One-point Perspective Transformation
From 3-Dimensional Space to 2-Dimensional Space
Implemented in the PostScript Laser Printer
Programming Language

An Honors Paper and Creative Project (CS 499/ID 499)

by

Joseph E. Coulson, Jr.

Director
Dr. J. M. McGrew

(advisor’s signature)

Ball State University
Muncie, Indiana
February 29, 1988

Date of Graduation:
February 26, 1988
Introduction

The PostScript page description language is a very powerful programming language which allows one to generate complex graphical outputs in addition to near typeset quality textual documents. PostScript is fast becoming the standard used in laser printer technology.

Chapter 1 of this paper is a brief introduction to an important concept in PostScript programming: the stack and postfix notation. Chapter 2 is a short summary of the PostScript commands to be used in the PostScript code discussed later. It is assumed that the reader has a general knowledge of programming practices and principals. Chapter 3 discusses the technique of mapping a three dimensional coordinate to a two dimensional coordinate with a preservation of perspective. PostScript code is given to accomplish this, and it is explained in detail.
Understanding Postfix notation

The PostScript programming language is one that is expressed in postfix notation, meaning that the operands are listed before the operator. For example:

100 80 moveto

This PostScript code would result in the current point coordinate being defined as the absolute position (x=100, y=80). The operands are 100 and 80; the operator is moveto. Consider the following PostScript code:

100 80 moveto
200 200 lineto

The result will be that a line is drawn on the output device from position (100,80) to position (200,200).

The interpreter, which reads the PostScript code that has been written and displays the results, uses a data structure known as the "stack." The stack is a data structure which lends itself well to implementing the postfix notation which is used in giving PostScript commands like those in the example above. A stack is also known as a LIFO (Last In First Out) structure. A good example of how a stack works can be seen in the stack of food trays found in a restaurant. The last tray that is put on the stack of trays will be the first tray that is taken out by the next customer, and so it is with the PostScript interpreter. Using the PostScript code example above, a 100 is the first item
pushed on the stack; next the 80 is pushed onto the stack; and finally the interpreter comes to the operator moveto. Operators are not pushed onto the stack, but rather they cause some action to take place. Only operands and results of operations (e.g. the result of the add operator to be mentioned later) are pushed onto the stack.

The way that the PostScript stack works is best understood in a graphical representation. Here is a step by step view of what happens when the following PostScript code is interpreted:

100
100 80

moveto is performed, operands taken off of stack
200

lineto is performed, operands taken off of stack
Chapter 2

Summary of PostScript Commands Used

add
example: num1 num2 add sum
Returns the sum of num1 and num2.

closepath
dexample: closepath
Closes the current subpath by appending a straight line segment connecting the current point to the subpath's starting point.

cos
dexample: angle cos real
Returns the cosine of an angle (degrees).

currentpoint
dexample: currentpoint x y
Returns the current point (i.e. the trailing endpoint of the current path).

def
dexample: key value def
Associates key with value. If key is already present, def simply replaces its value. Otherwise, def creates a new entry for key and stores value in it.

div
example: num1 num2 div quotient
Divides num1 by num2, producing a result that is always a real.

dup
example: anything dup anything anything
Duplicates the top element on the operand stack.

exch
dexample: any1 any2 exch any2 any1
Exchanges the top two elements on the stack.

grestore
dexample: grestore
Restores the graphics state in effect at the time of the matching gsave.

gsave
dexample: gsave
All elements of the graphics state are saved, including the CTM, current path, and clip path.

lineto
dexample: x y lineto
Appends a straight line segment to the current path.

moveto
dexample: x y moveto
Sets the current point in the graphics state.
mul  example: num1 num2 mul product
Returns the product of num1 and num2.

newpath  example: newpath
Initializes the current path to be empty, causing
the current point to be undefined.

repeat  example: int procedure repeat
Executes a procedure int number of times.

scale  example: Sx Sy scale
Multiplies the scale of the x-axis by Sx and the
scale of the y-axis by Sy.

setgray  example: num setgray
Sets the current color to a gray shade.

setlinewidth  example: num setlinewidth
Sets the current line width parameter.

showpage  example: showpage
Transmits the current page to the output device.

sin  example: angle sin real
Returns the sine of angle (degrees) as a real.

stroke  example: stroke
Paints a line following the current path and
using the current color.

sub  example: num1 num2 sub difference
Returns the result of subtracting num2 from num1.

translate  example: Tx Ty translate
Moves the origin of the coordinate system by Tx
units in the x-axis direction and Ty units in the
y-axis direction.
Chapter 3
Perspective Projections

The nature of a perspective transformation from three dimensions to two dimensions is to project a point \( P \) at coordinates \((x, y, z)\) to position \((x_p, y_p, 0)\) on a projection plane. Diagram 3.1 is a visual representation of what is to take place.

To obtain a perspective projection of a three dimensional object, points are projected along projection lines that meet at the center of projection. The center of projection is chosen to be along the y-axis to make calculations simpler.

The transformation equations can be obtained for perspective projection from the parametric equations describing the projection line from point \( P \) to the center of projection. The parametric form for the projection line is:

\[
\begin{align*}
x' &= x - x' u \\
y' &= y - (y + d) u \\
z' &= z - z' u
\end{align*}
\]

Parameter \( u \) takes values from 0 to 1, and coordinates \((x', y', z')\) represent any position along the projection line. When \( u = 0 \),
the equations yield point \( P \) at coordinates \((x, y, z)\). At the other end of the line \( u=1 \), and the result is the coordinate for the center of projection, \((0, -d, 0)\). To obtain the coordinate on the projection plane, it is needed to set \( z' = 0 \) and solve for parameter \( u \):

\[
\frac{y}{u} = \frac{y}{y + d}
\]

This value for \( u \) produces the intersection of the projection line with the projection plane at \((x_p, 0, z_p)\). Substituting for \( u \) results in the following perspective transformation equations:

\[
\begin{align*}
x_p &= x \times \frac{d}{y + d} \\
y_p &= 0 \\
z_p &= z \times \frac{d}{y + d}
\end{align*}
\]

Recall that the above two equations require that the center of projection be along the \( y \)-axis. Therefore, a minor adjustment must be made to allow the eye of the observer of a three dimensional object to view from other positions not directly on the \( y \)-axis. To do this, it is necessary to subtract the \( x_0 \) and \( z_0 \) values (distance left or right and up or down from the center of projection, respectively) from the three dimensional coordinates \( x \) and \( z \). This has the effect of shifting the object on the projection plane to reflect the fact that the observer's eye is not at the center of projection. The modified equations are:

\[
\begin{align*}
x_p &= (x - x_0) \times \frac{d}{y + d}
\end{align*}
\]
yp = 0
zp = (z - zo) \* \frac{d}{(y + d)}

The PostScript code on the following page is a complete program which will perform a perspective transformation of an object from three dimensional coordinates to two dimensional coordinates. A detailed description follows it.

Following the PostScript code listing are several runs of the program with various values reflecting the position of the observer (movements from left to right and also up).
% Draws two-dimensional perspective illustrations

% perspective transform - converts x,y,z into X and Y on stack:
/px {{/zz exch def /yy exch def /xx exch def yo dup yy add div
   xx xo sub mul yo dup yy add div zz zo sub mul} def

% save and restore previous x,y,z position:
/psave {{/zh zz def /yh yy def /xh xx def} def
/prestore {{/zz zh def /yy yh def /xx xh def} def

% pm perspective absolute move x,y,z and hold:
/pm {px psave moveto} def

% pd perspective absolute draw x,y,z and hold:
/pd {px psave lineto} def

% prm perspective relative x,y,z move with objrot rotation:
/prm {{zi exch def /yi exch def /xi exch def objrot cos xi mul
   objrot sin yi mul sub xh add objrot sin xi mul objrot cos yi mul
   add yh add z add px moveto psave} def

% prd perspective relative x,y,z draw with objrot rotation:
/prd {{zi exch def /yi exch def /xi exch def objrot cos xi mul
   objrot sin yi mul sub xh add objrot sin xi mul objrot cos yi mul
   add yh add z add px lineto psave} def

% default distances from observer to picture plane:
/xo -20 def % left and right
/y0 80 def % into picture; avoid small values
/zo -40 def % up and down; avoid large values
/objrot 30 def % relative xy object rotation

% ---- demo stuff follows ----
/inch {72 mul} def
2 inch 10 inch moveto
/Times-Roman findfont 18 scalefont setfont
(Perspective transformation from 3-D to 2-D) show
2 inch 9.5 inch moveto
(NO = -20, yo = 80, zo = -40, objrot = 30 degrees) show
300 400 translate % this is the center horizon
5 dup scale

% perspective grid
0 setlinewidth /startat 0 def 0 setlinewidth 2 setlinecap
19 {-30 startat 0 px moveto 30 startat 0 px lineto stroke
/startat startat 10 add def} repeat /startat -30 def
7 {startat 0 0 px moveto 0 100000 0 px lineto stroke
/startat startat 10 add def} repeat

% a non-putrid gray
106 35 {dup mul exch dup mul mul add 1.0 exch sub} setscreen
Perspective transformation from 3-D to 2-D

\[ x_0 = 0, \; y_0 = 80, \; z_0 = 40, \; \text{objrot} = 30 \text{ degrees} \]
Perspective transformation from 3-D to 2-D

$x_0 = 20$, $y_0 = 80$, $z_0 = 40$, $\text{objrot} = 30$ degrees
Perspective transformation from 3-D to 2-D

\[ \begin{align*}
x_0 &= -20, \\
y_0 &= 80, \\
z_0 &= 40, \\
objrot &= 30 \text{ degrees}
\end{align*} \]
Perspective transformation from 3-D to 2-D

\[ x_0 = -20, \; y_0 = 80, \; z_0 = 70, \; \text{objrot} = 30 \text{ degrees} \]
Definition of *px* procedure

The procedure *px* does the actual perspective transform. There should be three numeric values on the stack before this procedure is called - each value representing the x, y, z values, respectively, of a point in a three dimensional object.

As an example, the statement `/zz exch def` has the following effect on the stack:

```
50 20 100 /zz
50 20 /zz 100
50 20
```

The label for the variable *zz*, which is `/zz`, must occur on the stack before the value that is to be assigned to the variable *zz* - this is the purpose of the `exch` operator. The `def` operator then assigns the value to the variable.

Here is the stack execution representation of the procedure *px* given that arbitrary values x, y, and z are on the stack. Note that in reality actual numbers should appear on the stack rather than the symbols that are used here for illustrative purposes only:

```
x y z
x y z /zz
x y /zz z
x y
x y /yy
x /yy y
x
x /xx
/xx x

yo
yo yo
yo yo yy
yo (yo + yy)
(yo / (yo + yy))
(yo / (yo + yy)) xx
```
Definition of \textbf{psave} and \textbf{prestore} procedures

The procedure \textbf{psave} simply stores the values of the variables \textit{zz}, \textit{yy}, and \textit{xx} into the variables \textit{zh}, \textit{yh}, and \textit{xh} respectively. The procedure \textbf{prestore} has the effect of reversing the \textbf{psave} procedure.

Definition of \textbf{pm} procedure

The procedure \textbf{pm} is used to do a perspective absolute move. Before invoking this procedure the values for \textit{x}, \textit{y}, and \textit{z} should be on the stack. The \textbf{px} procedure is invoked and transforms the three dimensional coordinate into two dimensional coordinate values \textit{X} and \textit{Y}. The three dimensional coordinate is saved for future reference and then a move is made to the two dimensional coordinate (\textit{X}, \textit{Y}).

Definition of \textbf{pd} procedure

The procedure \textbf{pd} is used to do a perspective absolute draw. Before invoking this procedure the values for \textit{x}, \textit{y}, and \textit{z} should
be on the stack. The px procedure is invoked and transforms the three dimensional coordinate into two dimensional coordinate values X and Y. The three dimensional coordinate is saved for future reference and then a line is constructed to the two dimensional coordinate (X, Y).

**Definition of prm procedure**

The procedure prm is used to do a perspective relative move. Here is a stack execution representation of the procedure prm given that arbitrary values x, y, and z are on the stack. Note that in reality actual numbers should appear on the stack rather than the symbols that are used here for illustrative purposes only.

```
x  y  z
x  y  z /zi
x  y /zi z
x  y
x  y /yi
x  /yi y
x
x  /xi
/xi x

objrot
(cos objrot)
(cos objrot) x
((cos objrot) * xi)
((cos objrot) * xi) objrot
((cos objrot) * xi) (sin objrot)
((cos objrot) * xi) (sin objrot) yi
((cos objrot) * xi) ((sin objrot) * yi)
(((cos objrot) * xi) - ((sin objrot) * yi))
(((cos objrot) * xi) - ((sin objrot) * yi)) xh
(((cos objrot) * xi) - ((sin objrot) * yi)) + xh
# objrot
# (sin objrot)
# (sin objrot) xi
# ((sin objrot) * xi)
```
At this point the px procedure is invoked to transform the three dimensional coordinate on the stack to a two dimensional coordinate. A move is then made to that point and the original three dimensional coordinate is stored.

Rotation with respect to an arbitrary rotation point is shown in the figure below. The transformation equations for the rotated coordinates can be obtained from the trigonometric relationships in this figure as:

\[
\begin{align*}
x' &= x_r + (x - x_r)\cos\theta - (z - z_r)\sin\theta \\
z' &= z_r + (z - z_r)\cos\theta + (x - x_r)\sin\theta
\end{align*}
\]

To carry these equations over to three dimensions, set \( y' = y \). This will restrict the rotation about the \( y \)-axis and allows for simpler calculations. By modifying the above equations, one could cause rotation to occur about any of the axes.
**Definition of prd procedure**

The prd procedure is used to do a perspective relative draw. It acts in exactly the same manner as the prm procedure with the exception that the result is to construct a line to the two dimensional point.

**Definition of interactive variables**

The variables that affect the viewed position of the three dimensional object are defined here. The value for xo determines whether the observer is to the left or right of the center of projection; the value for yo determines whether the observer is far away or close up; the value for zo determines how far up or down the observer is; the value for objrot determines the degree of rotation of the three dimensional object with respect to the y-axis.

**PostScript program code**

After the PostScript procedures and variables have been defined, the PostScript program code begins. The first task is to translate the origin of the coordinate system to the center of the 8.5 inch by 11 inch output page. Next the x-axis and y-axis are scaled (magnified) by a factor of five (note: resulting units are 5/72 inch). This is a matter of preference and most importantly shows that any scale could be used to describe the three dimensional object.
Next a perspective grid is drawn as a special effect to show the relative position of the three dimensional object to be drawn. The grid also serves to show the vanishing point with respect to the center of projection.

The code following then sets a screen to be used in determining half-tone grayscale values for the pseudo-shading of the three dimensional object to be drawn. The default value for the "screen" could just easily have been used and would not have needed to be specified.

The rest of the code describes and draws the three dimensional object on the two dimensional output page with perspective. This code could easily be replaced with any description of a 3-D object (for example, the famous teapot). It is important to note that the object should be described by a move to an absolute position, and then all subsequent draws or moves should be relative.
References


