

# The digital component of the circle graphics habitat

by THOMAS A. DEFANTI  
*University of Illinois at Chicago Circle*  
Chicago, Illinois

## ABSTRACT

This real-time interactive computer graphics system derives from the author's dissertation<sup>1</sup> at the Ohio State University (National Science Foundation Grant GJ-204, Charles A. Csurí, project director). The system, called "The Graphics Symbiosis System" or "Grass" was first designed to help artists interactively explore computer art without the constant companionship of a programmer. Over the past three years, it has been expanded at the University of Illinois at Chicago Circle (Figure 1) and is now the image generation portion of a short-order full-color animated videotape production facility called "The Circle Graphics Habitat." Combined with Dan Sandin's Image Processor, the system<sup>2,3</sup> is sufficiently powerful and flexible to be used in real-time performance context<sup>4</sup> here at UICC.

## INTRODUCTION

The hardware is a standard PDP-11/45 computer with a Vector General 3DR display scope. In addition, we have a data tablet, thirty channels of analog input



Figure 1—The Grass user's console

devices (dials, slide potentiometers, joysticks, etc.) and several channels of analog output (built by Larry Leske) to drive the Image Processor.

Primary inspiration for the structure of the language was the DECsystem-10 operating system and its text editor TECO, and Ron Baecker's animation system "GENESYS."<sup>5</sup> About ten person-years of programming effort up to now has resulted in about ten thousand lines of code in assembler. Persons primarily responsible for implementing the software will be identified by their initials as we progress: Tom Chomicz (TC), Dean Daniele (DD), Tom DeFanti (TD), Mike Dearing (MD), Nola Donato (ND), Manfred Knemeyer (MK), Gerry Moersdorf (GM), T. J. O'Donnell (TO), Ralph Orlick (RO) and Bob Rocchetti (BR). Gerry Moersdorf was responsible for about half the code in the Ohio State implementation. Grass currently has six running installations.

This paper will communicate the program structure of Grass. It would be good, however, to first describe briefly what is expected of the system by university authorities and users. The system, with the Image Processor, is used by professors, instructors and advanced students in preparing animated educational materials in less than geological time. Educational animation makes demands on a system which are computationally and conceptually more complex than standard plotting. The system has to be both easily learned and powerful enough for use by expert programmers in animations like continental drift or a dynamic explanation of how television works. A friendly review of the Circle Graphics Habitat may be found in Reference 6.

Educational animation systems are exciting to develop but not always as much fun to use. Educational animation is rarely free and loose. Fortunately, to continue the good feeling of pure, uncompromised creation, we now have an annual event in April which, like a faculty recital in music, is a performance, but of animated 3-D images and color image processing along with live music (Figures 2 and 3). We believe that jamming on equipment like this demands more from the human engineering side of design than ordinary real-time graphics. More on the philosophy of

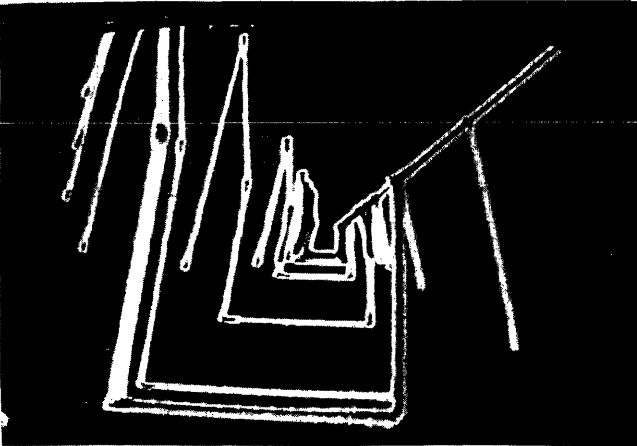


Figure 2—A still from 'Peano Boogie' by Sandin, Morton and DeFanti, a live performance. Colors are sky blue, raspberry red, lemon yellow and orange orange

the Circle Graphics Habitat may be found in Reference 2.

For the technical discussion that follows, it is assumed that the reader has knowledge of interactive graphics to the level found in Newman and Sproull.<sup>7</sup>

#### TECHNICAL DESCRIPTION OF THE GRASS LANGUAGE

Grass has two essential types of primitives: commands and pictures. Commands are kept as ASCII strings terminated by carriage-returns. Pictures (often called "sub-pictures" in the literature) are user-defined displayable lists of 3-D absolute vectors compiled into binary code acceptable by the Vector General display direct memory access processor. On disk, both commands and pictures are normally kept as ASCII

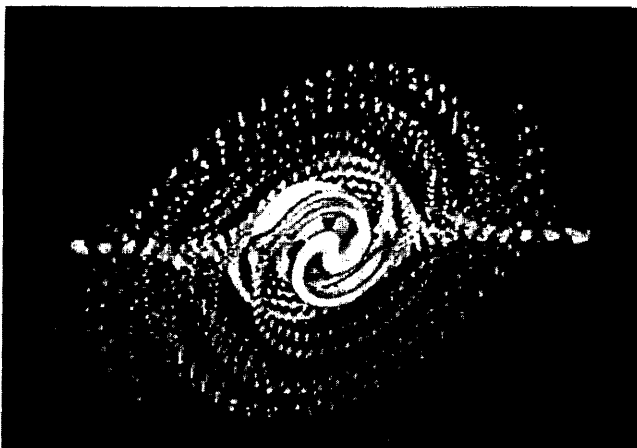


Figure 3—A still from 'Wednesday Night Spiral' by Sandin and DeFanti, a live performance

strings (we always wanted to be able to read what was on the disk). Both commands and pictures in Grass are higher-level primitives, and this concept is essential to the design of the system. In addition, both may be broken down into the lowest level components if desired (i.e., ASCII characters and x, y and z endpoints) and both may be grouped into hierarchies whose elements have much the same behavior as the primitives themselves. The user first learns the command primitives like ROTATE, MOVE and SCALE to manipulate graphical images. Non-programmers do not have to think of images as endpoints or manipulate instructions as individual characters. This differs considerably from the approach taken in typical FORTRAN graphics packages where the user is assumed to be an expert programmer.

#### USE AND CONTROL OF PICTURES

Pictures in Grass have user-assigned names and are displayable lists of vectors prefixed in core by a control block. The user indirectly communicates with the control block for each picture by using commands like ROTATE, MOVE and SCALE rather than by setting bits. At interrupt level, the system updates and writes these control blocks out to the display's internal registers which drive the hardware features of the display (see Reference 8 for a complete description of the Vector General Scope). Updating of the control blocks is done at programmed interrupt level six and the vector display is done by direct memory access over the UNIBUS. The maximum number of pictures concurrently displayed is an assembly feature, presently set to sixty. To the user, all pictures are effectively processed in parallel. The commands that set up to use the hardware are:

```
MOVE PIXNAME,x-DEVICE,y-DEVICE,
z-DEVICE
```

Translates picture named "PIXNAME" to current values of the x-, y-, and z-DEVICES until disabled (that is, the command has to be issued one time only, but any time the DEVICES are changed, the update is done automatically). A DEVICE is defined as either an analog input device or an integer variable. Example: MOVE TITLE,D2, D3,D4 moves the picture "TITLE" on dials 2, 3 and 4.

```
SCALE PIXNAME,DEVICE
```

Scales all three dimensions of the picture on the DEVICE until disabled. Also available are SCALE/X, SCALE/Y and SCALE/Z which scale in individual axes. Example: SCALE/X CIRCLE,A scales the circle horizontally on variable A.

**SETINT PIXNAME,DEVICE**

Sets the intensity of the picture to be continuously variable on the DEVICE until disabled.

**SETCUT PIXNAME,DEVICE**

Sets the z-axis depth cueing and z-axis cutoff feature of the scope until disabled.

The system hardware excels at rotation. Consequently, the ROTATE command, a very high-level primitive, has many options:

**ROTATE PIX,AXIS,SPEED-DEVICE**

Gives simple rotation about the x, y or z axis (specified by AXIS) at a constant speed of rotation determined by SPEED-DEVICE. If angular position rather than speed is desired, ROTATE/D is used for this and all the following ROTATE commands. (By MK and DD.) Example: ROTATE TETRA,Y,K rotates "TETRA" around the y-axis using variable K for the speed of rotation.

**ROTATE PIX,AXIS,SPEED-DEVICE,  
TILT-DEVICE**

This rotation uses the second DEVICE to control the angular position of the axis of rotation in the plane through the origin perpendicular to the AXIS specified. It is a strange rotation to describe on paper but it is a highly useful rotation for interactive use. (By MK and DD.)

**ROTATE PIX,AXIS,SPEED-DEVICE,  
TILT-DEVICE,X-DEV,Y-DEV,Z-DEV**

This rotation gives arbitrary origin rotation capability. It tends to produce elliptical rotations, the backbone of complex animated sequences (as in Disney's Fantasia). Again, it is non-intuitive and requires feedback to use. (By MK and DD.) Example: ROTATE COPTER,X,D0,D1,D2,D3,D4

**ROTATE PIX,7,SPEED-DEVICE,X1-DEV,  
Y1-DEV,Z1-DEV,X2-DEV,Y2-DEV,Z2-DEV**

The "seven-dial" rotate allows the user to specify the endpoints of an arbitrary axis of rotation. It is the rotation preferred by programmers describing scientific phenomena in terms of rotation. (By TC.)

**ROTATE/X PIX,DEVICE  
ROTATE/Y PIX,DEVICE  
ROTATE/Z PIX,DEVICE**

These rotations are compounded with any of the above rotate commands to produce more complex effects. Grouping of pictures allows further compounding of transformations.

**PATHMOV PIX,PATH-NAME,SPEED-DEVICE**

This command tangentially moves the PIX along a PATH (which is simply any picture, displayed or not) with the given speed. It is basically an extension of Baecker's p-curve.<sup>5</sup> (By TC.)

These commands stay in force and cause constant updating to the values of the DEVICES until disabled by the FIX or RESET commands. None of the above commands actually change the vector list since all the functions are done by the hardware.

Many commands do, however, change vector endpoints. The most used ones are:

SMOOTH—does a binomial smoothing of a vector list (BR).

PERSP—does perspective projection (TC).

CLIP—clips one picture against another (like film matting or video keying) (TC).

WINDOW—does normal 3-D windowing (TC).

SHADE—shades in outline with vectors (BR).

SOFT—carries out the hardware transformations on the vectors by software (DD).

The precise syntax of these commands is contained in the on-line HELP file, a copy of which may be had on request.

As is usual with sophisticated refresh graphics systems, the pictures appear to be parallel processed. They are controlled in parallel by turning dials (which are polled at interrupt level 30 times a second) or by manipulating variables. The user can also get at the individual endpoints of pictures with the GETPOINT and ZAPPOINT commands, build pictures with the PUTPOINT command, or draw them in using the tablet or other digitizers. (There is also complete software for text appearing on the Vector General (by BR and TO)).

In addition, the user can group pictures together and create a tree structure hierarchy of control on these grouped pictures. Pictures are grouped for convenient reference, and the groups respond to all the hardware transformation commands just as pictures do. Most often, the GROUP command is used to create multiply articulated structures like Professor Csuri's airplanes, helicopters and witches on propeller-driven broomsticks<sup>9</sup> as well as very complex rotations and translations. Grouping may be carried on to 59 levels (the same assembly feature as before). Note that the user does not have to know about tree structures, lists and pointers to use any of these commands because the system housekeeping does all the chaining. For knowledgeable users, the TREE command gives a schematic

of the hierarchy developed. A two hour-long videotaped lecture describes the internal workings of the system algorithms.<sup>10</sup>

Storage allocation (by TD and ND) is by a "best-fit" algorithm. About 10K of 16-bit words is available to the user for vectors, text, command strings and disk-resident command modules. Many Grass commands, especially user aids like TREE and software transformations like PERSP, SHADE and CLIP are not core-resident. Code (by GM and RO) automatically fetches the appropriate modules into core and executes them. They are automatically deleted. When done, the total overhead amounts to about a tenth of a second.

Garbage collection (by TD and ND) dynamically tries to maintain large blocks of storage. It is called by the user command "DELETE" or automatically invoked by system housekeeping whenever appropriate.

As an example, a videotape<sup>11</sup> to illustrate that 3-D rotations are not commutative (Figures 4, 5 and 6) was done using the following code:

```

GETDSK BFLY1           (get butterfly from disk)
COPY BFLY1,BFLY2      (make a copy)
ROTATE/D BFLY1,X,D1   (rotate on x axis)
ROTATE/Y BFLY1,D2     (then compound with
                      y rotation)
SETINT BFLY1,D3       (control intensity on
                      dial 3)
ROTATE/D BFLY2,D5     (rotate on y axis)
ROTATE/X BFLY2,D6     (then compound with
                      x rotation)
SETINT BFLY2,D7       (control intensity on
                      dial 7)
GETDSK 3DAXES        (get the axes up)
GROUP 3DAXES,        (group and call the
  BFLY1,SAM           group "SAM" BFLY2
                      gets in for free)
ROTATE/D SAM,X,S1,S2 (rotate the whole thing)

```

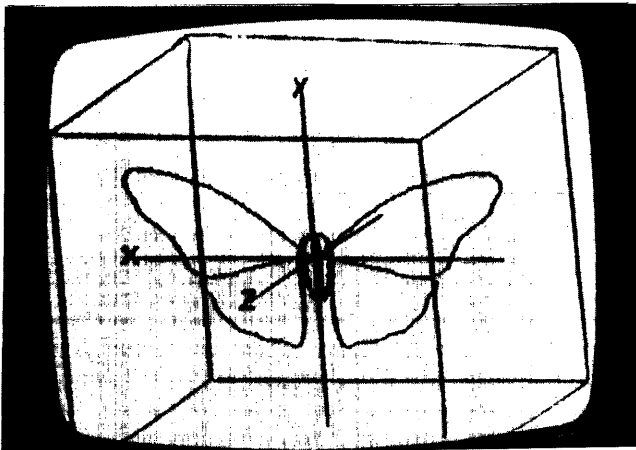


Figure 4—The butterfly in its original position

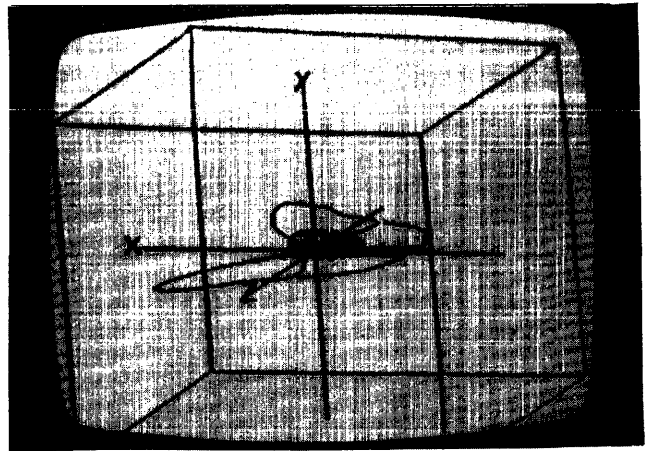


Figure 5—The same butterfly rotated ninety degrees around the x-axis and then ninety degrees around the y-axis

Now, by turning dials 1 and 2 enough to get ninety degree rotations about the x then y axis, the first butterfly takes one position. Turning dials 5 and 6 enough to get similar rotations about y then x axes, the second butterfly goes to a different position. It is quite easily seen that the two are not equivalent. That is, 3-D rotations are not commutative. (Rotating the group "SAM" allows the third dimension to be seen more clearly. The intensity controls allow independent fading of the two butterflies for clarity.) Given the existence of the butterfly and the axes on disk, the entire videotaped sequence took less time to produce (about fifteen minutes—some of which was dedicated to aesthetic judgment and color choice) than to describe.

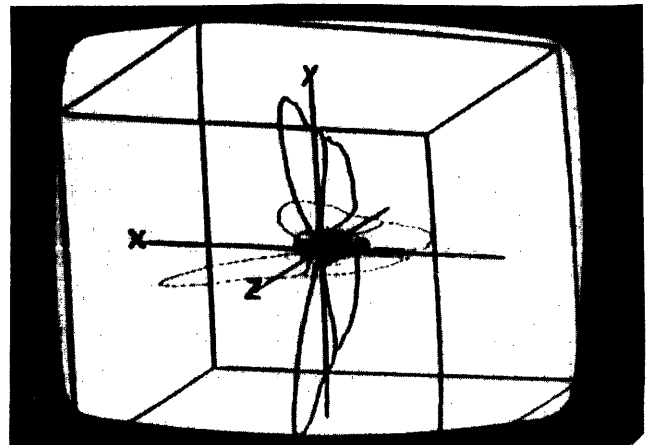


Figure 6—The same butterfly at half intensity contrasted with another butterfly rotated ninety degrees around the y-axis and then ninety degrees around the x-axis

## USE AND CONTROL OF COMMANDS

The foremost design criterion of the command language in Grass has always been habitability, a term adopted<sup>12</sup> which means the quality of a system that makes it easy to learn and use. In many ways, the linguistics of graphics languages are quite unnatural for describing animation and many people do much better by waving their hands. On the other hand, the power of linguistic structures is undeniable, especially when modeling scientific data.

Programming on someone else's system is always frustrating. What really matters is whether you get anything done while being frustrated. We have tried to design a system to help users at all levels to get things done.

The basic tenets of habitability are usually obvious—mnemonic command names, predictable syntax, good error messages, high feedback—but they are rarely implemented in full because the detail work in coding is immense. The leisure time for attention to such detail is hard to come by, especially if the users are overwhelmingly power-conscious and impatient programmers. In a short-order videotape laboratory where users are by and large professionals donating their time to improve the quality of their classes, non-alienation is the item of highest priority. General fun and productivity seem to follow. Again, more of this philosophy is found in Reference 2.

A command in Grass is a string of characters terminated by a carriage return (CRLF) or a semi-colon in the case of multiple commands on a line. In order for the system to process a command in any context, originating from anywhere, the string of characters is simply passed to the line processor (LINEP) which interprets the string, dispatching to the proper command module. There are two general formats:

```
COMMANDNAME ARG1,ARG2,ARG3,...
```

(for most commands)

Example: SCALE WIDGET,D0

```
VARIABLE=EXPRESSION
```

(for FORTRAN-style commands)

Examples:

```
A=A+10
```

```
K=K-D0/2 (note use of dial 0)
```

```
FA=ATN(FP)+SQR(FQ*(FD-FE))
```

If the command is not core-resident, it is fetched from the disk (all disk-resident commands are written in position-independent code), executed and deleted.

Commands are typed in line by line on the video terminal (VT05). Since many of the commands set up processes at interrupt level, the system may be used exclusively on a line-by-line basis as with a text editor. The previous rotation example was done this way, as was a twenty-minute film with witches chasing butterflies and airplanes flying around the globe, at a time (1972) when the language had only line-by-line capa-

bilities. Almost all programs written in the language start as a few commands typed in and tested one line at a time.

The next step in command processing is to take commands stored in a file so they in essence become like the roll of a player piano, or the paper tape for a milling machine. Many text editors allow this type of command usage. This grouping of commands is often called a "macro."

Macros in Grass are simply groups of commands—a concept easily grasped by all our users. In compiled languages, macros are called "subroutines," but this terminology was not chosen because Grass macros are often not subordinate to anything conceptually. Macros do not require preambles, declaration statements, end statements or other formalities associated with subroutines.

To give the player piano roll in Grass a variable fast-rewind and fast-forward, control is transferred with the SKIP command. Its argument (e.g., SKIP 3) specifies how many CRLF's to pass over forwards or backwards. The argument may also be a label, in which case the transfer is to that label local to the current macro. Transfer to other macros is with the DO command whose argument (e.g., DO SETUP) is a macro. If this macro is not core-resident, it is automatically fetched from disk and then interpreted. The CALL command is similar except that it uses the system area of the disk and tries to find a compiled version of the macro (see below for details on the compiler). As will be seen later, the syntactic form of the macro call with parameters is very close to the form of commands so that system macros can appear to be system commands to the user.

Conditional branching and command execution is done with the IF command whose syntax is simple:

```
IF VARIABLE=EXPRESSION,ANY COMMAND
```

Examples:

```
IF A=B,SKIP -5
```

```
IF D0 GT 0, IF A LT -100, DO FIXUP
```

In any case, control returns to the statement following the DO when the indicated macro is finished.

Macros may be generated in several ways. The system editor can be used to enter and change ASCII files on the disk or in core. Macros may also be created by typing a name followed by a colon as in the following example:

```
SETUP:<GETDSK GLOBE
SCALE GLOBE,D0
ROTATE GLOBE,X,TX,TY
      (using the tablet x and y)
GETDSK TITLE
MOVE TITLE,D6,D7,D8>
```

To execute this macro, one types "DO SETUP" or simply "SETUP" (providing it is not a system name). Often, immediate execution is desirable. The "un-

named" macro, entered thus:

```
<ANY COMMANDS
. . .
. . .
. . . >
```

is automatically executed upon typing the final angle bracket. In addition, when this type of macro finishes, it is automatically deleted. Only named macros may be stored on the disk.

The system uses a VT05 video terminal at 2400 baud for user communication. This choice was originally made (by MK) to keep the Vector General screen free for images to be recorded by a camera. One version of Grass uses a keyboard interfaced through the Vector General and space at the bottom of the screen for the same purpose. At any rate, the system can afford to be fairly wordy with messages without much delay.

Being strings, macros can generate other macros using the string variables (\$A through \$Z in Grass) and the string manipulation primitives given by concatenation and the SEARCH command (MD and TD). This feature, exploited only by fairly experienced users, allows very plastic fabrication of executable statements in Grass. String variables are executed by putting them alone on a line after construction:

```
$A='GETDSK GLOBE
SCALE GLOBE,D0'
$A
```

Passing parameters is usually clumsy in programming languages. Indeed, we have only recently implemented a habitable way of passing parameters between macros (by RO). Like everything else in the system, macros are used interactively. Rather than burden users (who often cannot yet write macros) with having to know which parameters to supply, macros are usually written to ask questions:

```
SETUP:<PROMPT "WHICH PICTURE DO YOU
WANT"
INPUT $A      (system types a "?" and waits
              user then types in a name)
GETDSK $A;SCALE $A,D0  (a multi-command
                       line)
ROTATE $A,Y,D1>
```

Effectively, any picture on the disk may be gotten, scaled and rotated with this macro. Similarly, one can input numeric values (e.g. INPUT FA) as numbers or expressions. The PROMPT command is the general typed output command, and may be used to print strings or numbers and combinations thereof:

```
PROMPT "THE SQUARE ROOT OF 1000 IS ",
SQR(1000)
```

However, once the user is familiar with the macro, he may type "DO SETUP,GLOBE" or simply, "SET-

UP GLOBE" which looks like a Grass command. Either construction may be imbedded in another macro without any reprogramming of the original macro—an important feature since many users cannot decipher complex macros written by others. Note that the PROMPTs are automatically suppressed as long as enough arguments are supplied. If the user leaves off an argument, the system will wake up the PROMPTs and start asking the questions again. Of course, overrides are available to force INPUTs or PROMPTS if desired. Arguments may also be passed in global variables.

Variables in Grass have fixed names and are either local to macros (LA-LZ, fixed; EA-EZ, floating) or global (A-Z, VA-VZ, WA-WZ, fixed; FA-FZ, floating; \$A-\$Z, string; and AA-AZ, fixed and floating arrays). Analog inputs are global (D0-D9, dials; S0-S9, slide potentiometers; JX,JY,JZ and KX,KY,KZ, joysticks; P0-P3, more dials). Digital inputs include the tablet (TX,TY,TZ or pen-press) and function switches (FS0-FS15). Analog outputs are global too (OA-OH). Variables are prenamed in Grass because otherwise the interpretive overhead for arithmetic would really be immense. Prenaming also eliminates the need for declarations of variables (except for array dimensions).

The external inputs are polled every  $\frac{1}{30}$  second by the system. Currently under construction is a flexible input box with variety of connectors and amplifiers to aid in prototyping new input devices such as your body or musical instrument to produce a set of Grass variables. Having so many analog input devices, by the way, may seem confusing to the reader, but these physical extensions to the system can be intuitive and therefore easy to use.

Macros work despite the parsing overhead because the primitives of the language are generally rather high level and parsing is only a fraction of the code executed in doing the command. The higher the level of the primitives, the more practical the interpreter becomes. Note also that often one does not care how long something takes to parse as long as it is done in say, less than  $\frac{1}{60}$  second.

However, for low level primitives like addition or expression evaluation, the interpreter may execute a thousand instructions to add one to a variable. This is a major reason compilers are still preferred for arithmetic calculations. As soon as we started doing scientific animations as well as computer art, a fast arithmetic capability became essential. Thus, The Habitable Compiler was written (by RO). It takes assignment statements and some commands (notably GETPOINT, ZAPPOINT and PUTPOINT) and compiles them into PDP-11 machine code which executes very quickly. Whatever the compiler does not understand, that is most commands, it keeps as ASCII strings which are passed to the resident interpreter during execution. Thus the compiler retains the benefit of the interpreter

and yet gains the speed of compiled code where essential. One usually debugs the macro first and then, if speed is a problem, compiles it. The compiler is disk-resident and rarely takes more than a second to load in and do its job. Compiled macros may be stored on the disk (in binary) and recalled at any time. In addition, macros that contain only arithmetic code are re-entrant so they can be set up to execute at interrupt level with the VIP (Very Important Program) command. VIPed macros are used when variables must be calculated perfectly in synch with the display refresh or when a higher priority task is useful. Few higher-level languages allow users to schedule subroutines at different priority levels.

To give a quick idea of the compactness of Grass macros, Donald Warren Collins wrote an architectural preview system<sup>12</sup> first on the IBM S/360 in FORTRAN taking some 132K 32-bit words, then in PDP-11 RSTS-11 BASIC taking 28K 16-bit words with multiple overlaying. Finally, he rewrote it in Grass, thus making it interactive. The Grass version took about 2K 16-bit words of storage, not including the 13K interpreter, of course.

## DRIVING MACROS

Most of the contribution this system has made to habitability in graphics is noticed when actually executing and debugging macros. The constant real-time user control combined with the analog input devices makes this system usable as a performing instrument. Some of this control comes from parallel processing of pictures and, if the user desires, macros as well.

Animation often involves several more-or-less independent things happening simultaneously. Grass can be asked to set up a ring structure of macros so they can be executed in parallel. With two macros, the commands are interleaved. More than two macros requires grabbing lines from each one, one at a time, round-robin fashion. In addition, unless specifically requested otherwise, a macro in this ring structure automatically starts over again when finished. All this housekeeping is initiated by the DOLOOP command (by TD):

```
DOLOOP MACNAM1,MACNAM2,...
```

where the MACNAMs may be macros or compiled macros. The unnamed macro is retained for convenience in setting up background jobs:

```
DOLOOP <A=A+D0/100  
B=A*2>
```

and so on. The system continuously listens to the VT05 keyboard for a line to be typed interactively and slips it in, executing it with no noticeable delay in most cases. Provisions for one macro waiting for another

to complete and selective removal from the ring also exist.

As might be imagined, there is some overhead associated with the DOLOOP feature, but compiling the macros usually helps make the overhead quite unnoticeable. The need for and desirability of a parallel execution structure in graphics is more fully developed in Reference 3. Alan Kay's SMALLTALK language for children also implements this type of parallelism for 2-dimensional graphics.<sup>13</sup>

Two major conceptual simplifications of using macros result from parallel execution. First, the user can develop small animation sequences separately and then combine them later with little or no reprogramming. Without the parallelism, a total rewrite of the macros would be required. Second, the user may wish to incorporate macros written by others in his animation sequence, macros whose logic he may not understand. Again, without the DOLOOP structure, this would be very difficult.

Grass also has well-developed "panic-button" control structures (by TD) to abort or temporarily interrupt macros. First, CONTROL-C (holding the control key while typing a "C") stops any macro or compiled macro, kills the DOLOOP structure, stops any output, cleans up any scratch files like unnamed macros and sets the user back to command input level. It is the most common way to exit a macro in an infinite loop.

CONTROL-W temporarily stops printout until pressed again (2400 baud is too fast to read) and CONTROL-O cancels any output.

The real crowd-pleaser in the system, though, is CONTROL-S. It suspends execution of any macro or compiled macro, even if DOLOOPED. The user is put in command level and he can type commands to check variables, or anything else. The macro continues when the user types "RESUME." Since performance graphics, especially the jamming variety, requires constant real-time debugging with two hundred people looking over your shoulder, CONTROL-S comes in very handy. The combination of background DOLOOPED macros and CONTROL-S give the user the impression he is always in control.

## ERROR MESSAGES AND DEBUGGING IN GRASS

Grass has about one hundred fifty error messages, only one of which is truly cryptic ("Undiagnosable syntax error"). When an error occurs, the whole command is printed out on the VT05 with a little arrow under the part that caused the error, followed by the error message. The user is then put into the same mode that CONTROL-S initiates, at which time he can correct the mistake and RESUME or type CONTROL-C to abort. Along with the feedback on the screen, these error messages and interactive fixups account for about ninety percent of the user debugging activity.

The harder-to-find logic errors are usually tracked down by the LIST command (by GM), the TRACE command (by TO) or sometimes the DOLOOP code. LIST simply prints each line of a macro as it executes. TRACE, adopted from SNOBOL, takes variable names as arguments and then prints out the value of the variables every time the variables are changed along with the location of the change. As a last resort, a macro to sense an elusive logic error may be parallel processed with the defective macro to discover the problem. For instance, assume variable A is never supposed to get to zero but it is anyway. "DOLOOP <IF A EQ 0,LIST>" will turn on the LIST feature as soon as A goes to zero. Obviously, more exotic bug traps can be constructed.

Error conditions may also be trapped and further processed by the macro writer. The ONERROR command sets up an asynchronous error recovery procedure for use when an error happens:

#### ONERROR VARIABLE,ANY GRASS STATEMENT

When the error occurs, the error number is put in the variable indicated for use by the error recovery routine and the Grass statement following the comma is executed in place of the line in error. Since errors can be as benign as not finding a file on the disk, system macros are written with ONERROR frequently to help the novice user. As might be imagined, considerable detail work was done to assure PDP-11 stack integrity when commands in error are replaced by ONERROR code. Note that the Grass statement in the ONERROR command can be a macro call like "DO FIXUP."

The expert user can take advantage of the asynchronous nature of the ONERROR command to speed up loops. For example, in the following program which zeros the z-value of each vector, the end-of-picture condition ( $K = -1$ ) must be tested:

```
ZEROZ:<PROMPT "NAME OF PIX FOR Z-AXIS
ZEROING"
INPUT $L
N=0
N=N+1
GETPOINT $L,N,X,Y,Z,K (set the nth point in
variables x,y,z and k)
IF K NE -1,ZAPPOINT $L,N,X,Y,0,K; SKIP -2
(if K = -1, it is the end of picture)
PROMPT "DONE ZEROING">
```

Now, if "ONERROR A,SKIP 2" is placed somewhere before the last four lines of this macro, the test for  $K = -1$  may be eliminated because the GETPOINT index (N here) will go out of bounds and generate an error. "SKIP 2" will be executed in place of the line in error and the control will pass to the last PROMPT.

Such tricks can even work with compiled macros, a housekeeping feat of some proportion.

In summary, Grass provides many ways of controlling, debugging and interacting with pictures and macros. Since so much of the control can be parallel, the user occasionally feels like he is conducting rather than watching a plotter. Music has always been a performing art and artists now have the tools to perform visual scores. The task now is achieving the control subtlety in performance graphics that we know and love in music.

#### CURRENT LIMITATIONS OF GRASS

Grass as a system has for some time been pushing against walls created by equipment speed, memory limitations and the nature of the refresh display. Currently, the PDP-11/45 we use has only 28K of usable memory, of which 3K is used by the disk operating system. The user is left with 10K of space to use. Given the software overhead, the maximum number of parallel full-screen vectors that can be displayed flicker-free is about a thousand, although they can all be rotating, moving and scaling. We have to operate in a flicker-free environment because our television-based system is not sophisticated enough to operate in anything but real-time.

Adding speed to the PDP-11/45 is fairly simple with an add-on cache memory, but not cheap. Adding to the memory requires memory management, and a considerable amount of reprogramming. And CRT's that are faster by an order of magnitude are still a gleam in the designer's eye.

A major limitation to the habitability of Grass is that not all users have linguistic skills—or type well. Non-linguistic approaches to subset problems like constructing complex 3-D pictures are possible using light-pen or tablet menus. We now have several small grants to investigate performance-time control structures for computer graphics.

#### REFERENCES

1. DeFanti, T. A., Dissertation, The Ohio State University, 1973.
2. DeFanti, T. A., D. J. Sandin and T. H. Nelson, "Computer Graphics as a Way of Life," *Computers & Graphics*, Vol. 1, No. 1, May 1975.
3. DeFanti, T. A., "Toward Loopless Interactive Graphics Programming," *Proceedings of the Conference on Computer Graphics, Pattern Recognition and Data Structures*, May 14-16, 1975 (IEEE Catalog #75CH0981-1C).
4. DeFanti, T. A., D. J. Sandin, et al., Interactive Electronics Visualization Event, Videotape, 90 mins., OIRD University of Illinois at Chicago Circle, 1975.
5. Baecker, R. M., Dissertation, M.I.T., 1969.
6. Nelson, T. H., *Computer Lib/Dream Machines*, 1974.



- 
7. Newman, W. M. and R. F. Sproull, *Principles of Interactive Computer Graphics*, McGraw-Hill, 1973.
  8. Vector General Inc., *Graphics Display System Reference Manual*, 1975.
  9. Csuri, C., et al., *Real-Time Film Animation*, The Ohio State University Research Center, February 1973.
  10. DeFanti, T. A., *Grass Internal Logic*, Videotape, 120 min., OIRD The University of Illinois at Chicago Circle, 1974.
  11. Donato, N. and T. A. DeFanti, 2-D and 3-D Transformations, Videotape, 25 min., OIRD The University of Illinois at Chicago Circle, 1975.
  12. Collins, D. W., "A Computer Graphics System for Modular Building Elevation Design," *Proceedings of the Second Annual Conference on Computer Graphics and Interactive Techniques—SIGGRAPH '75*, July 1975.
  13. Personal communication.

