretation of GRAIL files by means of the Vector General graphics system. The user will be able to select any frames in the GRAIL files for display, and one or more for modification. He will be able to perform the kinds of construction described in (3) above using a standard set of display input commands, and thereby modify the geometric model used as input to the CDC application program. Graphs of results generated by computational modules will be displayable to assist him in making decisions about the modification he makes.

GRAIL extensions for interactive graphics, then, will consist of (a) the enrichment of structures expressible in GRAIL file, including graphical subroutining, and (b) the ability to declare variables for use as coordinates, scale factors, and transformation arrays, and the assignment of descriptive strings to these variables.

Finally, the presently available mechanisms for intercomputer file transfer will be used to establish and maintain communication between graphic application programs on the CDC computers and the interactive driver program on the PDP-10.

Systems Programming Languages and Graphics Terminals[1]

Thomas Stuart
Courant Institute of Mathematical Sciences
New York University

Abstract

Faced with the design of an interactive graphics system, many researchers with widely differing applications have settled on a random access CRT display attached to a miniprocessor. In most cases this small computer is in turn linked to a larger time-shared facility, and it is the software implications of this overall hardware design with which this paper is concerned.

Such a setup is usually arrived at by a compromise between an estimate of required computational power and the reality of financial resources. The insertion of the small computer between the display device and the applications oriented large computer causes difficulties in three main areas:

(1) Decreased utility services, compilers, and other system software where it's needed.

(2) A more complex design for the graphics software, due both to an added communications channel and the division of labor between computers.

(3) A degradation of response time when interaction between display and application programs is necessary.

It is asserted here that the use of a systems programming language will aid in tackling each of these problems. With respect to the first, system software, one has an obvious application of the earliest arguments for a systems language. The extension of these arguments to the second difficulty, the design and implementation of a graphics system, receives major attention. Though a choice of programming languages would usually have no clear, direct effect on the third problem, response time, it does have an indirect influence since the response can never be completely independent of communications software. The disadvantages of a systems language are not to be ignored either. Finally, the use of a specific language, LITTLE, is described briefly.

Introduction

As with any new item of hardware, the development of graphics terminals brought a host of problems. Among the various types of terminals, one that has found a wider audience is the programmed beam CRT driven by a computer of relatively limited resources. Usually these resources are too limited for a complete resolution of the specific applications problem and, hence, a more versatile computer is used to handle the larger computational tasks.

At the outset, the advantages of such a design were not readily apparent.[2] Given a number of applications with varying graphical demands and varying computational requirements it was difficult for designers to choose an optimal system even when the system was to be used for a single application and its needs were well defined beforehand. The earliest solutions took one of two forms: if the computational tasks were large, plug the CRT into the large computer; if not, get a mini to drive the display. Neither solution was very practical with most problems.

When the memory and I/O facilities of a large computer are employed to drive the display directly the volume of data transmitted to maintain the image represented a serious drain on, and waste of, the resources available, degrading production in a time sharing system. On the other hand, if the driver is a more or less independent processor with a small memory set by a program in the main memory only as needed, the degradation this time occurs at the terminal, which has real time requirements undreamt of at IBM, CDC, or elsewhere. So long as an image is just being regenerated, the main processor need contribute little time or space to graphics, but whenever the picture changes in some manner not handled by the hardware of the CRT controller, then, within milliseconds, a generally large program must be in central memory and executing. The percentage of the time which must be devoted to this program of course depends on the application, but many problems will vary during the course of a job from a fraction of 1 per cent all the way up to 100 per cent. No time-sharing system which allowed this would be time-shared any longer. Essentially, the only types of interactive graphic display that can profitably be driven by a time-shared computer (with or without an intermediary processor) are displays of a relatively static nature,

alphanumeric displays, storage tubes, point plotters, etc., and we are not concerned with their special problems here. Whether a design lacking degradation on either end might be implemented is problematical, for such a facility would be a costly feature.

The other solution, driving the display from a stand alone minicomputer, was simply inadequate for the computational necessities of most problems, if not initially, then as additional projects were attempted. Expanding applications can outstrip attempts to upgrade performance of the minicomputer with great speed. However, this approach did possess the advantage of adaptability in some cases, for, if a large computer was near enough for a low cost, high volume transmission line, then the computational portion of the graphics problem could be undertaken, and all one had to face was reprogramming the system to reflect the minicomputer's shifted status - from central processor to intelligent terminal. The choice of this latter system, while it may in some cases involve substantial effort to convert, was eventually advantageous for its users, for the choice of a larger computer as driver has led to, or will lead to, the junking of both software and hardware.

The decision to employ this type of interactive graphics terminal - CRT attached to minicomputer attached to main computer - is not an easy one, since response time between applications program and display is degraded, and since it can only be more difficult to develop software, but it is a choice that experience forces, because there is no avoiding the need for the computational power of a large computer, and there are no projects that justify a dedicated mammoth and the concomitant waste. Once that basic choice is made there are still many questions to be answered concerning hardware characteristics of the display, the minicomputer, and the peripherals. Answers are impossible to generalize and will often depend more directly upon the specific application and financial backing. In any case, though these considerations are very important, the remainder of this paper is concerned with the other half of a system design, software.

Software

Software difficulties increase enormously with this terminal design. First, and most obvious, it is because there are two computers instead of one to program. Communication between them is always a non-trivial problem, especially when entering a time-shared community with some real time desires. The software interface between the graphics routines on the mini and graphic processing in the main computer is only one of the added interfaces. There are now application routines both on the time-shared computer and at the terminal, each requiring interfaces with the graphic routines where one sufficed earlier.

Perhaps not so obvious is another effect of the two computer choice, an immediate repercussion being that every time a new routine is written there is a choice to be made: where will it run? The answer to this question is in many cases neither apparent nor trivial. Factors influencing the choice will include -

(1) The memory available at the terminal, both in core and in storage devices,
(2) The access time of these devices,
(3) The speed of the link (or links) to the main computer,
(4) The priority this particular task would be accorded in the main computer at this time,
(5) The frequency of performance of the task for each application, and
(6) The volume of data to be processed during a particular job.

Whereas the first three of these factors tend not to change frequently and do absolutely determine the location of many routines, the latter three are time dependent variables and merit more attention. Of course, we always wish to carry out a process on the minicomputer when possible. Whether or not it is feasible depends not only on the hardware available, but the run-time environment. The fourth, fifth and sixth factors constitute that environment.

The priority of a task coupled with the time of day can sometimes determine the optimum location for executing a routine. Especially would this be true for processes requiring a few seconds on a mini and an order of magnitude less in the time-shared facility. In peak use hours, the mini is still a preferable location, whereas at night it never would be.

The fifth factor, frequency of task performance, becomes important as a determinant only when high. For an infrequently performed task the location is relatively unimportant when a choice is available. When the speed considerations point to the main computer as the prime site, then it still may be desirable to have a back-up routine for avoiding down time impediments. But when the frequency is high, speed assumes large importance and we have a boosting of the importance of the priority factor just cited.

The last effect, the volume of data, is the most unpredictable in its importance. It is here that many graphics applications will envelop a range of specific jobs that include some production runs requiring a large computer and some that do not. An ad hoc decision is usually made to program for the worst case, a decision which is often forced by the difficulty of reprogramming.

Specific cases which will be affected by one or more of these factors are, it should be emphasized, quite dependent on the hardware. That is, the affected routines will change from one system to another, but so long as a graphics project embraces a large number of tasks, there will be many falling in the range of interest.

Now that some of the complexities of the programming have been established, it may be reasonable to summarize, to characterize the problem.

First, the problem is many faceted. It embraces transformation and manipulation routines, access of various utility functions on both computers, an executive, probably some data analysis routines, more than likely a complex data structure.

Second, the program requires a formal structure for interfacing its divisions and also because it is to be written by more than one person.

Third, there is a high emphasis on the speed of execution, both because there are some real time criteria, and because it is designed for continuous or frequent production.

Fourth, it is likely to change frequently with changing needs, priorities, applications, and hardware.

Fifth, since it not only serves as a service program for large applications, but also carries out simple display problems as a stand along package, it must be supremely easy to use.

Sixth, some of the routines or tasks are to be written for two machines to best advantage.

In many interactive graphics projects a seventh characteristic would also be important, hardware independence. This takes either of two forms, an extension of the sixth characteristic to the whole of the graphics program because the package is to execute on two, different minicomputers, or, alternatively, some routines are to produce images on two, different display devices.

Then, it is appropriate to ask, what is the vehicle through which this problem is to be implemented, how shall it be programmed, in other words, which language?

Language

The above summary of the problem is a re-statement in specific graphics terms of the definition of a system problem given in the recent review article, "Systems Programming Languages".[3] Obviously, then, we have a system problem for which systems programming languages were developed. Obviously, then, we should use a systems language. Or perhaps not.

There are several very good reasons why a systems language should not be used. First, it doesn't exist. Though this is not likely the case in most large, time-shared systems, it is very frequently the case for many minicomputers. If it doesn't exist, then its creation is generally beyond the resources of a graphics project. Another reason is that it just isn't fast enough. Even though systems languages are designed for efficiency, they simply cannot match a routine coded in an assembly language. A systems language is also often machine dependent where there may be strong reasons to lean in the opposite direction, writing in FORTRAN, for instance. Then, too, FORTRAN has an audience in the applications field where there may well be need to interact with

and modify the graphics program itself. There are many other reasons for the persistence of FORTRAN in the graphics field, but as Newman and Sproull[4] state in an introduction to a discussion of the language in their text, Principles of Interactive Computer Graphics, "one language which has been used time and again as a graphics system basis is FORTRAN. If it were not for this, there would be little need to mention FORTRAN at all, for its performance in meeting our criteria is abysmal". The discussion that follows the quote is recommended reading.

My own list of most negative attributes in its use in graphics include slow speed, poor character handling, no macros, no bit or field operators, and an extremely poor base upon which to build a graphic language. None the less, being dissatisfied with FORTRAN is still insufficient incentive if no alternative is readily available, and some of the deficiencies pointed out in the aforementioned review[2] are serious. Some systems languages are no faster than FORTRAN; some are quite complicated to learn; most are very machine dependent. It is understandable that many will rely on the assembly language for the most crucial speed, bit and field needs, and just put up with FORTRAN's other attributes.

However, I was luckier. At the Courant Institute, Jacob Schwartz began development of a language called LITTLE in 1968.[5] The two basic goals of this language are machine independence and efficiency.[6] The LITTLE compiler aids the first goal by expressing programs as 'machine' language for an 'unknown machine', unknown, that is, at the time of program construction, with characteristics such as the word size considered a compile time parameter. The machine dependent section of LITTLE is a routine that maps the 'unknown machine' language onto the real machine, but this task is not at all extraordinary for any new machine, because the LITTLE compiler is written entirely in LITTLE. The approach bears some similarity to the 'abstract' machine of the STAGE2 system,[7] with the practical difference of less independence and more efficiency. Thus far, the second goal, efficiency, has only been measured on the CDC 6600 and is excellent. Efficiency results on the IBM 360 are expected shortly.

The structure of LITTLE includes MACROS, the usual arithmetic, logical and relational operations, bit and field operators, a single data type - the bit string,

the common IF...THEN...ELSE, DO, UNTIL, and WHILE constructions, conditional and unconditional transfers, local and global variables and a general program structure reminiscent of FORTRAN. In essence, if one took FORTRAN and added most of the options a systems programmer would want, one would have LITTLE.

From my own standpoint this structure has the additional advantages of making a link to existing FORTRAN routines easy, encouraging easy use to present FORTRAN - only programmers, and providing a vastly improved base for a graphics language.

Currently it is not a trivial exercise to write a machine block, with an estimate of four months of intensive work for a Honeywell Series 16 minicomputer, but this will decrease with the future expression of the 'unknown machine' in a formal language.

Acknowledgement

Jacob Schwartz, David Shields, and Edith Deak were all very patient and very helpful in introducing me to LITTLE. Elias Guth aided on the other end, getting the first code for the Series 16 machines out.

References

1. This work has been supported by PHS grant number NS-10072. General support by the AEC Computing and Applied Math Center under contract AT(11-1)3077 is also gratefully acknowledged.

2. Foley, J.D., "An Approach to the Optimum Design of Computer Graphics Systems", Comm. ACM, 14, 380 (1971).

3. Bergeron, R.D., Gannon, J.D., Schecter, D.P., Tompa, F.W., and Van Dam, A., "Systems Programming Languages" in Advances in Computers, 12, Academic, New York (1972), p. 177.

4. Newman, W.M. and Sproull, R.F., Principles of Interactive Computer Graphics, McGraw-Hill, New York (1973) p. 362.

5. Cocke, J. and Schwartz, J., Programming Languages and their Compilers, Courant Institute of Mathematical Sciences, New York University (1970).

6. Shields, D., "Guide to the LITTLE Language", LITTLE Newsletter No. 33, Courant Institute, New York University (1974).

7. Poole, P.C. and Waite, W.M., "Portability and Adaptability" in Advanced Course on Software Engineering, Springer-Verlag, Berlin (1973).

UNIVERSITY OF CALIFORNIA

Lawrence Berkeley Laboratory
Berkeley, California

AEC Contract No. W-7405-eng-48

A BARELY INTELLIGENT TERMINAL

Harvard H. Holmes

March 1974

A BARELY INTELLIGENT TERMINAL

H. H. Holmes
Lawrence Berkeley Laboratory
University of California
Berkeley, California 92720

ABSTRACT

A system for effective use of an intelligent terminal for graphics applications is described. It provides extensions to the basic hardware capabilities such as display subroutines and display scaling, and light pen tracking and inking. It supports a variety of local manipulations on display files which have been supplied by a host.

An interrogation facility allows the host to send a list of questions together with the ranges of acceptible answers to the terminal. Thereafter, a single command will invoke an interrogation of the user. Each answer, in turn, is checked for validity and is transmitted to the host only upon completion of correct input. Menus are used when the user must choose one of several alternatives. Each choice of an item may lead to a subsequent menu. These selections are accumulated and the host is interrupted only when the entire sequence is complete.

The editing operations allow changes to be made in the local display image, with or without sending these changes to the host. These operations are sufficient to allow a complete drawing to be constructed locally, without using the host at all. Our applications include ordinary graphs, symbolic modeling , and a drafting application.

The terminal hardware is a DEC GT40 display: a CPU, 8K of 16 bit memory, a display processor, and a communication line to the host. The local data structures comprise a display list, menus, and directories, supported by a simple brute-force memory allocation scheme.

Our goals for this terminal system are to provide fast response for display manipulations and editing. In addition, we anticipate a substantial reduction in computing load on the host for those applications primarily involving editing of displays. A substantial reduction in communication bandwidth makes remote use feasible, eg, at experimental sites via the ARPA net.

I. INTRODUCTION

One of the problems faced by graphics programmers today is how best to use the intelligent terminals which are being produced in ever increasing numbers. At large installations there is an enormous investment in existing hardware and software. One cannot just convert overnight to one of the new devices, but rather it must be integrated into the existing systems. And yet one cannot afford to overlook the possibilities offered by a new device.

We will describe the general tradeoffs involved in the selection and use of a graphics terminal and then we will describe the particular facilities which we plan to offer with our terminal, followed by a rough sketch of the projected implementation and some applications.

II. BANDWIDTH VERSUS INTELLIGENCE

The primary tradeoffs involved in choosing a graphics terminal can be characterized in terms of bandwidth and intelligence. These are two relatively independent variables which are easily understood. The selection of a bandwidth and intelligence will determine the computing load on the host and the applications served. Referring to Figure 1, we have arbitrarily chosen some benchmark tasks which are appropriate for a graphics terminal. These tasks are to display, edit or view in motion either simple or complex pictures. The response time for each of these tasks is considered to be 30 seconds for a display, 1 second for an edited display and 1/30 second for the display of each frame of a moving

display. A simple picture has 1,000 vectors and a complex picture has 10,000 vectors.

We have selected several milestones in our search for a definition of intelligence. Milestone 0 comprises no intelligence, for example a storage tube display or a television monitor. The cost per terminal is about $5,000. Milestone 1 adds refresh memory, a display processor and a CPU. Current systems, such as the DEC GT40 and the IMLAC PDS-4 are in the range of $15,000. These systems are intelligent terminals, able to alter a picture from coded commands and to search a simple data structure. Milestone 2 adds a disk, more memory and more processor and/or CPU power. These systems cost about $50,000. Depending on the application they may either be very intelligent terminals or a minimal satellite processor. Examples are the DEC GT44 and the IBM 1130 with a disk and 2250 display. These systems support a high level language and have enough power for continuous alteration of displays (animation) and for searching complicated data structures (disk resident). Milestone 3, in the range of $150,000, adds enough computing power to generate all the parameters for simple motion, or put another way, to simulate simple dynamic systems in real time. Large mini and midi computer systems fall into this category.

Bandwidth is the data transmission speed of the host to terminal connection. The cost of bandwidth is proportional to the rate and the distance involved. Telephone lines cover the range from 100 baud to 10 kilobaud. Acoustic couplers can be used at up to 300 baud, thus giving true portability. The ARPANET, using special lines, achieves 50 kilobaud. Computer I/O channels achieve bandwidths of 10 megabaud or more, but a channel connection usually requires that the equipment be in the same

room.

Our chart is constructed by assigning a bandwidth to each type of graphics activity: motion, editing or display of simple or complex pictures. Starting at the left of the chart we extend each activity to Milestone 1. After some reflection, we estimate the bandwidth reduction made possible by this much intelligence. To edit a simple picture, for example, we can keep the display in the terminal and transmit only change information. We estimate that this allows a bandwidth reduction of 30 to 1. Complex pictures cannot be so well structured in the limited memory of the terminal so we estimate a bandwidth reduction of 10 to 1 for editing complex pictures. The rest of the chart is constructed in a similar fashion. We then review the chart and make revisions as necessary to eliminate inconsistencies or clashes with common sense or experience.

We can now consider the computing costs of using the host. These costs include both the monetary costs and the costs in user frustration caused by poor response time. We have divided the hosts into two kinds: cheap and expensive. If we assume that the cost of the host is directly proportional to the bandwidth required, then we obtain the lines of constant cost as shown in Figure 2. We have also drawn lines of optimum cost effectiveness for the two types of hosts. If these lines are overlaid on Figure 1, they suggest what we all knew: as hosts become more expensive, the optimum tradeoff moves toward the more expensive terminal. Cheap, responsive hosts can make effective use of less expensive terminals. Unfortunately, the addition of several terminals will often cause the host to saturate, changing it from a cheap host to an expensive host. This is

an obvious reflection of the fixed capacity of the host. If a terminal is connected to a host, it would be wise to expect to process the extra workload in the terminal itself unless the host is suitably upgraded.

III. SOFTWARE SUPPORT

The introduction of terminals at our installation is providing an incentive to re-examine the software situation with a view to providing graphics wherever we now provide a teletype. We will not replace all our teletypes, of course, but we will provide enough graphics terminals so that they will be available to whoever needs them. While we have had CRT consoles for as long as we have had teletypes, they have not been as well received. The reason is that while the teletypes have gone out to the users' work area, the CRT consoles have remained isolated both physically and in terms of programming expertese. Now with the opportunity of providing graphics in the users' work area, we must also make the graphics terminal as easy to program as the teletype.

The first step in this direction has been the establishment of a standard device-independent interface at the FORTRAN subroutine level. We are now in the process of writing device drivers for each type of hardware to interface at this level. Each device driver can clear the screen, plot a sequence of lines, or plot a sequence of characters beginning at a specific point. Drivers for interactive terminals can also read user defined coordinates from a cursor or a tracking cross. This interface allows the user to specify his graphics device when the program is loaded. He may also control two or more devices. The second step in developing

graphics is to modify the several high level graphics languages at our installation to conform to this standard interface.

We will also develop some high level routines of our own: a menu routine, a questionaire routine and a modeling-edit routine. We will implement these on our intelligent terminal as part of the capabilities of the terminal itself. We feel that the implementation within the terminal will be easier and will provide much faster response while reducing the computing load on the host. This immediate response will make it much easier for the novice to learn to use this facility, since the results of each action will be instantly apparent. Frustration for the novice and expert alike will be reduced, since they no longer must wait for a response from the host. The menu facility will allow the host to define a tree structure of labeled nodes and send it to the terminal. When the menu is invoked via a short command from the host or internally within the terminal, the top of the tree is displayed as a menu of light pen sensitive items, either text or symbols; when an item is selected the branches below that node are displayed until a terminal item is selected. The entire path through the tree is then communicated to the host. In this way the user can rapidly select the items which lead him through a complicated command structure while the interaction with the host can be reduced to a few I/O requests.

Figure 3 is a sample menu. When the menu is invoked only the first two lines are shown; after choosing "output", the types of output are shown. After the type of output is selected, the user must confirm his selection. Until the selection is confirmed, the user may change his mind either by starting over at the top or by backing up one level at a time.

The questionaire is similar in spirit to the menu. It is a common facility in programs and it is implemented in the terminal to provide fast response and reduce the computational load on the host. The user is directed to a page of questions and as he answers, each answer is checked for validity. Default values may be provided and any value may be modified until confirmation.

The final and most extensive facility provided in the terminal is the modeling-edit facility which allows the host to send a picture to the terminal for modification by the user. Picture elements are grouped into subpictures and each subpicture may be included in any other subpicture. Any subpicture may be edited by the user; he may add or delete lines, alphanumerics and subpictures. Each change in the picture is sent to the host as it is made, so that when computation is desired, there is no time lost in sending the drawing back to the host. A simple application of this system is for layout of text, drawings, and other documentation. The host may supply some paragraphs of text and some drawings. If each item is a subpicture, then the text and drawings may be moved about on the screen until the proper composition is achieved. This especially suitable for preparing charts, tables and graphs for publication. The editing facility can achieve a pleasing layout far more easily and cheaply than multiple batch runs using trial and error.

The greatest gains can be made in the area of modeling. A typical sequence of the terminal would have the host initialize the terminal with symbols for electronics: resistors, transistors, etc. The user would connect these symbols to form a circuit and the circuit would be analyzed by the host. If the terminal had enough memory, the circuit could remain

in the memory ready for immediate editing as soon as the previous results had been viewed. Several response curves could also be accumulated for comparison by overlaying one on another. Programs for such analysis of symbolic drawings are already available, but the difficulties of using current terminals have prevented their widespread use.

IV. HARDWARE

Our terminal is a DEC GT40 having 8k of 16 bit memory, a PDP-11/05 CPU, a display processor, and a 2400 baud communication line to a CDC 6600. The CPU has six general purpose registers and a hardware stack. Interrupts are handled using an interrupt vector for each device and four priority levels are available. Each I/O device is assigned a pseudo-memory address and all of the CPU instructions may be used to manipulate data at the device address. The display processor includes a vector generator, a character generator and logic for direct memory access. A light pen is also part of the display processor. Light pen and other display processor - CPU interactions are handled with interrupts. The hardware does not provide a display subroutine jump so this is simulated as follows: (1) The display processor executes a stop and interrupt instruction followed by an address, (2) the interrupted CPU finds the address by reading the display program counter and (3) the CPU stacks the address and starts the display processor at the new address. If the new address is zero, then step 2 is reversed and the display processor is returned to a prior picture using the address from the stack.

The host is a CDC 6600 with 128k of 60 bit memory and the usual complement of I/O devices. All teletypes and the GT40 as well interface

through a PDP-8 which handles buffering and local control of the teletype lines. The PDP-8 is line oriented and will only pass information on to the host when a complete line has been received. Characters may be automatically converted from ASCII to the 6600 internal character code or they may be passed in image mode allowing every possible 8 bit character to be sent.

V. IMPLEMENTATION

The terminal program is implemented using the display file as the primary data structure. This file is continuously executed by the display processor so that each change in the data structure is immediately reflected on the screen. Memory is allocated in fixed sized blocks (usually 16 or 32 words) using a bit map to find unused blocks. As each block is filled with display code, the bit map is examined to see if the next block is free; if so, then the display code is continued into the next block. If not, the bit map is then searched for any empty block and a display jump is inserted from the old block to the new block. The bit map allows most blocks to be sequentially allocated and this avoids the overhead of a display jump most of the time. With 32 word blocks, the entire bit map for 8k words requires only a 16 word table, so the overhead required to find new blocks is very small.

The display file proper is organized into a master display list, a directory, and all the subpictures. A subpicture is displayed by putting a display subroutine jump for it into the master display list. The directory contains the names and the first addresses of all the subpictures. It also contains a response code which indicates what is to

be done when a light pen hit occurs on this subpicture. This response code identifies each item as a menu, a questionaire, an ordinary subpicture, or an internal element. This mechanism allows menus and questionaires to be stored as ordinary display elements, while providing proper response to these items. This scheme also allows the terminal to utilize menus and questionaires in its internal operation. The edit commands use this menu facility, for example.

Each subpicture is ordinarily closed, that is, it is defined entirely with relative beam positioning and the last operation returns the beam to its original position. This convention allows a subpicture to be modified subsequent to its inclusion in another picture without disturbing the location of items in the picture.

The terminal software has four discernable levels: memory management, display code generation, command interpretor, and interrupt routines. Every action by the user produces an interrupt which is put into a FIFO queue. As time permits, the command interpretor or the display code generation routines remove these actions and process them. The command interpretor uses the current state and a table to decide which of the code generator routines to call. The code generators ultimately call the memory management routines to alter the data structure.

A typical interaction would begin with the terminal initialized to display a menu. The user points to an item on the menu; after several hits, the interrupt routines put this action on the queue. The command interpretor removes the action, finds the response code and activates the proper routine. This routine may remove the menu and display another menu. The user may now ask to erase a line. The command interpretor sets the proper state and then waits for a light pen hit on a line. When the

hit occurs, the address of the line is passed to the proper code generation routine. This routine will remove the intensity bit from the line. It then must remove the line from the data structure if possible. It will search forward and backward to see if the line is surrounded by invisible lines. If so, these are combined and the deleted words filled with display no-operations. If an entire block of 32 words has been deleted in this manner, the block then must be de-linked and returned to the pool of available memory. The routine then returns to the command interpretor.

This system is implemented in assembly language using a cross assembler running on the host. The host also has available a text editor and a loader which operates over the communications line. Program development proceeds by typing in the new code to be added to the existing program. The code is then assembled and loaded into the terminal over the communication line. This usually takes less than one minute. The code is tested and any revisions can be made and immediately tested again. At the conclusion of the session the new source replaces the old one on the permanent file system. This is a great improvement over most minicomputer facilities; the line printers, magnetic tape drives, and other peripherals of the host are also immediately available.

IV. CONCLUSION

This system will enable an evaluation of the menu, the questionaire and the model editing facilities. If they are satisfactory, then they can be implemented for the PP (peripheral processor) driven CRT consoles

with a minimum of trial and error. This ability to proceed without trial and error is very important since the CDC 6600 operating system is not protected against errors in PP routines.

We feel that this system will go far toward enhancing graphical communication with the user. Our goal is for the user to regard these facilities with the same confidence that he has for the teletype and the text editor.

ACKNOWLEDGEMENT

Worked performed under the auspices of the U.S. Atomic Energy Commission.

REFERENCES

1. Holmes, Harvard., and Austin, Donald M., "Picasso: A General Graphics Modeling Program", ACM SIGPLAN: Symposium on Two-Dimensional Man-Machine Communication, Los Alamos, New Mexico, Vol. 7, No. 10, October, 1972.

2. Newman, William M., and Sproull, Robert F., Principles of Interactive Computer Graphics, McGraw-Hill, 1973.

3. van Dam, Andries, and Stabler, George M., "Intelligent Satellites for Interactive Graphics," NCC, 42, 1973, pp. 229-238.

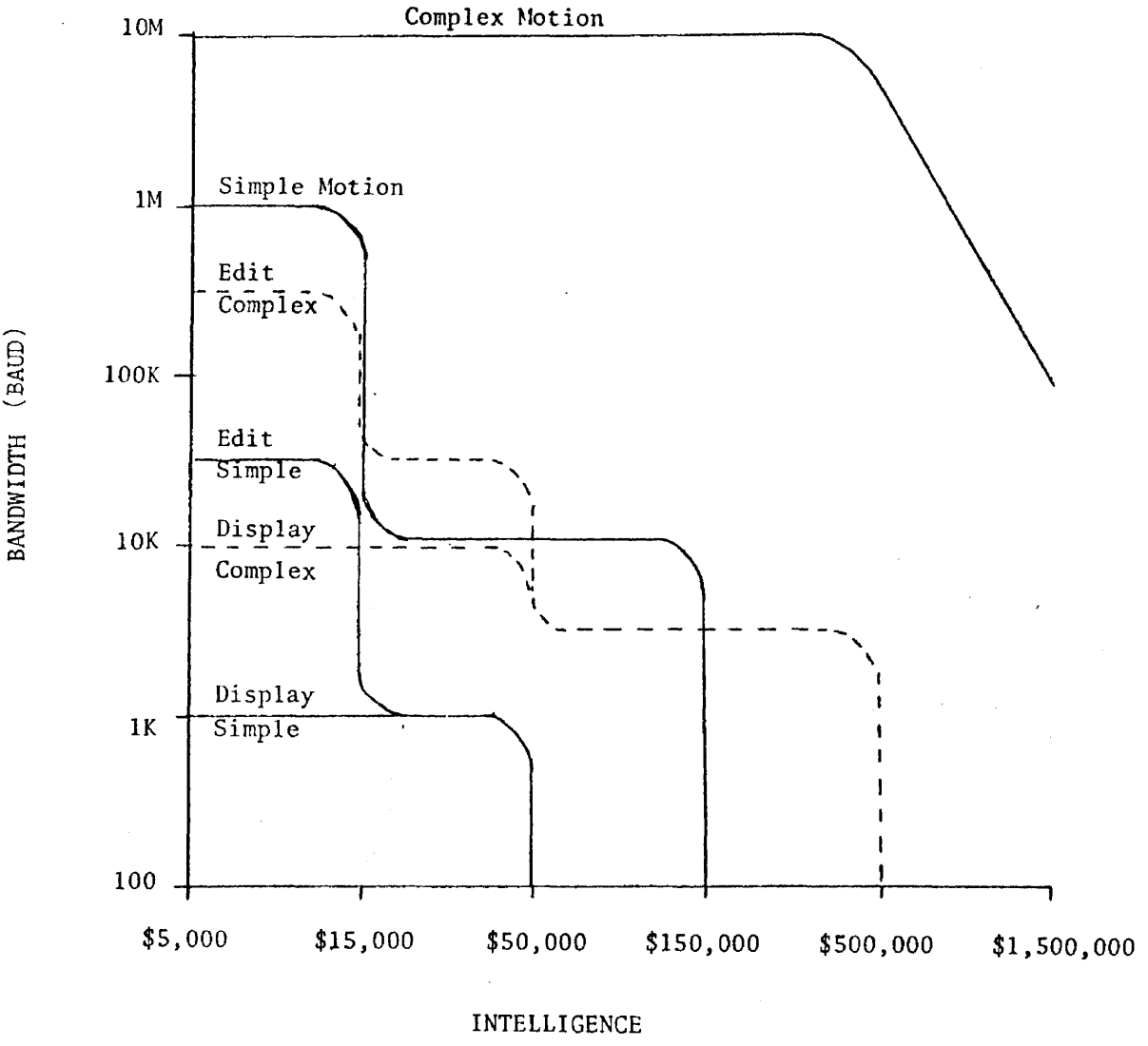


Figure 1. Bandwidth - Intelligence Tradeoffs for Selected Graphics Tasks

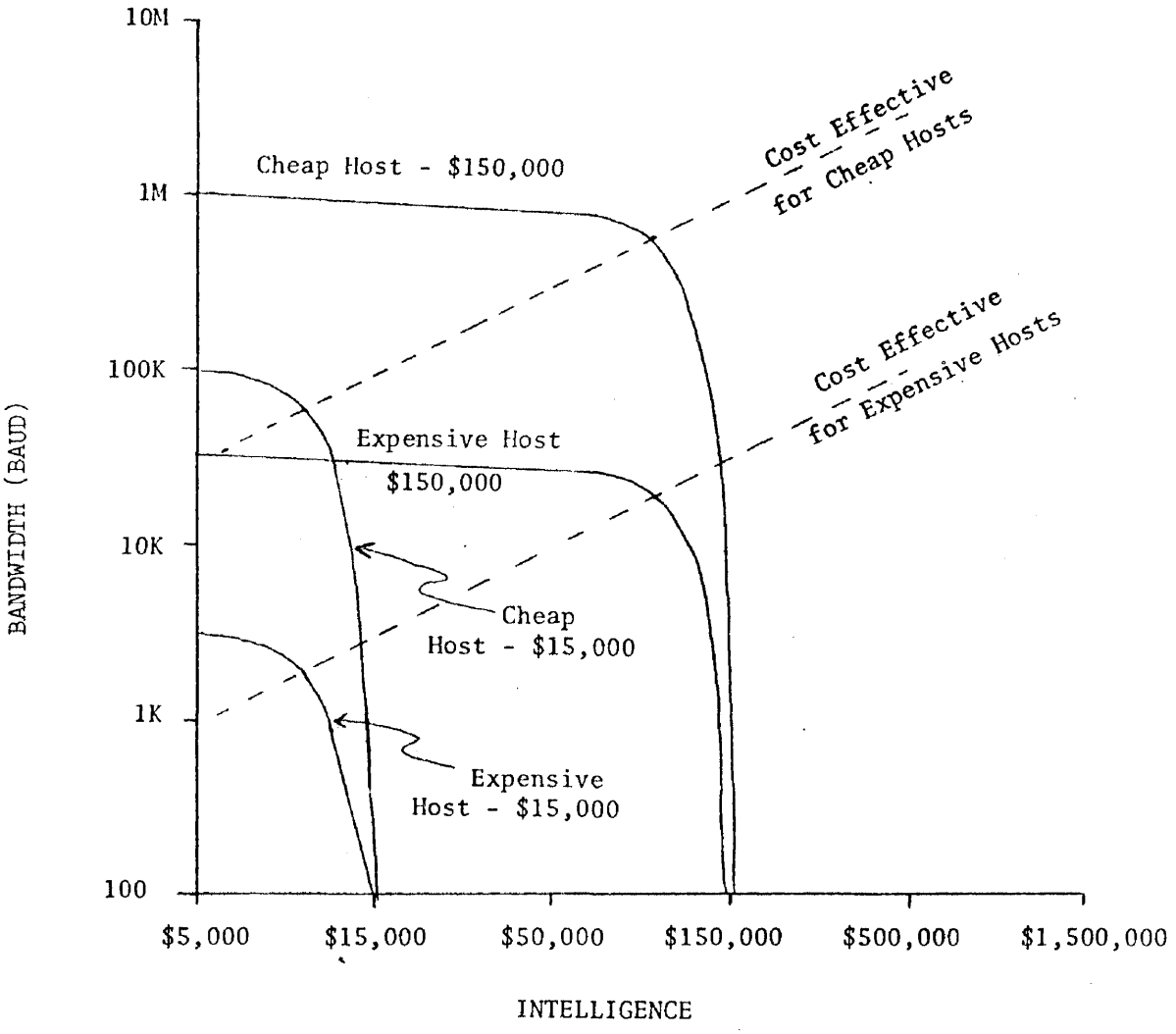


Figure 2. Lines of Constant Total Cost (Terminal Plus Computing) and Lines of Cost Effectiveness

SELECT NEXT COMMAND

INPUT	OUTPUT	COMPUTATION
	SELECT OUTPUT TYPE	
	HISTOGRAM	
	SCATTER PLOT	
	PERSPECTIVE	
	TIME SLICE	

Figure 3. A Sample Menu

Number of Magnets (0 to 8)? _ _ _ 4

Type of Magnet (Bending or Quadrapole)? _ B _ _ _

Number of iterations (1 to 100)? _ 1 0

CONFIRM (Y or N)

Figure 4. Sample Questionaire

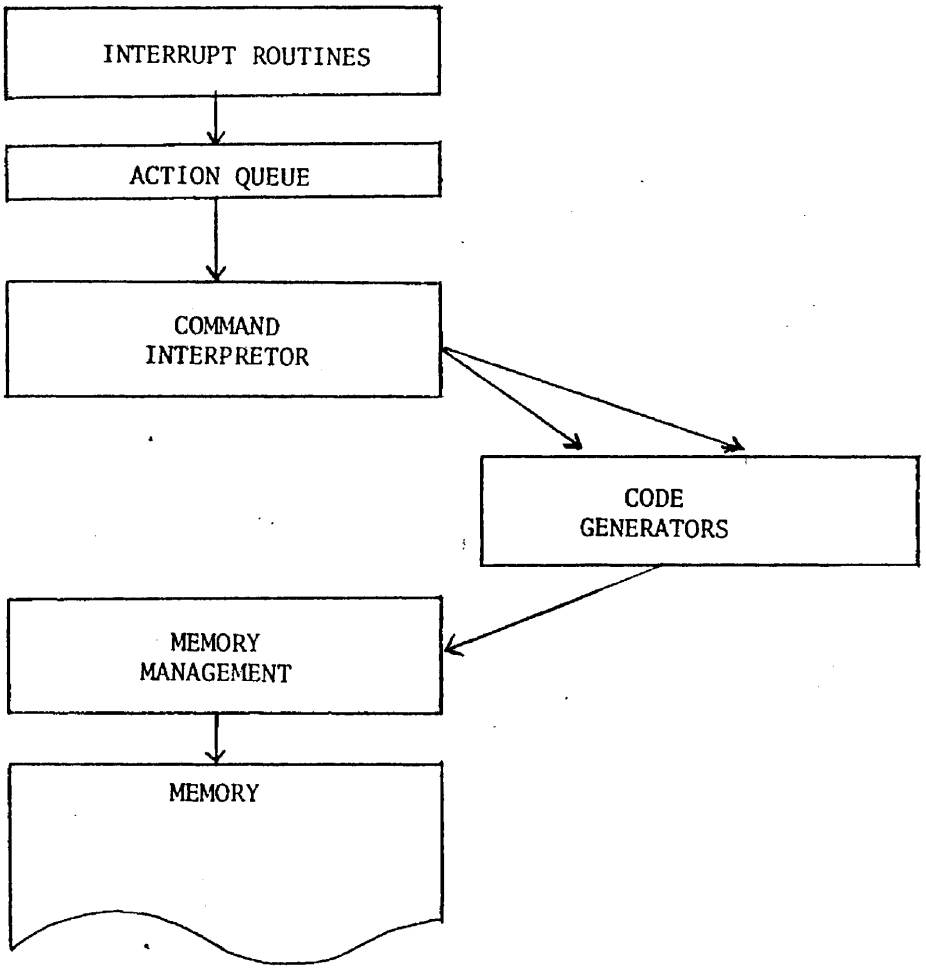


Figure 7. Terminal Program Structure

Sandia Interactive Graphics System - SIGS

R. Young

Abstract

A generalized interactive computer graphics system has been released for production use at Sandia Laboratories. The system utilizes five remote PDP-9 computers and Vector General 3D2 displays interfaced to a central CDC-6600 computer. The hardware configuration and basic system operating software are first described. The organization of the display file and its generation via FORTRAN callable display generation routines (DGR's) at the 6600 are then discussed. The manipulation of the display file and control of the graphics job via PDP-9 FORTRAN callable display manipulation routines (DMR's) and utility routines are described. A typical job run from the PDP-9 to the CDC-6600 is then presented. Current and future application programs using SIGS are then listed.

SANDIA INTERACTIVE GRAPHICS SYSTEM-SIGS

R. Young

Introduction

The Sandia Interactive Graphics System is the culmination of testing and evaluation of several graphics systems. The first graphics system implemented was a stand-alone DEC PDP7 computer with a 340 raster scan type CRT. Usage of this stand-alone system provided our graphics group with several basic facts regarding a general interactive graphics system. First such a system must be extremely generalized, flexible, and machine independent as possible to be of any extensive use for a production graphics system at Sandia. It must be fairly easy to learn and use for any level of application. The system must also be able to provide any application with sufficient computing speed, versatility, core memory, and mass storage in addition to a graphics display with comparable features.

The necessary computing power can be obtained by interfacing the graphics display directly or indirectly to a large computer. The direct interface method usually is an involved system programming task requiring a long design-to-implementation period, assuming the computer systems group will allow the modifications to be made to the operation system. In addition, the large computer will have the extra task of spending a large amount of time servicing the display generated interrupts and any "bookkeeping" required in the display file. Many installations have therefore taken the indirect approach by interfacing the graphics display to a smaller computer which is then interfaced to the large scale computer.

In this method, the small computer will service the display interrupts and provide the necessary "bookkeeping" for the display file while the large computer handles the computation tasks, generates the display file, and transmitts it to the small computer. This type of interface was the next type of graphics system implemented at Sandia. The previous stand-alone PDP7/340 system

was interfaced to a UNIVAC 1108 running under the EXEC 2 and later EXEC 8 system. A group of display generation subroutines was written for the 1108 and display manipulation routines for the PDP7. Our graphics system now had the necessary large scale computing power but was severely restricted by the slow raster scan speed, display work area, and lack of hardware features of the DEC 340 CRT.

A new graphics system was therefore planned using a UNIVAC 1108 interfaced to a DEC PDP9 with a LUNDY 32 display system. The LUNDY display generation routines were already coded for the 1108 when a management decision substituted a CDC 6600 for the large scale computer. Since the LUNDY display generation routines were coded in FORTRAN, it was an easy task to convert the routines to run on a CDC 6600. Unfortunately, the LUNDY display system failed to meet the acceptance tests. After evaluating the available displays on the market, a Vector General 3D2 display system was selected as a replacement for the LUNDY. Happily all four Vector General display systems were accepted and are currently in use. The fifth display will be interfaced to another PDP9 when the funds are available.

Hardware

The present interactive graphics system in use at Sandia utilizes a CDC 6600 as the central computer with four remotes using DEC PDP9 computers interfaced to Vector General 3D2 display systems. The central computer complex consists of a CDC 6600 main frame with 131K 60 bit words of central memory, 12 tape drives, 4 line printers, card reader, card punch, large system/scratch disk with 131 million character capacity, large user disk packs with 107 million character capacity, extended core storage of 500K word core, and 6674 data set controller which can handle up to four Bell 301B data sets.

The basic configuration for each of the four remotes consists of a DEC PDP9 computer with 16K 18 bit words of core memory, EAE, memory protect, 2 DEC tape transports, paper tape reader and punch,

card reader, KSR 33 teletype, magnetic disk memory with two platters giving 524K words of storage memory, four data channels, direct memory access multiplexer (allows up to four devices to use the DMA channel simultaneously), interface to a 301B data set, and a Vector General 3D2 display. All of the Vector General displays are equipped with an alpha-numeric keyboard, light pen with 3 microsecond response time, character generator, display hardware subroutining, 21 inch high speed CRT with 10 mil spot size, picture label scaling, intensity modulation, phosphor protect, and a PDP9 interface via the DMA multiplexer. Two of the displays also feature the control dials and data tablet options.

The communication between the central computer and the remote computer is made via Bell 301B modems on telpak lines providing a serial transfer rate of 40.8K baud with full duplex capability for each remote. The central computer is interfaced to the four 301B's using a CDC 6674 data set controller. The remote computer is attached to its 301B using a DEC DP01BJ interface which accesses memory through one of the four available data channels.

Systems Software

CDC's scope version 3.3 with Sandia modifications is the current operation system for all jobs run on the 6600. CDC's interactive graphics system (IGS) provides the software interface to scope. All 6600 user graphics jobs run under CDC's FORTRAN extended compiler. At the remote PDP9, a special Sandia coded executive device handler (DPB) interfaces with IGS. DPB handles all code translation, format conversion, communication synchronization, input-output file transmission, data transmission, and all display file transmission. In reality, DPB makes responses to all IGS status requests and issues directives to IGS which resemble a CDC 1700 computer, the computer normally used with the IGS system.

DPB also generates a software cyclic error code to comply with the hardware cyclic error code produced and checked by the 6674.

A Sandia coded device handler (VGI.) services all Vector General display interrupts, can start and stop the display, and returns information to the user which allows the display file to be modified and manipulated. Compilation of all user code to be executed at the remote PDP9 is done via DEC's FORTRAN IV compiler running under DEC's V5A resident keyboard monitor disk system. The display file is allocated the remaining PDP9 memory after the user's PDP9 program, necessary device handlers, display manipulation routines, and library routines have been loaded.

General Overview

The Sandia interactive graphics system can be thought as being divided into three major components: The CDC 6600 computer, The Vector General Display, and the PDP9 Computer.

The CDC 6600 is used to do calculations and to generate all of the Vector General Display commands via Fortran callable display generation routines (DGR's). The 6600 is used for the "Heavy Computation" and picture preparation. As the display commands are generated, they are transmitted to the allocated PDP9 display file memory via the IGS-DPB communications network. The Vector General display is started when the user 6600 program indicates that the display file is complete.

The PDP9 program is used to control the operation of the 6600 program and to manipulate the display file after it has been sent from the 6600. It is possible for the display file to be altered and redisplayed locally, or the 6600 will recompute an entire new display file with new parameters on command of the PDP9 program. It is improtant to note that the PDP9 program is in command. It should call in appropriate 6600 action when necessary and deal with the resulting display file, manipulating it where required. Data arrays may also be sent back and forth between the PDP9 and the 6600. This capability facilitates transferring new parameters to the 6600 which may be used in recreating the display file.

Display File The display file can be thought of as having a structure similar to that of a Fortran program. It is divided into four general areas: the Main Display Routine, the Vector General Stack, the Display Subroutine, and the DISTAB (Display File Description Table).

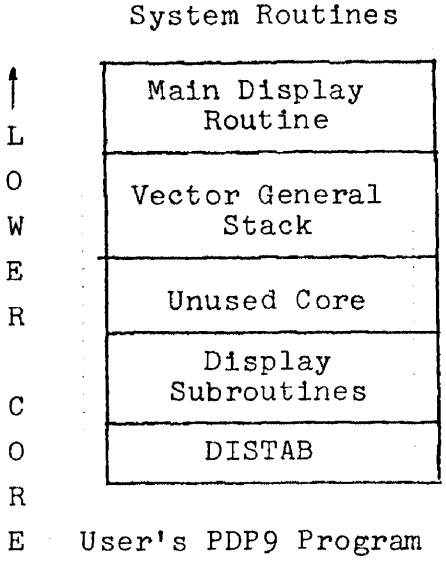
The Main Display Routine area is similar to a Fortran main program. In it, there may be statement numbers (called NAMES or TAGS) which can be used to point to any part of the display file (except the Vector General STACK area). Statement numbers can be used in a manner similar to a Fortran statement number in a GO TO statement.

NAME and TAG statement numbers can also be used to define display subroutines. NAMES and TAGS defined in this way are similar to defining a Fortran subroutine in that the NAME or TAG statement number corresponds to the Fortran subroutine name. A display subroutine must be written (defined) before it can be called. All defined display subroutines reside in the Display Subroutine area of the display file. Subroutine nesting is also allowed in the display file; however, the innermost level subroutine must be defined before it can be called by an outer level subroutine.

Each level of subroutine nesting requires a PDP9 word of memory in the Vector General STACK area of the display file. The STACK is used by the Vector General subroutine hardware to save the return address for each level of subroutine nesting. If subroutine nesting goes to 10 levels, then 10 words of PDP9 memory are needed for the STACK area. Subroutines are important in the display file for the same reason that the subroutine concept is useful in Fortran code. The subroutine allows repetitive use of code without the actual duplication of the display instructions. This is extremely useful in conserving core on the PDP9.

The DISTAB(Display File Description Table) is used by the Display Manipulation Routines (DMR's) to find any statement number NAME or TAG in the Main Display Routine or the Display Subroutine areas of the display file.

The total display file sent from the 6600 is arranged in PDP9 core as follows:



Statement numbers are divided into two basic classes, NAMES and TAGS. The basic differences between a NAME statement number and a TAG statement number are: A NAME statement number can be returned while retrieving light pen hit information, while a TAG statement number cannot; and the range of values for a NAME is $1 \leq NAME \leq 255$ as compared with a range for a TAG of $1 \leq TAG \leq 32767$.

There are four routines which the user's 6600 program can use to deal with subroutine setups and calls:

CALL DTAG(ISTNO,ISUB)	Define a tag.
CALL DNAME(ISTNO,ISUB)	Define a name.
CALL DEND(ISTNO)	Define the end of a subroutine.
CALL DCALL(ISTNO)	Generate a subroutine call.

where ISTNO = a unique statement number: 1 to 255 for a NAME and 1 to 32767 for a TAG.

ISUB=1 if this NAME or TAG is a pointer only; ISUB=2 if this NAME or TAG defines a subroutine. NAME and TAG statement numbers are useful in two basic ways: They can be used as pointers to any

portion of the display file (except the STACK) or to define display subroutines.

Each call to DTAG or DNAME makes an entry into the DISTAB. Each entry contains the statement number [NAME or TAG, with a flag indicating whether the NAME or TAG is a pointer (ISUB=1) or a subroutine definition (ISUB=2)] and a display instruction which points to the display code following the CALL to DTAG or DNAME. If the statement number is a NAME, then an additional entry is made in the DISTAB. This entry will allow the NAME value to be available to the user when retrieving light pen hit information.

If the user calls DTAG or DNAME with ISUB=2 (the statement number defines a display subroutine) then the display instruction in the DISTAB will point to the first word of the display subroutine. A call to DEND must then be made to finish defining the display subroutine. (This is similar to the Fortran END statement) and must be made before another call to DTAG or DNAME with ISUB=2). For ISUB=1, no call to DEND is necessary.

If a subroutine is defined using a NAME statement number, then that NAME value is only in effect to the end of the subroutine. One might well wonder why a TAG not pointing to a subroutine would ever be useful. Suppose that the user generates five pictures on the 6600 side but wishes to have only four of these displayed on the CRT. If a statement number (ISTNO) has been placed before the call which generated the picture he doesn't wish to view, then it is possible in the PDP9 program to cause that part of the display file to be "turned off." Without an ISTNO, there is no reference point to allow us to get at that portion of the display file from the PDP9 program. One's first inclination is to place ISTNO's all through the code. But note that ISTNO's require storage and should only be used where really necessary.

Suppose we have some code which draws a picture. Let's set this up as a subroutine using TAG 100:

```
CALL DTAG(100,2)
(code to draw the picture)
CALL DEND(100)
.
.
CALL DCALL(100)
```

Note several things in this example. First, the code between DTAG and DEND does not place instructions in the "main display" routine, but into the display subroutine area.

This section of our program merely sets up the procedures for a subroutine in the display file. The actual display file commands will only be executed when a call to DCALL is issued. The CALL DCALL(100) may be issued whenever required. However, we cannot CALL DCALL(100) until after the DEND statement defining that subroutine. Once set up, the subroutine may be DCALLed as often as necessary.

The display file structure looks very much like a Fortran program. The statement numbers allow the programmer to reference specific areas of the display file. Also, subroutine calls help prevent duplication of code in the display file. Enough TAGS should be used to supply necessary flexibility at the PDP9 but no so many that core is unnecessarily wasted.

Display Generation Routines - DGR's

There are three general levels of routines available to the graphics user under SIGS. The low level (system level) routines will give the user complete control and responsibility over his graphics display. Users are discouraged from using the low level routines whenever possible since they require a much better understanding of the Vector General and its display instruction set. The intermediate level routines should be used as they provide the user with considerable flexibility, but avoid the necessity of doing highly detailed graphics programming. The high level routines give much user convenience and maximum output with minimum programming with a sacrifice on flexibility. Only the intermediate level DGR's will be described here.

Two initializing routines must be called before any of the display code generating DGR's can be called. The first (CALL DINIT(3HTST or 3HRUN)) one gets the PDP9 display file boundaries from the PDP9. If DINIT is called with 3HRUN, the system will know that a display file or listings are to be sent to the PDP9. If DINIT is called with 3HTST, a debug run on the 6600 is assumed. For this case, no

communications link between the 6600 and a display is established. DINIT must be called before calling any other graphics routine. Also, only one call to DINIT with the argument 3HRUN is permitted in any one 6600 program. The second routine which must be called (call DISIZ(NAMES,ITAGS,ISUBS,ISTACK)) controls the size of the DISTAB and the minimum size of the Vector General display stack. All remaining PDP9 core is given to the display file.

For the intermediate level DGR's, all arrays of coordinate values must be in rasters (from -2048 to 2047). There are three routines available which convert user coordinates within specified boundaries to scaled integer raster values. The first (DBOUND) specifies the maximum and minimum user values which can be used in the two conversion routines. The first conversion routine (DFTOI) takes a floating point value and converts it to a scaled integer raster value (between -2048 and 2047). The second conversion routine does the reverse. Since these two routines use and produce absolute values only, two similar routines are provided to convert floating point values to scaled relative values.

Intermediate level DGR's have been written to allow use of most of the Vector General features. There are routines which position the beam invisibly for two dimensional relative or absolute Vectors, and also for three dimensional relative and absolute Vectors. The circle-arc routines can draw a circle or arc leaving the beam at the end of the circle or arc or at the center of the circle or arc. A multitude of Vector drawing routines provide the capability of drawing Vectors in two or three dimensions, as relative, absolute, incremental, and auto-increment in X, Y, or Z. A character routine allows the entire 192 character set to be used in the four available character sizes. A tracking cross routine used with the DTRACK DMR routine at the PDP9 allows any item to be moved with the light pen until any key on the display keyboard is pressed. A group of load register type routines allows selective blinking, light pen enabling-disabling, and modification of the

coordinate scale, picture scale, X, Y, Z offsets, intensity offset, intensity scale, rotation coefficients, and temporary general registers. Of course, all of these routines can be used with the statement number and subroutine defining DMR's (DNAME, DTAG, DCALL, and DEND). Another group of DMR's provide for the swapping of designated arrays between the 6600 and the PDP9. Presently up to eight arrays can be swapped with a maximum size of 93 PDP9 words for each array. The size limitation is caused by the size of the communication buffers used in IGS and DPB. Real, integer, double precision (PDP9 only), and data (no conversion) types can be swapped.

Display Manipulation Routines-DMR's

The PDP9 display manipulation routines allow the user to interact with the display file in a variety of ways. The light pen routines permit identification and selection of items in the display file by name register or temporary general register values. The name values are defined at the 6600 via the DNAME routine while the TGR values are defined by the DLDLPNO routine.. Any display keyboard value can be obtained by use of the DKEYBD routine. Often a user will generate more information in the display file than he desires to see simultaneously on the display. The DBREAK and DLINK routines have been written to provide the user with an easy method to selectively "turn on" and "turn off" portions of the display file. Both routines will work on any portion of the display file which have been identified by statement number names or tags.

The DCHANG routine allows a character string (which has been identified by a statement number name or tag) displayed on the CRT to be changed using the display keyboard. If the character string happens to be numeric it may be desirable to convert it from ASCII to a fixed or floating point PDP9 number for future computation. Conversions are accomplished using the DFNUM and DINUM routines.

Many times it is useful to be able to directly set or retrieve the contents of one or a consecutive string of the Vector General display controller registers. To set a display controller register or string of registers, the DSETR routine can be used. To retrieve a register or string of registers, the DLISTR routine should be used. The first register must be identified by a name or tag

statement number.

In order to simplify the format problems which can arise by using such routines as DLISTR and DSETR, two conversion routines are available. DVGTO9 converts a display controller register to a signed integer PDP9 value. The D9TOVG routine performs the reverse operation. (applies only to registers 8 to 13, 17, 19 to 31, and 70 to 79).

The DROT routine is available which can change the rotation coefficients of any portion of the display which was first identified at the 6600 by a name or tag statement number and followed by a call to the 6600 DROT routine.

If light pen tracking is necessary, then the PDP9 DTRACK routine can be used in conjunction with the 6600 DCROSS routine. DCROSS generates a subroutine defined tracking cross with a user specified name statement number. The tracking cross is turned off via a call to the 6600 DBREAK routine. The call to DTRACK at the PDP9 attaches the tracking cross to any user item defined by another statement number name or tag. Tracking continues until any key is depressed on the display keyboard.

For the graphics systems which have the control dials option, a DIALS routine returns the value of any designated control DIAL to the user in VG format. For systems also equipped with the optional data tablet, the X and Y coordinate values for the location of the stylus plus its vertical location above the data tablet can be obtained via the DTABLT routine.

The PDP9 DMR's also provide the swapping of designated arrays to and from the 6600. The DESIG routine is used by both the 6600 and the PDP9 to specify which arrays are to be swapped. Their size, dimensionality, and type. Each time DESIG is called in both computers, a correspondence is established between the arrays designated in the two computers. The calls should agree in number, order, array type, dimensionality, and array size. In order to store any array from either computer (all arrays are stored on disk at the 6600 using CDC's data manager), the DSTOR routine is used. when it is necessary to retrieve an array from either computer which has been "DSTORED", then the DFETCH routine may be used.

Job Flow and Control

The CDC 6600 operating system program management is based on the use of "control points" as a method of handling the allocation of resources to each user program. There are seven control points available to the system. The control points on the 6600 used for interactive graphics are used in the following way.

Control Point	
1	JANUS (input/output)
2	Export High Speed (graphics communication)
3	IGS (graphics execution)
4	BATCH
5	BATCH (execution of BATCH jobs)
6	BATCH
7	BATCH

Every job (graphics and non-graphics) is loaded from the input queue into central memory at a BATCH control point. The order of loading from the input queue into central memory at a control point is determined by job priority and job size. Sufficient information is retained by the operation system for jobs executing at each control point in order to allow the system to timeshare the use of central memory, roll jobs to and from disk, provide I/O (read tapes, Print, etc.), and other necessary tasks.

Before running an interactive job, the user will usually read in his PDP9 and 6600 programs using the card reader via PIP and store each program as a file on the disk. The PDP9 program is then compiled and the object O/P is stored as another file on the disk. At this point, the user can make any DAT slot assignments for any device handler (Dectape, Disk, Line Printer) needed for his graphics run. The PDP9 is then loaded and execution begins.

Following are the chronological events in the life of an interactive job.

1. The PDP9 program calls DSTART which establishes hardware and software communications with the 6600. DSTART puts out two messages via the PDP9 teletype. "Line Active"

indicates the PDP9 has received a hardware response from the 6600; "6600 Active" indicates the PDP9 has received a software response from the 6600.

2. The PDP9 program then calls DJOB which sends the 6600 program to the 6600. At this point the PDP9 program usually needs a display file from the 6600 before it can continue. The program can wait via a Fortran loop, or it can go to "Sleep" via a call to DSLEEP and wait for the 6600 to respond.

3. The 6600 job is placed in the input queue, and at this time the job name is printed on the PDP9 teletype.

4. Execution begins. (The job is assigned a BATCH control point.)

5. BATCH execution finishes.

6. The job is swapped to the graphics control point, IGS. IGS operates as a very high priority BATCH job along with the normal 6600 work load. Under IGS, the individual graphics jobs are timeshared independently from the rest of the BATCH work load. At this time, a short listing is produced consisting of a banner page and dayfile and is placed in the output queue. A message is printed at the PDP9 teletype saying that the job is in the IGS queue.

7. Interactive execution begins. The 6600 program performs the necessary computations and creates a display file. When the 6600 program has completed its appointed task and is ready to give control back to the PDP9 program, the user can call DWAKE. DWAKE indicates to the PDP9 that the 6600 program portion currently in execution has finished and the 6600 has rolled out the job. The display is started if the 6600 has (RE) generated a part of the display file. The PDP9 program can now continue execution. When the PDP9 program has completed its current tasks, it can call its version of DWAKE (the PDP9 version of DWAKE does not halt execution or initiate a wait loop) and then

do a Fortran loop or call DSLEEP. The 6600 program is then rolled back into core and execution continues. This exchange of control between the 6600 and the PDP9 may continue for as many cycles as is necessary to complete the graphics job.

8. Interactive execution finishes. The 6600 program usually terminates the job upon command from the PDP9 through a parameter sent via a swap array.

9. The listing is placed in the output queue.

10. At the first opportunity, export will take the listing out of the output queue and try to send it to the PDP9.

The following details will be helpful in understanding the events and are keyed to the events they describe:

1. Once communication is established, it will be broken if:
   a. The 6600 goes down.
   b. The line to the 6600 fails.
   c. The PDP9 stops (failure, program error, stop statement, program stop key, etc.)

2-3. The job does not enter the input queue until it all arrives at the 6600. If communication is broken before the job goes into the input queue, the 6600 throws away the information received up to that point and forgets all about the PDP9.

3-4. Operator intervention is often necessary to get to step 4 in a reasonable length of time. It is often expedient to call the operators on the "hot line" and call attention to the fact that the job is in the input queue.

4-5. Control cards down to and including the card containing the file name specified on the overlay (0,0, FNAM) card (example: FNAM) are executed at the BATCH control point.

3-6. If communication is broken at this time, execution will proceed until call DINIT (3HRUN) is executed at the graphics control point and will hang, waiting for the PDP9 to become available again.

7. This is the first opportunity to generate a display file and put a picture on the display.

7-9. If communications is broken at this time, the job will abort and the chain of events will stop at 9, at least temporarily.

9-10. If there is a listing in the output queue that came from a PDP9 that is in communication, event 10 will occur unless a call to DSPDIR has been made to stop it. If someone leaves a listing in the output queue as a result of breaking communications at the wrong time (as above), the listing will be sent back at the first opportunity, even though the opportunity is hours later, and the current graphics user is another person.

It is often very useful to know how far along the 6600 has progressed on a job, or to send some special directive to the 6600. A special routine, DSPDIR, has been created for this purpose. DSPDIR can be used to:

1. End communication (this will also cause the 6600 job to be killed).
2. Search the input and output queues and divert any jobs or listings from that PDP9 which sent the directive. A diverted listing is printed at the 6600.
3. Print out on the PDP9 teletype the current status of your job on the 6600. This is useful when one is waiting around for the 6600 to do something.
4. If a listing (not a display file) is currently coming across from the 6600, stop it. The listing is rewound and placed back in the output queue. No more listings will be sent during this communications session.

Another routine which is convenient to use in conjunction with DSPDIR is DSWICH. This routine returns a value set in the PDP9 accumulator switches 1 through 17 at the instant that accumulator switch 0 is flipped from off to on. Use of these two routines to control PDP9 and 6600 control is much better than reading characters from the PDP9 teletype since the Fortran formatting package occupies

much core and should be avoided where possible. A typical PDP9 program could look as follows:

```
C     GET AN ENTRY FROM THE ACCUMULATOR SWITCHES
          CALL DSWITCH(I)
          I = I + 1
C     CHECK FOR A BAD SWITCH SETTING
          IF((I.LT.L. OR (I.GT.5)) GO TO 4
C     BRANCH TO SWITCH SETTING
          GO TO (5,6,7,8,9),I
C     END COMMUNICATIONS AND TERMINATE JOB
5         CALL DSPDIR(0)
          GO TO 4
C     SEND 6600 LISTING TO THE 6600 PRINTER
6         CALL DSPDIR(1)
          GO TO 4
C     PRINT THE JOB STATUS ON THE PDP9 TELETYPE
7         CALL DSPDIR(2)
          GO TO 4
C     STOP A 6600 LISTING FROM COMING ACROSS
8         CALL DSPDIR93)
          GO TO 4
C     CONTINUE THE PDP9 JOB
4         CONTINUE
```

Current and Future Applications

DAVINCI (anonym for deleting, adding, verifying, and integrating network circuit images) is an interactive graphics program that allows the user to edit precision art work data. In addition to several manipulation capabilities it has window and pattern capabilities. The input to DAVINCI is a deck of Gerber plotter instructions that are translated into a data base used by the graphics system to produce a display

of the circuit geometry. The output is a Gerber plot tape. DAVINCI is a production program being used by precision graphics personnel. It takes about one day to train these people to use DAVINCI. In the past, if a printed circuit needed more than 15% revision the board was redigitized. With DAVINCI this is not done. Lead time is reduced on the average from several days to several hours.

Much of the 6600 computer time at Sandia is devoted to the solution of complex hydrocode problems. Hopefully, a considerable savings in computer time can be realized by graphically monitoring the development of hydrocode problem solution. Mesh elements, for example, can be monitored and altered before a degenerate solutions results. The time saving here is to be realized by stopping the execution of obviously degenerate runs and changing parameters in real time on other runs to obtain a successful solution. Since these programs use a rather large amount of computer time (several hours is typical) and core (large is on the order of 300000 octal words of central memory), even a small percentage reduction in time would be significant. Graphic hydrocide applications are presently in the development stage.

A typical data reduction process at Sandia requires several distinct reduction programs to convert raw measured quantities to final results which can be interpreted in a more meaningful way. The total data reduction process usually requires several days and often many partial reruns to complete. A graphics data reduction process is in the development stage which will reduce both turn-around time and reruns. The general steps in the process will be as follows:

1. Digitize or reformat and edit raw data from telemetry systems, PCM systems, or tracking systems, etc.
2. Examine the edited data via the Vector General CRT to select parameters for the next processing step.
3. Perform the selected operation on the data such as filtering, integration, or generalized mathematical transformations.

4. Plot or display the data derived from the preceeding step to verify proper parameter selection.
5. Repeat the two previous steps, if necessary, until satisfactory results are obtained.
6. If parameters were properly chosen, then the next process step is considered and parameters are selected for that step.

This interactive, iterative process is continued until the desired reduction step has been completed. From one to twenty steps may be required.

GAIN (graphic and for investigating networks) is used as a front end for sceptre a circuit analysis program widely used in this country. GAIN provides the circuit designer with a capability to interactively define circuit topology on the display screen and specify parameters to control the execution of sceptre. At the conclusion of the interactive session, GAIN generates the appropriate input data stream to allow sceptre to analyze the circuit which was defined. GAIN is designed to eliminate, as much as possible, computer-and-man time wasted because of incorrect sceptre input data.

The Nielsen/Air Force Flight Dynamics Laboratory computer program, SOURCE, has been recently adapted to interactive graphics. The new program displays graphical output created by program source which computes source and sink distributions to model axisymmetric bodies in subsonic compressible flow. Permits the user to rapidly to see both the desired shape and the resultant shape and change the input data interactively. The time required to generate an acceptable source-sink distribution for a body has been compressed from approximately two weeks to approximately one hour. This system has been used to develop a model for the fuselage of the F-4D aircraft and will soon be used to develop a model of the body of the SLA B57 Tiger prototype weapon. These models will be used to theoretically analyze the store separation of the B57 Tiger from the 4F4D aircraft prior to the full scale drop test program.

RLOCUS is a computer program used by the Exploratory Measurements Division. This application is similar to the program "SOURCE" in that the user can modify parameters for his 6600 program at the display. The graph that is returned to the display after each 6600 run indicates to him the proper parameter choices for the next run. It is not unusual for him to execute 40 to 50 times in a one hour session reducing the solution time from 2 or 3 weeks to 1 hour.

APPENDIX A

6600 Display Generation Routines - DGR's

Initialization

DINIT(3HRUN)

DISIZ(NAMES,ITAGS,ISUBS,ISTACK)

Conversion

DBOUND(AMIN,AMAX)

DFTOI(F,I)

DFTOIR(F,I)

DITOF(I,F)

DITOFR(I,F)

Positioning

DMBA2(IX,IY)

DMBA3(IX,IY,IZ)

DMBR2(IX,IY,IZ)

DMBR3(IX,IY,IZ)

Circle Arc

DARC(IVM,IRAD,IANGL,NDEGS)

DARCCTR(IVM,IRAD,IANGL,NDEGS)

Vector

DVAAX2(IVM,INC,IA2,NMPTS)

DVAAX3(IVM,INC,IA2,IA3,NMPTS)

DVAAY2(IVM,INC,IA1,NMPTS)

DVAAY3(IVM,INC,IA1,IA3,NMPTS)

DVAAZ2(IVM,INC,IA1,NMPTS)

DVAAZ3(IVM,INC,IA1,IA2,NMPTS)

DVA2(IVM,IA1,IA2,NMPTS)

DVA3(IVM,IA1,IA2,IA3,NMPTS)

DVECT(IVT,IVM,IA1,IA2,IA3,NMPTS,ID)

DVIAX2(IVM,INC,IA2,NMPTS)

DVIAY2(IVM,INC,IA1,NMPTS)

DVI2(IVM,IA1,IA2,NMPTS)

DVI3(IVM,IA1,IA2,IA3,NMPTS)

DVRAX2(IVM,INC,IA2,NMPTS)

DVRAX3(IVM,INC,IA2,IA3,NMPTS)

DVRAY2(IVM,INC,IA1,NMPTS)

DVRAY3(IVM,INC,IA1,IA3,NMPTS)

DVRAZ2(IVM,INC,IA1,NMPTS)

DVRAZ3(IVM,INC,IA1,IA2,NMPTS)

DVR2(IVM,IA1,IA2,NMPTS)

DVR3(IVM,IA1,IA2,IA3,NMPTS)

Statement Number and Subroutine

DCSIZ(NDIST,NDISP)

DFSIZ(NWORDS)

DNAME(ISTNO,ISUB)

DTAG(ISTNO,ISUB)

DCALL(ISTNO)

DEND(ISTNO)

DBREAK(ISTNO)

DLINK(ISTNO1,ISTNO2)

DTRUNK(ISTNO)

DJUMP

Character

DCHAR(IHV,ISIZ,ICARA,NCHR)

DISVAL(IHV,ISIZ,VALUE,FRMT)

Load Register

DBLINK(I)

DCOSCL(FNUM)

DLDLPNO(LPNO)

DLPON

DLPOF

DOFSET(IX,IY,IZ)

DPICSCL(INUM)

DSET2D(INUM)

DSET3D(INUM)

DROT(ALPHA,BETA,GAMMA)

Array Swapping

DESIG(ARRAY,ITYPE,I1,I2,I3)

DFETCH(ARRAY)

DSTAT(NSTAT)

DSTOR(ARRAY)

Tracking

DCROSS(ISTNO1) used with DMR DTRACK

Communication

DWAKE(I)

APPENDIX B

PDP9 Display Manipulation Routines - DMR's

Initialization

DSTART(ISO,ICI,FILEC,ILO,FILEL)

DJOB

Conversion

DFNUM(IARAY,FLT)

DINUM(IARAY,INT)

DVGT09(IVG,IPDP9)

D9TOVR(IPDP9,IVG)

DCNVRT(IPDP9,XVALUE)

Statement Number and Subroutine

DCHANG(ISTNO,IARAY)

DLISTR(ISTNO,IREGNO,IARAY)

DSETR(ISTNO,IREG,IARAY)

DMOVE(ISTNO,IX,IY,IZ)

DROT(ISTNO,ALPAH,BETA,GAMMA)

DTRACK(ISTNO1,ISTNO2,IXEND,IYEND) used with 6600 DCROSS

DBREAK(ISTNO)

DLINK(ISTNO1,ISTNO2)

DRCHAR(ISTNO,IPDP9)

Display Interaction

DFO

DIALS(IREGNO,IVALUE)

DTABLT(IX,IY,ILOC)

DKEYBD(IVALUE)

DLPREG(IVAL)

DLPHIT(IVAL)

Array Swapping

DFSIG(ARRAY)

DFETCH(ARRAY)

DSTAT(NSTAT)

DSTOR(ARRAY)

Communications

DWAKE

DSLEEP

DSPDIR(N)

UNIVERSITY OF CALIFORNIA

Lawrence Berkeley Laboratory
Berkeley, California

AEC Contract No. W-7405-eng-48

CONFERENCE PROGRAMS WITH INTERACTIVE GRAPHICS

Donald M. Austin

March 1974

Conference Programs with Interactive Graphics

Donald M. Austin
Lawrence Berkeley Laboratory
University of California
Berkeley, California 94720

ABSTRACT

Multi-user conference programs provide interaction between users at remote terminals through the mechanism of a time-sharing or multiprogrammed host computer system. Extension of the conference program idea to include terminals with graphic input and output capability will provide a more natural medium of interaction and information exchange for a large class of problems.

The major problems to be solved in developing this type of network graphics facility are the interfacing to a variety of graphics terminals through a device-independent graphics system, providing reasonable data transmission rates necessary for interactive graphics, and the design of a suitable man-machine interface language to handle a variety of problem areas. Operating system requirements for both shared-program and shared-file conference systems are investigated, and the implementation of such a system based on the BKY 6000 operating system at LBL is explored.

I. INTRODUCTION

Conference programs are interactive programs which allow several users to interact with each other through the mechanism of a time-sharing or multiprogrammed host computer operating system. Representative examples are the single-host MOTIF program on Dartmouth University's DTSS [1] and the FORUM program of Institute for the Future which runs on the ARPANET system under TENEX [2]. These programs provide textual communication between participants with the additional advantage of having computational and data base facilities of computer systems as an integral part of the activity. On a network of multiprogrammed system with remote terminals, users may interact with each other and a data base in a real-time environment or on a delayed basis via a storage file mechanism.

Graphics is a natural extension to text based conference programs. The availability of graphics terminals has reached the state that this extension is both practical and useful for problems in which communication by pictures is the natural method. Interactive graphics applications cover most areas of problem solving, but the programs are usually written for specific systems and a relatively small number of these satisfy both criteria of extensibility and transportability.

The major problems to be solved in developing this type of network graphics facility are the interfacing to a variety of graphics terminals, providing reasonable data transmission rates necessary for interactive graphics, and the design of a suitable man-machine interface language to handle a variety of problem areas.

II. APPLICATIONS FOR INTERACTIVE CONFERENCE GRAPHICS

Some more or less specific examples of interactive conference graphics programs will help define the problems associated with implementation of such a system. Applications may be separated into two rather broad categories which characterize the nature of the interactive graphics - display with commentary and game situations.

A. Display with Commentary

Perhaps the simplest applications of interactive conference graphics are the analysis programs which display a picture (graph, set of data, schematic, flow chart, etc.) and allow the users to select features of interest and comment on them. A further extension of this category includes features such as windowing, zooming, overlaying plots and guiding the analysis by command menus and parameter setting. Many existing interactive applications could be extended to permit easy and natural communication of ideas between remote users. Conference manipulation of graphical data bases, such as urban planning, transportation network design, or architectural design, is an area which overlaps this category and the game situation area.

B. Game Situations

This category includes programs in which input from more than one user is required. Obvious game situations are those usually associated with the term, such as chess, economic modeling games and other decision-testing applications. A broader definition encompasses teacher-student and question-answer applications in a graphics based problem area.

III. DEVICE-INDEPENDENT GRAPHICS SYSTEMS

A. Types of Terminals

In order to be very useful, conference graphics programs require that the host computer be able to communicate with a variety of interactive graphics terminals and hard-copy devices. These terminals can be classified as follows:*

1. "Dumb" terminals, which perform i/o only.
2. "Semi-intelligent" terminals, which have a limited instruction set display processor and local memory.
3. "Intelligent" terminals, which have central processors as well as display processors and local memory.

For the first class, the host computer must speak to the terminal in its language. There is no sub-picture capability and each change in the display must be done in the host computer. An example of this type is the Tektronix 4012.

The semi-intelligent terminal has a limited subpicture capability and at least a "start address" operation code, so that portions of a display can be changed selectively without re-transmitting the entire picture. Translation of the graphics data into display commands must still be done by the host computer.

The intelligent terminal has a fairly powerful computer and is capable of performing many graphics operations locally, including translation of graphics data into display commands, simple transformation

---

*cf. Ref. 3 for a review of terminal classifications

of subpictures and editing.

In a conference system utilizing a variety of terminals, most operations will be reduced to the lowest common denominator, for if each user is to have an identical display all operations on the graphics data structure necessary to generate a new display must be done by the host for the lesser endowed terminals. However, a reduction in data transmission can still be realized by utilizing the full power of each type of terminal in the conference. For example, suppose a zoom operation is called for. For "dumb" terminals, the appropriate transformation of the data structure, translation into terminal display commands and the transmission of the new picture to the terminal must be carried out by the host. For the intelligent terminal, however, only a simple parametrized zoom command need be transmitted and all the other operations can be carried out locally.

Graphics input devices available fall into three categories:

1. Character input devices, usually a keyboard with 6, 7, 8 or 12 bit character codes.
2. Numerical input devices, such as potentiometers, or function keyboards.
3. Two-dimensional input devices, such as light pens, joy-sticks, mice, tracking balls, data tablets, thumb wheel cursors.

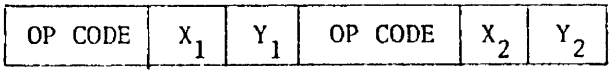
(There exist some three-dimensional input devices, such as the Lincoln Wand and 3-D joy sticks, which form a fourth category, but these are usually too exotic to be useful with 2-D terminals.)

B. Requirements for Device-Independent Graphics Systems

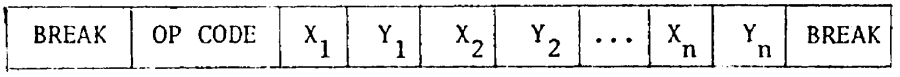
Given the above constraints it is evident that applications programs for conference graphics should be based upon a device-independent graphics system. Such a system consists of general high level routines for creating displays such as grid, smoothed curves, etc., plus some low level routines for translating a graphics data structure into terminal-specific display instructions. Ideally, the system should meet the following requirements:

1. Allow full use of available hardware features, such as character generators with variable sizes, fonts and orientations, and vector generators with variable line widths and intensities.
2. Allow for support of several devices simultaneously, including hard-copy devices operating in parallel with the various types of terminals.
3. Allow for high level graphics operations such as picture sub-routining and incremental display modification.
4. Allow modular selection of high level routines and have small memory requirements for low level routines.
5. Allow for the various categories of input devices.

A common implementation of a device-independent graphics system employs a high level intermediate display language with a set of graphics commands in either a fixed-length format, such as



or a string format, such as



Translation of the intermediate display file into device specific commands can be done as a separate job step or in line by specifying at load time the proper library of low level subroutines. Through picture subroutining, a mixture of the two methods can be used.

The study by the Network Graphics Group for the ARPA computer network covers most aspects of device-independent graphics protocol [4]. The protocol proposed by this group is to be implemented at various levels of sophistication, and provides features for interfacing with all types of terminals.

IV. OPERATING SYSTEM REQUIREMENTS

In order to implement conference programs on a host computer system, the operating system must contain certain features. Two possible conference systems will be discussed - the shared-program system and the shared-file system. The applications possible with these two systems share considerable overlap (the shared-file system has more general possibilities), but operating system requirements differ considerably for the two.

A. The Shared-Program System

For a single set of related applications, particularly in game situations, the most efficient method of conferencing is the shared-program system. In this system, multiple terminals are connected to a single job running at a single control point (thus a single user operating system is even suitable if one has the resources to tie up a host computer with interactive jobs). Features required of the host operating system are:

1. Multiple-terminal interface.
2. Multiple-terminal connection to a single job.
3. An interrupt or polling capability which allows the host computer to service any one of the connected terminals with reasonable response time (including log on and log off).
4. For interactive graphics programs, a device-independent graphics system and an appropriate set of interpretors.

The third requirement is perhaps the most troublesome. The concept is simple enough - it requires that the program be informed by the terminal handler whenever any terminal logs on or off the conference program.

In addition, it requires that the program be able to post reads to all connected terminals and be activated (rolled into central memory) when any input is forthcoming.

Typically such a program would consist of an applications module, a high level graphics module, an executive module and a set of interpretor modules, as depicted in Figure 1. The applications module operates on some data base to produce interesting data, which is fed to the graphics module for creation of a display file. The executive module directs input to the interpretors for translation and transmission to the terminals. Terminal response is fed back to the executive for further action.

B. The Shared-File System

The shared-file system is somewhat more general than the shared-program system in that communication is between separate jobs, each of which may include different applications programs and data bases. Features required of the host operating system are:

1. Multiple-terminal interface.
2. A multiprogrammed or time-sharing system.
3. Special file types accessible by more than one program simultaneously.
4. An interrupt or polling facility which allows a host program to service any of the conferee's with reasonable response time (including log on and log off).

This system is the basis of the conference systems mentioned in the Introduction (DTSS [1] and FORUM [2]) and seems to suit a wider variety of operating systems than the shared-program system.

A schematic of this system is depicted in Figure 2. The executive module reads the input files from the terminals and creates a global

VI. A PROPOSED IMPLEMENTATION

The computer center at LBL offers the following facilities relevant to interactive conferencing:

1. Interconnected CDC 7600, 6600 and 6400 with over a billion (60 bit words) of on line mass storage.
2. A terminal handler system being expanded to 256 terminals with data rates up to 9600 bps.
3. A variety of interactive terminals, including Tektronix 4012's, DEC GT40's, CDC 250 VISTA consoles, plus several hardcopy devices.
4. ARPANET connection (soon).

The BKY operating system currently allows an implementation of a shared-file system through a facility called "shadowed" COMMON files. This facility allows a job to capture a COMMON file created (and temporarily released) by another job, obtain read-only access (i.e., SHADOW the file) and return it to the system. The originating job then recaptures the file by the COMMON operation and retains write access. Anything written on the file can be immediately read from the shadowed file. Thus in Figure 2, the executive program shadows all the input files for the connected terminals, and all the terminal programs shadow the global display file simultaneously. Polling is accomplished by periodically reading the system File Name Table into executive program memory space and checking a list of prespecified file names for users logging on or off the conference. By maintaining an updated list of file pointers, the programs can determine when new input is available on a given file.

On the BKY 6000 system, interactive jobs are automatically rolled out of memory after a period of inactivity. Thus, while the terminal programs can be rolled in on demand, the exeuctive program, which is not connected to a terminal, must execute a recall loop in order to relinquish the central processor to other jobs. This becomes unnecessarily expensive for long periods of inactivity. One solution, albeit a rather clumsy one, is to have a chairperson terminal connected to the executive program. It is then the chair's responsibility to insure that response time is maintained at a reasonable level. A much more elegant solution is to provide a peripheral processor (PP) program which resides in one of the 20 PP's attached to the 6600. This PP program can perform the polling function by "waking up" the executive program when new input is forthcoming. Going one step further, the same PP program is capable of doing direct memory-to-memory block transfers, eliminating the need for auxilliary storage files (at least for input from the terminals, which tends to be smaller than the global display file).

The shared-program system has already been implemented for the primitive Berkeley Remote Facility, and a new system under development for the implementation of the ARPANET connection at LBL.

VII. CONCLUSIONS

Conference programs with interactive graphics on a variety of terminals offer a useful method of communication between users at remote sites. The problems involved have for the most part been solved in one way or another, and all that remains is fitting the pieces together into a coherent system.

The shared-program system allows several terminals to connect to a single job, offering features usually associated with conference or game situations, where all users are interacting with the same data base. The addition of graphics broadens the applications possible with this system to include many problem areas not feasible with text-only systems.

The shared-file system connects several interactive jobs and thus provides several host-sized computer facilities to the conferees. This system is in fact a natural extension of computer networks and is considerably more general than the single-host, shared-program concept, since only the graphics and file transfer protocols need be specified. Program languages, analysis programs and data bases available to the users can be as varied as required for a particular application.

ACKNOWLEDGEMENT

Work performed under the auspices of the U. S. Atomic Energy Commission.

FIGURE 1. DIAGRAM OF THE SHARED-PROGRAM ENVIRONMENT

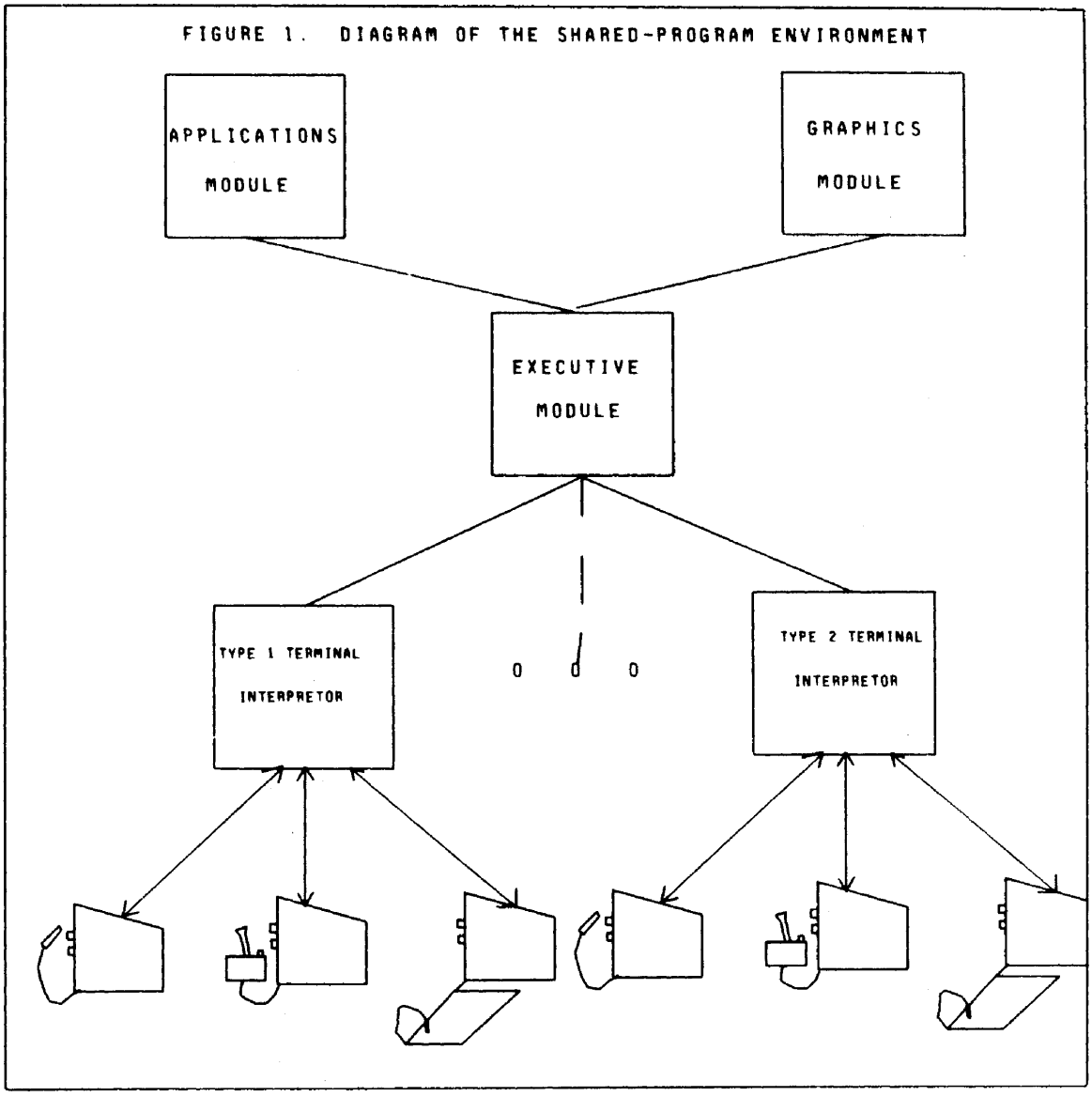
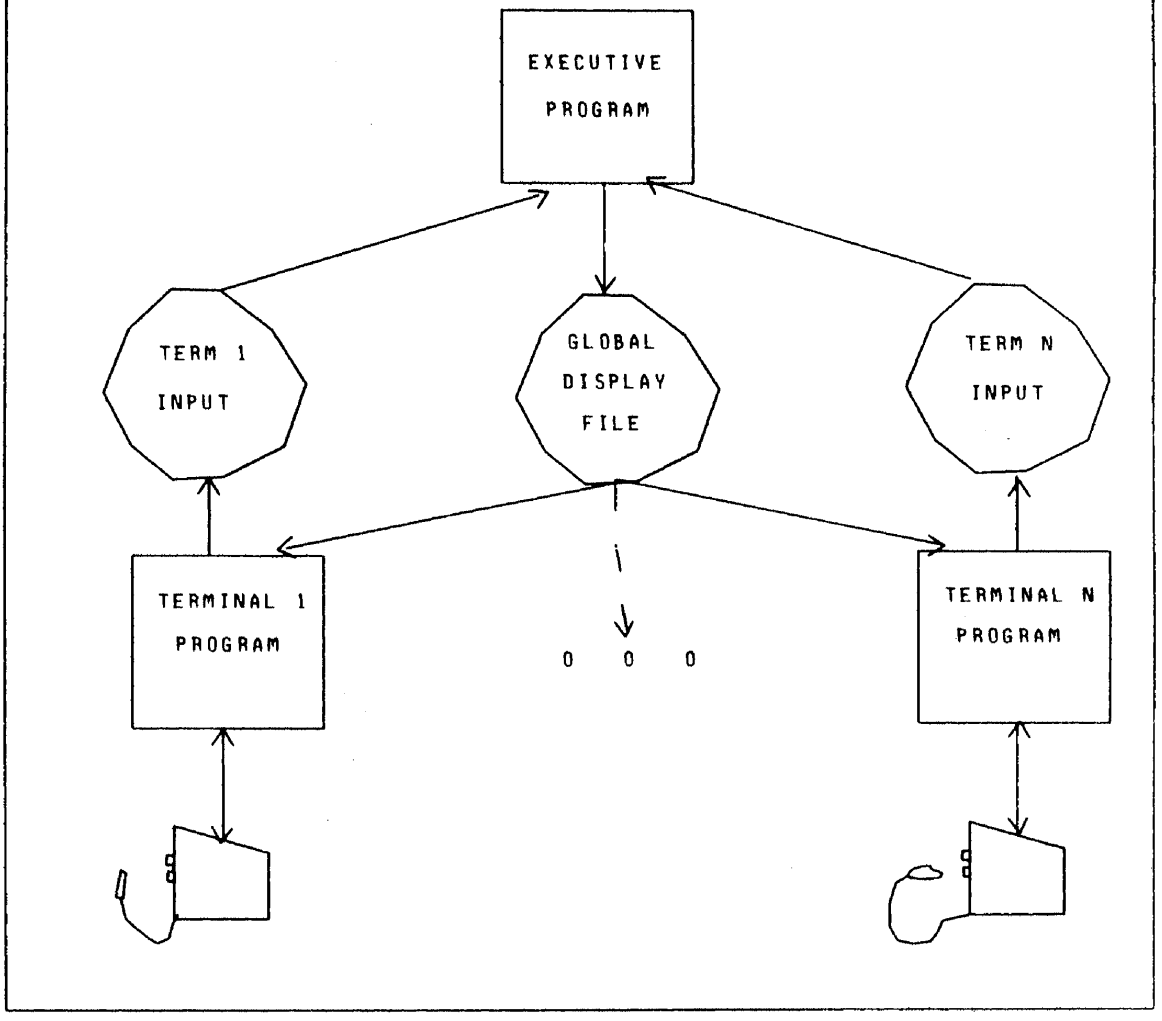
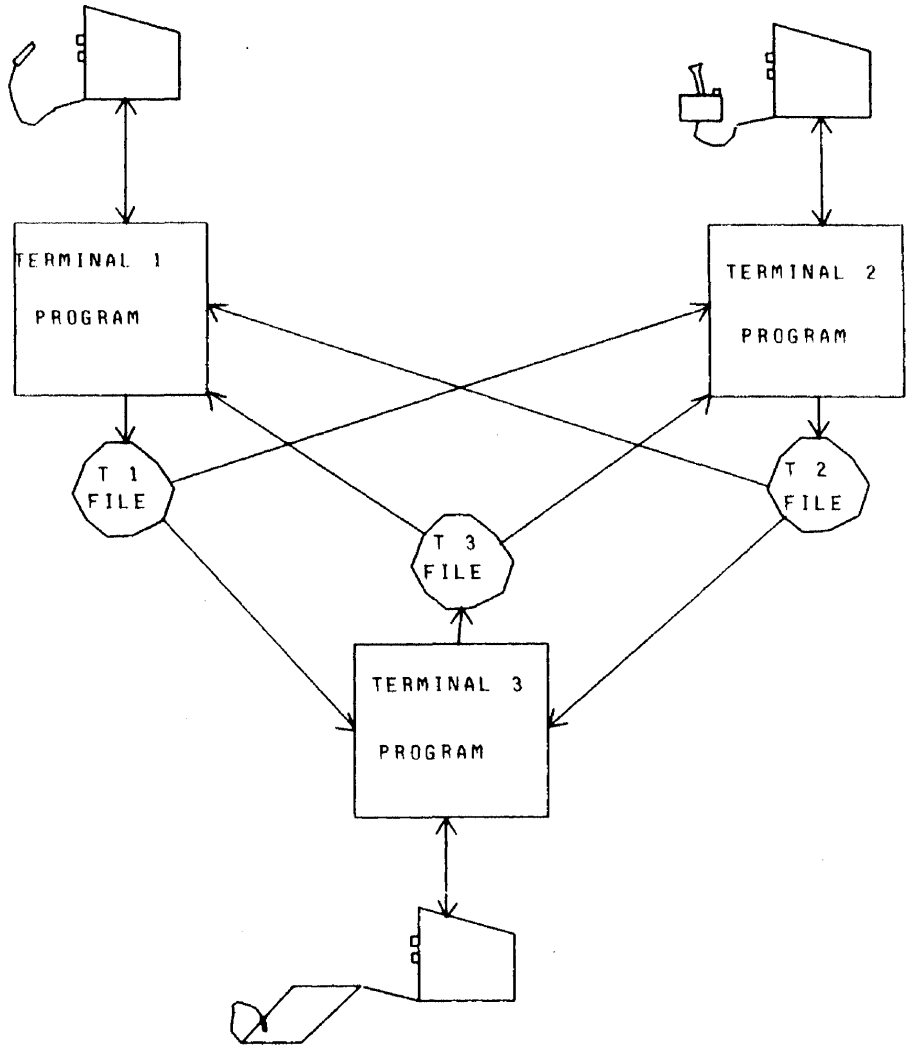


FIGURE 2. DIAGRAM OF THE SHARED-FILE ENVIRONMENT



DEMOC

FIGURE 3. DIAGRAM OF THE DEMOCRATIC ENVIRONMENT



REFERENCES

1. McGreachie, J. S., Multiple Terminals Under User Program Control in a Time-Sharing Environment, Comm. ACM 16, 10 (Oct. 1973), 587-590.

2. Amara, R. and Vallee, J., FORUM: A Computer-Based System to Support Interaction Among People, Institute for the Future, Menlo Park, Calif. 94025.

3. van Dam, A., Intelligent Satellites for Interactive Graphics, Proceedings of AFIPS, 42 (June, 1973) 229-238.

4. Michener, J. and Sproul, B., Proposed Network Graphics Protocol, Network Graphics Group Note No. 5, NIC No. 19933, ARPA Network Information Center, Stanford Research Institute, Menlo Park, Calif. (Oct. 1973).

Graphics Applications for Finite Element Code Processing

V. K. Gabrielson
Sandia Laboratories
Livermore, California

ABSTRACT

This paper describes the application of interactive graphics to mesh generation and to output display processing of finite element codes.

INTRODUCTION

An interactive graphics terminal has been used for a number of finite elment applications for several years. For mesh generation, it has been used primarily to verify mesh designs, reducing the time required to create a desired mesh. In evaluating and processing output data, the terminal has been used to scan the large amount of data finite element codes produce, permitting detailed study (in given regions of the structure) of any of the stress and strain vectors produced by the code. Displacement plots of the node points can be constructed for the best visual presentation.

Currently, two types of terminals exist at SLL (Sandia Laboratories, Livermore), an interactive terminal which requires a dedicated control point and memory on the CDC 6600, and two DVST (Direct View Storage Tube) displays, which are used with the CDC 6600 Intercom system.

Since both systems are competing with the CDC 6600 batch processing, the emphasis is on small memory requirements for these interactive graphics applications, effective data handling procedures internal to the code, and adapting the structural codes to interface with the graphics.

Mesh Generation

The use of mesh generation programs as separate input processors is quite common for finite element codes. In general, they are coded for specific finite element codes; any generalities usually result from effort spent by the programmer to make them adaptable to more than one application. The FEMESH code has been used in this capacity for several years at SLL. The code is designed primarily for finite element codes in which the node points are mapped onto the (i,j) unit grid. The code described here for the DVST terminals was an adaptation of this code.

Features of the FEMESH code which make it attractive for terminal applications are: small memory size, mesh can be designed in sections, a simple meshing algorithm, small input data set, and few limitations on size of completed mesh.

Features of the DVST terminal for which the code was designed include: keyboard input, a display tube of 1024-760 rasters, tracking cross, and a heat sensitive printer for recording data displayed on the screen. The CDC Intercom software is used, and the system shares a CDC 6600 Intercom control point which competes with normal batch processing and other Intercom users. Applications programs are written in Fortran using system subroutines for the tracking cross, displaying text, and constructing line vectors. A DVST is a write-only display (the display can be refreshed only by erasing the total screen), which restricts interactive capability. The use of the Intercom system allows the user to communicate directly with the SCOPE operating system on the CDC 6600, permitting the use of UPDATE and file processing programs from the terminal. In addition, the Intercom text EDITOR program can be used. The Intercom system restricts

programs to less than 60000_8 words of memory, and neither ECS (Extended Core Storage) nor tape storage can be used.

The mesh code used at the terminal has the following capabilities: input data can be entered at the terminal or accessed from a permanent disk file; a graphical display of the current data set can be generated; the entire data set or any part of it can be listed on the display with the graphics display; editing of the data set can be done by inserting, deleting, and changing data records; the tracking cross can be used to extract coordinates of data points; and areas of the display can be enlarged, using a "zooming" option. The current input data set is always stored, to ease recovery from errors and to allow the job to be performed at discrete times. The node point data set of the completed mesh design can be processed for several finite element codes.

The following gives a brief description of the meshing procedure used in the code. The structure to be meshed is represented as a body of revolution and may be divided into two-dimensional regions of common materials. Each region (PART) is treated independently of others in the code, and is defined as four sides consisting of point sets representing line and arc segments. Each region is then subdivided into an (M x N) array of quadrilateral elements which produce the mesh.

Finite element codes require each element to be defined by its four node points. The node points on the boundaries of the region are computed from the data sets defining the sides. The (M-1) x (N-1) set of node points internal to the region are computed as follows. For a given internal point $(x,y)_{m,n}$ such as illustrated in Figure 1, the x coordinate is computed as

$$x_{m,n} = (K_1 x_{m,1} + K_2 x_{m+1,n} + K_3 x_{m,n+1} + K_4 x_{1,m})/(K_1 + K_2 + K_3 + K_4)$$

where

$$m = 2, \ldots, M; \quad n = 2, \ldots, N.$$

A similar equation applies to $y_{m,n}$. The K_i's are weights whose numeric values are determined as a ratio of the node points' relative location (m,n) from each of the region's sides to the smallest subdivision of the given node row or column. For equally spaced nodes, this reduces to

$$K_1 = 1/(n-1) \qquad K_2 = 1/(M+1-m)$$

$$K_3 = 1/(N+1-n) \qquad K_4 = 1/(m-1) \ .$$

For rectangular regions this produces a proper orthogonal mesh. For more general regions, adjustments to the K's are made by weighting functions w_i which are applied to the respective K_i and apply over all node points internal to a region.

For further enhancement of the mesh in given regions, the subdivisions along any of the given surfaces can be proportioned. Appropriate adjustments are made to the K's for such definitions, resulting in the same proportioned relationships for the internal mesh points in the region.

The following example illustrates the use of the mesh code with the DVST terminal. The boundary data sets of Figure 2 were input at the terminal, and a display verifying the data is shown in Figure 3. The SCAN option is used to list the data and insert data specifying the number of subdivisions in each region. Data is again listed on the display for verification. If errors are noted the data is edited; if correct, the mesh is constructed on the display as shown in Figure 4. Adjustments to the mesh are made by using the WEIGHT and RATIO options

by inserting their specifications in the data set using the editing procedure. The user may iterate in this sequence until satisfied. If desired, the editing procedure allows him to add new regions and change boundary data sets of previously defined regions. (Also, interactive features such as the tracking cross and windowing as illustrated in Figure 5 can be used.) To use this data in the finite element codes, the regions are mapped onto an (i,j) unit grid. This option is available using a mapping procedure for locating each given region on the (i,j) grid. Displays of these results are constructed on the terminal for verification and illustrated by Figure 6. After the mapping is verified, the data can be processed for a given finite element code application.

The feasibility of using a DVST type terminal for mesh generation has been shown. The program has the features of being small in memory size, with small input data set, and it can be applied to complex structures due to region definitions. Few limitations are imposed on the size of the problem since the code does not require two-dimensional arrays. The code was used effectively for the problem illustrated which created proper meshes for an ablation calculation and stress analysis of a nosetip of an aerodynamic test vehicle.

Output Processing Using an Interactive Terminal

An interactive graphics terminal has been used for processing output data files for the SASL code for several years. The output file of this finite element code for linear elastic axisymmetric stress analysis included the boundary data set, the (r,z) coordinates of the node points, the geometry data required for each element, the displacement of each node point, and the stress and strain functional values for each element. The code uses the Q4 integration

element, which computes the displacements at the four node points and computes stresses and strains at the integration point within the element. The code can be used for applications having nonlinear material properties; this involves an iteration process in which an output data set can be generated for iteration. The output file for each data set includes displacements for each node point and 22 stress and strain vectors at each integration point.

The program for processing this data at an interactive terminal provides the capability for the analyst to display the mesh, access any of the data sets in the file, and construct displacement plots of the node points as functions of the boundary data set, magnification factors, and regions of the structure such as shown in Figure 7. For any given data set, any of the 22 stress or strain vectors can be accessed and displayed as contour plots over the two-dimensional space describing the structure. The analyst can select contour levels, and contours over regions of the structure can be displayed in detail using windowing techniques. Several options are available for contour display as shown in Figures 8-10.

Features of the interactive terminal for which the code was designed are a CRT (Cathode Ray Tube) display of 1024 x 1024 rasters, 12 function interrupt switches, and a typewriter keyboard. The program, processed by the CDC 6600, is written in Fortran, with subroutine calls to graphics system programs which produce necessary code for the 8192-word memory of the terminal controller used for generating the display on the CRT. The terminal requires a dedicated control point and sufficient memory to process the program on the CDC 6600.

This direct coupling with the CDC 6600 has the advantage of high-speed processing, allowing extensive calculations to be made between interactions. Having access to large data stores and ECS makes it attractive for output processing applications. The unit is also directly tied to a CRT plotter used with the CDC 6600, and film records can be made of any display at the terminal.

The disadvantages of direct coupling are that it increases the cost of processing, and that the system in its present status cannot be expanded. Although no restriction of program size is required, practical processing requires codes to be under 50000_8 words of central memory and 200000_8 words of ECS.

The implementation of the output processor for the SASL code to be applicable to any size problem required special efforts in coding. Since only one function value existed for each element, knowledge of the element's neighbors was required for contour plotting. Therefore, contour plots were designed using the (i,j) unit grid as the basis of the contour map.

Figure 11 illustrates the problem, in which the X's imply node points and O's, integration points. The contour area is noted by the dotted line, and is divided into four triangles in which the functional values can be determined at each vertex. The use of triangles for contour plotting simplifies the logic, since a contour line can only intersect two of the sides.

Since the maximum (i,j) varies greatly between problems, a form of dynamic dimensioning was used which changes all arrays to desired dimensions prior to compiling. ECS was used effectively for storing the node points associated with each element, and data for contouring was stored as a function of the j lines of the i,j grid. Storage required for 1000 element data sets is around 50000_8 words; 3000 element data sets required 75000_8 words. (Less than 100000_8 words of ECS is used for any size problem.) The code has been used effectively for processing this type of data at SLL, and it is the primary means of evaluating output from this particular code.

Output Processing for the GNATS Code

Since the terminal was used successfully as a means to provide greater flexibility for processing data from a finite element code, it was considered as the primary output processor for a new nonlinear, large deflection, stress code recently developed at SLL. From the experiences learned in the preceding application, considerable effort was given to the design of the output data set. This code has incremental loading capabilities which produces an output data set for each increment. It also is designed with the higher order Q8 integration element with many options as to number of node points and integration points used per element. For example, Figure 12 shows an element with eight node point displacements and nine integration points internal to the element. For this case, the data computed which may be of interest for postprocessing is over 300 items of data per element per data set.

The structure of the output data file is as follows:

(a) file containing all parameters of the data set,

(b) a load set file for restarting purposes,

(c) a data record for each element containing all coordinates, displacements, stress and strains for data set one, and

(d) a data record for each element containing data for data set two, etc.

The graphics code written to process this data has the same capabilities as noted before for contour plotting, mesh display, and displacements. The basic difference between the two applications is that sufficient data exists within each element providing the means to treat each element independently for contour plotting. For this application, the basic contour element of Figure 10 is used on the Q8 element. In this case, the functional values and coordinates must be determined at the node points of the element by fitting a function over the two-dimensional space by a least squares procedure. This capability eliminates the need for the (i,j) unit grid required in the SASL applications.

ECS is used more effectively since all element data can be stored for a given data set. This increases the speed of contour processing and the code can process any size problem without change in dimensions. The code currently requires about 50000_8 words. The ECS required is a function of the problem size and type of element used. The use of ECS is a tradeoff with central memory requirements and speed of data transfer. The code is programmed such that the random access feature of ECS is not required and data could be processed from sequential disk storage.

Since contours are computed independently over each element, the function is averaged at the node points to obtain a continuous contour such as shown in Figures 13 and 14. A contour can be obtained that is not averaged which can result in a very discontinuous function at the element boundaries such as shown

in Figure 15. This display aids in determining the quality of the integration option used, coarseness of the mesh, and the type of methods used to evaluate the function at the node points. In addition, contours can be constructed on displaced meshes such as shown in Figure 16, which illustrates a problem of large distortion. To improve the accuracy of the contour function over the element, a refinement of the triangle evaluation has been implemented permitting 8- and 16-triangle options to be used. The triangle option used is a function of the integration option and coarseness of the mesh.

Several of these options have been experimental since the code is in the development stage. The only means to evaluate many of these options is by graphical display. The use of the interactive terminal provides a convenient means to do this quickly, and in the detail necessary for proper comparisons.

Conclusion

The use of interactive terminals for the applications shown has been considered quite valuable at SLL. The relatively high cost of processing pays off in the ability to reduce time to analyze data, check validity of solutions, and produce quality graphical outputs that emphasize special aspects of the problem.

The reduction of time was clearly shown in the mesh application for the nosetip analysis. The time span for the analyst to construct a mesh using the DVST program obtaining a usable data set for the structural analysis program was about three hours. The same reconstruction was required on another code for which the DVST application was not available. The procedure required processing a large code for mesh generation and verifying the results from the film records. The span of time resulted in over two days to complete the task.

A similar type reduction of time can be obtained with evaluating output data files from finite element codes. The amount of calculations required to produce the plots is large but when processed by the CDC 6600, no serious problem with speed is encountered. About two to three seconds of central processing time is required to produce the contour plots shown. The elapsed real time to display this data at the terminal is normally between five to ten seconds, which results in no appreciable wait time for the analyst.

The experience gained by having the power of the CDC 6600 available provided the ability to learn techniques on file structure and reducing memory size. The current programs will continually be improved for new applications and equipment as it becomes available. The programs should be adaptable to interactive systems which use PDP-type computers for processing terminal requests but use the CDC 6600-type computer for computation and file storage.

List of Figures

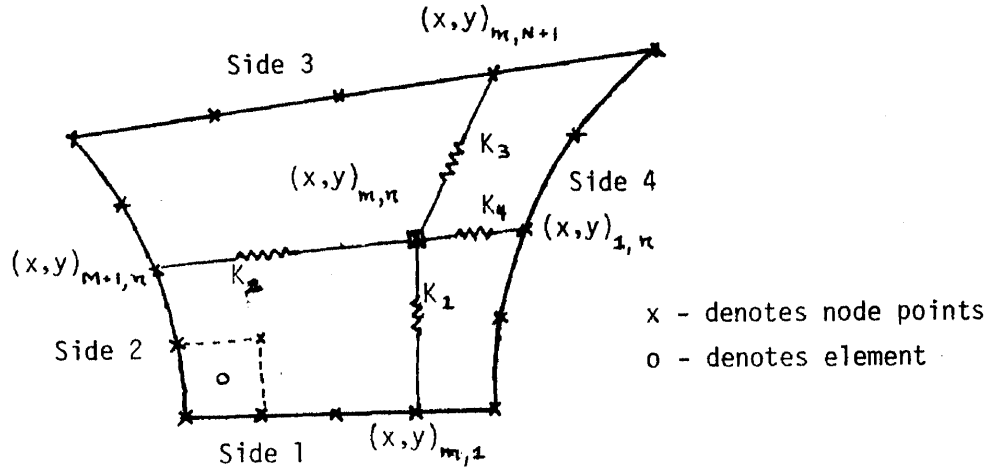


Fig. 1 - Representative region and mesh definitions

Fig. 2 - Input data set

```
PART   1  1  1
SIDE   1  1  2   2.18   0.0    1.25     0.0
SIDE   1  2  2   1.25   0.0    0.0      0.0
ARC    1  3  3   1.25   0.0    1.25   180.0    96.0
SIDE   1  3  2   1.12   1.24   2.0      1.34
SIDE   1  4  2   2.0    1.34   2.0      0.92
ARC    1  4  3   2.18   0.92   0.18   180.0   270.0
SIDE   1  4  2   2.18   0.75   2.18     0.0

PART   2  1  1
SIDE   2  1  2   3.0    0.0    2.18     0.0
SIDE   2  3  2   2.18   0.75   3.0      0.75

PART   3  1  1
SIDE   3  1  2   3.0    0.75   2.18     0.75
ARC    3  2  3   2.18   0.92   0.18   270.0   180.0
SIDE   3  2  1   2.0    1.0
SIDE   3  3  2   2.0    1.0    3.0      1.1
SIDE   3  4  2   3.0    1.1    3.0      0.75

PART   4  1  1
SIDE   4  1  2   3.0    1.1    2.0      1.0
SIDE   4  3  2   2.0    1.34   3.0      1.44
END
```

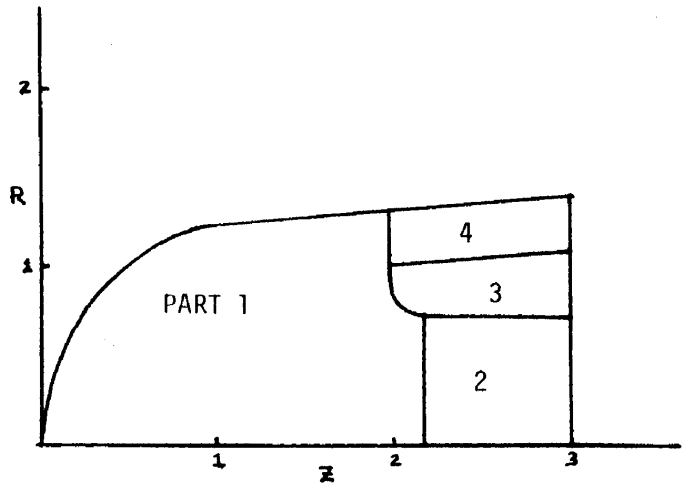


Fig. 3 - Display of input data set

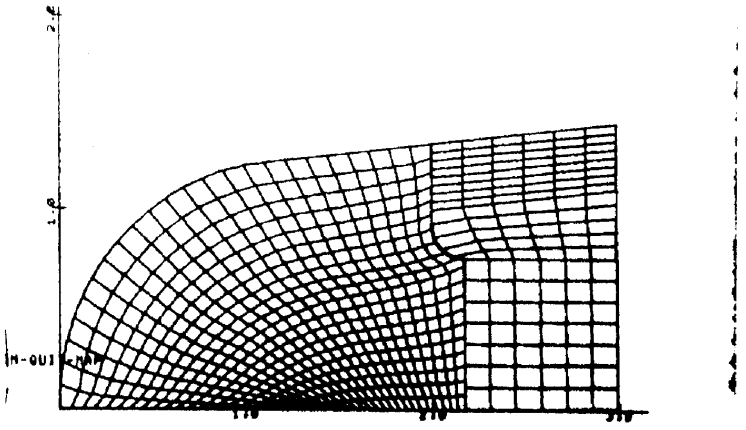


Fig. 4 - Display of mesh using equal subdivisions

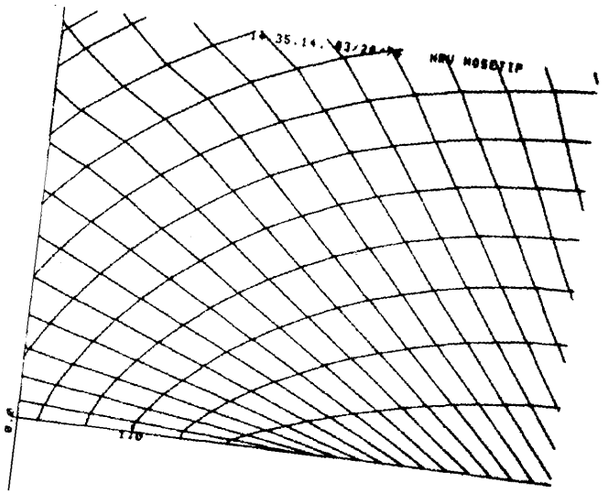


Fig. 5 - Illustration of a windowing of Figure 4

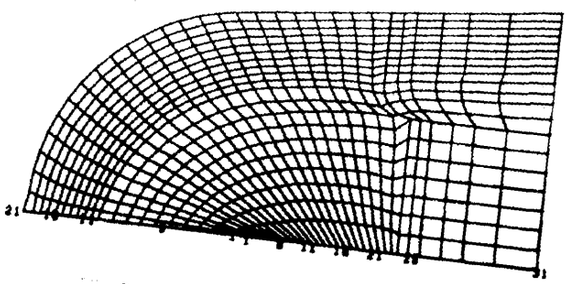


Fig. 6 - (i,j) line display of completed node point data set

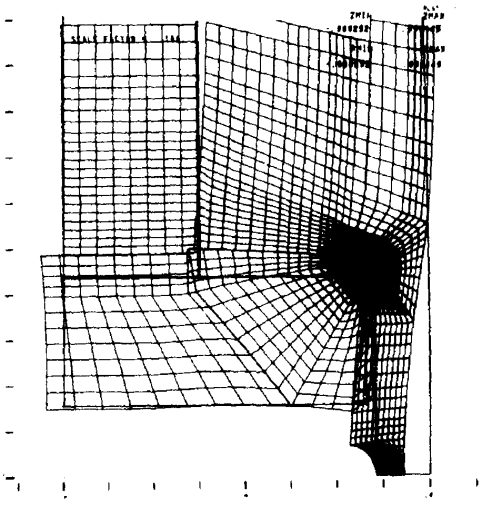


Fig. 7 - Example of displaced mesh option with SASL data

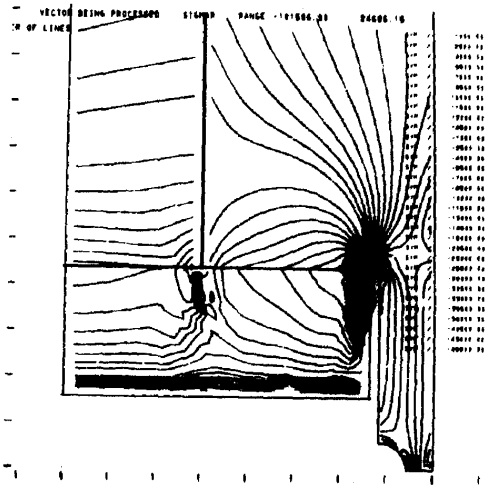


Fig. 8 - Example of CONTURS over large complex problem with SASL data

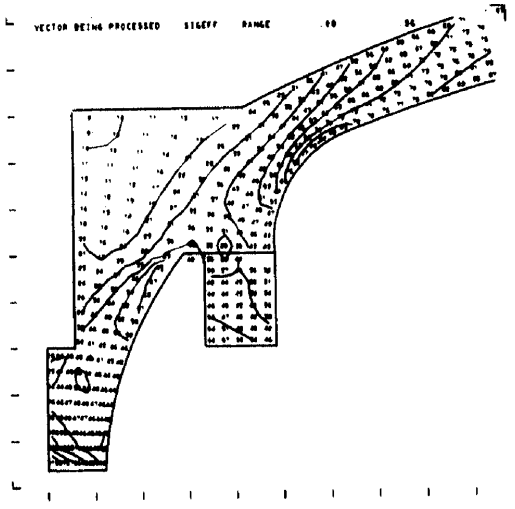


Fig. 9 - Example of CONTURS using divisor option with SASL data

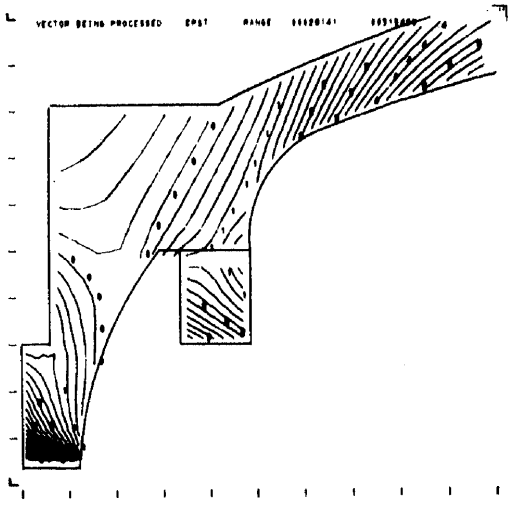
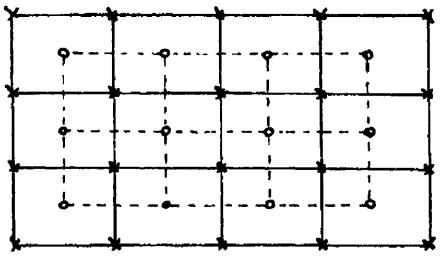
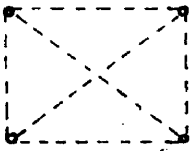


Fig. 10 - Example of contour identification option with SASL data

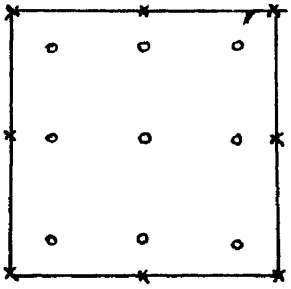


SASL CONTOUR AREA

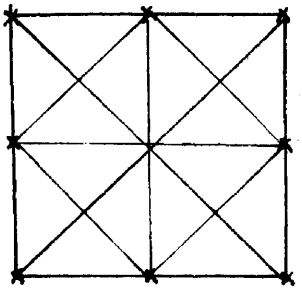


BASIC CONTOUR ELEMENT
4 TRIANGLES

Fig. 11 - Details of SASL contour requirements



Q8 ELEMENT



BASIC CONTOUR ELEMENT
16 TRIANGLES

Fig. 12 - Q8 element with nine integration points

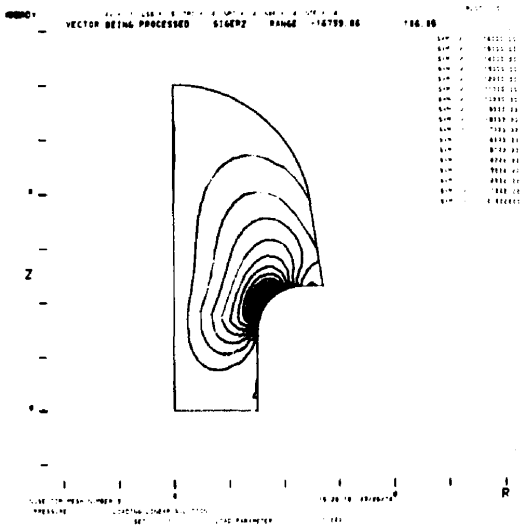


Fig. 13 - Display example of contour option using GNATS data

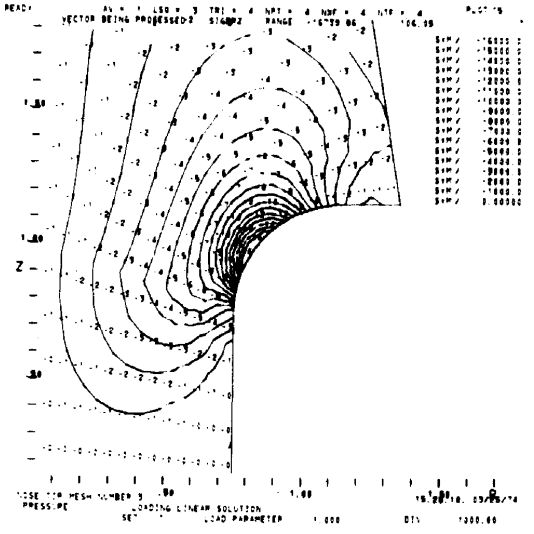


Fig. 14 - Windowed display of Figure 13 with divisor option

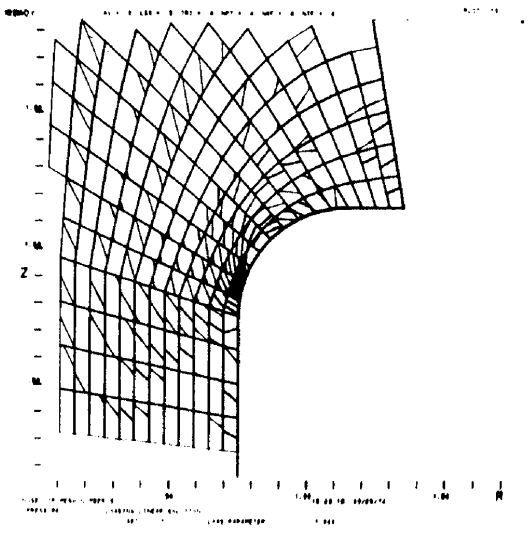


Fig. 15 - Example of contour result when function is not averaged at node point

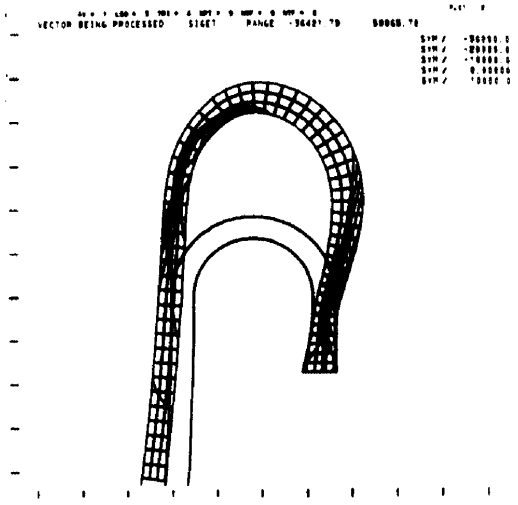


Fig. 16 - Example of contours on problem having large deflection

Bertram Bussell

The Inverse Window:
A Solution to Non-Rectangular Windowing in Interactive Graphics

1. WINDOWING

The operation of windowing in the field of interactive graphics has been widely explored [1, 2, 3, 4, 5]. Various hardware and software techniques for a range of windowing capabilities (sometimes referred to as scaling, zooming, or scissoring) have been studied; both two-dimensional and three-dimensional windowing capabilities have been derived. The purpose of a windowing procedure is to create a subset of real-world data which is of interest to the user and which he desires to display on a graphics terminal CRT; in this manner, a windowing procedure is essentially a "filter", removing from the real-world data set all data outside the area of interest of the user. Windowing does not reduce or enlarge the size of the picture on the CRT. Subsequent processing operations affect the size and orientation of the image on the CRT, while the windowing operation merely isolates the subset of real-world data that is of interest to the user.

The real-world area lying inside the window is defined by some window boundary. The boundary can be an arbitrary closed curve, possibly created by freehand sketching with a light pen or tablet. This general window boundary offers the most flexibility to the operator, since he is free to place the boundary in the real-world independently of geometric constraints which other techniques require. However, subsequent processing of this window boundary (to determine which real-world data is inside and which data is outside the window) is difficult due to the need for

---

*This research was supported by the U.S. Atomic Energy Commission, contract No. AT(11-1) Gen 10, Project 14 and is a portion of the research contained in [6, 7].

explicit specification of the boundary value at every resolvable point touched by it. Certain approximations to a continuous curve, such as replacement of the curve with a polygon, or approximation of the area bounded by the curve with contiguous rectangular windows, as discussed below, can simplify windowing calculations. No significant work utilizing an arbitrary boundary window has appeared in the literature to date.

A more formal way in which the window boundary can be specified is by a functional description. Functional specification is useful where the window area naturally fills a simple mathematically described curve (e.g. a circle or an ellipse), as might be the case for real-world data associated with optical scanning or processing. Functional specification can be thought of as a parametric version of an arbitrary boundary, with the advantage that coordinates along the boundary may be calculated.

A third type of window boundary specification is the use of an n-sided polygon. A general polygon is the most flexible, although an orthogonal polygon, one whose sides are parallel to the real-world data axes, offers reduced processing requirements. No general purpose windowing procedures have been developed for the polygon boundary. The general polygon is a useful approximation to the arbitrary or functional boundary; the approximation can be made as accurate as necessary by increasing the number of sides on the polygon. The orthogonal polygon is a special case of the general polygon, but the loss in generality is offset by the decreased processing requirements. The inverse window, described in section 2 below, uses an orthogonal polygon boundary.

Finally, the most commonly used type of window boundary specification is a rectangle, whose sides are parallel to the axes of the real-world coordinate system (a rotated rectangle becomes a special case of the polygon). This type of window forms the basis for most of the other window processing procedures. For example,

in the case of the arbitrary window boundary, contiguous rectangular windows could be concatenated to approximate the area bounded by the continuous curve, as an approximation to that region. Each elementary component, or rectangular window, is relatively easy to process. In general, the accuracy of the approximation is inversely proportional to the size of the rectangular window and therefor also proportional to the windowing time.

A single circumscribing rectangle is often a practical alternative to the continuous boundary, and in any case forms the basis for the multiple contiguous windows and the inverse window process. We define, then, as the first graphical element, the display window. The display window is some rectangular area, wholly contained within the real-world, which precisely circumscribes the graphic data of interest for display on the graphics terminal. The boundaries of this window are specified by the two X coordinates XO, XM, and the two Y coordinates, YO, and YM. These four coordinates are expressed in real-world dimensions.

An important consideration is the relationship of the window boundary values to the real-world coordinate system dimensions and values. Since the window boundaries are expressed in real-world coordinates, it is actually unnecessary to know the units or dimensions of the real-world data. In the process of extracting the real-world data which lies within the window, various arithmetic operations are performed on the ratios of the window boundary values to the real-world values. Hence, as long as all coordinates are expressed in identical units, there is no inherent requirement in the display system that data be expressed in any particular units.

Let us examine further the elementary aspects of windowing. Figure 1 illustrates what we call a basic window component, a line connecting two real-world points P_1 and P_2. In the illustrated case, the line connecting these two points intersects the display window boundary at a point P'_{12}. Since only the data within the

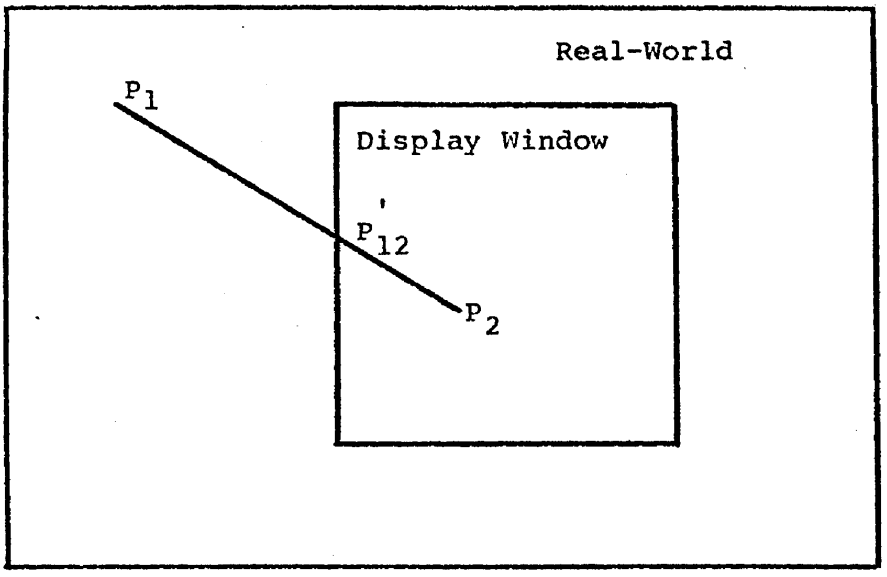


Figure 1 Basic Window Component

window is to be displayed, the graphics terminal should never need to "see" the line segment from P_1 to P'_{12}. The determination of window boundary points (WBP's), like P'_{12}, is the purpose of a windowing algorithm. The coordinates of WBP's must be determined to specify the visible segment to the graphics terminal.

The determination of window boundary points is not equivalent to so-called "edge violation detection" contained in some graphics terminals. Edge detection implies that all of the display data is being sent to the graphics terminal vector generator for processing, and that an error or attention condition arises when some element of this data attempts to generate a line which would fall off the physical boundaries of the CRT.

The process described here is much more flexible and more important to real-world data processing. In the first place, we are concerned with a virtual edge, the boundary of the display window, rather than a physical edge, such as the hardware limit of the CRT deflection system. Secondly, a system of this design will be processing large amounts of data; it might be normal for the display window to contain a small percentage of the total real-world data and we would like to reject unnecessary processing of data outside the display window; hence, we have a greater problem than just detecting boundary crossings. Simple edge detection schemes will not work in this environment.

Several more complex cases exist in windowing. Consider the line segments shown in Figure 2. The line from P_1 to P_2 crosses the window in two places; hence there are two WBP's for this line and the line from P'_{12} - P''_{12} is displayed on the CRT. Another important case arises when the segment just considered continues to P_5. Of the two initial lines P_1 - P_2 and P_2 - P_5, we are interested in viewing only the two segments P'_{12} - P''_{12} and P'_{25} - P_5.

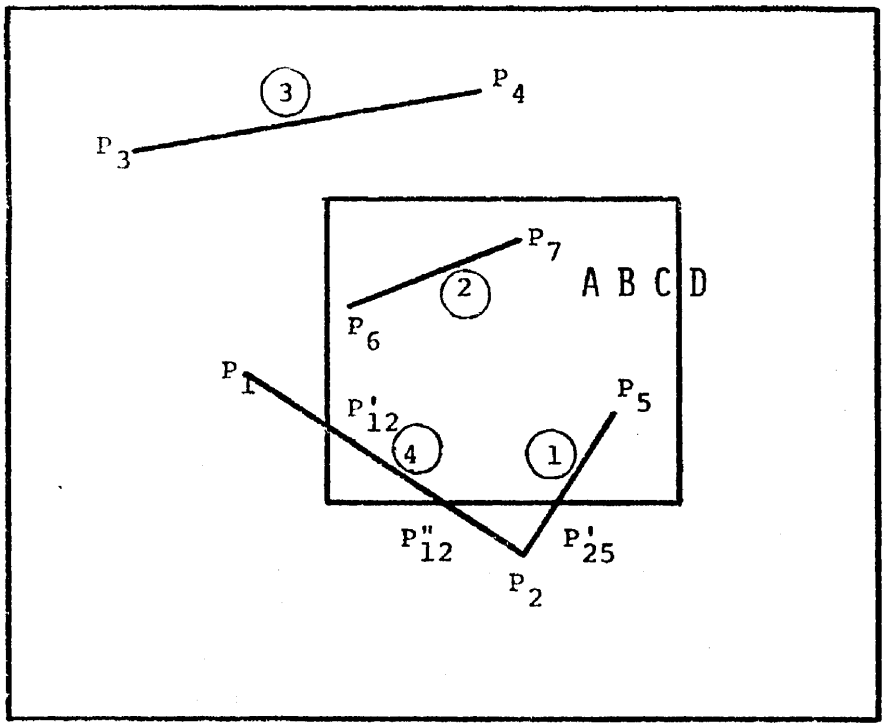


Figure 2 More Complex Window Components

It is common in many graphics terminals to specify only the next sequential endpoint of a line segment, and as a result the original real-world data is usually similarly structured, showing two line segments, one stretching from P_1 and P_2, and the second from P_2 to P_5. However, the windowing operation will produce <u>three</u> segments: The visible segment from P'_{12} to P''_{12}, the invisible segment from P''_{12} to P'_{25}, and the visible segment from P'_{25} to P_5.

Another important case is also illustrated. Line segment $P_3 - P_4$ lies entirely outside of the window. No sophisticated techniques are required here to determine window boundary points. However, in the many application areas where data is distributed uniformly over the real-world, the majority of the real-world data will probably fall outside of any particular display window. This condition implies that segment $P_3 - P_4$ will be the most frequently arising window component. Any algorithm for windowing must dispose of this component with minimum overhead.

The fourth type of line segment is one wholly contained within the window, such as $P_6 - P_7$. These four line segment cases exhaust all possible cases which a windowing algorithm must consider. Table 1 summarizes these cases.

Case	End Point 1	End Point 2	WBP_1	WBP_2
1	Inside Window	Outside/On Window	TBD	None
2	Inside Window	Inside Window	None	None
3	Outside/On Window	Outside/On Window	None	None
4	Outside/On Window	Outside/On Window	TBD	TBD

(TBD = To Be Determined)

Table 1 <u>Windowing Cases</u>

A possible fifth case, that of both end-points outside the window with the resulting line segment crossing the window boundary at a single point (which must, therefore, be one of the four corners), is handled as a degenerate form of Case 4; that is, $WBP_1 = WBP_2$.

Finally, the case of functionally specified graphical information which crosses the window boundary needs some consideration. This type of data, exemplified by the character string in Figure 2, is usually drawn on the CRT with the use of a special hardware generator. This generator normally has a rigid format; for example, it cannot draw an arbitrary portion of a character. Therefore, windowing this type of data involves determining the set of characters inside the window, the set outside the window, and the character, if any, which is on and clipped by the window boundary. Those characters outside the window are clearly discarded, but further consideration must be given to the character which lies on the boundary. Several alternatives exist. One is to delete this character, or the string containing it, altogether; however, if two contiguous windows share this same character, and each deletes the character, then the composite window will have lost some information without the knowledge of the user.

Another approach is to invoke a line-segment replacement procedure to draw on the CRT only the portion of the character within the window; in this manner, two contiguous windows would each display their portion of the character, and the composite would have no information loss. However, this scheme requires that a software subroutine be present to effectively simulate the hardware function generator; this is costly of time and high-speed memory, and a large data table is usually required to define the allowable character set.

A third alternative is to adopt the operating procedure that a character is displayed only if it falls entirely inside one or more windows. The special function generator can now be used to

create the character; however, for this convenience, a more global test must be made to determine if the window edge that intersects the character is an edge common with another window, and then if the character is contained entirely within the contiguous windows.

Each of these schemes, and indeed the entire process of piecing together contiguous rectangles, has the disadvantage that the final display list is fragmented and not easily related to the original real-world data set; in addition, certain functionally generated items may be lost (or expensively retained). A novel approach to this contiguous window problem is presented by the inverse window. The inverse window is defined to be some rectangular area in the real-world inside of which data is to be discarded, rather than preserved. When used in conjunction with a normal circumscribing rectangular window, inverse windows permit a polygonal-shaped area of the real-world to be displayed without the problems of special functionally-generated data outlined above.

Consider Figure 3, in which it is desired to display the data contained within a continuous window boundary within the real-world. The first process is to develop the circumscribing rectangular window, as illustrated. The data within this window is extracted from the real-world data set, and is used to form a sub-real-world within which further windowing operations will occur.

The continuous boundary is then approximated, to any desired level of accuracy, by orthogonal line segments forming a closed orthogonal polygon. The corner points of this polygon define the coordinates of a series of inverse rectangular windows, indicated by shaded areas in Figure 3, in which the data from the subreal-world is to be removed and discarded; the remaining data represents, to a close level of approximation, the data from the original real-world which is contained within the continuous boundary. The line and function data within the polygonal boundary is preserved

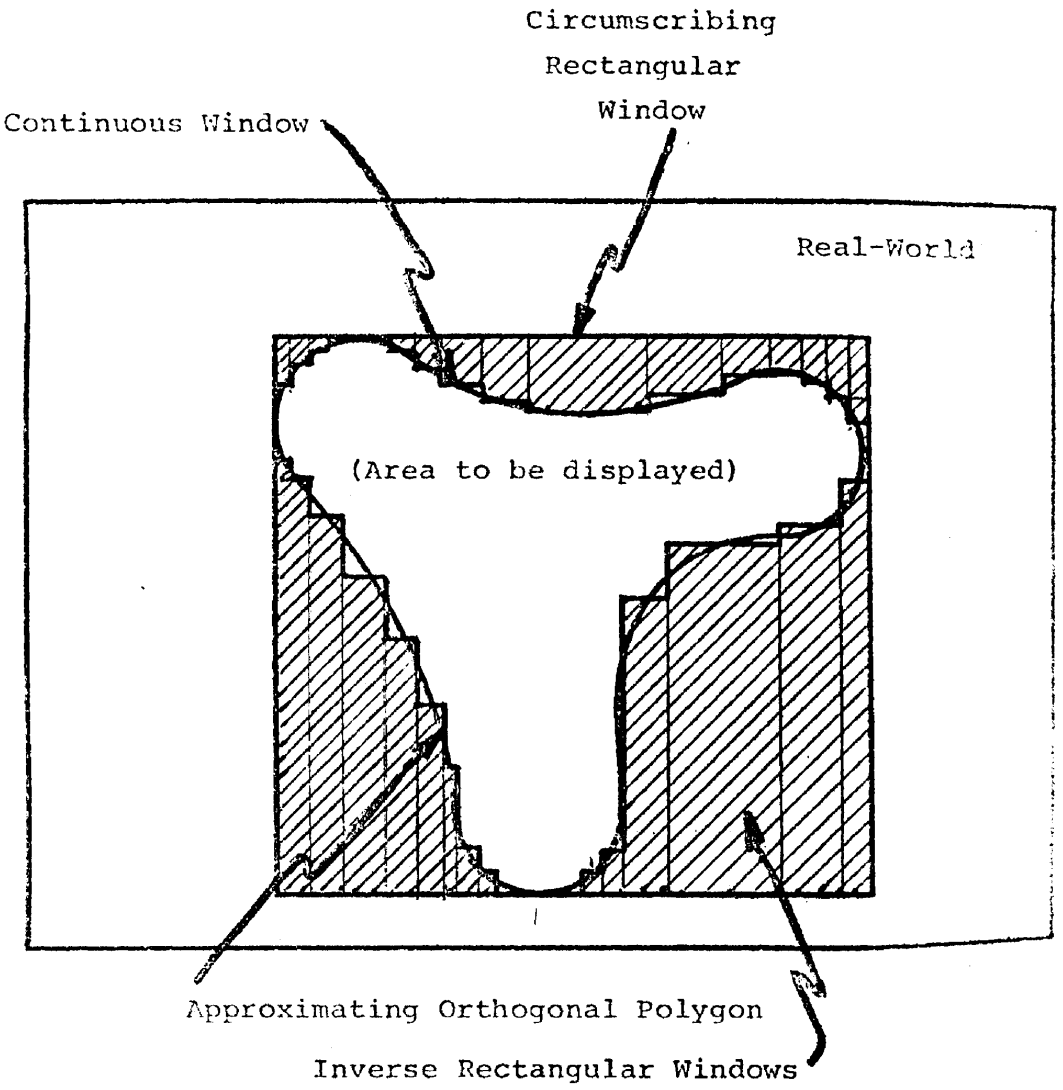


Figure 3 Inverse Windows

in data structure; that is, a single straight line in the real-world is still a single line in the display list, rather than the several lines which would have been generated by the use of contiguous rectangles.

A factor which makes the inverse window approach even more suitable for approximating continuous or polygonal window boundaries is the ease with which the conventional rectangular window algorithm can be modified to process the "inverted" case. This technique is explored in Section 2.

2. THE INVERSE WINDOW [6]

The concept of inverse windows is illustrated in Figure 4. First, the entire real-world data set is passed through the normal windowing operation to derive a subreal-world data set; this first windowing operation is identical to the normal rectangular window described in the previous section. In the case of inverse window processing, this first rectangular window is referred to as the circumscribing rectangular window. It is this circumscribing rectangle which is depicted in Figure 4.

Within the circumscribing rectangle is some orthogonal polygon whose sides are either horizontal or vertical. By definition, the orthogonal polygon which defines the area of data to be preserved is exactly circumscribed by the circumscribing rectangular window. Therefore, the orthogonal polygon will have one of its sides co-linear with each of the four sides of the circumscribing rectangle. We can then derive a set of inverse rectangular windows, the set $\{R_i\}$, which covers the area within the circumscribing rectangular window containing data to be discarded (shown as the shaded area in Figure 4).

The inverse window processor can be treated simply as another type of windowing processor, substitutable for the common rectangular window processor. This relationship is illustrated in Figure 5,

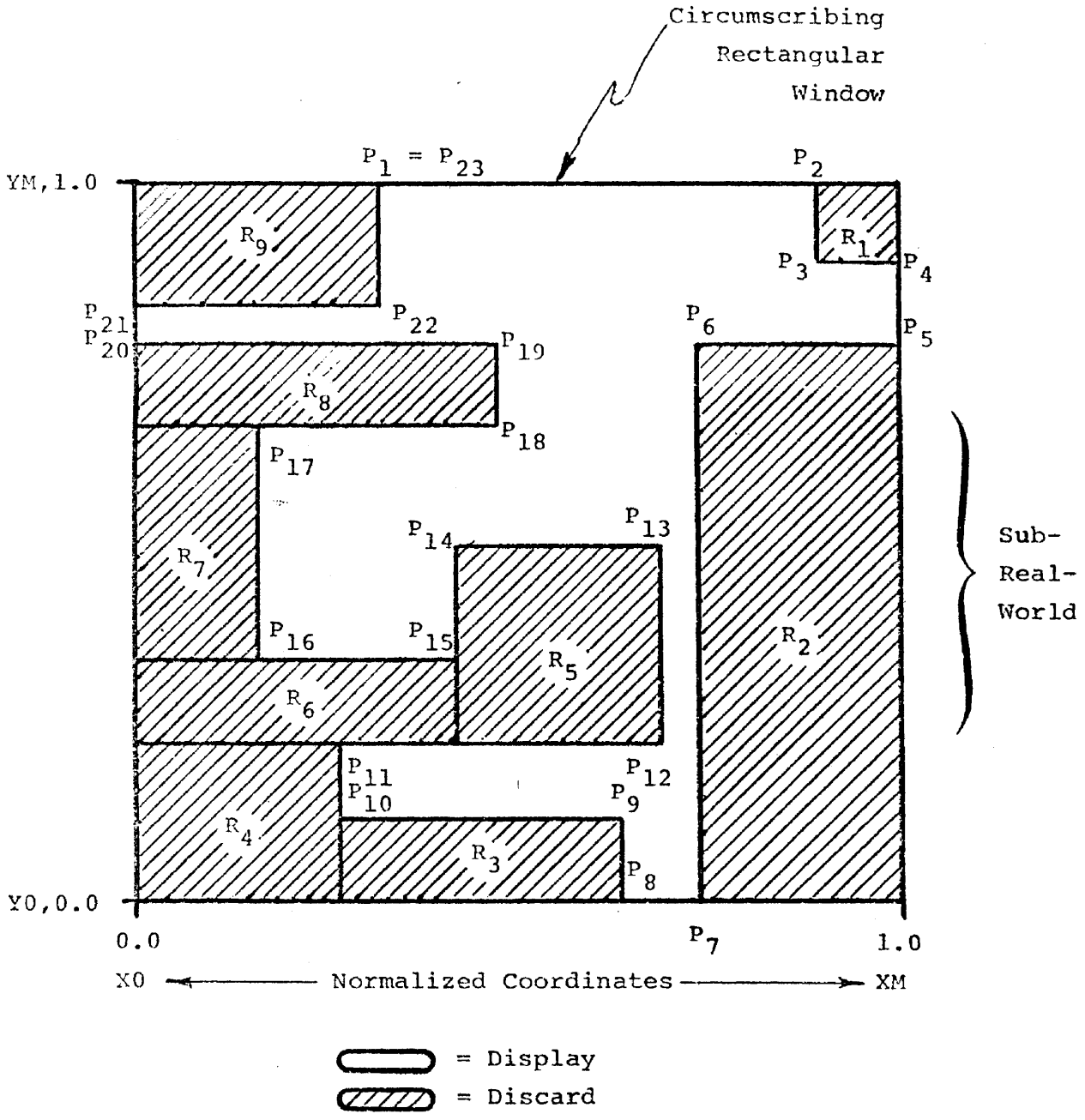


Figure 4 Inverse Window Set

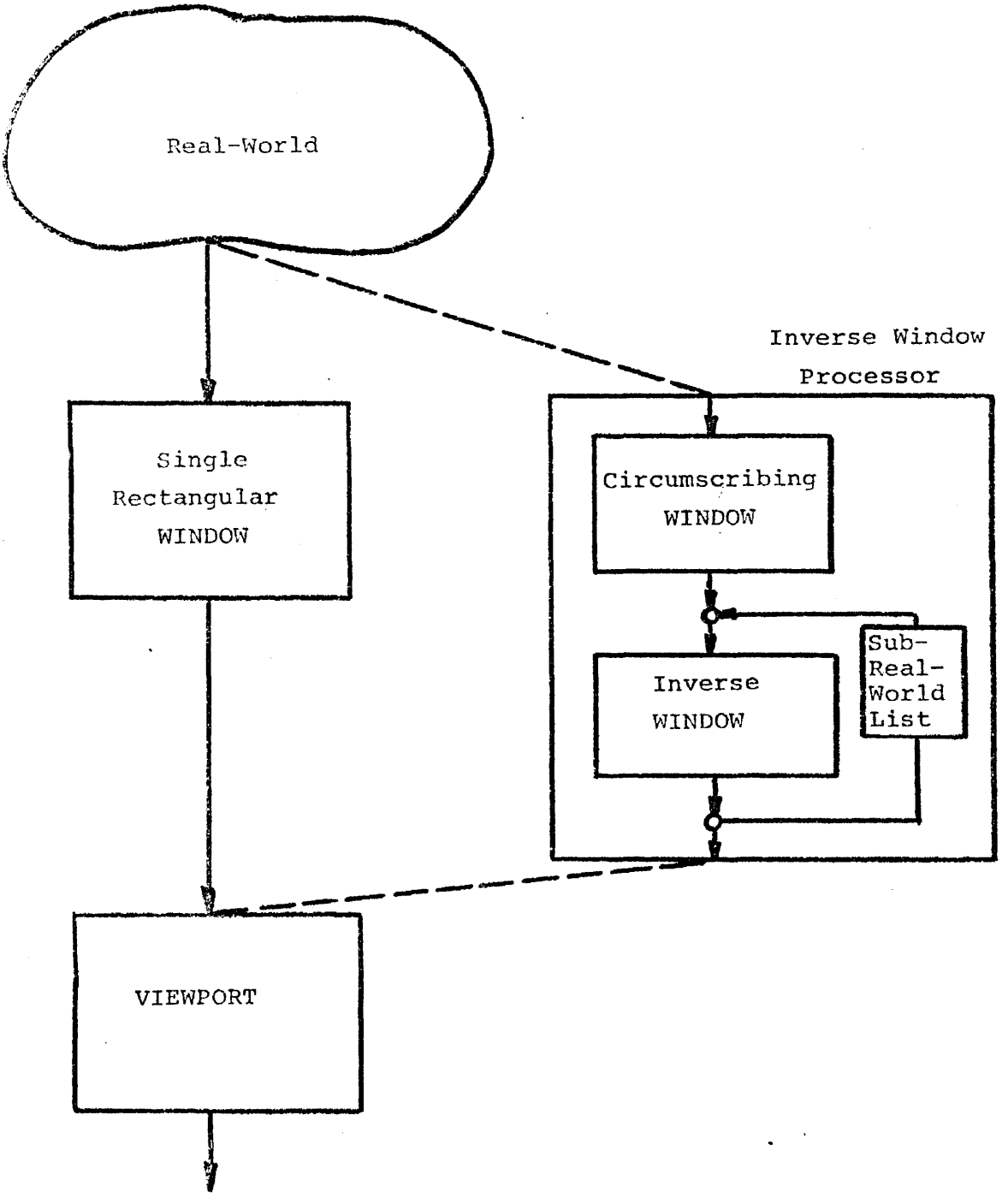


Figure 5 Inverse Window Processor Module

where the single rectangular window processor is shown as an alternative to the inverse window processor. Within the inverse window processor, there are several sequences of operations. First the circumscribing window must be determined. This isolates the subreal-world data set and is passed on to the viewport processor. Next, the set of inverse windows which exactly covers the area of data to be discarded must be derived (that is, the set $\{R_i\}$ in Figure 4 must be found). Finally, the data within each rectangular inverse window must be determined and discarded, leaving in the end only the data within the polygonal window for display on the CRT.

The boundary for the orthogonal polygon may be determined in one of several ways. It might be specified by interactive input, wherein a light pen or tablet is used to specify the set of points P_i which define the polygon; an input verification program would be used to guarantee that all input segments are either horizontal or vertical, and that the set is closed.

The boundary for the orthogonal polygon may be specified by
a) interactive input
b) the graphic system or user program
c) approximation to a more general window boundary.

To derive the set of rectangles $\{R_i\}$ which covers the area of data to be discarded requires a preprocessing of the list of points which defines the boundary polygon. This preprocessing insures that the list of boundary points is ordered; e.g. ordered in a clockwise fashion, with the first point representing one of the points on the circumscribing rectangle (such as point P_1 in Figure 4). In addition, the list must be closed, so that the last in the list is identically equal to the first entry. Finally, no two sides of the polygon are permitted to intersect (that is,

only a single, connected area within the polygon is treated here).

Proceeding clockwise around the polygon, it is then possible to determine which quadrants of the coordinate system (centered at each successive point of the polygon boundary) are inside or outside the area in which data is to be preserved. Using this information, it is possible to derive a procedure for "walking" around the polygon boundary to determine, at each point in the boundary data list, the appropriate inverse window(s). As each rectangle is tentatively determined, a check must be made to determine how far that rectangle can extend before intruding into the area which is to be preserved. For example, the rectangle R_5 in Figure 4 must be constrained so that it does not extend into the display area bounded by the points P_9, P_{10}, P_{11}, and P_{12}. It is possible, of course, to overlap the inverse windows when necessary.

Once the set of inverse windows has been determined for the given polygon, the actual inverse windowing operation can begin. As the first step, the data within the circumscribing rectangle is extracted from the real-world and forms a data set termed the subreal-world. The data within this subreal-world is maintained entirely in normalized coordinates, since that data set is just the output of a normal rectangular window processor.

The actual process of inverse windowing is easily implemented with slight modification to the windowing tools generally present. In each data case, the inverse window performs the dual of the operation performed by the normal window processor. Because of this dual characteristic, only slight modifications of the software and hardware algorithms which implement the normal rectangular window processor are required. These modifications consist of accepting line segments instead of discarding them (and vice-versa), and by-passing the scaling operation at the output of the window processor when the processor is in the inverse mode. By not scaling

the output, the output data is preserved in the same coordinate system as the input data; since this data is already the normalized data set from the circumscribing window, the final output set after i iterations (one for each R_i) through the inverse window processor will also be in the same normalized coordinate system, as required for subsequent viewport processing.

Timing comparisons were made between clipping with hardware and a software clipping divider on a 360/91 [7]. The cases listed refer to those in Table 1. All times are in milliseconds.

Case	Remarks	Time (ms) Hardware	Software Normal	Software Inverse	Software Time / Hardware Time
1	non-orthogonal	.01675	1.04	1.01	60
1	orthogonal	.01675	1.02	1.01	61
2	trivial acceptance	.0075	.098	.09	13
3	non-trivial rejection	.00325	.15	.15	46
3	trivial rejection	.00075	.090	.093	120
4	non orthogonal	.01675	2.03	1.96	121
4	orthogonal	.01675	1.98	1.96	118

Table 2

These results indicate that the hardware algorithm is approximately two orders of magnitude faster than the software implementation provided here.

An illustration of the effect and operation of the inverse window process is illustrated in Figure 6 [7]. Here, the real-world data is represented by a neural network in Figure 6a. The super-

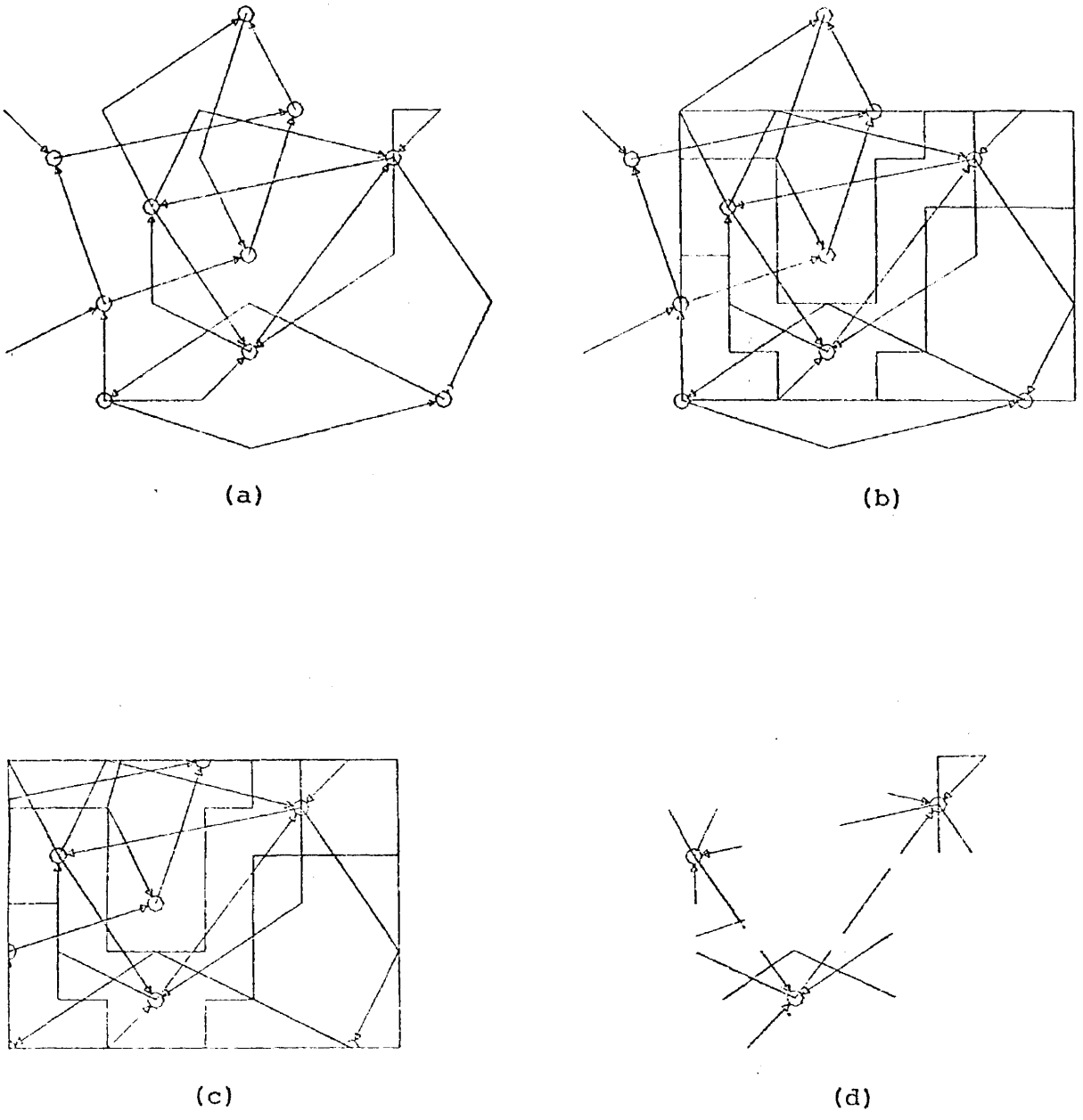


Figure 6 Inverse Windowing Example

posed non-rectangular window and circumscribing rectangle is shown in Figure 6b. Figure 6c shows the clipping outside of the rectangle. The complete windowed display is shown in Figure 6d. Under software implementation, CPU time for this example was under 500 milliseconds. It is reasonable to assume that the hardware implementation would be two orders of magnitude faster.

Inverse windowing represents a significantly new technique for determining the contents of non-rectangular windows. The alternate technique of using contiguous rectangles within the area to be preserved is much less desirable because it generates a longer display list containing segmented real-world vectors. Another alternate technique which should be mentioned is a "pure" computation of the intersection between a line segment and the polygon (or continuous) boundary. This is clearly a costly, "hidden-line" processing procedure, and does not make use of existing hardware or software which is necessary and available for the normal window operation. The author has been informed [8] of another proposed procedure which permits convex polygonal windows. In this procedure the data is rotated until sides of the window are parallel to one of the screen's coordinate axes. The clipping procedure is then applied to a half-space. One rotation and clip for each polygonal side finally removes all extraneous vectors. No results or reports of this procedure have been seen. Although novel, this windowing technique is not as versatile as that described here, where convexity of the polygon is not at issue.

REFERENCES

1. Coggan, B. B., "The Design of a Graphic Display System," UCLA Department of Engineering Report 67-36, August 1967.

2. Sproull, R.F., and Sutherland, I. E., "A Clipping Divider," Proceedings of the FJCC, 1968.

3. Sutherland, I.E., "Sketchpad, a Man-Machine Graphical Communi-Cation System," Lincoln Laboratory Report 296, MIT, January 1963.

4. Thornhill, D. E., et al., "An Integrated Hardware-Software System for Computer Graphics in Time-Sharing," Report Number ESL-R-356, MIT Electronic Systems Laboratory, December 1968.

5. Newman, W. M. and Sproull, R. F., Principles of Interactive Graphics, McGraw Hill Book Company, New York, 1973.

6. Taxin, H., "Interactive Graphics in the Computer Aided Design of Digital Systems," Ph.D. Dissertation, UCLA, Dec. 1970.

7. Buchness, R., "Non-rectangular Windowing Using Interactive Graphics," M.S. Thesis, UCLA, Dec. 1973.

8. Private Communication from W.M. Newman.

APPLICATIONS OF COMPUTER-GENERATED PERSPECTIVE PLOTS*

by

Melvin L. Prueitt

Los Alamos Scientific Laboratory
of
The University of California
Los Alamos, New Mexico 87544

*This work was performed under the auspices of the U.S. Atomic Energy Commission

Key Words and Phrases: Perspective Plots, Hidden-line Removal, Computer Graphics.

APPLICATIONS OF COMPUTER-GENERATED PERSPECTIVE PLOTS

By

Melvin L. Prueitt

INTRODUCTION

The most important characteristic of the modern computer is its ability to handle large quantities of numbers quickly. But this very feature poses a problem to slow-witted man. No one can assimilate the vast volume of information which the computer can generate. The problem is compounded by the fact that coming generations of computers will be even faster.

One solution, which is largely used today, is to selectively print out only the most important items of information. Even then the quantity of data printed is often far too large, and much of it is never read. Furthermore, the quality of the output is usually not such that it is readily assimilated by the analyst. That is, human beings were not designed by nature to perceive relationships among numbers in a printed table.

But the visual system was designed to translate line drawings into subjective "reality." The mental hardware almost instantly translates a curved line on a two-dimensional surface into a form which allows the observer to perceive relationships among various points of the curve. It is true that some precision is lost in going from a table of numbers to a plot of the same numbers, but roughly two orders of magnitude more information is presented to an observer by a curved line than by a single number.

Even two-dimensional curves are often insufficient to represent the

large amount of data that needs to be displayed. Two orders of magnitude more information than the simple curve may be presented (with somewhat less precision) by a perspective projection of a three-dimensional surface. The visual portion of the human brain incorporates the necessary hardware to reconstruct a three-dimensional surface from a two-dimensional perspective drawing.

This visual hardware is more adept at interpreting perspective plots than isometric plots. Figure 1 compares an isometric plot (a) with a perspective plot (b). The isometric plot is a representation of the well known "optical illusion stairway." In attempting to reconstruct the figure, the brain finds that it is ambiguous. The logic circuits of the brain then presents to the consciousness an inverted view of the stairway. Not finding that view superior to the first one, it switches back. Figure 1(b) is much more satisfying to the human optical system. It doesn't have to work so hard in interpreting the geometry.

In cases where one is not troubled by the ambiguity of an isometric plot, proper interpretation may still be difficult. Objects in the background appear too large in relation to features in the foreground. The effect is familiar to anyone who uses a telescope or binoculars. Greater magnification in a telescope implies closer approach to an isometric image.

Since the computer can produce a perspective plot just as easily as an isometric plot, it is recommended that the former be used.

A desirable feature of perspective plots is the removal of hidden lines. Figure 2 illustrates the confusion that can arise when hidden lines are not removed. With all lines present, the picture takes on the characteristics of an X-ray photograph. By removing the hidden lines,

the figure gives the appearance of an opaque surface. Some information is lost, but comprehension is gained. Hidden portions of the surface may be studied by having the computer rotate the surface.

APPLICATIONS

The principal value of perspective plots arises from the fact that man can quickly evaluate the shape of a surface. He can see trends and relationships that might be difficult or impossible to perceive in a set of numbers. The following examples are offered to inspire greater use of perspective plots as computer output.

At the Clinton P. Anderson Meson Physics Facility at the Los Alamos Scientific Laboratory a magnetic field was generated in the path of the accelerator beam. The question arose: How good was the magnetic field? About 25,000 measurements were made at various points in the field volume to determine the characteristics of the field. But what can one see from 25,000 numbers? The problem was resolved when Professor Gordon Lind from Utah State University (who was visiting at Los Alamos) fed the numbers into a computer and used our PICTURE program to plot them. Figure 3 shows one component of the magnetic field over the area of a "slice" taken through the field volume. This makes an interesting plot, giving the "shape" of the magnetic field. But the surprising features of the plot were the unsuspected striations running across the field. Without the plot, the striations probably would have gone undetected until some serious anomaly occurred in later operation. The eye picks it out immediately in the plot.

Dr. Marvin Mueller of LASL used PICTURE to plot calculated laser absorption on a material as a function of angle of incidence (left to

right in Figure 4) and depth of penertration (front to back). He could not only see the over all absorption characteristics at a glance, but he observed for the first time, ripples along the ridge which denoted interference phenomena.

Sometimes one is fooled by examining isolated values in a two-dimensional array of numbers. Dr. Robert Rowell of the University of Massachusetts sent me some computer-generated data on Mie scattering to be plotted. He also sent a sketch, shown in Figure 5(a) to give a rough idea of the way the final plot should look. After examining the sketch, I assumed that the surface would gradually change from a flat shape to an undulating shape from front to back -- that there would be valleys running from front to back. I was surprised when I saw the computer plot of the data (Figure 5(b)).

Figure 6 is a plot of CN cross-sections as a function of photon frequency and temperature made by Dr. Athel Merts and Norman Magee of LASL. The data for the plot were generated as part of their work in determining cross-sections in stellar atmospheres. Such plots are useful not only for study of the physical behavior of matter and energy, but they are valuable tools of communications for describing results to colleagues.

Figure 7 also exemplifies the pedagogical value of perspective plots. Dr. Paul Stein of LASL worked with three chemical equations for gases which were recognized to be conic equations. Dr. Carson Mark worked out the geometrical interpretation and showed that they could be represented by cones with parallel axes. The simultaneous solution of these equations represented the chemical equilibrium of the chemical solution. The question arose: Did the set of simultaneous equations have a solution? That

question could be answered by examining the corresponding cones. That is, do three cones whose axes are parallel intersect in a point? It was not immediately obvious to many people that the cones do intersect in a point. With a plot, Dr. Stein could point out to others how they intersect. Then it became obvious.

This figure also shows how the computer can be used to generate patterns on the surface. Patterns can be used to identify various parts of the surface when a color plotter is not available.

Stanley Marsh of LASL wanted to know the shape of fragments of material as they traveled away from the center of an explosion. He took a high speed radiograph of such particles and scanned the negatives with a densitometer. The resulting numbers were fed to a computer and plotted. Figure 8 is actually a plot which shows the thickness of a particle moving at 20,000 kilometers/hour. This provides some idea of the shape of the particle.

Professor Gordon Lind made some plots of the nuclear spectra for several isotopes. In Figure 9 the energy increases to the right. From foreground to background, each row of three lines represents a different isotopes. The physicist can use such plots to study the relationships among energy levels in different nuclei.

Some people use perspective plots to detect errors in large computer programs. Dr. Don Baker and Lawry Mann plotted some 6000 values of computer calculations from a large plasma program. It is rather difficult to scan 6000 numbers for each of several variables and for each of many time-cycles to detect misbehavior in the computation. In Figure 10, the "canyons" around the periphery of the plot were immediately spotted to be improper calculation rather than real physical phenomena.

Often one single point on a plot projects far above the others, signifying an error. One point would be very difficult to find in 6000 numbers.

Mathematicians find perspective plots useful for displaying functions of two variables. Figure 11 shows a step function and its two-dimensional Fourier Transform calculated by Dr. James Breedlove and George Wecksung of E G &G. Figure 12 represents the summation of several bivariate normal distribution functions.

Geologists would find perspective plots useful for displaying underground formations. Figure 13 is a simulation of the earth's surface above and a geological layer below. Besides the use of such plots for his own study, the geologist could use them as audiovisual aids in describing his conclusions to colleagues.

Perspective plots provide a more familiar rendering of terrain than topographical maps. Figure 14 is a topographical map of the ocean floor of the Bering Sea. Daniel J. Brown of NORFISH at the University of Washington produced the plot of Figure 15 which shows the same region of the ocean floor in perspective.

Daniel Brown also sent me the plots shown in Figure 16 depicting mountain terrain and the location of a planned highway.

Besides the foregoing examples of applications which involve data as a function of two variables, perspective plots may also be made of data which are normally considered to be functions of one variable by utilizing another variable such as time. For example, if one has a shock wave data relating relating pressure with distance along a shock tube, time may be chosen as a second variable and the perspective plot would provide the history of the pressure waves on one plot.

If one has a function of three variables, such as a magnetic field

in space, a "slice" may be taken and the magnetic field intensity would represent the height of the plot while the two space coordinates would be represented by the base dimensions of the plot (see Figure 3). Then if the slice is moved through the field in the direction normal to its surface, a movie of the perspective plots could be made giving the magnetic field strength throughout the space. Alternatively one might want to plot the shape of a surface which has a constant magnetic field.

CONCLUSION

Computing power is largely wasted when the output consists of printed numbers. In order to make better use of our computers, a concerted effort should be made to force the computer to communicate to us in terms which we naturally understand: pictures. With graphical output, the investigator gets at the solutions faster and understands them more clearly. Perspective plots are presently the best way to communicate large amounts of information to man from the computer.

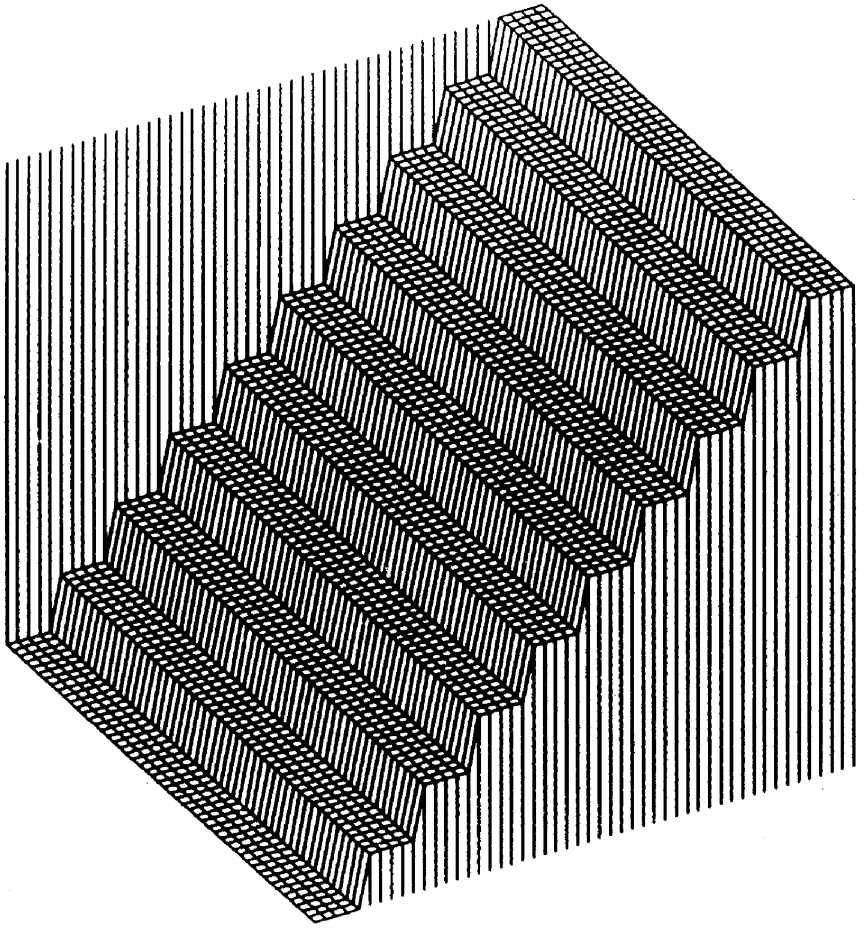


Figure 1. (a) Isometric projection of optical illusion stairway.

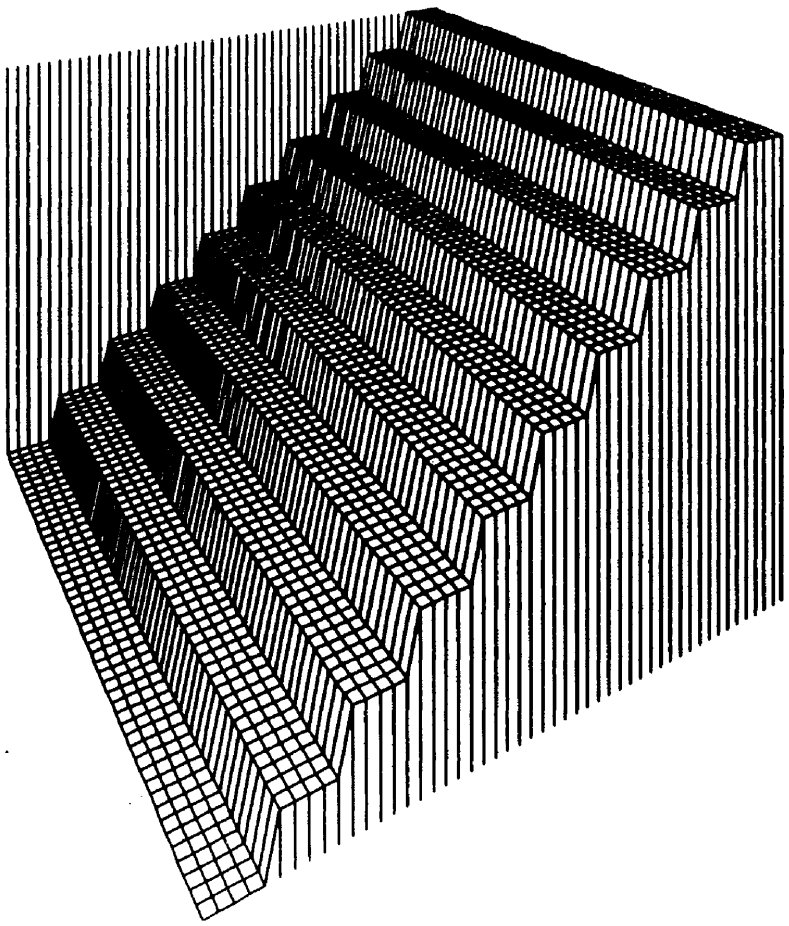


Figure 1. (b) Perspective projection of stairway.

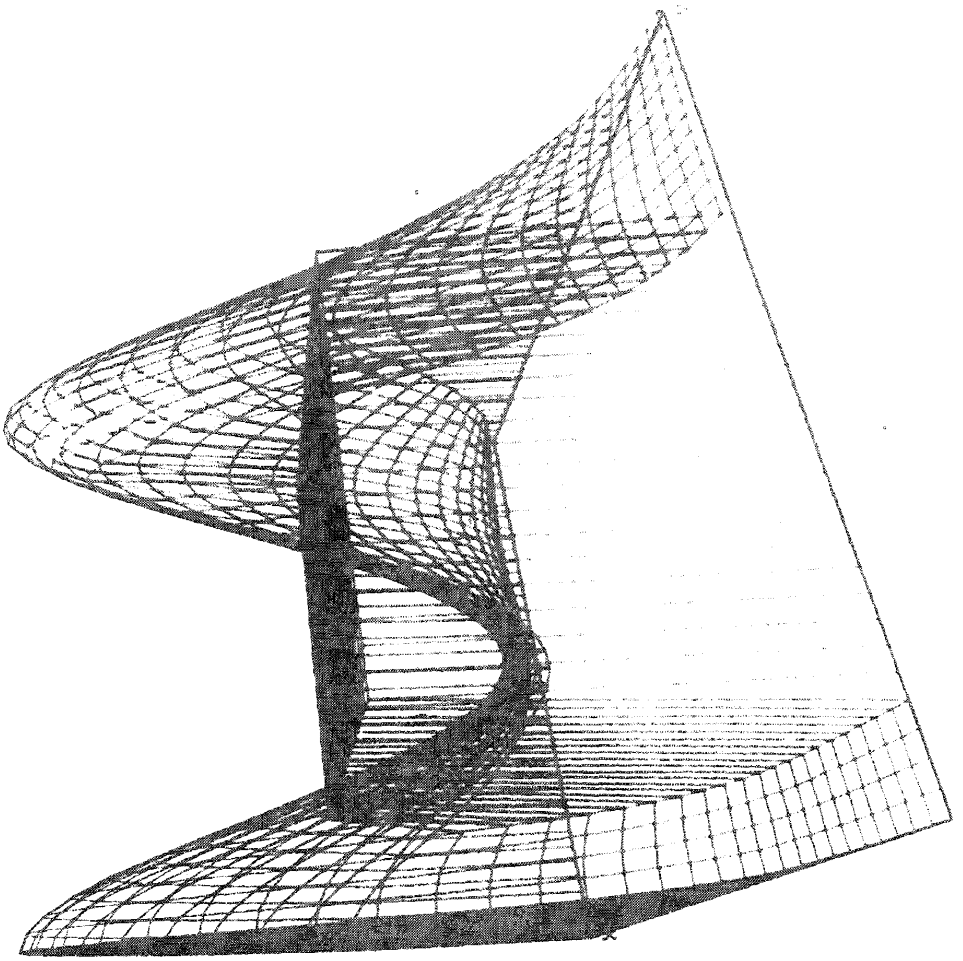


Figure 2. Perspective plot with hidden lines not removed.

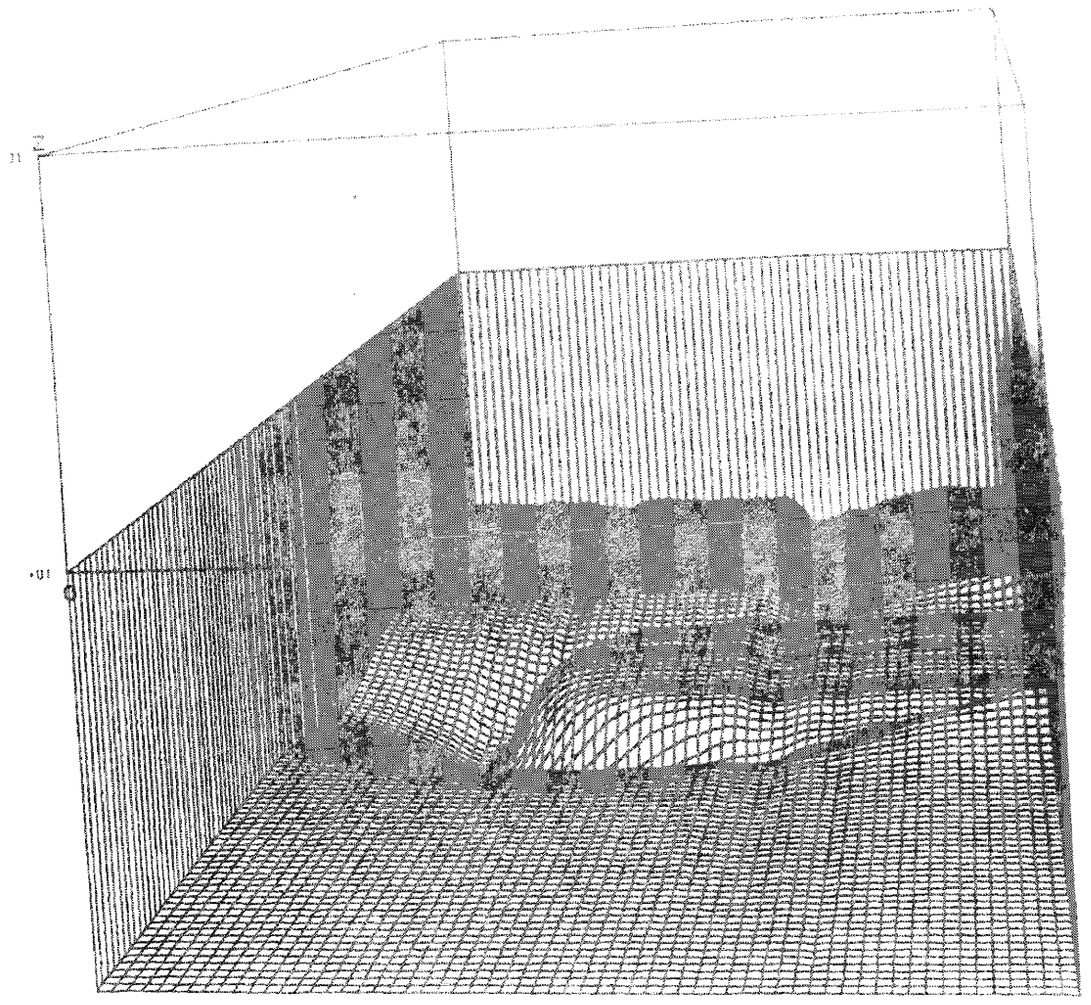


Figure 3. Plot of one component of a magnetic field.

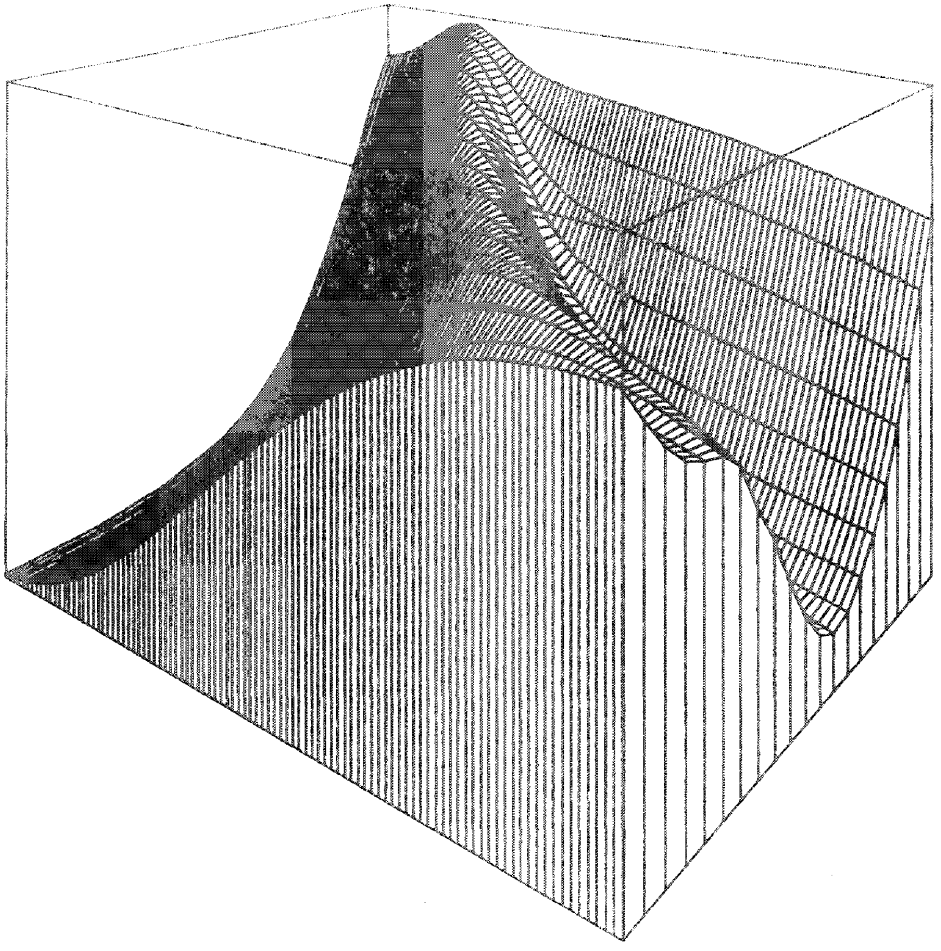


Figure 4. Laser absorption in a material.

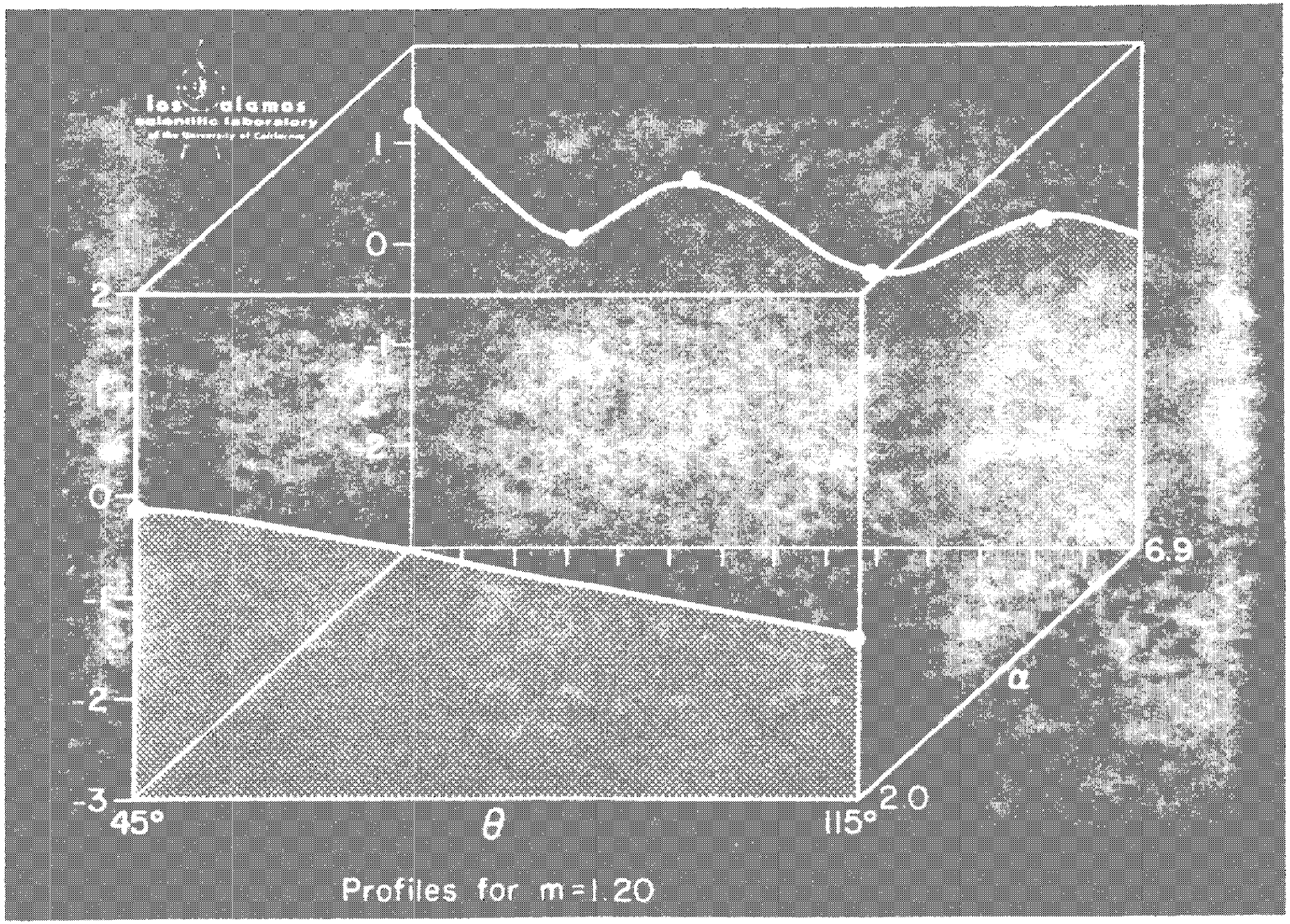


Figure 5. Mie scattering: (a) Isometric plot of a few points.

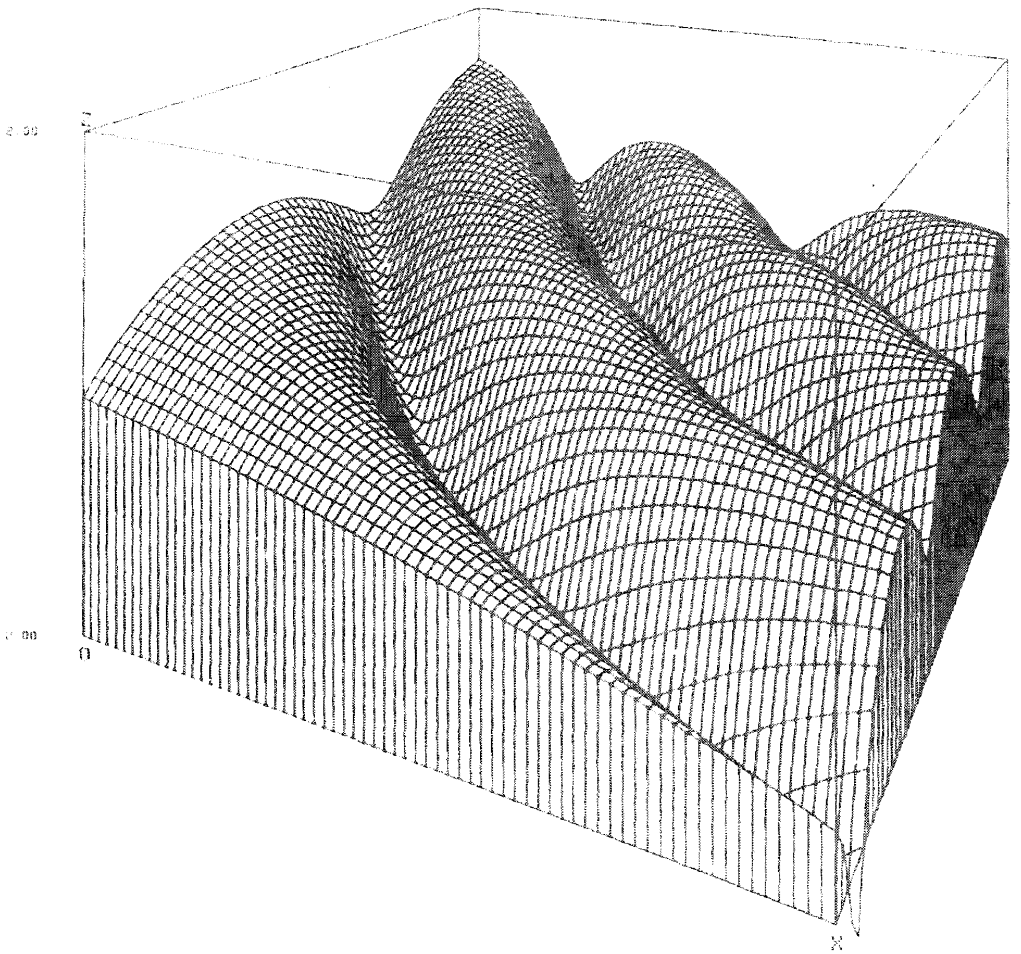


Figure 5. Mie scattering: (b) Perspective plot of 3500 points.

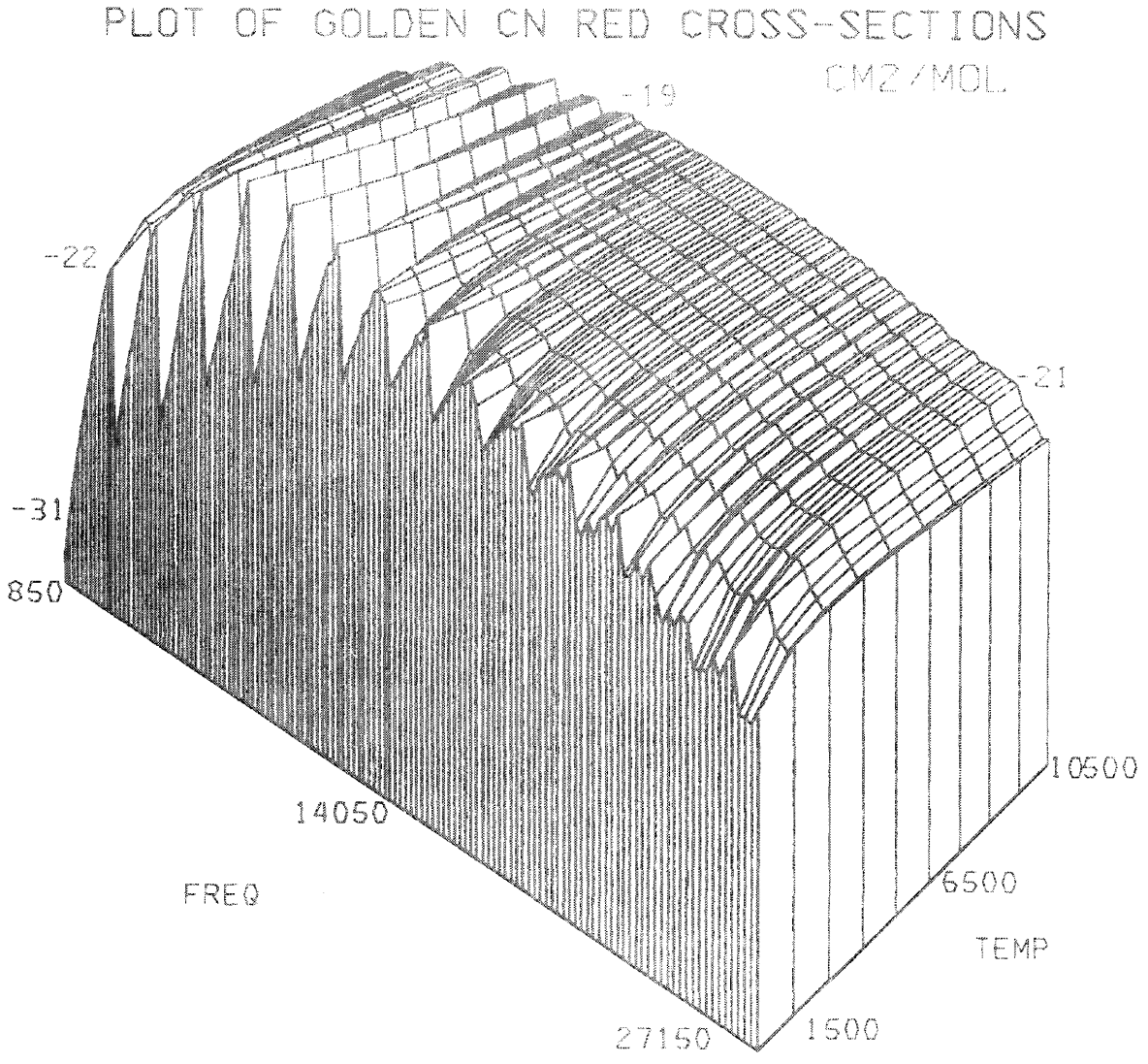


Figure 6. CN cross-sections.

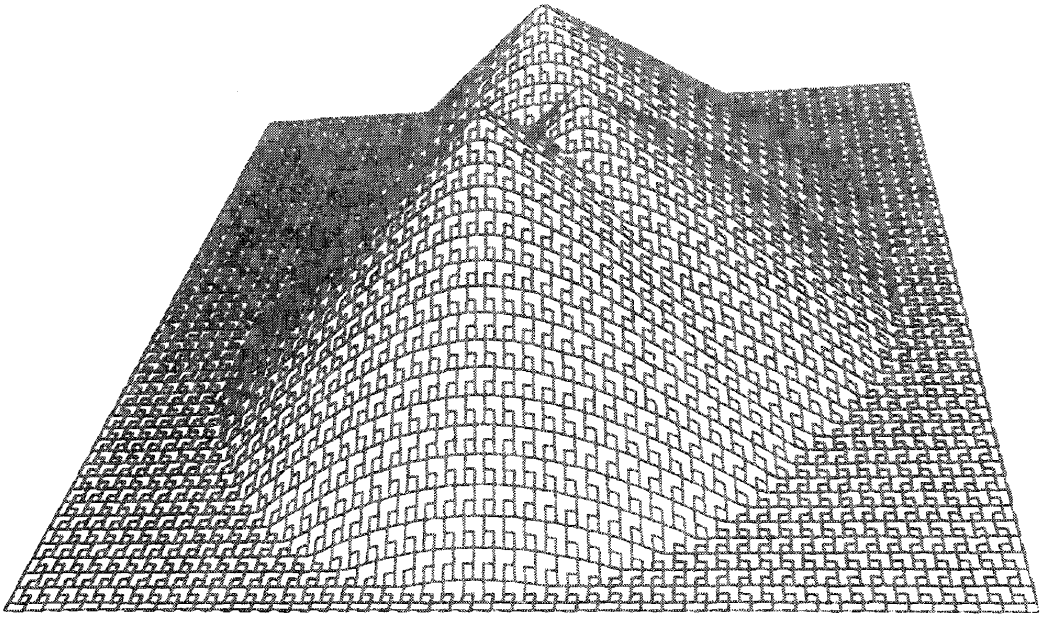


Figure 7. Cones representing chemical equations.

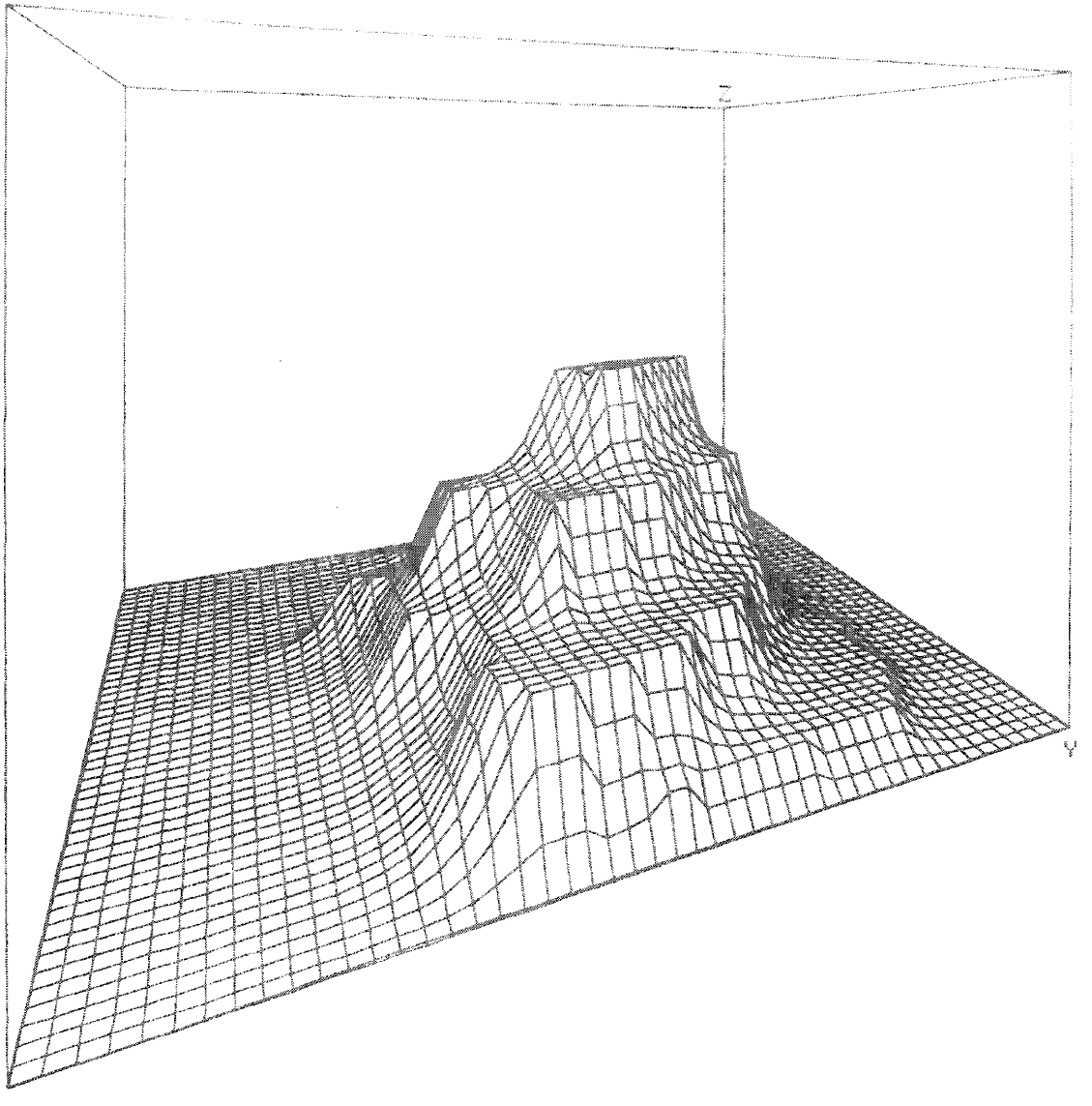


Figure 8. Thickness plot of a explosion fragment taken from a densitometer reading of a high speed radiograph.

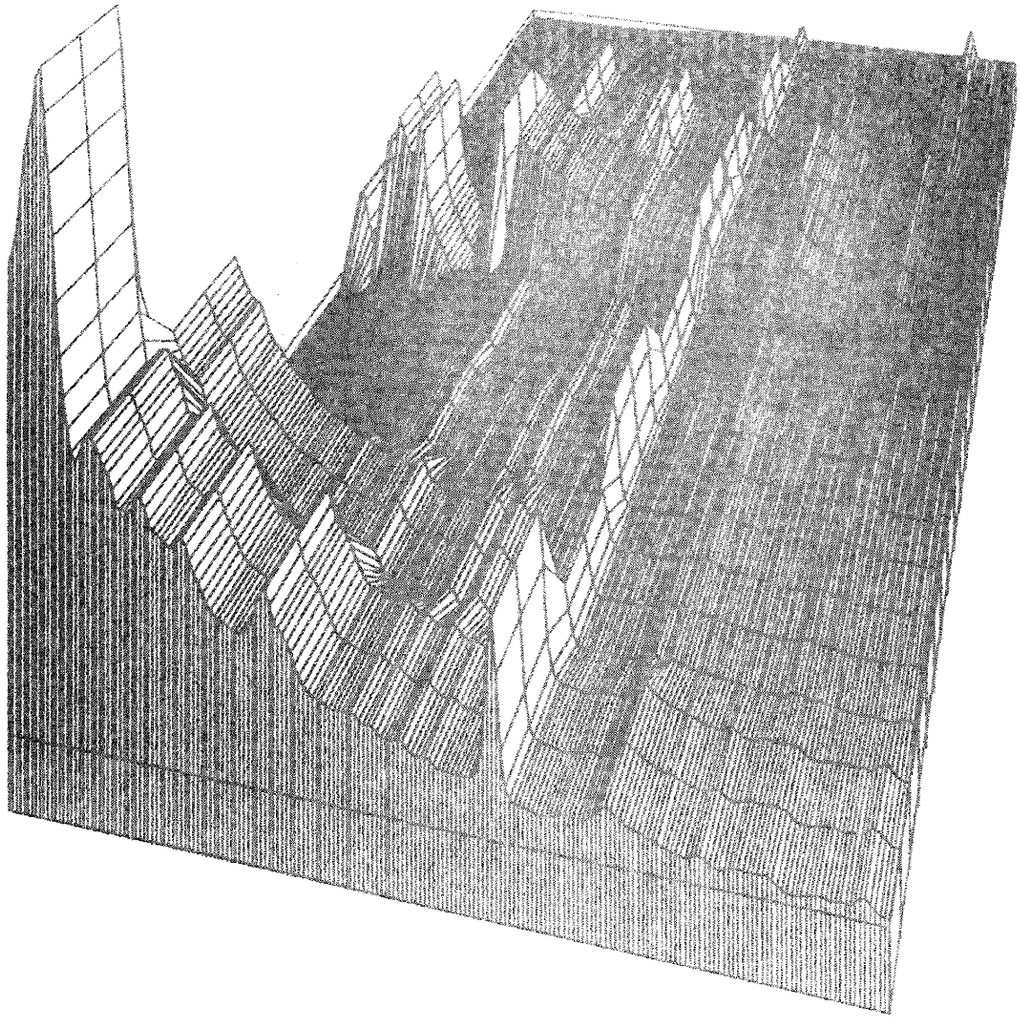


Figure 9. Nuclear spectra of B isotopes.

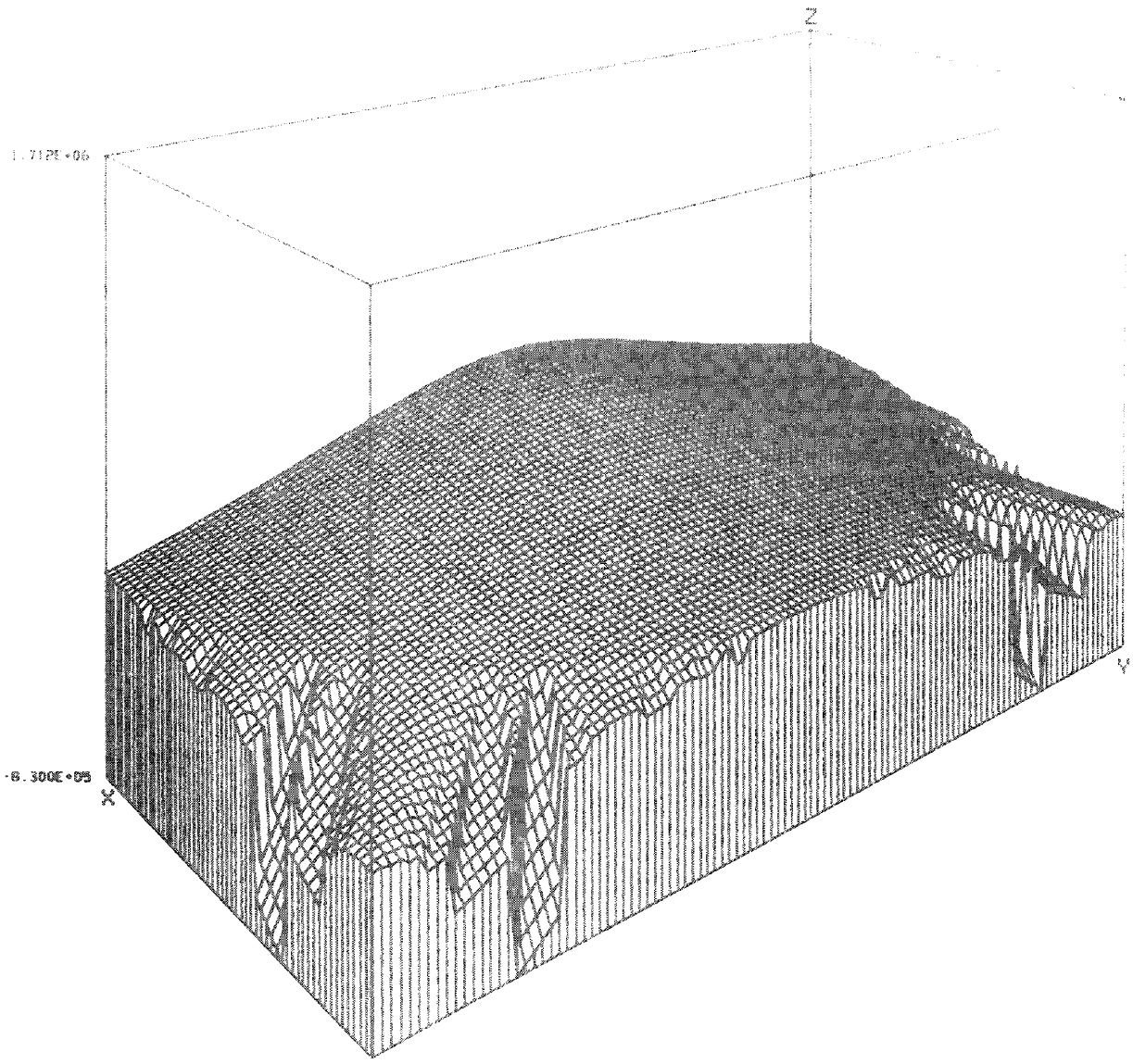


Figure 10. Calculation errors showing up on a plot of the gradient of a magnetic field in a dense plasma.

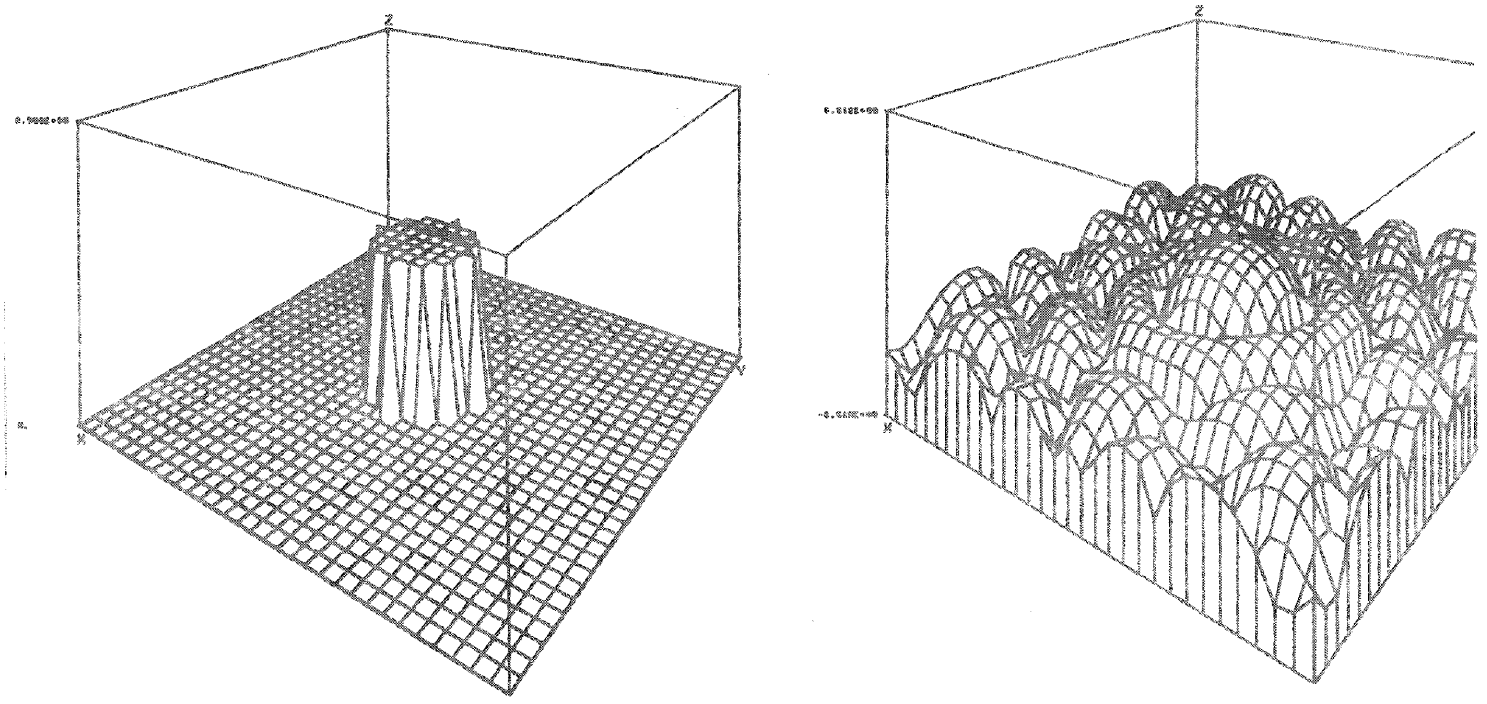


Figure 11. Step function in two variables and its Fourier transform.

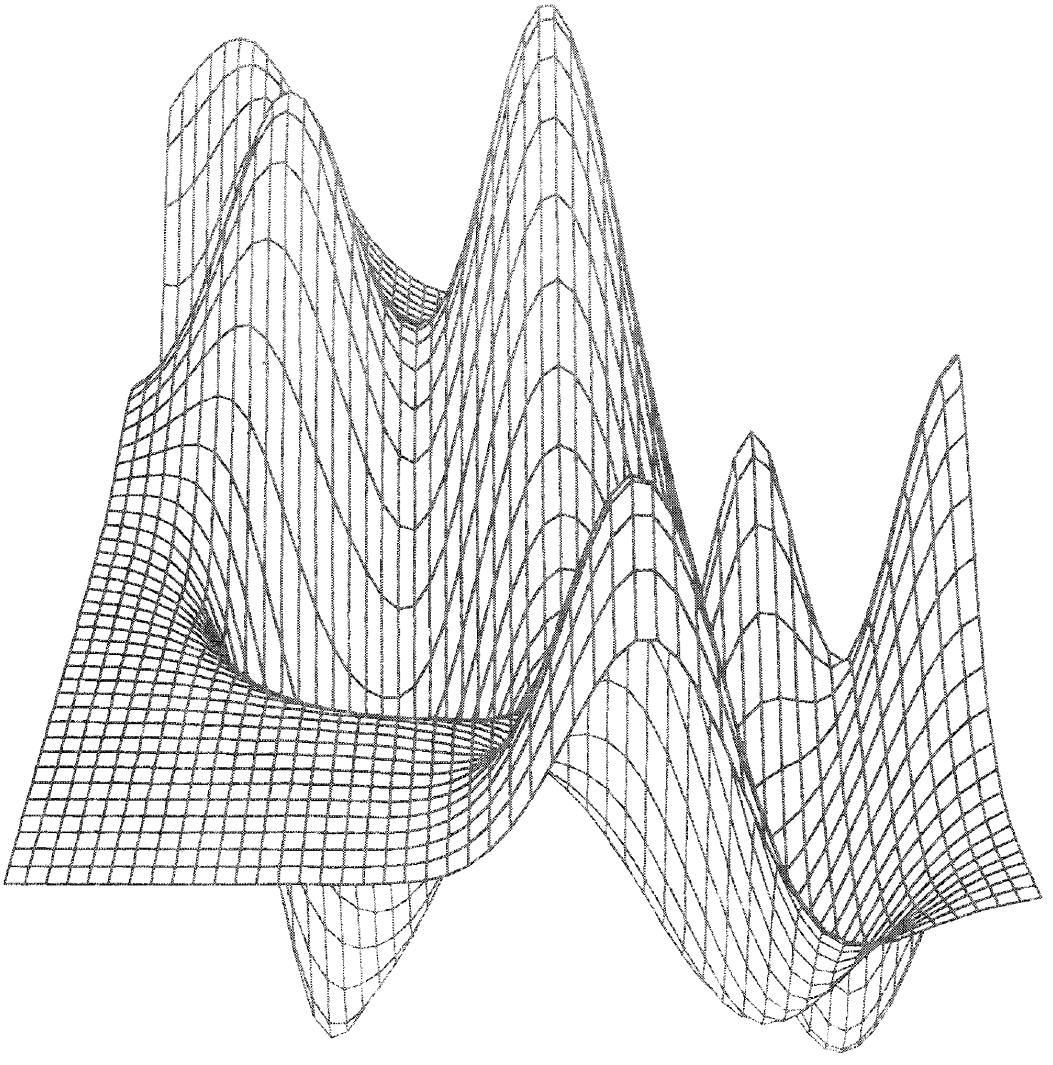


Figure 12. Summation of several bivariate normal distribution functions.

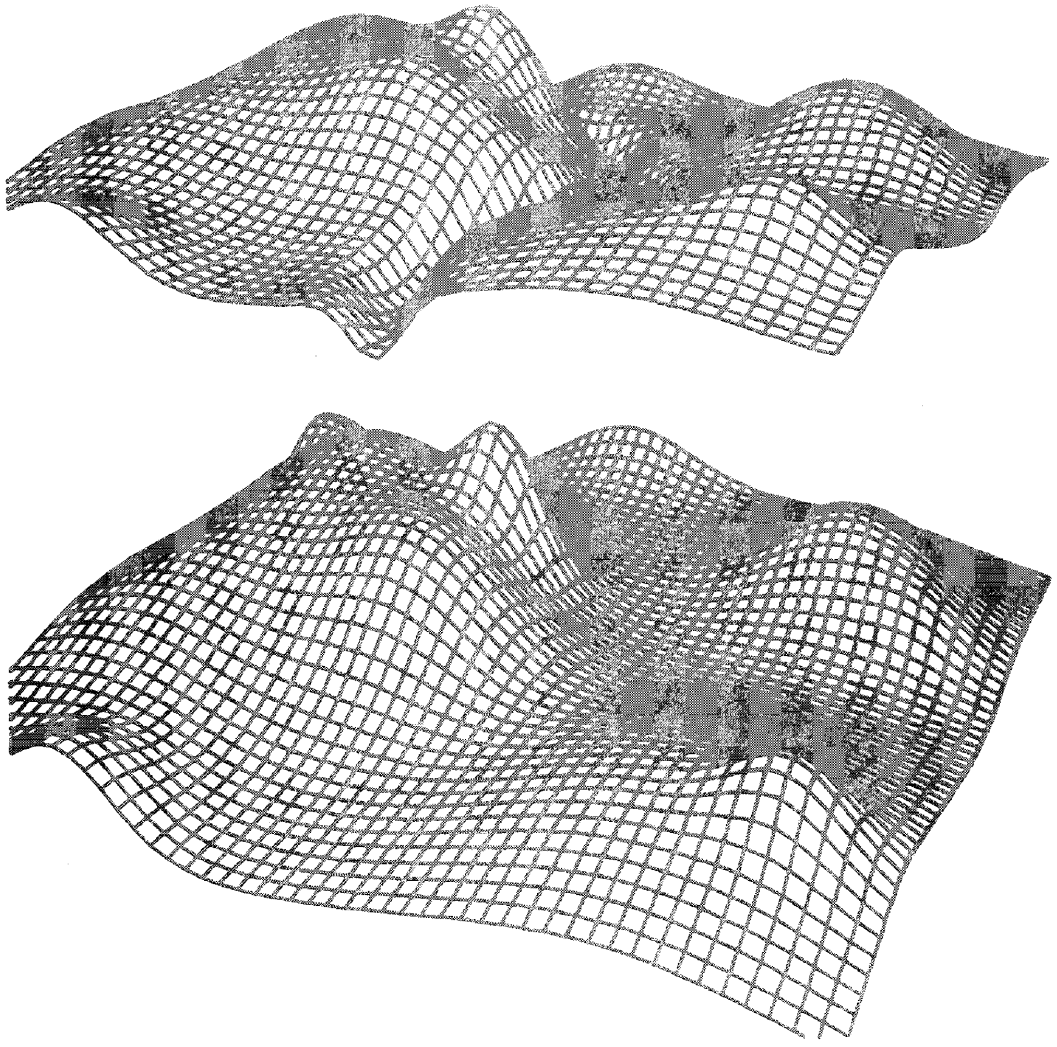


Figure 13. Simulation of a geological problem showing ground surface above and geological layer below.

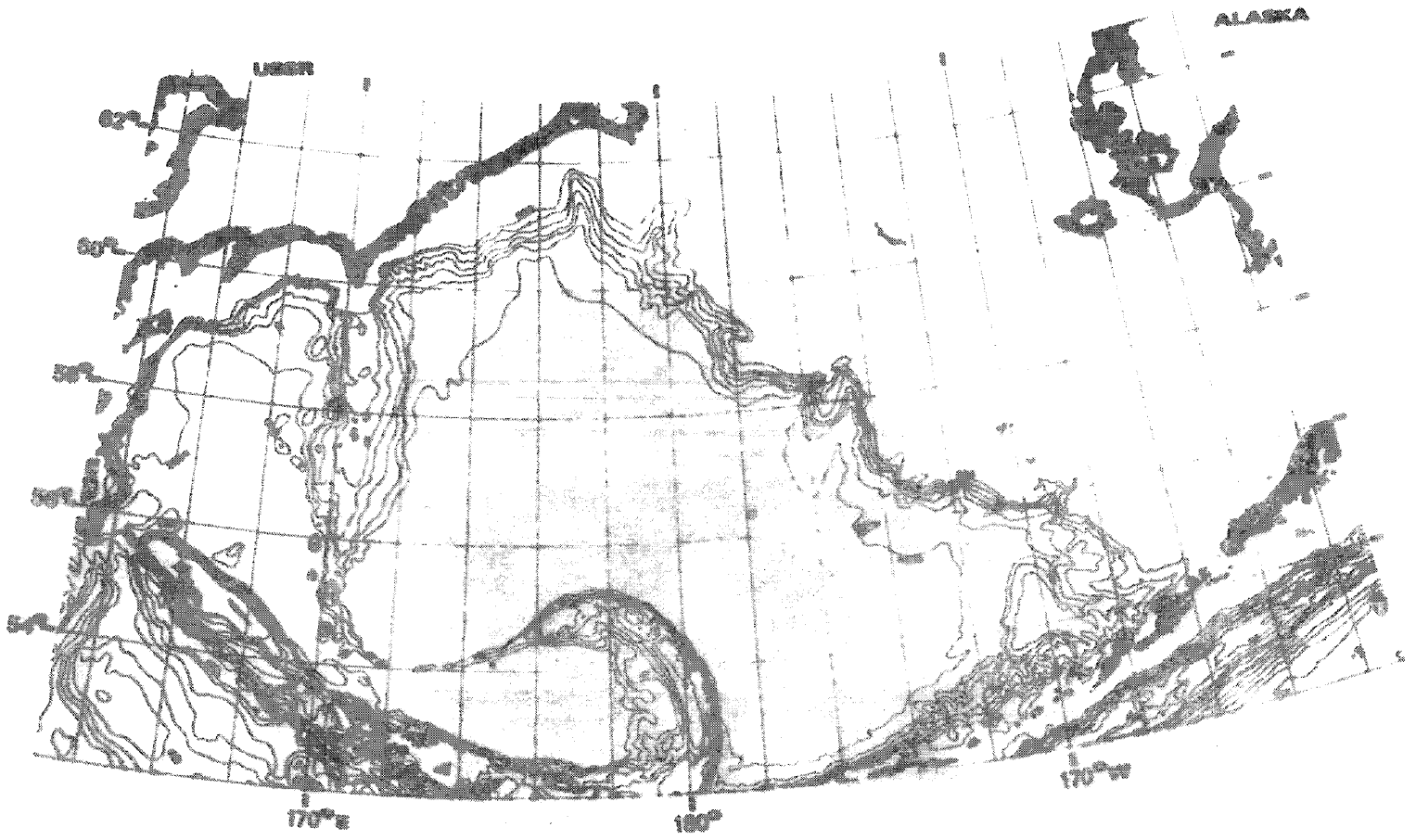


Figure 14. Topographical map of ocean floor in the Bering Sea.

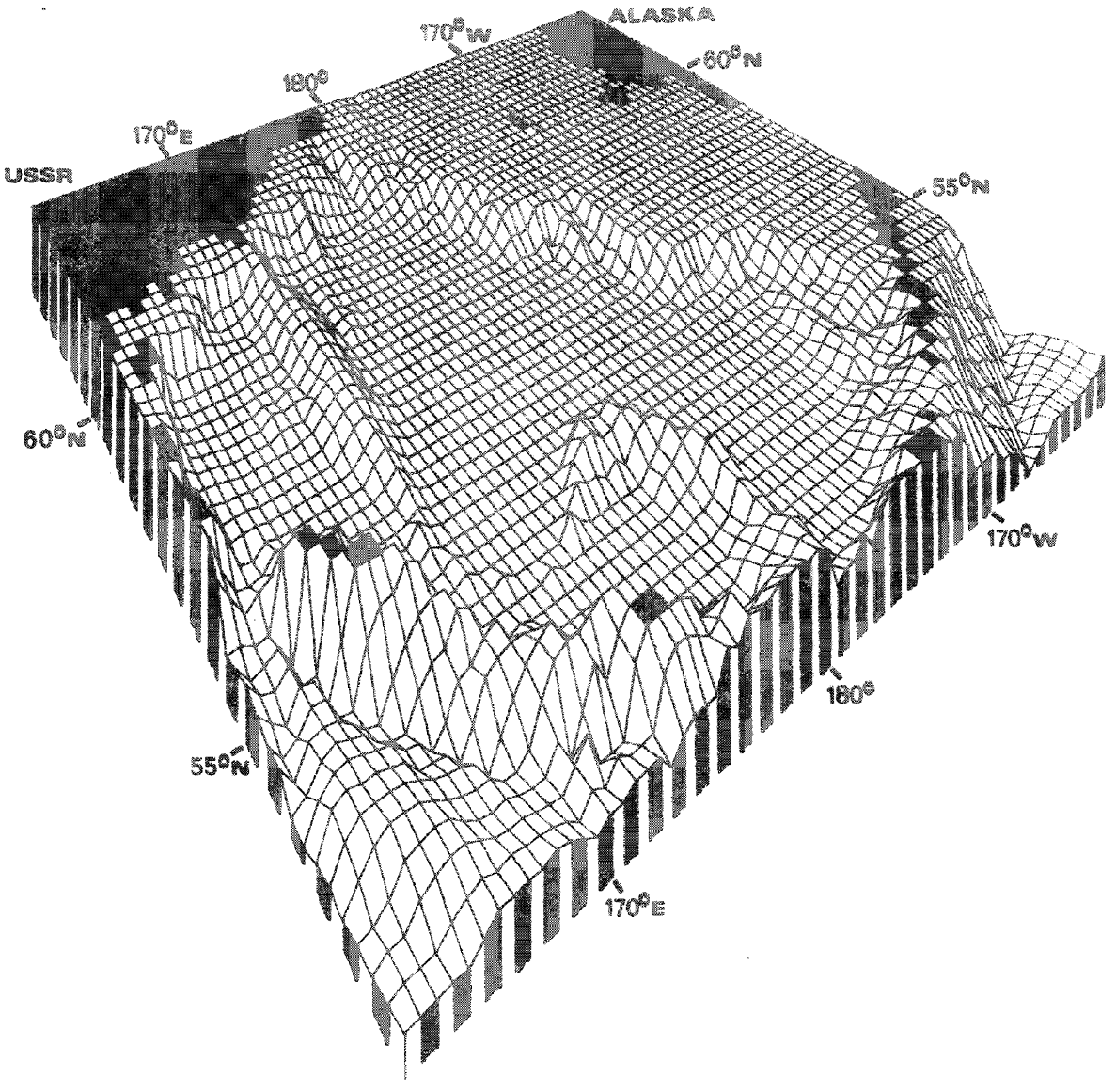


Figure 15. Perspective projection of the ocean floor region shown in Figure 14.

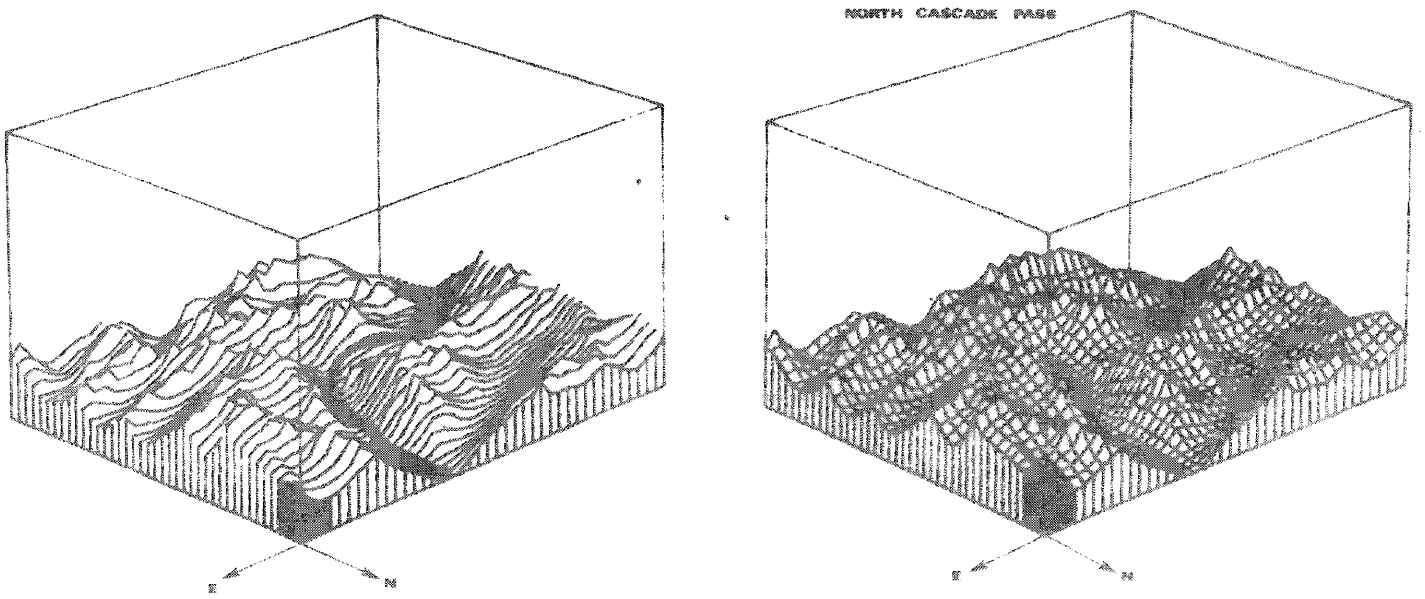


Figure 16. Plots of mountainous terrain. Planned highway is shown in plot at left.

Abstract

Picture Processing Techniques Applied to Electron Microprobe Data

H. D. Jones, 8441
W. B. Estill, 8314

In the traditional mode of operation, the electron microprobe acquires data by scanning an electron beam across a sample in television fashion. The X-rays emitted by the sample are counted and displayed "on-line" to give a picture that qualitatively describes the distribution of chemical elements over the sample surface. Data obtained in this way is very noisy, and has little quantitative use.

We have devised a new technique, whereby data is acquired digitally at a lattice of points on the sample. The data is converted "off-line" to picture form using a CalComp microfilm plotter. The resulting picture is less noisy than those obtained in the traditional mode. Moreover, the data can be analyzed quantitatively, and standard picture processing techniques can be used to reduce blur or to smooth out noisy data.

ABSTRACT

Efficient Curve Fitting Using Interactive Graphics

J. F. Lathrop
8441

It frequently occurs that an analytic fit of experimental discrete data is desired. This may be to provide a program with an efficient means of finding data values, or to interpolate the data, or to estimate derivatives of the data. While numerous least squares fitting routines exist, it is still impossible to guarantee that the fit have the subjective property of "looking right" and most users are unable to explain what they mean by this.

We describe two programs in use for the last two years that allow the user to interactively vary the parameters of the fit and obtain good fits with the desired properties. Both programs use polynomial splines (piecewise polynomials) of arbitrary order to fit the data. The first program allows the user to specify knot locations (the place where successive polynomials meet), observe the fit, and then modify the fit by adding or deleting knots. The second allows the user to also constrain the fit to possess specified values or derivative values at particular points or to be bounded (or derivatives bounded) at specified points.

The discussion will include implementation methods and the hardware systems used.