Methodologies for Preparing and Integrating PostScript Graphics

J.T. RENFROW

Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109
TRENFROW@jpl-pds.jpl.nasa.edu

ABSTRACT

Graphical material represented in the PostScript language can be incorporated into TEX-based documents and ultimately printed on a printer having a PostScript interpreter. Successfully completing this incorporation process requires an understanding of PostScript and its use in programs to generate graphics and in programs associated with document preparation. It also requires an understanding of the particular DVI-to-PostScript translator program being used. PostScript concepts such as a stack of dictionaries, commands such as "translate", "rotate", "scale", and the idea of saving the state of virtual memory must be mastered. Different graphics generation programs use somewhat different mechanisms to produce the PostScript description of a graphic object. The graphics programs Adobe Illustrator, Cricket Draw, and MacDraft illustrate the diversity of possible PostScript output files. Text manipulation programs, such as AWK, can be used to examine these PostScript output files, establish the correct origin for these files when incorporated into documents, and prepare them for inclusion in TEX-based documents. A number of special cases need to be considered depending on the use of landscape or portrait modes in both the TEX-based document and the graphic object.

1. Introduction

Incorporating PostScript code into a document being processed by TEX is challenging initially but once the techniques and approach are worked out you will be able to incorporate PostScript code produced by graphics programs very rapidly and easily. To set up the process initially may require someone who is willing to "hack" or experiment a little. After this, anyone who can use TEX should be able to incorporate this type of PostScript code into a document with relative ease.

In this paper I will illustrate how to incorporate PostScript code representing graphical figures — lines, circles, shaded figures, text, etc. — into documents. I will not illustrate how to incorporate images but the techniques used here should apply to that graphical form also. I will illustrate the techniques using three common software packages: MacDraft, Adobe Illustrator, and Cricket Draw. I have chosen these three packages because they require somewhat different approaches as one incorporates their graphical material into the TEXed document.[1]

It seems that the success of incorporating PostScript graphics material into TEX documents can be very dependent on the hardware/software configuration. For that reason I am carefully documenting the configuration I use:

a. Hardware - IBM PC/AT, Macintosh Plus, QMS 810 Laser Printer
b. Software - DOS 3.3, MacLink Plus, Mac OS 4.2, Laser Prep 5.0, Cricket Draw 1.1, MacDraft 1.2a, Adobe Illustrator 1.1, MicroTEX Version 1.5A1, DVIPS 4.0.4, AWK (MS-DOS MKS Toolkit Version 2.2c from Mortice Kern Systems, Inc.)

If you are using other software (for example, a different DVI-to-PostScript processor), you will probably have to do things differently. Hopefully you will have enough courage or insight after reading this paper

[1] The phrase "TEXed document" means a document that was produced using the TEX typesetting system.

that you will try to incorporate graphics using your configuration.

I employ a uniform approach to the generation of PostScript and TEX material. The graphical material is generated on a Mac Plus using one of the three packages previously identified. Using techniques to be described later I generate a file containing the PostScript code associated with the graphical material. This file is transferred over to the IBM AT using MacLink Plus. The new file is processed by several AWK programs which insert, change, or remove PostScript commands as appropriate. Via these AWK programs I am also able to preview the diagram itself so that I can determine information needed to properly center the diagram on the printed page. Commands using the `\special` command in TEX are placed in my text file. These commands are then passed through untouched by TEX through to the `dvi` file. The `dvi` processor (in my case DVIPS from ArborText), then interprets the `\special` commands, includes the appropriate PostScript files, and prepares a PostScript file for the printer. This file is then sent via a parallel communication cable to the printer. The resulting document contains the integrated text and the graphical material.

These techniques of incorporating graphical material into TEXed documents may not be useful in every case. If you are producing only one version of a document that contains only one or two diagrams then it may be simpler to paste in a copy of the graphical material. These techniques become useful when you have many figures to include in a document or when multiple versions or multiple drafts of the document will be prepared.

The remaining sections of this paper explain the techniques used to incorporate graphical material. First an explanation of relevant PostScript concepts is presented, including the concept of a PostScript header file. After this, the techniques for producing the PostScript file representing graphical material are presented. The techniques for manipulating and converting this PostScript file are then explained. Finally some techniques useful in printing the document are discussed. Appendix A contains some AWK code that is used in processing the PostScript files.[2]

2. Understanding PostScript

PostScript is a language used to describe where to put certain objects on a page. Of course this is a very simplified statement of the capability of PostScript, but it is sufficient for the mental model that I am trying to help you build for PostScript. I am including this material to help you *read* PostScript and write a few simple commands in it. To learn more about it you should consult either the second or the fourth reference in the Bibliography.

The initial coordinate system for PostScript has 72 units to the inch, with the origin at the lower left corner of the page. Thus the position (144,72) would correspond to a point 2 inches to the right of the lower left corner and 1 inch above the lower left corner.

PostScript uses post-fix notation; that is, the parameter or arguments are given first and then the command follows. For example, `100 100 moveto` tells PostScript to move its current point to position (100,100). PostScript also assumes a stack architecture — parameters are placed on a stack and then removed or manipulated by the operators.

It is important to understand how PostScript knows what to do when it sees an operator. The simple answer is that the PostScript interpreter looks up the operator in a dictionary to see what it means. That is a simple model to follow. In fact, a user can create a stack of dictionaries for the PostScript interpreter to use. The interpreter looks up the command in the top dictionary first. If it finds the entry there then it does what the dictionary entry says. If it does not find the entry in the first directory it looks in the second directory. If it finds the entry there then it does what the dictionary entry says. The interpreter continues this process through all the directories on the dictionary stack until it finds some definition for the operator. If the interpreter has searched all the directories on the dictionary stack and it has not found the operator, it signals that a bad operator has been given. If the same operator name appears in two different dictionaries then the definition in the dictionary that is above the other in the stack will be used. This provides a very easy way to make sure that PostScript does exactly what you want with the commands you use. You create a new dictionary on top of all the other dictionaries and put your special commands in it. This is the way that most drivers

[2] A full set of documentation for this work, including complete examples, can be obtained by sending a message to either the electronic or physical address of the author.

for PostScript work. We will talk about these special dictionaries in the next section.

The PostScript commands that you need to learn for this work fall into a few categories:

1. Translating, scaling, or rotating the material to be placed on a page.
2. Creating a dictionary and putting it on the dictionary stack and then removing it.
3. Defining a new operator.
4. Saving and restoring the state of the entire PostScript interpreter or just the graphics state.

2.1 PostScript Movement Operators

These operators will be used to adjust your graphical material to appear on the printed page at the right location. *The methodology advocated in this paper is to place the bottom center of the diagram at the coordinates, (0,0).* The commands used are **currentpoint**, **moveto**, **translate**, **scale**, and **rotate**.

a. **currentpoint** – This puts on the stack the coordinates of the "PostScript pointer" at the current time. This is useful when PostScript knows where it is but you don't.

b. **moveto** – This command moves the current location of the PostScript writing pen to the coordinates you designate. For example **72 72 moveto** moves the graphics pen to the point one inch up and one inch in from the bottom left corner of the paper.

c. **translate** – This command translates the origin to a new point. For example, **144 144 translate** moves the origin up two inches and over two inches. A common use for this command in the context of placing graphical material into documents is **currentpoint translate**. You may have positioned your diagram so that the bottom center of your diagram is at (0,0). When the DVI-to-PostScript program tries to put your diagram on the page the current point may be (212,345.33). By putting **currentpoint translate** at the beginning of your graphics PostScript file, you have removed any need to know exactly where your diagram will go on the page.

d. **scale** – This command scales the coordinate system. It can be used to shrink the coordinate system or expand it. The shrinking or expanding may be non-uniform (i.e., the shrink may be bigger in the x direction than in the y direction). For example, **2 2 scale** expands all the coordinates by 2. Now the point (72,72) would be two inches over and two inches up. We have to use the **scale** operator because DVIPS does some necessary scaling that we, nonetheless, must undo. Another useful but initially confusing example is **1 -1 scale**. This produces a flip of the graphics along the x-axis. We have to use this when dealing with CricketDraw output.

e. **rotate** – This command rotates the coordinate system. In this paper it will be used to convert diagrams from landscape to portrait and vice versa.

The diagrams presented in Figure 1 illustrate how these operators can be used to place the bottom center of the figure at the origin. For each diagram, additional code has been created to put an axis and rectangles indicating four 8 1/2 × 11 inch pages so that the reader can better see how the transformations are taking place. Two new operators (**makedarkrectangle** and **makedarktriangle**) have been defined to produce the geometric figures.

Determining which transformation should be applied first and which second, etc. has always been counter-intuitive to me. As you can see from the examples, you should first take the code that draws the figures, then precede it by the first transformation. Then precede all this by the second transformation, and then precede all that by the third transformation, and so on.

Figure (1a) represents the initial diagram which consists of one rectangle and one triangle. Figure (1b) represents the diagram with the bottom center of the diagram shifted to the origin. Figures (1c) and (1d) represent a translation of the original code to put the left side center at the origin and then a rotation to put the diagram "bottom centered" at the origin when the diagram is in a landscape position. Figures (1e) and (1f) represent the same initial translation to the origin as in Figure (1b) but then a scaling by .5 in each direction to shrink the diagram so that it is centered at the origin but in a smaller size. This last technique is useful for adjusting a diagram when it is initially too large to fit in the space allocated for it in the document.

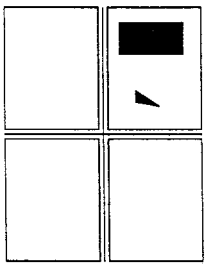

Figure 1a:
makedarkrectangle
makedarktriangle

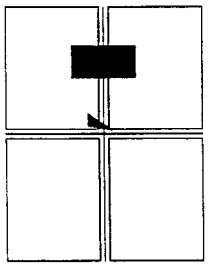

Figure 1b:
-300 -150 translate
makedarkrectangle
makedarktriangle

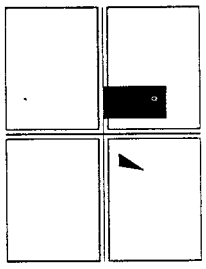

Figure 1c:
-100 -400 translate
makedarkrectangle
makedarktriangle

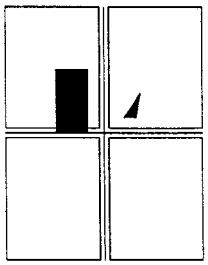

Figure 1d:
90 rotate
-100 -400 translate
makedarkrectangle
makedarktriangle

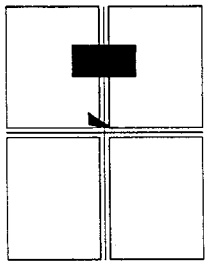

Figure 1e:
-300 -150 translate
makedarkrectangle
makedarktriangle

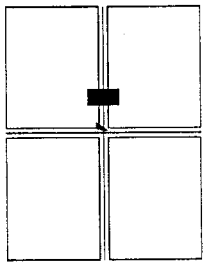

Figure 1f:
.5 .5 scale
-300 -150 translate
makedarkrectangle
makedarktriangle

Figure 1: Illustrations of PostScript movement operators

2.2 PostScript Dictionary Operators

This paper covers only a few of the many PostScript operators used for manipulating dictionaries. The operators discussed here are used to create a new dictionary, and to put a dictionary on the stack of dictionaries being used by PostScript and to remove a dictionary from this stack. You need to realize that PostScript can store dictionaries "over in the corner" when it does not have them on the active stack of dictionaries. Via the appropriate command, a dictionary "over in the corner" can be brought out of storage and put back on the stack. All its entries are still intact.

The way to create a brand new dictionary is via a command string such as **`/mydict 200 dict def`**. This defines a new dictionary, which I have arbitrarily named **`mydict`**, which has room for 200 entries and which is currently stored "over in the corner".

The way to bring a dictionary (for example, one named **`myoldone`**) that is "over in the corner" to the active stack is to give the command string **`myoldone begin`**. Notice that we did not have a "/" at the beginning of **`myoldone`** because the name is already known to PostScript.

The way to remove a dictionary from the active stack is to give the command **`end`**. This removes the top dictionary from the stack and places it "over in the corner".[3]

2.3 PostScript Definition Operators

Just as macros are used in TEX to define useful combinations of basic commands, so procedures are used in PostScript to combine useful combinations of PostScript operators. A simple form of this definition is **`/myownprocedure`** { *some PostScript code* } **`def`**. This process causes the name **`myownprocedure`** to be added to the topmost dictionary in the stack of dictionaries and the corresponding PostScript code to be entered as the meaning of **`myownprocedure`**.

With this form of defining a procedure, when PostScript encounters **`myownprocedure`** it will find the corresponding PostScript code in the dictionary and proceed to look up the *current* definition of

[3] I have continued to use the phrase "over in the corner" to impress upon you that dictionaries don't disappear just because they are removed from the stack.

each of the PostScript operators and then execute them. This in fact is the way that TEX behaves when macros are expanded normally.

There is another way to define procedures in PostScript which can be a bit confusing at first but is very helpful once it is understood. This involves the use of the **bind** operator. If a PostScript expression such as `/moveit { 100 100 translate } bind def` is used, then the **bind** operator looks up all the operators — in this case **translate** — in all the dictionaries. If it finds that **translate** has been re-defined from its original meaning (by having been defined again in some new dictionary) then **bind** does nothing. If it finds that **translate** has not been re-defined then it associates **translate** with its original definition and in fact causes this name to point directly to that code. This has two beneficial results. The first benefit is that when the **moveit** procedure is called, the PostScript interpreter will not have to go through all its dictionaries for the definition of **translate**. It can immediately execute the code associated with it. The second benefit is that even if between the time that the **moveit** procedure is defined and the time that it is executed another dictionary is added to the stack of dictionaries which re-defines **translate**, the **moveit** procedure is unaffected.

It is important to understand the necessity for using the **bind** operator at times. In some cases header files can interfere with one another and the person producing the document must load header files in such a way that the interference is avoided. This actually occurs when the Adobe Illustrator header file is loaded on top of the DVIPS header file. This will be discussed more later. This problem was solved by loading the Adobe Illustrator header file permanently, while the DVIPS header file dictionary was not on the dictionary stack, and the **bind** operator was used on the affected Adobe Illustrator code. The new release of DVIPS (Version 4.6) from ArborText has fixed this problem.

2.4 PostScript State-Preserving Operators

When you insert PostScript code for diagrams into PostScript code that will cause the document to be typeset, the inserted code should make sure that when it finishes execution, the PostScript interpreter is in the same state as it was when the inserted code began execution. In TEX we could accomplish this by grouping, using { and }. In PostScript this is accomplished by using the **save** and **restore** operators. The way that these are used is as follows:

```
/vmsave save def
many lines of PostScript code
vmsave restore
```

In the remainder of this paper this convention is referred to as a "save–restore" encapsulation. Other distinct combination of letters could have been used instead of **vmsave**. The first line of code defines a "save" object, which reflects the state of the PostScript interpreter's virtual memory at that time. The last line of code restores the state of virtual memory to this same state. All new definitions are forgotten, all new graphics operations are forgotten, etc. **Warning**: the operand stack is not fixed so you must make sure that your code leaves the operand stack just as it found it.

Usually any decent program producing PostScript code leaves the stack clean. These two lines of code are usually placed at the very beginning and end of the PostScript code for the diagrams that you are inserting. Sometimes the DVI-to-PostScript program will automatically insert these.

Another "save and restore" sequence used for saving the graphics state is **gsave** and **grestore**. These operators are easier to use than **save** and **restore** but are less powerful. The way that they are used is as follows:

```
gsave
many lines of PostScript code
grestore
```

The graphics state (i.e., the line width, the fill value, the current path, the current point, etc.) after the **grestore** will be the same as it was immediately before **gsave**. Any new definitions created will not be removed. Only the graphics state is restored.

3. Understanding and Using Header Files

Understanding header files can be easier if one is familiar with the concept of a file of macros for TEX. For example, the `plain.tex` file discussed in Appendix B of the *TEXbook* is such a set of macros. Both the header files and the files of macros are created to contain abbreviations for very commonly used sequences of commands. In both cases these files are loaded into the "interpreter" before the other code is sent to the interpreter. This can simplify the complexity of the other code that is sent since the code can use these abbreviated forms of commands rather than laboriously repeat the full sequence each time they are used.

Just about any program that produces PostScript output has a header file associated with it. DVIPS, Cricket Draw, Adobe Illustrator and MacDraft each have a header file associated with them. The header file is independent of the actual diagram and document that is produced. Thus, once a header file is analyzed and conquered (you feel like using that language after you have dealt with some header files) you never have to worry with it again. You don't have to reconsider it each time a new diagram or document is produced. However, you need to make sure that the version of the header file that you have created and stored (perhaps some time ago) is the same one that the application is currently using. This is particularly important with the standard Laser Prep file used by MacDraft and most of the other Macintosh applications. Compare version numbers, which should be contained in each file.

Two nice examples of header files are those associated with Adobe Illustrator and Cricket Draw. In the following paragraphs I will tell you how to produce these header files and how to load them into the printer.

3.1 Adobe Illustrator Header File

The header file for Adobe Illustrator can be produced simply. Create a diagram using Adobe Illustrator. For example, a diagram consisting of a single line is fine. Save this diagram. The Illustrator file will appear as a simple text file of PostScript commands. Using a text editor, open and edit this file. Extract the header file as follows. There will be some material at the beginning of the Illustrator file that is specific to the particular file and this can be removed. The header file then begins with the line

```
%%BeginProcSet:Adobe_Illustrator_1.1 0 0
```

and ends with the line:

```
%%EndProlog
```

Thus the whole header file is:

```
%%BeginProcSet:Adobe_Illustrator_1.1 0 0
% Copyright (C) 1987 Adobe Systems Incorporated.
% All Rights Reserved.
% Adobe Illustrator is a trademark of Adobe Systems Incorporated.
/Adobe_Illustrator_1.1 dup 100 dict def load begin
/Version 0 def
/Revision 0 def
% definition operators
/bdef {bind def} bind def
/ldef {load def} bdef
/xdef {exch def} bdef
% graphic state operators
/_K {3 index add neg dup 0 lt {pop 0} if 3 1 roll} bdef
```

(several more pages of definitions)

```
% font construction operators
/Z {findfont begin currentdict dup length dict begin
{1 index /FID ne {def} {pop pop} ifelse} forall /FontName exch def dup
```

```
length 0 ne
{/Encoding Encoding 256 array copy def 0 exch {dup type /nametype
eq
{Encoding 2 index 2 index put pop 1 add} {exch pop} ifelse} forall} if
pop
currentdict dup end end /FontName get exch definefont pop} bdef
end
%%EndProcSet
%%EndProlog
```

3.2 Cricket Draw Header File

To create a header file for Cricket Draw also is easy. Create a diagram in Cricket Draw consisting of one line or some other trivial diagram. While this file is open, choose "New" from the "File" menu and check the "PostScript" button in the dialog box, indicating you wish to create a PostScript file (as opposed to a "Draw" file). A new window will be created. Under the "Goodies" menu that has appeared with the activation of a PostScript window, select the entry "Create PostScript". This will display the PostScript code for the diagram, but not the header code. Now choose "Save As ..." from the "File" menu. Give this file a name and check "Complete" when indicating how to save the file. Exit Cricket Draw and use a text editor to examine the PostScript file. Like Illustrator, there will be code at the beginning and at the end of the file that is not a part of the header. This should be removed. The remaining material will be the header. The header file should look like:

```
/$cricket 210 dict def
$cricket begin
2 setlinecap
/d /def load def
/b {bind d}bind d
/l {load d}b
/e /exch l
/x {e d}b
/C /closepath l
/CP /currentpoint l
```

(several more pages of definitions)

```
/shadow {so}d
/charshadow {sc}d
/fountain {df}d
/offsetcalc {oc}d
/MakeOutlineFont {of}d
/MakeUnderlineFont {uf}d
/leftshow {ls}d
/rightshow {rs}d
/centershow {cs}d
/fullshow {ss}d
/coordinatefont {cf}d

%-----------------------------------
```

One last line should be added to the header file in order to make it complete. That is the line

```
end
```

and it is needed to close off and remove the dictionary that is being created for Cricket Draw.

3.3 MacDraft Header File

To create the header file for MacDraft may require Herculean strength. This header file is the standard Laser Prep file used by most applications that run on the Macintosh. I have included a list of suggestions on how to construct this file in Appendix B. It is definitely a non-trivial process.

We can now assume that we have three header files created. The next step is to load them into the PostScript interpreter's memory. There are four different techniques that can be used.

1. The header file can be downloaded directly into the permanent memory of the PostScript interpreter. The advantage to this is that the header file is accessible whenever you or anyone else needs it, until the power to the printer is turned off or the printer is reset. Another advantage is that you don't have to worry with how your DVI-to-PostScript processor would handle these header files. Still another advantage is that you will not be loading the graphics software's header (a.k.a. dictionary) on top of the DVI-to-PostScript package's header (a.k.a. dictionary). This last feature has already been referred to in Section 2.3. The disadvantage is that the header file takes up space in the printer's memory. On printers with a reasonable amount of memory (e.g., 2MB), this does not appear to be a problem.
   To use this process requires placing the following additional line at the beginning of the header file:

   ```
   serverdict begin 0 exitserver
   ```

   The assumption here is that "0" is the password for your printer (it will be unless some printer guru has gone in and changed it). It may appear that you have placed an additional dictionary ("serverdict") on the stack and therefore you should place an additional "**end**" at the end of your header file. *You do not need to do this.* However, you should send a **<ctrl-d>** character to the printer at the end of this file to indicate that the end of this special file has been reached. I have done this on my system by creating a special file with only one character in it, namely **<ctrl-d>** (the HEX notation for this is 04). I send the header file and then this very small file to the printer.
2. The header file can be downloaded via a Macintosh computer attached to the printer. This technique is really useful only in downloading the header file associated with MacDraft and is the standard Laser Prep file for the Macintosh. This is by far the cleanest and simplest way to download this very complex file. Simply attach the Macintosh computer to the printer and print a Word document or a MacWrite document or a MacDraw diagram or a MacDraft diagram to the PostScript printer. This causes the header file to be downloaded permanently to the printer.
3. The header file can be downloaded as a part of printing the document itself. This process is dependent on the particular DVI-to-PostScript processor being used. If you are printing diagrams on multiple pages of the document, then you want to make sure that the header code survives from one page to the next one. If my header file is in **header.ps**, then I can place the following **\special** command at the beginning of the text for my document:

   ```
   \special{ps:  plotfile header.ps global}
   ```

   and the header file will be correctly loaded. This particular command sequence is used by DVIPS and will most likely not work with other DVI-to-PostScript processors. The user should consult the manual for the particular DVI to PostScript processor used.[4]
4. If you are going to print only one diagram then you may want to include the header with the actual file itself. This technique is discussed later when the actual process for including a graphics file is considered.

We can now assume that the header file is loaded into the printer or else will be included with the diagram itself.

---

[4] There is a problem in using this technique (no. 3) with the version of DVIPS and Adobe Illustrator referenced in this paper. DVIPS's header re-defines a basic PostScript operator **concat**. When Adobe Illustrator's header is installed with the DVIPS dictionary already on the stack (as technique 3 would do), this operator, which is used by Adobe Illustrator, is not bound even though **bind** is used. The wrong definition of **concat** will be used. If the Adobe Illustrator header is loaded while the DVIPS dictionary is not on the stack, the **bind** operator will make sure that **concat** is defined properly and permanently. When the Adobe Illustrator dictionary is finally called during operations and placed above the DVIPS dictionary, the correct definition of **concat** will still be used.

4. Producing PostScript Graphics

This section concerns the production of the PostScript code for the diagram itself. This production process depends on the application used.

a. Adobe Illustrator – Using Adobe Illustrator, produce the diagram. Save the diagram. This entire file can be sent from the Macintosh computer to the IBM PC for processing.

b. Cricket Draw – Using Cricket Draw, produce the diagram. Follow the procedures discussed with respect to Cricket Draw in the section on producing headers (Section 3.2). Instead of saving the diagram using the "Complete" option the file should be saved using the "brief" option. This will produce only the code for the diagram and not the header file.

c. MacDraft – Using MacDraft, produce the diagram. Select the "Print" command. Position the cursor over the "OK" button but don't click it. At the same time, press down the "Apple" or the "Four-Leaf" (different ways to describe the same key) key and the "F" key — then click on the "OK" button. The computer may sound some warning signals because these keys are pressed but this can be ignored. The message to look for after this sequence of key strokes and mouse clicks are performed is a message box which says "Creating PostScript file". A file will be created called "PostScript*n*" where *n* is an integer between 0 and 9. This file should be re-named to some meaningful name associated with the diagram.

5. Transferring the Graphics

Any of the files produced on the Macintosh must be sent to the IBM PC. I use MacLink Plus which makes the job very easy. If you decide to use Kermit or some other file transfer protocol, then you will have to cope with the fact that on the Mac the `EOL` (`end-of-line`) is designated by just `CR` (`carriage return`) (0D in HEX) while in the IBM PC world it is designated by `CRLF`, (`carriage return-line feed`, 0D0A in HEX). There are fixes around this but a file transfer package such as MacLink Plus simplifies the work.

6. Analyzing and Converting the Graphics Files

The graphics files that were produced on the Macintosh need to be altered in some standard ways to prepare them for incorporation into documents that will be processed by TEX. Each of the three graphics packages requires different changes. Also complicating the conversion is the fact that a diagram may have been produced in either portrait or landscape orientation and the document that will contain it may either be in portrait or landscape orientation. To handle all this repetitous file conversion work, I use the AWK programming language. It is ideally suited for this work. It is powerful, it runs on an IBM PC, it does text manipulation, and it can handle files in a combined interactive/batch mode.[5]

The following generic steps must be taken for all PostScript files sent over from the Macintosh:

1. Appropriate scaling information must be placed in the file to compensate for the uniform magnification provided by the DVI-to-PostScript processor.
2. The "bottom center" of the diagram must be determined and the graphics file must be adjusted to contain this information. It may be desirable to produce a test print for the diagram to make sure that it is correctly centered.
3. All unneeded PostScript code in the file must be removed. Any needed additional PostScript code must be added.

The matter of scaling depends on the DVI to PostScript processor. DVIPS magnifies all the PostScript code based on the magnification level indicated in the original document (i.e., what `\magstep` was used). For example, if `\magnification = \magstep1` is used, then a scaling factor of 1/1.2 must be included in the graphics file. This is done by placing the sequence `.8333 .8333 scale` in the appropriate (to be shown later) place in the graphics file.

The diagram that has been constructed on the Macintosh will need to be translated so that the bottom center of the diagram is at the coordinates (0,0). This makes the placement of the diagram in the document very easy. Since the diagram will not normally be constructed so that the bottom center of the diagram has these PostScript coordinates, a simple statement must be added to the file

[5] Complete AWK programs to handle all the various permutations of graphics packages and document and display orientations can be obtained via electronic mail from the author.

to adjust the origin. The following approach is used to determine the correct coordinates to use in making this code addition:

1. The original PostScript file is manipulated by an AWK program that adds code to produce a pair of very long lines that form a right angle on the page. The vertex of the right angle is printed at approximately one inch up and in from the lower left corner of the page. This modified code is sent directly to the printer. The resulting output contains two things — the diagram and a pair of reference lines. By visual inspection and by using a ruler with a scale in points, the person preparing the material can measure the relative displacement from each of these reference lines to the bottom center of the diagram.
   It is unwise to use a corner of the paper as the reference point. The printer may have a systematic offset that it applies so that a corner of the paper does not correspond to (0,0). The orientation of the paper as it goes through the printer may be a little askew and this can also distort the measurement process.
2. A second AWK program is used to interactively ask the user for these two displacement coordinates and then to insert the correct PostScript **translate** command into the file to accomplish this needed translation.
3. A third AWK program can be used to print out this modified PostScript file to verify that the translation is correct.

Each graphics production package requires different changes to the contents of the file that it has produced:

a. Adobe Illustrator – The header file information must be removed. The correct translation of bottom center must be added. The proper scaling must be added to compensate for the magnification found in the document. A "save–restore" encapsulation of the PostScript file must be put in place.

b. Cricket Draw – A correction (`1 -1 scale 30 -762 translate`) must be inserted to correct for the fact that Cricket Draw does a flip of the diagram about the x-axis. A line of PostScript code must be added for each font used in the Cricket Draw diagram. The fonts used in the diagram will be listed in the header of the document. The AWK program can read these lines, remember the fonts, and then insert the needed lines at the appropriate place in the file. For example, if the two fonts used are Times-Roman and Helvetica, then the two lines to be added are:

```
(Times-Roman) coordinatefont
(Helvetica) coordinatefont
```

   The correct translation of bottom center must be added. The word **showpage** must be removed from the file so that printing will occur only after the entire page is done. The proper scaling must be added to compensate for the magnification found in the document. A "save–restore" encapsulation of the PostScript file must be put in place.

c. MacDraft – The correct translation of bottom center must be added. The proper scaling must be added to compensate for the magnification found in the document. The line `F T cp` , which occurs near the end of the file, must be removed or "%ed out". This line causes the page to print prematurely. A "save–restore" encapsulation of the PostScript file must be put in place.
   If the header code is to be included with the code for the diagram itself then the following alterations of the above additions must be used. The header material (minus the code **serverdict begin 0 exitserver**) should be inserted immediately after the line **/vmsave save def**. This will allow the header to be loaded before the code that calls on it is executed.

Appendix A to this paper contains the "before and after" codes for a simple diagram (the word "Hi") for each of the three graphics packages.

7. Incorporation of Graphics Files into TEX Files

The files containing PostScript code for the graphical material must now be placed in the output stream that will be sent to the printer. This is done in a two-stage process and is dependent on the particular DVI-to-PostScript processor used. The following information applies to DVIPS.

The DVIPS processor prepares an environment in which you can place your diagram. When the `\special` command is given, DVIPS restores most of the pure PostScript environment (except the magnification level). You make the current point the center of your drawing environment by giving the command `currentpoint translate`. You will notice that this sequence has been added to all the sample files shown in Appendix A.

My placement methods for diagrams uses the "bottom center" approach. I determine the bottom center of the space on the page in which the diagram should fit. I find this approach much easier than if I have to specify the lower left corner of the diagram. Supposedly all the PostScript files should tell you where the bottom left corner of the diagram's "bounding box" is, but most of the programs don't do this correctly. Generally this space for the figures or diagrams is created through the use of a `\midinsert`, `\topinsert` or `\pageinsert`. If the diagram is 3 inches high then I can use the following code:

```
. . . and this is shown in Figure 1.

\midinsert
\vskip 3truein
\centerline{\hbox to 0pt{\special{ps: plotfile fig1.ps}}}
\medskip
\centerline{Figure 1 -- My First Figure}
\endinsert

A new paragraph . . .
```

Sometimes the diagrams are larger than will fit on one whole page using `\pageinsert`. As was mentioned earlier, these diagrams require additional scaling factors to reduce the size of the diagram. These scaling factors can be combined with the scaling factors used to correct for the document magnification. For example, if `magstep1` is used and we want to reduce the diagram to 85% of its original size anyway then, when using DVIPS, the correct scaling would be `.7083 .7083 scale` since $.7083 = .85 \times (1/1.2)$.

8. Additional Comments and Guidelines

The incorporation of graphics into documents definitely proceeds in two phases if it is to be successful. The first phase involves constructing the header files and building the AWK programs that will manipulate the PostScript files. This phase is tedious, subject to numerous errors, and will need to be completed only once. The second phase involves the production and conversion of the actual diagrams. This part is relatively easy. After approximately 25 diagrams were prepared in MacDraft, I converted and centered all of them in approximately one hour.

Sometimes it may help to start learning how to insert PostScript code by incorporating some pure PostScript code. You have no header to worry about. For example, the following code will cause one diagonal line to be constructed:

```
/vmsave save def
currentpoint translate
newpath
0 0 moveto
72 72 rlineto
stroke
vmsave restore
```

Once you have gotten this to work, you will have more confidence to tackle the bigger job of PostScript output from applications. Since PostScript code is simply ASCII text, you can always examine a (small) output file and get some idea of how the PostScript code for graphics has been inserted into the other PostScript code.

I feel that it is important to use AWK programs in order to perform this work. This allows for

the almost complete automation of this process. Dealing with all the cases of landscape diagrams being placed in portrait documents or landscape diagrams being placed in landscape slides requires either that you remember all the subtle but necessary corrections to make or that you depend upon a program to make them.

This paper has not addressed all the desirable features one would like in incorporating PostScript files. There are numerous other features that can be considered:

1. The PostScript code can be instructed to read the magnification setting that is current in the printer and scale the diagrams accordingly. Currently I need to know beforehand the magnification level that will be used in the document that will receive the graphics files.
2. Another form of auto-scaling can be built into the process. The TEX macros would be used to pass the size of the space available to the graphics file. Specially written PostScript procedures would be invoked to combine this information with information describing the size of the diagram and these procedures would then set the scaling factors correctly.
3. The positioning of text generated by TEX and to be placed in a landscape orientation when the body of the text is in a portrait position could be implemented. This would be useful in placing captions for landscape diagrams correctly in portrait-oriented documents.

I have discussed how you do this work but now you need to actually do it. What tools are available to help you? Your greatest new need will be to try to understand what is going on inside the PostScript printer, that inscrutable "black box". There are several possibilities:

1. There is a package called "LaserTalk", produced by Emerald City Software. It is excellent for sending code to a PostScript printer and getting error messages back. It is available for the Macintosh and IBM environments.
2. There are various error handling routines that can be downloaded to your printer. Then, when you send PostScript code to your printer and there is an error, *supposedly* a sheet will be printed showing the offending command and the state of the stack at the time the command was issued. Unfortunately there are times when you commit an error that will not allow this error handler to be invoked.
3. Communicate between the computer and the printer via a serial port that is set up to handle communication *from* the printer as well as *to* the printer. ArborText includes such a communications utility with the DVIPS program, called SPR. The value of this is that the PostScript interpreter will most always be able to use this method to send you the first offending command. It will not be able to use this method to send you any more information but at least this is a start.

A final comment: getting started with the process of incorporating PostScript code can be a trying or challenging experience. But once the process is established, incorporating graphical material can be easy, benefical and very time-saving.

9. Acknowledgment

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

Bibliography

Adobe Systems Incorporated. *PostScript Language Reference Manual*. Reading, Mass.: Addison-Wesley, 1985.

Adobe Systems Incorporated. *PostScript Language Tutorial and Cookbook*. Reading, Mass.: Addison-Wesley, 1985.

Aho, Alfred V., Brian W. Kernighan, and Peter J. Weinberger. *The AWK Programming Language*. Reading, Mass.: Addison-Wesley, 1988.

Holzgang, David A. *Understanding PostScript Programming*. San Francisco, Calif.: SYBEX, 1987.

Appendix A: PostScript Code and AWK Programs

The following material is the PostScript code for a simple diagram (the word "Hi") before and after modification for incorporation into a document. The code is included for Adobe Illustrator, Cricket Draw and MacDraft. In some cases large amounts of text has been removed and replaced by three lines consisting of a single period. This is done simply for economy of space in this paper. Normally no code would be removed.

1a: Adobe Illustrator Code Before Modification

```
%!PS-Adobe-2.0 EPSF-1.2
.
.
.
%%DocumentFonts:Courier
%%+Helvetica
%%BoundingBox:111 -395 252 -320
%%TemplateBox:288 -360 288 -360
%%EndComments
%%BeginProcSet:Adobe_Illustrator_1.1 0 0
% Copyright (C) 1987 Adobe Systems Incorporated.
% All Rights Reserved.
% Adobe Illustrator is a trademark of Adobe Systems Incorporated.
/Adobe_Illustrator_1.1 dup 100 dict def load begin
/Version 0 def
/Revision 0 def
% definition operators
/bdef {bind def} bind def
.
.
.
% font construction operators
/Z {findfont begin currentdict dup length dict begin
{1 index /FID ne {def} {pop pop} ifelse} forall /FontName exch def dup
length 0 ne
{/Encoding Encoding 256 array copy def 0 exch {dup type /nametype
eq
{Encoding 2 index 2 index put pop 1 add} {exch pop} ifelse} forall} if
pop
currentdict dup end end /FontName get exch definefont pop} bdef
end
%%EndProcSet
%%EndProlog
%%BeginSetup
Adobe_Illustrator_1.1 begin
n
%%BeginEncoding:_Helvetica Helvetica
[
39/quotesingle 96/grave
128/Adieresis/Aring/Ccedilla/Eacute/Ntilde/Odieresis
.
.
.
%%EndEncoding
%%EndSetup
```

```
0 g
.
.
.
%%Note:
/_Helvetica 72 12 0 0 z
[1 0 0 1 148 -380]e
(Hi)t
T
%%Trailer
_E end
```

1b: Adobe Illustrator Code After Modification

```
/vmsave save def
%!PS-Adobe-2.0 EPSF-1.2
.
.
.
%%DocumentFonts:Courier
%%+Helvetica
%%BoundingBox:111 -395 252 -320
%%TemplateBox:288 -360 288 -360
%%EndComments
%%BeginProcSet:Adobe_Illustrator_1.1 0 0
% Copyright (C) 1987 Adobe Systems Incorporated.
% All Rights Reserved.
% Adobe Illustrator is a trademark of Adobe Systems Incorporated.
%%EndProcSet
%%EndProlog
%%BeginSetup
Adobe_Illustrator_1.1 begin
n
%%BeginEncoding:_Helvetica Helvetica
[
39/quotesingle 96/grave .
.
.
.
%%EndEncoding
%%EndSetup
currentpoint translate
.8333 .8333 scale
-320 288 translate
0 g
.
.
.
%%Note:
/_Helvetica 72 12 0 0 z
[1 0 0 1 148 -380]e
(Hi)t
T
%%Trailer
```

```
_E end
vmsave restore
```

2a: Cricket Draw Code Before Modification

```
%!PS-Adobe-2.0
%%Creator:Cricket Draw 1.1
%%Title:Untitled #1
%%CreationDate: 5/15/89   4:16 PM
%%DocumentFonts: Helvetica
%%BoundingBox: 0 0 612 792
%%Pages:(atend)
%%EndComments
/vmstate save def
/$cricket 210 dict def
$cricket begin
2 setlinecap
%%LoadPrep:Cricket Procedures

%----- Begin Main Program -----%
gsave  % Text Block
0.000 1  -1  0.000 126.500 114.000  fixcoordinates
/myshow /show load def
0 setgray
-63 -11 moveto
/|_____Helvetica findfont 72 scalefont setfont
{
(Hi) show
} leftshow
grestore
%------ End Main Program ------%
showpage end
vmstate restore
%%Trailer
%%Pages: 1
```

2b: Cricket Draw Code After Modification

```
%!PS-Adobe-2.0
%%Creator:Cricket Draw 1.1
%%Title:Untitled #1
%%CreationDate: 5/15/89   4:16 PM
%%DocumentFonts: Helvetica
%%BoundingBox: 0 0 612 792
%%Pages:(atend)
%%EndComments
/vmstate save def
currentpoint translate
.8333 .8333 scale
1 -1 scale -100 -130 translate
$cricket begin
2 setlinecap
%%LoadPrep:Cricket Procedures
```

```
(Helvetica) coordinatefont
%----- Begin Main Program -----%
gsave  % Text Block
0.000 1  -1  0.000 126.500 114.000  fixcoordinates
/myshow /show load def
0 setgray
-63 -11 moveto
/|______Helvetica findfont 72 scalefont setfont
{
(Hi) show
} leftshow
grestore
%------ End Main Program ------%
 end
vmstate restore
%%Trailer
%%Pages:1
```

3a: MacDraft Code Before Modification

```
%!PS-Adobe-2.0
%%Title: Untitled-1
%%Creator: MacDraft
%%CreationDate: Monday, May 15, 1989
%%Pages: (atend)
%%BoundingBox: ? ? ? ?
%%PageBoundingBox: 30 31 582 761
%%For: Tom Renfrow
%%IncludeProcSet: "(AppleDict md)" 66
%%EndComments
%%EndProlog
%%BeginDocumentSetup
md begin

T T -31 -30 761 582 100 72 72 1 F F F F T T psu
(Tom Renfrow; document: Untitled-1)jn
0 mf
od
%%EndDocumentSetup
%%Page: ? 1
op
.
.
.
2 F /|______Helvetica fnt
bn
-1.95671 0.(Hi)ashow
0 0 pen
254 147 gm
(nc 0 0 730 552 6 rc)kp
0 gr
104 146 194 227 0 rc
F T cp
%%Trailer
```

```
cd
end
%%Pages: 1 0
%%EOF
```

3b: MacDraft Code After Modification

```
%!PS-Adobe-2.0
%%Title: Untitled-1
%%Creator: MacDraft
%%CreationDate: Monday, May 15, 1989
%%Pages: (atend)
%%BoundingBox: ? ? ? ?
%%PageBoundingBox: 30 31 582 761
%%For: Tom Renfrow
%%IncludeProcSet: "(AppleDict md)" 66
%%EndComments
%%EndProlog
%%BeginDocumentSetup
/vmsave save def
currentpoint translate
.8333 .8333 scale
-250 -600 translate
md begin

T T -31 -30 761 582 100 72 72 1 F F F F T T psu
(Tom Renfrow; document: Untitled-1)jn
0 mf
od
%%EndDocumentSetup
%%Page: ? 1
op
.
.
.
2 F /|_____Helvetica fnt
bn
-1.95671 0.(Hi)ashow
0 0 pen
254 147 gm
(nc 0 0 730 552 6 rc)kp
0 gr
104 146 194 227 0 rc
%%F T cp
%%Trailer
cd
end
vmsave restore
%%Pages: 1 0
%%EOF
```

Two AWK programs are included as examples to show the utility that the AWK language can provide for manipulating PostScript files. The first program reads in a PostScript file produced by MacDraft and puts in the code necessary to generate two reference lines. This altered file can be sent to a PostScript printer and the resulting diagram (with reference lines included) can be used to determine the "bottom center" of the diagram. It is assumed that the MacDraft header has already been loaded into the printer.

```
#Add the vmsave header
#Add the code to generate the reference lines
/^%%BeginDocumentSetup/ {print $0
     print "/vmsave save def"
     print "gsave"
     print "newpath"
     print "72 720 moveto"
     print "360 0 rlineto"
     print "stroke"
     print "newpath"
     print "72 720 moveto"
     print "0 -360 rlineto"
     print "stroke"
     print "grestore"
     next}
#Add the restore command at the end
/^end$/ {print $0
     print "vmsave restore"
     next}
#Print the lines which don't contain an end of file marker
$0 !~ "\032" {print $0}
```

The second AWK program converts a Cricket Draw file from the original form transferred from the Macintosh to the form that can be included in the PostScript code for the document. Before the user runs this AWK program, it is assumed he has run another AWK program like the one above (but designed for Cricket Draw) which helps the user determine the "bottom center" of the diagram. The AWK program interactively queries the user for the two needed offset measurements. The results of the transformation are written to a file specified by the variable **outfile**. This variable is specified by the user in the command line that invokes the program. An interesting feature of this program is that it saves font name information from one part of the text file and then writes it out later.

```
#Ask user for offset information
BEGIN {print "How many points from the left line"
          print "is the center of the figure?"
          if (getline > 0) xoffset = $1
          print "How many points up from the bottom line is"
          print "the bottom of the figure?"
          if (getline > 0) yoffset = $1}
#Build the list of fonts needed in Cricket Draw
#Note: This command will be executed only after the
#/^%%DocumentFonts:/ command has been used.
/^%%LoadPrep:/ {print "%------------------------" > outfile
               print "%Encode PS Fonts to match Mac Fonts"  > outfile
               for (name in fonts)
                    print "(" name ") coordinatefont" > outfile
               print "%------------------------" > outfile
               next
               }
```

```
#Put in the needed offset information
/^\/\$cricket/ {
     print "currentpoint translate" > outfile
     print "/xoffset " xoffset " def" > outfile
     print "/yoffset " yoffset " def" > outfile
     print "/specialmag " specialmag " def" > outfile
     print "specialmag dup neg scale" > outfile
     print "xoffset 100 add neg 500 yoffset neg add neg translate" > outfile
     next}
#Turn off the routine that gathers font information
/^%%BoundingBox:/ {action = 0; print > outfile ; next}

#Turn on the routine that collects font information
/^%%DocumentFonts:/ {action=1; fonts[$2]=0; print > outfile ; next}

#Another line that will contain font information
action == 1 {fonts[$2] = 0; print > outfile ; next}

#Remove the showpage command so that the page is not
#printed prematurely
/^showpage/ {print "end" > outfile ; next}

#Print the other lines which don't contain an end of file marker
{if ($0 !~ "\032") print > outfile }
```

Appendix B: Suggestions on Creating MacDraft Header File

Producing a working version of the MacDraft Header File is a challenging task. Basically this involves producing a working version of the Laser Prep file on the Macintosh.

The first step is to capture the header file. This can be done by creating a file in Word or MacWrite that consists of some small amount of text — for example, the letter "a". Select the "Print" command. Just before you click on the "OK" button, depress the "Apple" or "Four leaf" key and simultaneously the "K" key and then click on the "OK" button. Ignore the strange sounds produced by the machine. A new file called "PostScript*n*" will be produced on the system and you can find it with the Macintosh Desk Accessory "Find File". Open this file with some text editor.

Now you have to pare this header file down some. There are several large code segments at the end of the header. These look like:

```
currentfile ok userdict/stretch known
not and{eexec}{flushfile}ifelse
373A767D4B7FD94FE5903B7014B1B8D3BED0
2632C855D56F458B118ACF3AF73FC4EF5E81F57490.
.
.
.
0000000000000000000000000000000000
0000000000000000000000000000000000
cleartomark
currentfile ok userdict/smooth4 known
not and{eexec}{flushfile}ifelse
.
.
.
F94E00EE41A71C59E5CAEED1EDBCF23D1DBA1
EE99B9BB356492923BD8B1BA83A87CEB0E07377A3
0000000000000000000000000000000000
0000000000000000000000000000000000
cleartomark
%%EndProcSet
```

These segments can be removed. The rest of the file can remain. If you plan to download the file permanently to the printer, then you need to add the line `serverdict begin 0 exitserver` at the beginning of the file. When you transfer this file from the Macintosh to the IBM PC and edit it with a text editor, you must may very sure that some of the long lines do not get broken up incorrectly. One particularly bad section is the part that lists the names of special characters (e.g., `/agrave /acircumflex /adieresis /atilde /aring /ccedilla /eacute /egrave`). Text editors may break these lines in the very middle of a word and this causes the PostScript interpreter to think that there is a new name and also a command which it does not understand. Text editors may also break -1 into - and 1 which will cause an error. Once you get all these edits made and no lines broken improperly, then you can send the header to the printer to see if you have any errors. At this time it is nice to use a utility like LaserTalk to analyze the code as it is being sent.

The suggestions don't work magic but they may help.