SESSION REPORT

61	A564	DI3000 User Experience	34
SHARE NO.	SESSION NO.	SESSION TITLE	ATTENDANCE

Graphics Education	Gail Garrison	GFC
PROJECT	SESSION CHAIRMAN	INST. CODE

250 North Street, White Plains, New York (General Foods Corp.) 914-335-1267
SESSION CHAIRMAN'S COMPANY, ADDRESS, and PHONE NUMBER

Abstract:

Part I: What is DI 3000 and Pretty Pictures?

DI3000 is a system of device-independent FORTRAN-callable graphics tools. Its capabilities range from presentation graphics to Mandelbrot sets.

Part II. A DI3000 under CMS User Experience or How I Learned VS/FORTRAN, Assembler, and PER

A naive applications programmer learns what it's like to be among the first on your block to install a software package written for VAXes under CMS.

Part III. The June 1 "Slam Dunk" Tape

The somewhat less naive programmer describes how easy it is to install the new Precision Visuals, Inc. (PVI) "CMS Slam Dunk Tape", provided you know what stuff on it is Cornell-dependent.

SPEAKER: Wendy Alberg (CUN)
Cornell University
Ithaca, New York

(The information in this section is drawn heavily from the Introduction to the DI-3000 User's Guide and the DI-3000 Short Course -- Instructor's Manual.)

Precision Visuals set out to create a flexible and durable graphics system when they designed DI3000. They wanted to make it device independent, portable, and make it adhere to existing and future standards for graphics software.

Different graphics devices have different special capabilities. Plotters produce permanent copies of your graphics output. One kind of screen may be able to draw circles, another may be able to fill a polygon with one of a menu of patterns, and yet another may allow you to use 4096 different colors at once. Rather than write a separate graphics system for each device, Precision Visuals chose to write one device independent sysstem that attempts to simulate the features of all devices on every device whenever possible. If a device can't erase lines in hardware, for example, the device independent system will attempt to simulate that function in software. If it can't simulate the function (for example, trying to let you use 4096 simultaneous colors on a plotter) the system will ignore it. The advantage of a device independent system is that graphics ouput produced by a given application on one device will be very similar, if not identical, to graphics output produced by the same application on another device, eliminating the need to spend hours modifying an application each time you must switch devices.

Since DI3000 is a library of subroutines, a programmer can build up his or her own combinations of graphics primitives in endless different ways, unrestricted by patterns pre-programmed into the system. A turnkey system may produce a pie chart with a single keystroke, for example, but may be limited in the ways it can display it. A subroutine library allows you (with more effort) to customize the pie any way you like - adding plums, for example. Precision Visuals chose to write the DI3000 subroutine library in FORTRAN in order to take advantage of the fact that since there are FORTRAN compilers on almost any system, DI3000 can potentially be made to run almost anywhere.

Establishing graphics standards and making DI3000 conform to those standards has several advantages. When people first started writing graphics software, they wrote it to take advantage of hardware features of the graphics device(s) they happened to have. Such software is useless for different graphics devices and often for the same device on a different system. If a graphics system can be written to be device independent and to adhere to a widely accepted standard, the same program can be run on different devices and/or different systems. A programmer used to writing graphics applications on one computer can easily write them on another. Software that conforms to the standard will continue to run, and a graphics device, once the graphics system has a device driver for it, will never become obsolete. Such a graphics system would be extremely durable.

Currently, DI3000 supports thirty or more devices and the list grows steadily. Precision Visuals also claims to support at least nine different computer systems. At Cornell, we're running DI3000 under CMS, with various degrees of success, on a Tektronics 4013 (green storage tube), a Tektronix 4027 (8 simultaneous color raster device), an HP7221 (8 pen flatbed plotter), an AED 512 (Advanced Electronic Design's 256 simultaneous color raster

device), and an ACT-1 (Advanced Color Technology) ink-jet printer.

What can you do with your DI3000 application program? Your program creates graphics objects for a sort of generic device that Precision Visuals calls the "virtual device". In theory, this device possesses every feature of any real graphics device. In practice, defining such a device may be difficult as long as hardware companies keep developing new and better devices. Your application builds graphics objects from DI3000's graphics primitives: such things as lines, the current cursor position, or the size of a letter. It combines primitives into segments that are treated as user-defined primitives (rather like the way an exec in CMS can be considered a user-defined CMS command).

Using modeling transformations, you can rotate, shrink, chop corners off, or otherwise alter these segments. Once the segments look the way you want them to, you can use a viewing transformation to translate a rectangular region (or "window") of your "world coordinates", the coordinates of the sheet of graph paper you designed your object on, to a "viewport" in "virtual coordinates", the normalized (x and y range from -1 to +1) coordinate system of the virtual device. Once you've translated an object to virtual coordinates, you may use an image transformation to rotate, shrink, or otherwise alter the image once more.

You may use another viewing transformation to define what Precision Visuals refers to as a "virtual camera". Just as you might set a real camera close to or far away from the object you want to photograph, and just as you might walk around the object searching for the best angle from which to take your picture, so you can use a viewing transformation to get a close up or far away view of the object from any angle you like. If you think of the image of the object that will appear in the viewport as what you'd see if you looked in the range finder of a camera, and think of the virtual coordinate system as a photograph album in which you'll paste that viewport once you've taken the picture, you've got the concept.

Finally, DI3000 defines six different types of input from the virtual graphics device, allowing you to create truly interactive graphics applications.

As you write your application, you think in terms of the omnipotent virtual device. Not until you actually run the program do you tell it what real device it will be using. In fact, with the Metafile Generator system, you may save graphics output as if it had been displayed on the virtual device. Later, you can go back and use the Metafile Translator to display that output on any real device at all. Saved metafiles are analogous to load modules in the sense that you don't need to rerun your application program to run a device with its output.

So much for the metaphysical details of DI3000's design. What are the basics you have to work with?

DI3000 has six non-text primitives and nine text primitives. The non-text primitives include standard basic objects: lines, polylines, polygons, and markers; and abstract concepts: current position and move. A polyline is a collection of line segments. The current position is defined in world coordinates; move changes the current position.

Non-text primitives have various attributes. Color, linestyle, linewidth, and intensity are just what you'd think they are. Pen is a conglomerate of the first four attributes. Polygons can have an edge style, an interior (fill) style, and an interior color and intensity. The marker symbol is an attribute of the current position, rather than the other way round as I'd expect. With these primitives and some modeling transformations, you can build just about any image you'd care to.

The only other thing you'd probably like to add to your image is text. In the real world there are two ways to generate text. One is to let the hardware generate its internal characters; the other is to draw each letter segment by segment ("stroke-generated"). DI3000 recognises two types of each. Hardware-generated text can be drawn character by character or as an entire string. Stroke-generated text can be uniformly spaced or proportionally spaced (the latter is used for publication-quality text).

Text has its own set of attributes. Although I haven't seen it stated, I assume that the usual non-text attributes like color and intensity apply to text as well. In addition, characters have a font, or typeface, and size; they are justified with respect to the current position and have a character path that determines the direction the string goes (right, up, down, left). If you get tired of strings going in only four directions, you can change their character base, the direction the base of the string goes in relation to the world coordinate system. Strings may also have a character plane, defined at some angle to the world coordinates; depending on the way the characters are generated, they will lie in that plane. Strings have a string extent, the width and height of the string; it's useful to know what it is if you're trying to put a string "inside" another object. Finally, strings have a character gap, the space between letters in the string, that lets you stretch the letters out or squeeze them close together.

Segments, the collections of graphics primitives that make up an object, have attributes, too. Segments can be temporary or retained. Temporary segments vanish when you clear the screen and are useful for experimentng with combinations of primitives. Retained segments have names and survive screen clears (although they aren't saved between DI3000 runs). You can make them invisible, copy them into other segments, or make them a different intensity from other segments you're displaying (called highlighting). One of their most interesting attributes is that you can use the device input functions to PICK named segments and build pictures or complicated objects from them. Our part-time graphics support person is currently writing a slide-generating utility that lets you PICK character fonts, type sizes, justification, color - any text attributes - and construct a slide image from a file of character strings.

Besides PICK, DI3000 supports five other input functions. A button returns an integer value to the application program; a key on the keyboard could be a button. A locator returns a device coordinate; the location of the cross-hair cursor on a Tektronix terminal, for example. A valuator returns a floating point number; the intensity of the point on the screen given by a locator perhaps. A keyboard returns a string (a response to a question from your program maybe). Stroke gives you a string of virtual coordinates; the input

from a mouse or bit pad, for example. You can also ask the device for its characteristics; depending on the device driver you'll get different answers. You may ask DI3000 what the last error was, or what the names of the current saved segments are. Your program can ask the device to pause; the device will wait for some sort of input to continue. You can send the device a message, or send it an escape sequence. Because of the way DI3000's device drivers are written, the driver will ignore any message or escape sequence it doesn't understand, rather than stopping your program.

DI3000 is smart enough to tell when you aren't using it correctly. Whenever you make a mistake calling a DI3000 routine, the system writes an error message to the error log. There are different severity levels for errors. You can decide what level of error will stop your program. That means when you're debugging you can see many of your errors at once, instead of having your program die with the first error, fixing that one, and running it again to find the next error. Imbedded in DI3000 routines is debugging trace code. There are different levels of tracing, too; you may choose to see all the messages, some of them, or none. You can turn the debugging level up or down as much as you want within a sigle run, with the advantage that if you're homing in on a bug you can look at detailed messages from the suspicious parts of your program without getting a ton of verbiage from the parts you know are ok.

In addition to the basic DI3000 package, Precision Visuals puts out two other software tools, Grafmaker and Contouring. Grafmaker does presentation graphics (pie charts, bar charts, etc) quickly and easily. You can make use of the DI3000 routines to customize your graphs.

The Contouring system produces contour plots. A little surprisingly, Contouring is the most popular part of DI3000 at Cornell. People who used to use the NCAR routines have switched to DI3000, perhaps because the same person who wrote NCAR long ago also wrote the Contouring system more recently. John Hubbard and David Zimmerman from the Mathematics department are using Contouring in combination with ADABAS and our Floating Point Systems Array Processor in a novel application. They are exploring the characteristics of Mandelbrot sets, spaces in the complex plane of fractal dimension. They are producing some extraodinary pictures as well as doing abstract mathematics.

DI3000 sounds like a really spiffy graphics system, so we were eager to put it up under CMS. Someone from Precision Visuals apparently assured us that it would run under CMS. In fact it eventually did run under CMS, but only after a lot of sweat and many modifications. The major problem with the initial "CMS" release of DI3000 3.0 was that the installation instructions lulled me into a false sense of security. I got the impression that DI3000 did understand EBCDIC and would talk to my devices and the operating system fluently, that it was written in IBM FORTRAN, and that although each new obstacle I encountered was worse than the one before, it would surely be the last.

The problems we encountered in that first release of DI3000 were that the distribution tapes were incomplete, the design of DI3000 doesn't suit IBM systems in general and CMS in particular, the source contained non-IBM FORTRAN, there was no suggestion of what the user interface should look like, the installation documentation described steps common to all machines very well but lacked system-specific instructions for the rest of the steps, and initially, Precision Visuals wasn't very supportive of our struggles.

The first problem was trying to decide how large a minidisk I'd need to build the DI3000 system. Precision Visuals's prior IBM experience was with TSO, a system with essentially unlimited disk space. There was no way to tell how much minidisk space I might need to work on DI3000; the five cylinder minidisk I defined initially was way to small for even the source files. After a few iterations, I arrived at 30 cylinders of 3350 as a workable minidisk size.

The state of the distribution tape should have made me suspect that installing DI3000 was going to be harder than installing a typical software package. The systems that are easiest to install have been tested on the operating system they're being distributed for and are therefore shipped as object modules. The systems that are the most troublesome are shipped as source. Not only did Precision Visuals send us source, they sent us the wrong source or neglected to send parts of the source. The TSO-specific routines that Precision Visuals thought they had included weren't there. We asked for a Tektronix 4013 device driver; they sent us a 4014 driver that, due to the hardware differences between the two devices, never could have run. Most humorous was the file containing the ASCII I/O routines; it was written in ASCII rather than EBCDIC.

Once we'd copied the files onto the minidisk, Precision Visuals's installation instructions said, "Make the necessary changes to the machine dependent source." It took us two months to comply with this simple instruction, and in trying to do so we ran into most of the problems we were going to encounter. These problems divided themselves neatly into 5 categories: making software designed for a VAX work on an IBM system, finding and correcting non-IBM FORTRAN in the source, trying to make the I/O routines tolerant of CMS, and dealing with Precision Visuals's support group and documentation.

One problem that anyone with an IBM machine will have with DI3000 is that each source file on the DI3000 tape is composed of many subroutines. If you compile all these routines as one file and generate a CMS TXTLIB or OS object

library from the resulting TEXT file, you'll get a library with only one entry point. The loader would load the entire library into memory each time you used DI3000, even though you might need only a few of the routines. Precision Visuals suggested separating the source files into individual subroutines. I wasted a lot of time writing EXECs and EDIT run files to do that; there were just too many routines to keep track of, and it was too easy to miss one routine when I was trying to compile them all. Gary Buhrmaster, one of our programmers, solved the problem by writing an assembler routine to take the TEXT file you get when you compile one of the DI3000 source files and create a new TEXT file with ESD cards in front of each subroutine. The loader can then load only the routines it needs, saving you memory. You can use this new TEXT file to generate a TXTLIB with one entry for each subroutine. The assembler routine he wrote, ULOADM, is on Precision Visuals's CMS tape.

Since the rest of the world uses ASCII in their machines, it makes sense that DI3000 writes all its I/O buffers directly in ASCII. That would be fine if IBM machines spoke to their ASCII devices in ASCII. Unfortunately, most IBM systems communicate in EBCDIC and rely on a communications controller of some sort to convert output to devices to ASCII and input from devices to EBCDIC. If we allowed DI3000 to send out its ASCII buffers directly, the controller would cheerfully try to convert them into ASCII again and send garbage to the graphics device. To outwit the controller, DI3000 must translate its output buffers into EBCDIC so the controller can convert them back to ASCII for the device, and convert the ASCII buffers the controller receives from the devices and converts to EBCDIC back into ASCII. It is not very efficient to do things this way. but it's also probably not worth Precision Visuals's time to translate all DI3000's buffers into EBCDIC.

Precision Visuals was aware of the communications controller problem, so they supplied routines to do the character conversions. Every installation's translate tables are different, however, so you have to insert your tables into the DI3000 code. Since they weren't writing specifically for IBM machines, Precision Visuals didn't put sequence numbers on the code, so there was no way to make an update file containing your tables to apply to subsequent DI3000 releases. It turns out you can't use Precision Visuals's conversion routines anyway, since they tried to use LOGICAL*1 arrays for I/O buffers in a way that wouldn't work. To avoid extensive recoding, we used Gary Buhrmaster's revised versions of the PLOT10 ADEIN and ADEOUT routines. These routines call separate assembler routines (DSECTs) containing the conversion tables; you still have to change the tables; more about how to do that later. Precision Visuals now includes Gary's ADEIN and ADEOUT on their CMS tape.

DI3000 uses BYTE (LOGICAL*1, although the BYTE statements were left in the code) arrays to hold its buffers of ASCII characters. Both the original DI3000 character translation routines and Gary's modified PLOT10 translation routines used INTEGER variables. Since the two types are incompatible in IBM FORTRAN, we changed all BYTE variables to INTEGER variables. It solved the problem, but increased the buffer storage 300%.

We ran into even more problems getting around some of the limitations of IBM FORTRAN. One routine tried to initialize a variable in COMMON with a DATA statement. Apparently Precision Visuals was aware that you can't do that in IBM FORTRAN, because at the end of one of the assembler routines to perform logical AND or OR was a strange little CSECT that initialized the same COMMON variable. They didn't bother to document it. I think we ended up using the CSECT; it was easy, if not straightforward.

Another bit of leftover non-IBM FORTRAN was the "Q" format. "Q" format returns the number of characters in the string just input. We wrote a DDSTRP routine to simulate that function, but so many people were working on it at the same time that it doesn't work. It's on the CMS tape, though.

The last IBM FORTRAN limitation we had to deal with was the problem of dynamic FILEDEFs. DI3000 is parsimonious with its I/O units; it originally tried to use the same FORTRAN unit numbers for seven different files (six font files and the error message file). VAX FORTRAN lets you use an OPEN statement to dynamically allocate the chosen file to the unit number and open the file. Remnants of that syntax were left in the version we got, although the OPEN statement had oddly been changed to CALL OPEN. There was some confusion about whether we were missing an OPEN routine or whether that call was just bad syntax (it was just bad syntax).

There is no facility in IBM FORTRAN for dynamically assigning a file to a unit number. There was no facility in FORTRAN IV, the version of FORTRAN that DI3000 3.0 was written in, for explicitly opening a file. Since it makes sense not to FILEDEF all the FONT files to separate unit numbers if you really only wanted to use one of them, Ben Schwarz, another of our programmers, wrote four routines, TOKEN, CMSCMD, CUOPEN, and CUCLOS, that tokenize a line of text, send the line to CMS as a command, open a file, and close a file. We used TOKEN, CMSCMD, and CUOPEN to select a font file, issue a FILEDEF command for that file, and open the file while running DI3000.

We were then faced with the problem of what to do if the application used more than one font file in a run. If you want to reuse a FORTRAN unit number for another file under CMS, you have to close the current file first (otherwise CMS assumes that the second file has exactly the same attributes as the first). We used Ben's CUCLOS routine to close the file. Because the attibutes for the font files and the error message file were so different, we assigned the message file its own new unit number and issued a FILEDEF for it before running DI3000.

The JFILES routine lets you change the unit number(s) for one or more of DI3000's internal files (error message file, debugging output, graphics input, graphics output) dynamically. In this case, however, it was impossible to modify the routine to do dynamic FILEDEFs to these new units since we had no way of knowing (without rewriting DI3000 extensively) what the attributes of the files were and what media they were to reside on. We ask our users to issue their own FILEDEFs for these unit numbers before running DI3000 if they want to use JFILES to change them.

Finally, the buggy version of the VS FORTRAN compiler that we were using at the time (1.1.?) hindered our efforts to put up DI3000. Whenever you passed a character string to a subroutine, that early VS FORTRAN passed an implicit string length as the next parameter to the subroutine. Subroutines written in VS FORTRAN were clever enough to pick up that extra parameter automatically; subroutines, like CMSCMD, written in assembler weren't that clever. We also rediscovered an early bug in the compiler's computed GOTO handling code. The

compiler was using the storage area reserved for computed GOTO addresses as scratch space for implicit type conversions in assignment statements (e.g. the statement RVALUE = IVALUE could cause DI3000's computed GOTO to branch to the middle of your graphics data because it used the computed GOTO's branch table as scratch space to convert the integer value in IVALUE to floating point). (I did say in the title of this section that I'd learned to use PER while installing DI3000!) The moral of all that is, if you're going to develop a machine-specific version of software that doesn't yet run on your machine, at least use a compiler you are certain is stable.

CMS's limitations and quirks gave us trouble with the device drivers. We had to use ADEIN and ADEOUT rather than FORTRAN READs and WRITEs for most of the device driver I/O because FORTRAN in CMS uses the CMS RDTERM macro which can only read 130 characters/input line (no more, no less), and CMS thinks that each WRITE to the terminal must end with a carriage return/line feed, even when you're doing graphics output.* We had to change the length of all device driver's I/O buffers to 130 because that was the length RDTERM could use. We had to set off any terminal settings that would allow the terminal to receive asynchronous messages (handled by the DI3000 EXEC discussed later).

CMS also imposed restrictions on what kind of user interface we could set up for DI3000. Questions we need to answer were: How do you store DI3000 itself? In separate TEXT files? TXTLIBs (how many? how do you select the ones you want to use?) Since the routines that make up a device driver have the same names as the ones for all other device drivers, how do you store a device driver? How do you pick the correct device? How do you get around the fact that CMS lets you GLOBAL only up to eight TXTLIBs at a time? Since Precision Visuals had no CMS experience, they didn't suggest a possible user interface. We had to develop one ourselves.

What we came up with was the DI3000 EXEC. We put each section of DI3000 (DI3000, levels A, B, and C of segment storage support, Contouring, Grafmaker, and each device driver) in a separate TXTLIB. To use DI3000, you ask for the section(s) your application program uses; the EXEC GLOBALs the TXTLIBs corresponding to those sections. We got around the problem of needing to GLOBAL more than eight TXTLIBs by GLOBALing only the DI3000 libraries, LOADing the program, GLOBALing the FORTRAN and CMS runtime libraries, and using INCLUDE VSCOM# to pull them into core.

Since the routines that make up each device driver do not have unique names, presently there is no way to use more than one device in a single DI3000 run. Earl tells me that the TSO version of DI3000 uses concurrent device support and that he's certain we could get it running under CMS, but I haven't looked at it.

---

*Joe Shapiro, our part-time graphics support person, has since discovered that if you issue strange-looking FILEDEFs for the terminal you can READ and WRITE about 1600 characters before getting a CR-LF. Once you can output that many characters it becomes less impractical to strip off the CR-LFs. He thinks we'll be able to use FORTRAN READs and WRITEs for the device driver input and output after all, but he's still working on it.

Earl came up with a set of execs to do the same thing as the DI3000 EXEC. The difference is that you need to remember many commands rather than just one. There's more about the DI3000 EXEC and exactly what it does in the next section.

As I mentioned above, the DI3000 documentation was not much help when it came to trying to solve CMS-dependent problems, because the machine-dependent parts of the documentation were missing or vague. Along with the instruction that advised us to "verify [the] correctness [of the machine dependent routines]", there was a note declaring that "Code which may have to be changed for various files is heavily commented and easy to follow." There are about six printed inches of code in which those easy to follow comments might be located. It would have been helpful to have had a list of the subroutines containing the comments. Since any graphics system will do a lot of I/O, and since I/O varies greatly from machine to machine, it would have been helpful to have had a list of all I/O routines, at least, along with an index listing which DI3000 source file these routines could be found in, to give you an idea of which routines to start looking at. Earl has since told me that the name of each routine tells you which source file to find it in. An explanation of these naming conventions in the installation manual would have been handy.

The documentation about the device drivers was particulaly vague. It was not clear for a while whether Precision Visuals had supplied device drivers or whether they had supplied a skeleton for each driver that we were to flesh out ourselves. The notes sounded like they were trying to instruct people in the art of device driver writing, but without much detail. Once I believed the drivers they had sent might work without a lot of additional coding, I still couldn't tell from the documentation that the drivers could handle synchronous messages, but couldn't handle asynchronous ones. It never was clear what parts of the drivers needed to be changed. We did need to rewrite the I/O routines to call ADEIN and ADEOUT. Bad documentation for one of the routines calling ADEIN (MGRINP) resulted in setting the value of the constant "6" to "0". The installation documentation said that the variable RSPLEN was the length of the input string the device returned. The subroutine comments said that RSPLEN was the maximum length of the input string returned. The comments in the routine were correct. The installation documentation should be fixed in the next release of DI3000.

Now that Precision Visuals has all the modified-for-CMS software, installing DI3000 under CMS should be much easier in the future. If from now on they can make their CMS tapes "Slam Dunk" tapes, we won't need as much detailed documentation as we needed, but didn't have, this time.

At first, support from Precision Visuals people was not good. There was a TSO person on the staff, but no one who knew CMS. When I asked if anyone else had ever put up DI3000 under CMS, they said that a few had, but that Precision Visuals couldn't tell anyone else who they were. The only person they told me about was Samuel Chan at the University of Houston who was having trouble with the BYTE variable types. Even if Precision Visuals couldn't have told me who else had successfully installed DI3000 under CMS, I am surprised that the other installations hadn't written up notes from their experience that they could give Precision Visuals to share with other CMS installations.

When I finally ground to a halt trying to install the device drivers and we were seriously considering sending DI3000 back, Precision Visuals rallied and sent their itinerant consultant, Earl Billingsley, out to install DI3000 in return for a tape with the modified source and our utilities. Earl, an impressive professional, whipped into town, worked on the device drivers for a week, and got all the tests to run without having ever worked on a CMS system before. Precision Visuals is now distributing the modified DI3000 source and utilities that came out of this experience- more on that in the next section.

Since December 1982, Precision Visuals CMS support has improved dramatically. They are aware of the problems of installing DI3000 under CMS, and have hired Ken Tallman to be their resident DI3000 under CMS expert. Ken seems eager to learn all he can about DI3000 and CMS. He came to Cornell with Earl in June to install some DI3000 updates and a new AED device driver. He documents his work, so Precision Visuals should have CMS-specific documentation out in future releases of DI3000. It is nice to have this assurance of Precision Visuals's commitment to CMS. It was nice to have had the benefit of Earl's experience.

The hard part of getting DI3000 to work under CMS seems to be done. There are still a few things that don't work, but now Precision Visuals is distributing a version of DI3000 that doesn't need to be extensively modified in order to run. There are some modifications that any CMS shop will have to make for every release of DI3000, and there are some things on the current distribution tape that work only at Cornell. The next section of this paper talks about what you'll have to do to put up Precision Visuals's CMS tape for DI3000.

The June, 1, 1983 "Slam Dunk" Tape:

How to Really Install DI3000 Under CMS

Once Earl finished installing DI3000 for us in December '82, Precision Visuals took all the utilities and modified routines he sent back and combined them into a "Slam Dunk"* tape for CMS. The contents of this tape went up relatively easily at Cornell (before we got rid of MVT), but probably won't go up without some nasty surprises anywhere else unless you keep reading. There are things you may want to do differently and things you must change.

The first thing you do is get a large CMS minidisk to work on. Thirty cylinders of 3350 space was enough for us to put DI3000 up on if we used T-disks for holding some of the source files while we compiled them. The DI3000 Grafmaker, Contouring, and device driver TXTLIB's and two sample contouring DATA fils currently occupy 1543 cylinders of 2048-byte blocks on the 3375 that contains our 3081's Y-disk. PVI may tell you a number of cylinders to use; if you haven't bought all the parts of DI3000 or if your tape isn't the June 1 version go with PVI's number.

The next thing you do is set up the tape. On the tape are two EXECs, DISET and GMSET, for setting up the DI3000 and Grafmaker tapes, respectively. It is nice to have such EXECs if you discover you need to keep going back to one of the tapes to unload something you've forgotten; however, DISET and GMSET use an obsolete SETUP processor that ran on Cornell's heavily modified MVT system. Write your own SETUP EXEC.

Once the tape is set up, read the appropriate files off it. PVI supplies two EXECs to do that. The DI3RD EXEC reads particular files from the DI3000 tape; the GCRD EXEC reads files from the Grafmaker-Contouring tape. Unless PVI custom-tailors these execs for the features of DI3000 you ordered, DI3RD and GCRD will work only if you have exactly the same files on your distribution tape(s) as Cornell has on theirs. Compare the list(s) of tape contents PVI sent you with the lists in the handout. You may need to modify DI3RD and GCRD to read only the files you received into the correct CMS files. If you want to be able to ue PVI's BLD.... EXECs to compile the source and build your TXTLIBs, retain the fileids that DI3RD and GCRD give the source files when they load them off the tape. DI3RD and GCRD use a utility. DIFI, that should work without modification on any CMS system.

Now you'll need to make the necessary changes to the source. Precision Visuals supplies two routines, Q3ATOI and Q3ITOA, to translate EBCDIC to ASCII and vice versa. They tell you to alter the routines by changing the translate tables to match your system's tables. These two routines don't work well with IBM FORTRAN; you would have to do some recoding. The CMS-specific file on the tape provides four other routines that are easier to install than PVI's original translation routines. ADEIN and ADEOUT do the ASCII input and EBCDIC output, calling AS2EB and EB2AS to do the translation. All you need to do is modify the translate tables in MAS2EB ASSEMBLE and MEB2AS ASSEMBLE to match your installation's translation codes. John Baird's TABLES MEMO file that may

---

*A technical term meaning "installs itself painlessly".

or may not be on your tape has good instructions for choosing the values to put in the tables. Ignore the values in the sample tables in that memo, however; they're idiosyncratic Cornell values guaranteed to be wrong for your system. Earl and Ken already modified the device driver I/O routines to call ADEIN and ADEOUT.

Some of the device drivers contain non-IBM FORTRAN in the debugging code. Compile each driver once to find out where the errors are. Edit the source and either fix the code or, since debugging code isn't an essential part of the routines, delete it.

We continue to have problems with I/O to and from devices, particularly with input. JKEYBD, JLOCATE, and JPICK have given us the most problems, due to STRG4 and DDSTRP, two routines that are supposed to simulate VAX FORTRAN's "Q" format by stripping trailing blanks from an input string and returning the length of the resulting string. Truncation or round-off errors in JCONWV may be causing DI3000 to occasionally interpret coordinates input with a cross-hair cursor as being outside the current viewport. In general, the device drivers are the part of the code that need the most work, and that you should be most suspicious of.

Better device drivers may be on the way. Joe Shapiro, one of our part-time programmers, has become intrigued with how the device drivers work and has cleaned up some of the code. For example, by recoding LOWIOL, the main I/O routine, he reduced the number of FORTRAN WRITE statements required to put up a menu screen for his interactive transparency-generating program from 800 to six. He has improved the speed of the HP7221 plotter driver by 200%. With luck, we can afford to pay him long enough for him to develop some really efficient device drivers for use with CMS.

Once you have made the changes to the source, you need to compile it and store it in TXTLIBs. PVI suppiles eight execs to compile, load, and build a library of the seven sections of the DI3000 system (DI3000, Contouring, Metafile Generator, Metafile Translator, and Level A, Level B, and Level C of segment storage support). They call them, respectively, BLDDILIB, BLDCTLIB, BLDMFG, BLDMFT66, BLDMFT77, BLDLEVA, BLDLEVB, and BLDLEVC. Before you run any of them you'll need to GLOBAL the assembler MACLIB(s) yourself. BLDDILIB, BLDCTLIB, BLDMFG, BLDMFT66, and BLDMFT77 try to compile the routines DI3RD or GMRD EXEC read off the tape. If you didn't use those execs to copy the source off the tape, be sure you have given the DI3000 source files the same names those two execs do. Again, if you don't have the same tape files as the ones Cornell got you'll need to edit the BLD.... execs and delete references to the files you don't have.

These same caveats apply to BLDLEVA, BLDLEVB, and BLDLEVC. You may not have bought all three levels of segment storage support, so of those execs, run only the one(s) you need. Unlike the other four BLD.... execs, these three leave the compiled code in a TEXT file rather than storing it in a TXTLIB. I find these TEXT files clutter up an already cluttered Y-disk, so I just TXTLIB GEN each into its own separate TXTLIB. Any segment storage level uses all its routines whenever you call it, so you can get by with only one entry point for the whole TXTLIB rather than needing a separate entry for each routine in that level.

These eight execs use one or more of three utility execs. COMP66 compiles FORTRAN with the LANGLVL(66) option of the VS FORTRAN compiler. If you don't have VS FORTRAN, you'll want to change the name the exec uses for the command invoking your FORTRAN compiler. Even if you have VS FORTRAN you may want to change some of the compiler options the exec uses.

COMP77 compiles using the LANGLVL(77) option of VS FORTRAN. Again, you may want to change some of the compiler options used.

The third utility, ULOADM, takes the TEXT file created when you compile one of the DI3000 source files containing numerous subroutines and converts it to a TEXT file with a separate ESD card for each routine. What that means is that if you TXTLIB GEN the original TEXT file into a TXTLIB you'll end up with a library with about 200 names listed and only one entry point; if DI3000 loaded one of those routines it would have to load them all. If you TXTLIB GEN the ULOADMed TEXT file you'll get a TXTLIB with 200 names and 200 entry points; DI3000 can then load only the routines it needs, saving you space. I don't know what use ULOADM DATA is, although Ken and Earl maintain you need it.

PVI includes two other execs for doing compilations, BCOMP and COMDITST. COMDITST compiles DI3000 test routines. You may need to change the name of the command it uses to invoke FORTRAN (FORTVS).

BCOMP sends a compilation job to Cornell's obsolete MVT system to compile while you do something useful. It is garbage; can't possibly work anywhere else. Write your own batch compile exec, if you have the facilities; otherwise compile in CMS while you eat lunch or perform some other useful task away from the keybored.

As far as I can tell, PVI has no execs specifically for compiling device drivers. Compile them your favorite way, or use one of the BLD.... execs as a template for your own device driver compiling exec, and use ULOADM and TXTLIB GEN to create a separate library for each driver.

PVI does provide execs to GLOBAL the necessary libraries, load and run your application program with DI3000. They also provide the DI3000 EXEC, which does the same thing in a more friendly way (fewer command names to remember).

The DI3LB, GMLB, and CONTLB EXECs GLOBAL the TXTLIBs you need to run DI3000, Grafmaker, or Contouring, respectively. Inside each EXEC, the names of the libraries on the GLOBAL command line from "VFORTLIB" to the end of that line may be peculiar to Cornell. These are the FORTRAN and CMS standard run time libraries. Use your system's names if they're different.

The DI3LD, GMLD, CONTLD, and METRNS EXECs load your program with the DI3000 base routines and the segment storage level and device driver you ask for, then execute the program. For some reason, they also always load the Metafile Generator routines; wasting space if you don't need the Generator, and causing an error if your shop didn't purchase the Generator. Additionally, CONTLD issues FILEDEFs for the PVI sample DATA fies; they're large enough that you may not want to keep them on your Y-disk.

DI3LD, GMLD, CONTLD and METRANs use the ERRFI utility EXEC to set up a FILEDEF for the DI3000 error messages file. At Cornell, we called this message file "DI3000 MESSAGES" rather than using PVI's name "ERROR MESSAGES" to avoid confusion with other message fies on the Y-disk.

DI3000 EXEC is much nicer*, but depends on VM/SP and EXEC2. You'd need to rewrite the EXEC if you don't have both; probably not worth the effort. You'll otherwise need to change only four lines in the EXEC to make it work on your system. First, we use the line beginning "SMSG USE ..." at Cornell to collect data about how often a package is used per month. You don't need this line, so delete it. Second, the EXEC uses Cornell's HELP processor rather than IBM's to display info about the DI3000 command. You need only replace the line "HELP DI3000(S" with a call to your own HELP processor, or with the syntax of the DI3000 command. We have routines at Cornell to transform IBM-format HELP into Cornell format HELP; we might be able to persuade them to work in reverse. Third, at Cornell, rather than GLOBAL a lot of TXTLIBs by hand whenever we want to compile and/or load an applications program, we run an EXEC to GLOBAL the relevant libraries. The DI3000 EXEC expects you to give it the name of such an EXEC or uses "FORTLIBS EXEC" by default. You need only replace the line "EXEC &LIBS" with a statement that GLOBALs the list of VS FORTRAN and CMS runtime libraries. For increased flexibility, you could write your own one-line FORTLIBS EXEC. Fourth, PVI calls their file of error messages ERROR MESSAGES; we call it DI3000 MESSAGS to avoid confusing it with other message files on the Y-disk. If you don't like "DI3000 MESSAGES", change it to "ERROR MESSAGES" in the exec.

PVI includes some other execs and routines in their collection of CMS-specific files. RENSUF EXEC is left over from a predecessor of ULOADM and is garbage. The conversion tables in CMSTABLES.DAT on the tape are another version of Cornell's translate tables; use them as models, but ignore the values in them. As I mentioned before, STRG4 FORTRAN and DDSTRP FORTRAN probably don't work. XGLOBAL EXEC, a utility to add to or change the names of currently GLOBALed libraries, is included on the tape, but doesn't seem to have been used by any of the runtime execs.

The chart on the pages that follow contains all the information presented in this section in a format that's easier to use as a reference. Following the chart is an outline of the DI3000 EXEC, then the complete listing of Cornell's DI3000 distribution tapes. (which you also have in your handouts). Using these lists, you can check whether the files on your tape are in the same order as the files on our tape, and thus tell how much you'll need to modify PVI's ....RD, BLD...., and ....LD utility execs to get them to work for you.

Precision Visuals has come a long way toward a true CMS version of DI3000 in a short time. We no longer need to worry about encountering non-IBM FORTRAN in the source code or ASCII files on the tape. The utility EXECs make it easier to build the DI3000 libraries, especially if you're not familiar with how all DI3000's components fit together. The DI3000 EXEC or the DI3LD, GMLD, CONTLD, and METRNS EXECs provide a well-defined user interface where there was nothing before.

All this is not to say that DI3000 works perfectly from CMS. There are still bugs to work out of the device driver code and the I/O routines. The utility EXECs are still in a rough form, and need to be documented better. Now that we have a rough idea of what steps are necessary to put DI3000 up under CMS and how CMS users will interface with DI3000, someone at PVI needs to take Earl's and Ken's notes and write up CMS-specific installation and user's manuals.

Precision Visuals has demonstrated their commitment to CMS and IBM machines by hiring Ken Tallman to work full-time on making DI3000 work from CMS. Their support and their product is enormously improved from what it was just nine months ago. That's enough to make me hope we may see that true CMS version of DI3000 in another nine months.

---

*I'm biased, though, since I wrote it.

MODIFICATIONS TO DI3000 EXECS AND UTILITIES

In general, read the comments at the top of each utility before you try to use it.

I. Setup Tape

DISET EXEC GMSET EXEC	Set up DI3000 and Grafmaker/Contouring tapes, respectively. Depends on obsolete Cornell setup processor. Write your own tape setup EXEC.

II. Read the Tape

DI3RD EXEC GCRD EXEC	Unless the contents of your tape(s) exactly matches the contents of Cornell's (see attached tape lists), modify these two EXECs to load the correct disk files from the correct tape files. Retain PVI's fileids if you want to use the BLD..... EXECs to compile, link, and build libraries from the source.
DIFI EXEC	Utility routine for DI3RD and GCRD. Should work ok; only difference could be the tape density (the exec uses 1600 bpi).

III. Change the Source

Q3ATOI FORTRAN Q3ITOA FORTRAN	PVI's routines to translate ASCII to EBCDIC and vv. Contains translate tables consisting of decimal codes for ASCII or EBCDIC characters. These routines don't work well with IBM FORTRAN; you have to do some recoding.
ADEIN ASSEMBLE ADEOUT ASSEMBLE MAS2EB ASSEMBLE MEB2AS ASSEMBLE	These are Gary Buhrmaster's routines; Earl and I agree that they're easier to install than Q3ATOI and Q3ITOA. PVI's tape contents document says you need to supply these routines from the PLOT10 library. They are actually included on the tape in the IOASCCMS.FOR file; you need only supply the values for the translate tables, MAS2EB and MEB2AS ASSEMBLE. (These files might be called AS2EB and EB2AS instead.) The translate table entries are the hexadecimal codes for the characters to translate to. The file TABLES MEMO, if it's on your tape, has good suggestions for how to decide what values to put in the translate tables. Since translate tables are truly installation dependent, build your own tables using the sample attached to this handout as a model and your own installation's translation codes. Keep the assembled text for these routines in the DI3000 TXTLIB.
CONVERSION TABLES (CMSTABLES.DAT)	Garbage. These are Cornell's own ideosyncratic translate tables, guaranteed to be wrong for your system.
CUOPEN FORTRAN CUCLOS FORTRAN CMSCMD ASSEMBLE TOKEN ASSEMBLE TEDON ASSEMBLE TEDOFF ASSEMBLE	Utilities to open a file, close a file, send a command to CMS, tokenize a line to pass to CMS, and turn TERMINAL settings on and off, all dynamically from within DI3000. Essential for DI3000's on-the-fly FILEDEFs and terminal I/O. Keep the text for them in the DI3000 TXTLIB.
Device Drivers	Some of the debugging code has non-IBM FORTRAN; compile the routines to find out where the bad code is. Edit the source and either fix the code or comment it out. In general, the device drivers and I/O routines are the ones most likely to give you problems. Test JKEYBD, JLOCATE, and JPICK thoroughly.

IV. Compile Source and Build Libraries

BCOMP EXEC	No longer works even at Cornell. Write your own exec to send the compilation off to your batch system (if you have one) or compile the routines interactively while you go to lunch.
BLDDILIB EXEC BLDCTLIB EXEC BLDMFG EXEC BLDMFT66 EXEC BLDMFT77 EXEC	Compiles, links, adds DI3000 routines to a TXTLIB with one entry per routine. You must GLOBAL the Assembler macro libraries yourself before running any of the BLD..... EXECs. Take the names of the routines your installation didn't buy out of these EXECs and use only the EXECs corresponding to the pieces of DI3000 you bought. BLDDILIB handles the basic DI3000 routines, BLDCTLIB applies to Contouring, BLDMFG applies to the Metafile Generator routines, and BLDMFT66 handles the Metafile Translator routines.
BLDLEVA EXEC BLDLEVB EXEC BLDLEVC EXEC	Build the level A, B, and C segment storage TEXT files, respectively. PVI leaves the text in TEXT files; I feel that adds to the clutter of TEXT files on the Y disk. I TXTLIB GEN each TEXT file so you get only the level you explicitly GLOBAL and don't get the text accidentally when you meant to load your own routine called "LEVELA". You may not have bought all levels.
COMP66 EXEC COMP77 EXEC	Utilities that compile DI3000 source using VS FORTRAN LANGLVL(66) and LANGLVL(77) respectively. Called by BLD..... EXECs. You might want different compile options. The command to invoke FORTRAN on your system might not be "FORTVS".
COMDITST EXEC	Compiles test routines. You might want different compile options. The command to invoke FORTRAN on your system might not be "FORTVS".
ULOADM EXEC ULOADM MODULE	Utility that takes TEXT file made up of many routines but only one entry (ESD) and produces a TEXT file of many routines with one entry (ESD) per routine. Called by the

BLD..... EXECs. You (or the BLD..... EXEC you're using) can TXTLIB GEN a TXTLIB with one entry per routine from this TEXT file.

V. GLOBAL Libraries Before Running Application

DI3LB EXEC GMLB EXEC CONTLB EXEC	Library names from "VFORTLIB" to the end of the line in these EXECs may be Cornell-specific. Substitute the names of the VS FORTRAN and CMS runtime libraries from your system where you need to.

VI. Load and Run Application

DI3LD EXEC GMLD EXEC CONTLD EXEC METRNS EXEC	These are PVI's utilities for running DI3000, Grafmaker, Contouring, or the Metafile Translator. Not as flexible as the DI3000 EXEC. All except METRNS always load the Metafile Generator, too. CONTLD always sets up FILEDEFs for PVI's Contouring sample DATA files.
ERRFI EXEC	This is a utility called by the ....LD and METRNS EXECs. It sets up a FILEDEF to ERROR MESSAGES *, DI3000's error message file.
RUNTST EXEC	Runs the DI3000 test program of your choice.
DI3000 EXEC	GLOBALs libraries for, LOADs, and optionally RUNs any DI3000 application program. Requires VM/SP and EXEC2, but is more flexible than PVI's runtime execs. You definitely have to make the following four changes to use it. 1) Delete the line containing SMSG USE. 2) If the syntax of your HELP is different from "HELP DI3000 (SYNTAX" you'll need to change that line or replace it with lines that type the syntax of the DI3000 command. 3) We use an exec named FORTLIBS to GLOBAL the VS FORTRAN and CMS runtime libraries; either replace the "&LIBS" line with a GLOBAL command for these libraries, or write your own FORTLIBS EXEC consisting of a line to GLOBAL these libraries. 4) We renamed the DI3000 error message file DI3000 MESSAGES instead of ERROR MESSAGES to avoid confusing it with message files for other packages on the Y disk; either rename the file or change "DI3000 MESSAGES" to "ERROR MESSAGES".

VII. Miscellaneous

XGLOBAL EXEC	Utility that lets you change or add to GLOBALs already in effect. Not sure why it's included.
RENSUF EXEC	Left over from a predecessor to ULOADM; garbage.
STRG4 FORTRAN DDSTRP FORTRAN	Probably don't work, and thus probably the roots of JKEYBD problems. The assertion in the tape contents document about "non-fence" characters in relation to STRG4 is absurd. It was a comment from an old version of STRG4 written before I knew that the CMS RDTERM macro couldn't handle lines longer than 130 characters.

ADEIN and ADEOUT expect an assembler DSECT of the following form that they use as a translate table. Unlike the translate tables from PVI's Q3ATOI and Q3ITOA, which contain decimal values for the characters to translate to, the DSECTs contain hex values.

```
EB2AS    CSECT
*
*-------------------------------------------------------------------------
* This is part of Cornell's EBCDIC => ASCII table.  It probably is wrong
* for your installation.  Use it as a model to build a similar table
* containing your system's translation values. For example, if your
* system translates the EBCDIC value 'F0' to the ASCII value '30', put
* '30' in row F, column 0 of the table.
*-------------------------------------------------------------------------
*
*              0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
*              |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
         DC   X'00,01,02,03,1A,09,1A,7F,1A,1A,1A,0B,0C,0D,0E,0F'  -0
         DC   X'10,11,12,1A,1A,1A,08,1A,18,19,1A,1A,1C,1D,1E,1F'  -1
          .     .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .     .
          .     .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .     .
          .     .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .     .
         DC   X'30,31,32,33,34,35,36,37,38,39,1A,1A,1A,1A,1A,1A'  -F
*              |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
*              0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
         END
```

What the DI3000 EXEC Does

The DI3000 EXEC is written in EXEC2 and depends on features of VM/SP. It uses the following CP and CMS commands:

CONWAIT	GENMOD	LOAD	SET
DROPBUF	GLOBAL	MAKEBUF	START
EXECIO	HELP	QUERY (STACK	
FILEDEF	INCLUDE	SENTRIES	

The DI3000 EXEC works like this:

1. Checks for ? among arguments and prints help if ? is found. Uses Cornell's HELP processor, but it should not be too difficult to convert the help file to IBM HELP format.

2. Checks for R/W A disk.

3. Sets default options. You can change the defaults in this section.

4. Parses command line

   a. Checks that at least the minimum abbreviations were requested.

   b. Compares each option against list of legal options. You can change the list of acceptable options.

   c. Parses all options, even if a bad one is found. Does not load or run routines if one or more bad options found.

   d. Save current CMS settings and turns off any settings that could cause an asynchronous message to appear in the graphics area if certain message-sensitive devices were requested. You can modify this section to turn other settings off or include other devices in the list of sensitive ones.

5. Sends a message to USE to keep track of the number of times this exec was successfully invoked. Delete this line.

6. CLEARs and sets FILEDEFs for TERM and the error message file. Cornell calls the file DI3000 MESSAGES; PVI calls it ERROR MESSAGES.

7. GLOBALs TXTLIBs corresponding to the sections of DI3000 requested.

8. LOADs application program using the CLEAR option.

9. INCLUDEs VS FORTRAN and CMS runtime libraries (we had too many TXTLIBs to GLOBAL them all).

10. GENMODs and/or RUNs loaded module depending on options requested.

11. Cleans up (restores any saved settings, CLEARs FILEDEFs).

Throughout, the exec checks for error conditions and uses EXECIO to print CMS-style messages.

SHARE

SHARE SESSION REPORT

SHARE NO.	SESSION NO.	SESSION TITLE	ATTENDANCE
61	A733	Using the Fortran Extended Error	
PROJECT		SESSION CHAIRMAN	INST. CODE
Fortran		W. Horowitz	CSS

D & B Computing Services 187 Danbury Rd. Wilton, Ct. 06897

SESSION CHAIRMAN'S COMPANY, ADDRESS, AND PHONE NUMBER

USING THE FORTRAN EXTENDED ERROR HANDLER

May 1983

Pat Hennessy

Hughes Aircraft Co.
2000 E. El Segundo Blvd. E1/F128
El Segundo CA 90245
(HUG)