1
Postscript Tutorial

Reprinted from *The Geometry Toolbox* by Gerald Farin and Dianne Hansford, published by AK Peters, Wellesley, MA, 1998.

The figures in this book are created using the PostScript language. This is a mini-guide to PostScript with the purpose of giving you enough information so you can alter these images or create similar ones yourself.

PostScript is a page description language.[1] A PostScript program tells the output device (printer or previewer) how to format a printed page. Most laser printers today understand the PostScript language. A previewer, such as Ghostview or xpsview, allows you to view your document without printing – a great way to save paper.

Before proceeding, check if you have a previewer available. If not, there are free versions available. Ghostview for instance can be found at

`http://www.cs.wisc.edu/~ghost/index.html`.

In case you would like more in-depth information about PostScript, see [2, 1] or check out

`http://www.cs.indiana.edu/docproject/programming/postscript/`

[1] Its origins are in the late seventies, when it was developed at Evans & Sutherland, Xerox, and finally by Adobe Systems, who now owns it.

for more help.

1.1 A Warm-Up Example

Let's go through the PostScript file that generates Figure 1.1.

```
%!
newpath
    200 200 moveto
    300 200 lineto
    300 300 lineto
    200 300 lineto
    200 200 lineto
stroke

showpage
```

Figure 1.1 shows the result of this program: we have drawn a box on a standard size page. (In this figure, the outline of the page is shown for clarity; everything is reduced to fit here.)

The first line of the file is "%!". Nothing else belongs on this line, and there are no extra spaces on the line. This command tells the printer that what is to come is in the PostScript language.

The actual drawing is done with the `newpath` command. Move to the starting position with `moveto`. Record the path with `lineto`. Finally, indicate that the path should be drawn with `stroke`. These commands simulate the movement of a "virtual pen" in a fairly obvious way.

PostScript uses *prefix notation* for its commands. So if you want to move your "pen" to position $(100, 200)$, the command is `100 200 moveto`.

Finally, you need to invoke the `showpage` command to cause PostScript to actually print your figure. This is important!

Now for some variation:

```
%!
```

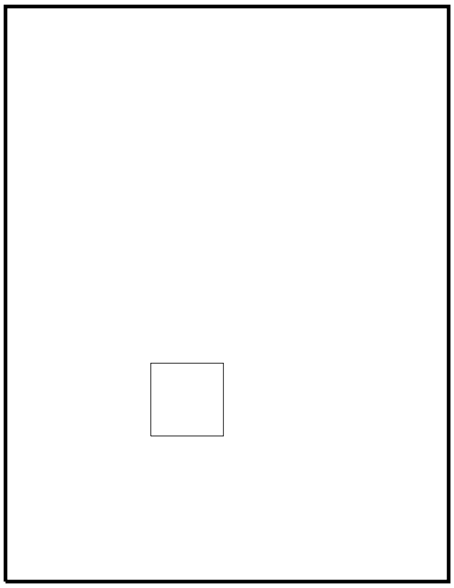

Figure 1.1.
A simple PostScript example.

```
% This is a comment line because it begins with a percent sign

% draw the box
newpath
    200 200 moveto
    300 200 lineto
    300 300 lineto
    200 300 lineto
    200 200 lineto
stroke

80 80 translate
0.5 setgray

newpath
    200 200 moveto
```

```
    300 200 lineto
    300 300 lineto
    200 300 lineto
    200 200 lineto
fill

showpage
```

This is illustrated in Figure 1.2. The `80 80 translate` command moves the origin of your current coordinate system by 80 units in both the $\mathbf{e}_1$- and $\mathbf{e}_2$-directions.

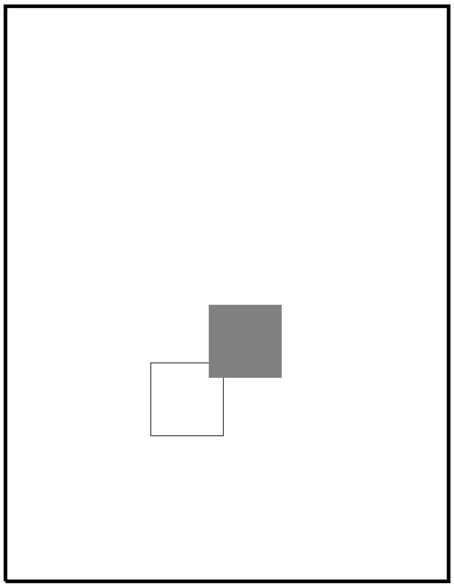

Figure 1.2.
Another simple PostScript example.

The `0.5 setgray` command causes the next item to be drawn in a gray shade (0 is black, 1 is white).

What follows is the same square as before; instead of `stroke`, however, there is a `fill`. This fills the square with the specified gray scale. Note how the second square is drawn on top of the first one!

Another handy basic geometry element is a circle. To create a circle, use the commands

```
250 250 50 0 360 arc stroke
```

This generates a circle centered at $(250, 250)$ with radius 50. The `0 360 arc` indicates we want the whole circle. To create a white filled circle change the command to

```
1.0 setgray
250 250 50 0 360 arc fill
```

Figure 1.3 illustrates all these additions.

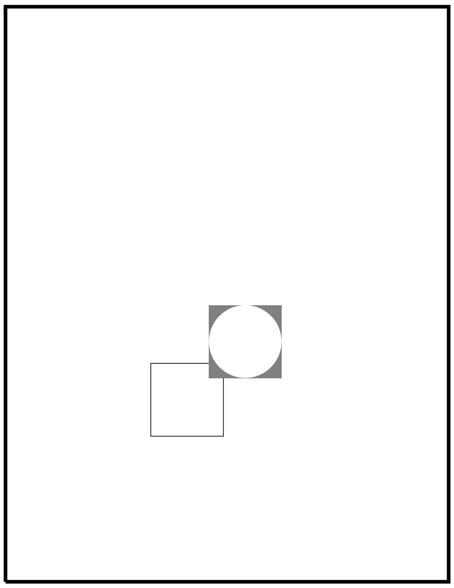

Figure 1.3.
Yet another simple PostScript example.

1.2 Overview

The PostScript language has primitive graphics operators: shapes such as line or arc, painting, text, bit mapped images, and coordinate transformation. Variables and calculations are also in its repertoire; it is a powerful tool. Nearly all the figures for this book are available as PostScript (text) files at the book's ftp-site. To illustrate the aspects of the file format, we'll return to some of these figures.

We use three basic scenarios for generating PostScript files.

1. A program (such as C) generates geometry, then opens a file and writes PostScript commands to plot the geometry. This type of file is typically filled simply with move, draw, and style (gray scale or line width) commands. Example file: `Bez_ex.ps` which is displayed in Section **??**.

2. The PostScript language is used to generate geometry. Using a text editor, the PostScript program is typed in. This program might even use control structures such as "for loops." Example file: `D_trans.ps` which is displayed in Section **??**.

3. A screen dump creating a bit mapped image is made part of a PostScript file. This is a "save' option in the Netscape Web browser, for example. Example file: `Flight.ps` which is displayed in Chapter **??**.

There are also figures which draw from more than one of these scenarios.

Since PostScript tells the printer how to format a page, it is necessary for the move and draw commands to indicate locations on the piece of paper. For historical reasons, printers use a coordinate system based on *points*, abreviated as pt.

$$1 \text{ inch} \equiv 72\text{pt}.$$

PostScript follows suit. This means that an $8\frac{1}{2} \times 11$ inch page has the following extents,

$$\text{lower left}: \ (0, 0) \qquad \text{upper right}: \ (612, 792).$$

Whichever scenario from above is followed, it is always necessary to be sure that the PostScript commands are drawing in the appropriate area. Keep in mind you probably want a margin. Chapter **??** covers the basics of setting up the dimensions

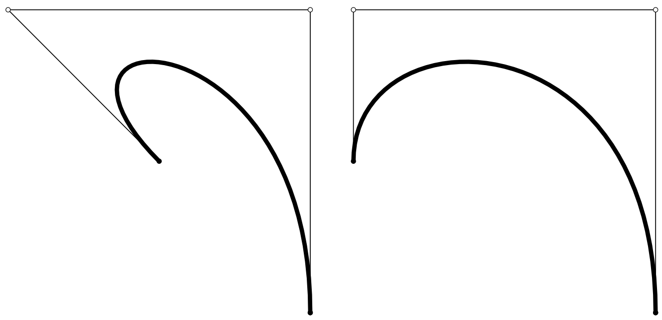

Figure 1.4.
Bez_ex.ps

of the 'target box' (on the paper) and that of the 'source box' to avoid unwanted distortions. Getting the geometry in the source box to the target box is simply an application of affine maps!

1.3 Affine Maps

In our simple examples above, we created the geometry with respect to the paper coordinates. Sometimes this is inconvenient, so let's discuss the options.

If you create your geometry in a, say C, program, then it is an easy task to apply the appropriate affine map to put it in paper coordinates. However, if this is not the case, the PostScript language has the affine map tools builtin.[2]

[2]There are many techniques for displaying 3D geometry in 2D; some "realistic" methods are not in the realms of affine geometry. A graphics text should be consulted to determine what is best.

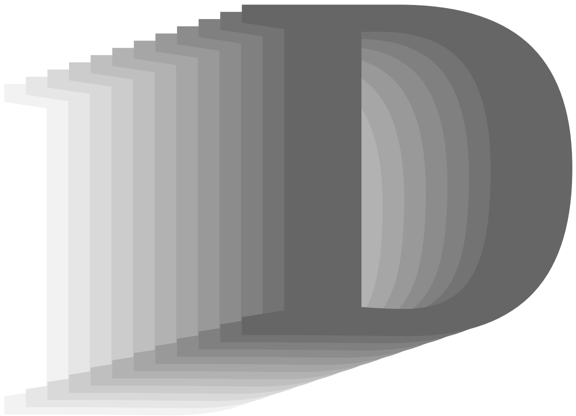

Figure 1.5.
`D_trans.ps`

There is a matrix (which describes an affine map, i.e., a linear map and a translation) in Postscript which assumes the task of taking "user coordinates" to "device coordinates". In accordance to the terminology of this book, this is our source box to target box transformation. This matrix is called the *current transformation matrix*, or CTM.

In other words, the coordinates in your PostScript file are always multiplied by the CTM. If you choose not to change the CTM, then your coordinates must live within the page coordinates, or "device coordinates". Here we give a few details on how to alter the CTM.

There are two "levels" of changing the CTM. The simplest, and most basic ones use the `scale`, `rotate`, and `translate` commands. As we know from Chapter **??**, these are essentially the most basic affine operations.

Unless you have a complete understanding of the CTM, it is

Figure 1.6.
`Flight.ps`

probably a good idea to use each of these commands only once in a PostScript file. It can get confusing! (See Section 1.6.)

A scale command such as

```
72 72 scale
```

automatically changes one "unit" from being a *point* to being an inch. A translate command such as

```
2 2 translate
```

will translate the origin to coordinates $(2, 2)$. If this `translate` was preceded by the `scale` command, the effect is different. Try both options for yourself!

A rotate command such as

```
45 rotate
```

will cause a rotation of the coordinate system by 45 degrees in a counterclockwise direction.

These commands are used in the ftp-site files, `D_scale.ps`, `D_trans.ps`, and `D_rot.ps`.

Instead of the commands above, a more general manipulation of the CTM is available, see Section 1.6.

1.4 Variables

The figures in this book use variables quite often. This is a powerful tool that allows a piece of geometry to be defined once, and then affine maps can be applied to it to change its appearance on the page.

Let's use an example to illustrate. Take the file for Figure 1.2. We can rewrite this as

```
%!

% define the box
/box {
        200 200 moveto
        300 200 lineto
        300 300 lineto
        200 300 lineto
        200 200 lineto
} def

newpath
box
stroke

80 80 translate
0.5 setgray

newpath
box
fill

showpage
```

The figure does not change. The box that was repeated is defined only once now. Notice the `/box {...} def` structure. This defines the variable `box`. It is then used without the forward slash.

1.5 Loops

The ability to create "for loops" is another very powerful tool. If you are not familiar with the prefix this might look odd. Let's look at the file `D_trans.ps`, which is displayed in Figure ??.

```
%!
%%BoundingBox: 90 100 375 300
/Times-Bold findfont
70 scalefont setfont
/printD {
  0 0 moveto
  (D) show
} def

100 100 translate

2.5 2.5  scale
.95 -.05 0 {setgray printD 3 1 translate } for

showpage
```

Before getting to the loop, let's look at a few other new things in the file. The `BoundingBox` command is not used by PostScript; this is to help in the placement of the figure in a LaTeX file. The `Times-Bold findfont` command allows us to access a particular set of fonts – we want to draw the letter D.

Now for the "for loop." The command above

```
.95 -0.5 0 {...} for
```

tells PostScript to start with the value 0.95 and decrement by 0.5 until it reaches 0. At each step it will execute the commands within the parenthesis. This allows the D to be printed 19 times in different gray scales and translated each time.

1.6 CTM

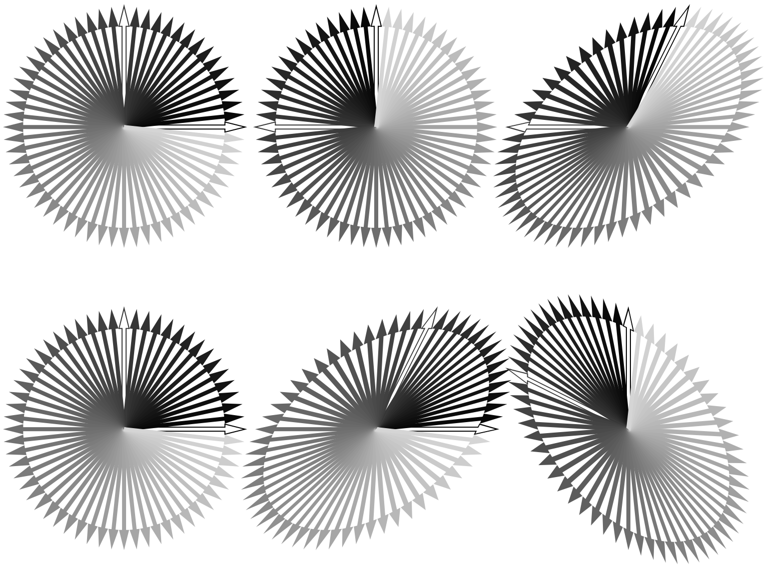

Figure 1.7.
`Nocomm.ps`

One of the most complicated figure files is `Nocomm.ps`, which is illustrated in Figure **??**. Here is its listing.

```
%!
%%BoundingBox: 50 50  350 250
% show that matrix multiply does not commute

% define a set of unit vectors with varying gray scale.
% if a vector points at 0 or 90 degrees it gets painted
% in a special way.
/vectors
```

```
{       /inc 5 360 div def
        /deg 0 def
        /gray 0 def
        0 6 354
        {
                /gray gray inc add def
                gray setgray
                newpath
                0 0 moveto
                0 2 lineto
                100 2 lineto
                100 5 lineto
                120 0 lineto
                100 -5 lineto
                100 -2 lineto
                0 -2 lineto
                0 0 lineto
                deg 90 eq  deg 0 eq or
                {stroke}{fill}ifelse
                6 rotate
                /deg 6 deg add def
        }for
}def


%definition of the two matrices:
% rotate 90 deg
/mat1[0 1 -1 0   0 0] def
% shear in x-dir
/mat2[1  0  0.5  1 0 0] def

% inverse matrices defined - computed below
/mat1inv[0 0 0 0 0 0]def
/mat2inv[0 0 0 0 0 0]def

% create the inverse of the shear and rotation
mat1 mat1inv invertmatrix
```

```
mat2 mat2inv invertmatrix

% move to place on page and scale to fit
100 200 translate
0.35 0.35 scale

% plot the vectors without a transformation
        vectors

% plot vectors with rotation
        250 0 translate
        mat1 concat
        vectors

% undo the rotation
        mat1inv concat

% plot vectors with rotation then shear
        250 0 translate
        mat2 concat
        mat1 concat
        vectors

% undo shear then rotation -- order important!
        mat1inv concat
        mat2inv concat

% move to a new row and plot vector without a transf.
        -500 -300 translate
        vectors

% plot vectors with shear
        250 0 translate
        mat2 concat
        vectors

% undo shear
```

```
        mat2inv concat

% plot vectors with shear then rotation
        250 0 translate
        mat1 concat
        mat2 concat
        vectors

% undo rotation then shear (not needed but to illustrate)
        mat2inv concat
        mat1inv concat

showpage
```

This one is commented a bit more than the original! Hopefully the comments help you get a hang of the language. What makes this figure more complicated than others is that it manipulates the CTM directly rather than only using the `translate`, `scale`, and `rotate` commands.

Here is the tricky part. PostScript uses a transformation matrix for "left multiply." This book works on the basis of "right multiply". [3] Specifically, this book applies a transformation to a point $\mathbf{p}$ as

$$\mathbf{p}' = \begin{bmatrix} a & c \\ b & d \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix},$$

whereas Postscript would apply the transformation as

$$\mathbf{p}' = \begin{bmatrix} p_1 & p_2 \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix}.$$

So the Postscript CTM is the transpose of the matrix in this book.

[3] The computer graphics community typically uses left multiply whereas mathematicians typically use right multiply.

A matrix definition in Postscript has the translation vector included. The matrix has the syntax

$$[a, b, c, d, t_x, t_y].$$

From the listing above, let's look at the part where we want to rotate, then shear.

```
% plot vectors with rotation then shear
        250 0 translate
        mat2 concat
        mat1 concat
        vectors
```

From the notation in this book, we would write

$$\mathbf{p}' = SR\mathbf{p},$$

where S is a shear matrix and R is a rotation matrix. For left multiply, this becomes

$$\mathbf{p}' = \mathbf{p}^T RS.$$

So our transformation matrix T is $T = SR$, but the Postscript matrix is $T = RS$! This is reflected in the Postscript code segment by concatenating (multiplying) the S matrix `mat2` to the CTM first, then concatenating the R matrix.

This same idea is reflected in the procedure to "undo" the shear and rotation from the CTM. To restore the CTM we apply,

$$S^{-1}R^{-1}RS(CTM),$$

which means that the rotation gets taken off first, then the shear. This is the opposite order they were put on.

That should be enough detail to put you well on your way to becoming a PostScript expert!

Bibliography

[1] Adobe Systems Inc. *PostScript Language Reference Manual*. Reading, MA: Addison-Wesley Publishing Company, Inc., 1985.

[2] Adobe Systems Inc. *PostScript Language Tutorial and Cookbook*. Reading, MA: Addison-Wesley Publishing Company, Inc., 1985.