

CS557

CPLOT

January 5, 2000

One of the best ways to develop a working knowledge of curves and surfaces is to actually implement them in a computer program. Since information on how to write the basic graphics routines for such a program is incidental to our subject, you will be provided with the skeleton of a computer graphics program, called *CPLOT* (for “Curve Plot”), which performs simple tasks such as drawing a line, a circle, or a Bézier curve. CPLOT can easily be enhanced. Much of your learning will come from implementing those enhancements.

CPLOT is written in C and can be obtained as follows. The computer prompts are written in **bold face** type and your keyboard input is printed in **typewriter** font. If you are logged in to a CS UNIX workstation, create a new directory for CPLOT, and **cd** into that directory. Then:

```
cp tom/557/cplot/* .
```

You will now find the source files and some example input files in your directory. Optionally, you can download the files from your web browser at <http://students.cs.byu.edu/tom/557/cplot>.

To create the executable for CPLOT, type

```
make
```

CPLOT is executed using the command

```
cplot data_file
```

The graphics will automatically appear on your screen, assuming you are using a graphics terminal with Tektronix emulation. You can create a graphics window on your UNIX workstation by typing

```
xterm -t &
```

To instead produce a PostScript file of the graphics, type

```
cplot data_file -p output_file
```

To obtain a printed hardcopy of this figure, type

```
lp output_file
```

or to view it using GhostScript, type

```
gs output_file
```

Many of your projects will involve making changes to CPLOT. After you make a change to the source code, you can recompile it by once again executing the makefile. Simply type:

```
make
```

This skeleton version of CPLOT has several commands implemented. As a rule, only the first four characters of each command are significant. Most commands are followed by some data values. The commands for drawing circles, line segments, and Bézier curves are:

CIRC

x,y,r Plot a circle of radius r centered at (x, y)

LINE

n

x_1, y_1

x_2, y_2

...

x_n, y_n Plot the $n - 1$ line segments $(x_i, y_i) - (x_{i+1}, y_{i+1})$.

STORe

n,ncurve

x_0, y_0, w_0

x_1, y_1, w_1

x_2, y_2, w_2

...

x_n, y_n, w_n

Store the control points of a degree n rational Bézier curve and assign it a curve number of **ncurve** for future reference. The reason that curves are stored rather than plotted immediately (such as is done for circles and line segments) is to allow us to more easily perform such operations as subdivision and degree elevation

(which you will later learn about and implement). The w values are *weights*, whose function we will study when we discuss rational curves. For a conventional *polynomial* Bézier curve, simply set all w values to 1.

CPLOT

ncurve

Plot curve # $ncurve$ as a Bézier curve.

The file `eg1.dat` contains the following drawing commands:

```
stor
3 1
20 26 1
30 15 1
40 15 1
50 26 1
cplo
1
circ
35 35 30
line
4
18 30 10 30 10 40 18 40
line
5
22 30 22 40 28 40 28 36 22 36
line
3
38 30 32 30 32 40
line
9
40 32 40 38 42 40 46 40 48 38 48 32 46 30 42 30 40 32
line
4
54 30 54 40 50 40 58 40
```

These commands define one cubic Bézier curve, one circle, and five sets of line segments which sketch the letters “C-P-L-O-T” as shown in Figure 1. Refer to any such collection of circles, line segments, curves, etc. as a *scene*. The Cartesian coordinates used in the CPLOT commands are defined in the *scene coordinate system*.

Two further CPLOT commands define the *window* and *viewport*:

VIEW

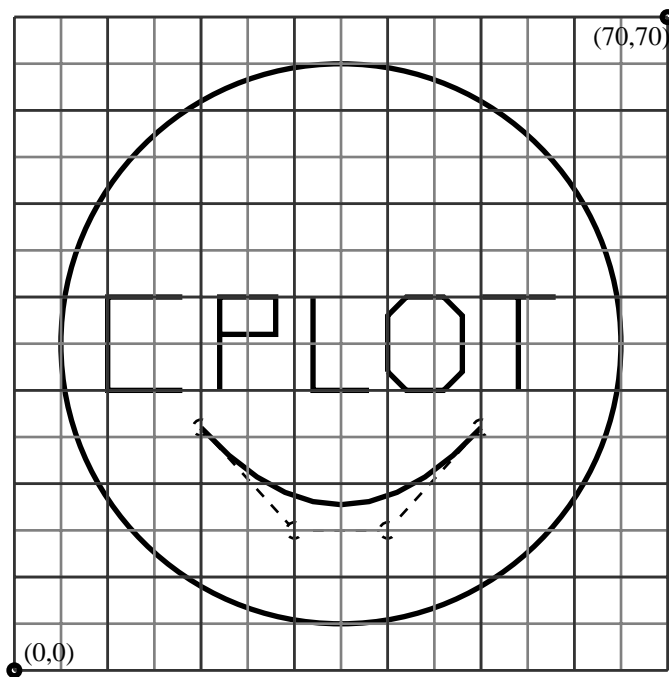


Figure 1: Scene from file `eg1.dat`

$x_{min}, x_{max}, y_{min}, y_{max}$

WIND

$x_{min}, x_{max}, y_{min}, y_{max}$

BORD Plot the viewport rectangle.

A window can be thought of as a rectangular hole in a large piece of cardboard which can be positioned over a region of the scene. Only that portion of the scene within the window is visible. The window is defined by specifying the values $x_{min}, x_{max}, y_{min}, y_{max}$ in scene coordinates.

A viewport is a second rectangle that specifies where the portion of the scene contained in the window should be plotted on the page. The page is taken to be a square sheet of paper, with page coordinates running from (0,0) in the lower left corner to (1,1) in the upper right corner. The viewport is defined in terms of its values $x_{min}, x_{max}, y_{min}, y_{max}$ in page coordinates.

The value $\frac{y_{max}-y_{min}}{x_{max}-x_{min}}$ is called the *aspect ratio*. If the window and viewport have the same aspect ratio, the resulting picture will not be distorted.

The window and viewport commands must appear prior to any scene commands (circ, line, cpl), since scene commands are executed immediately. The file `eg1.dat` uses the following window and viewport commands:

```
wind
0 70 0 70    /* The entire scene */
view
0 .5 .5 1    /* Upper left fourth of the page */
bord        /* Draw a rectangle around the viewport */

(the scene commands appear here)

wind
5 65 20 50   /* A smaller window, aspect ratio = .5 */
view
.5 1 .6 .85  /* Upper right region of the page, aspect ratio = .5 */
bord        /* Draw a rectangle around the viewport */

(the scene commands are repeated here)

view        /* Use the previous window */
.2 .45 0 .5 /* Lower left region of the page, aspect ratio = 2 */
bord        /* Draw a rectangle around the viewport */

(the scene commands are repeated here)

wind
0 70 0 35    /* Bottom half of scene; aspect ratio = .5 */
```

```

view
.5 1 .15 .4 /* Lower left region of the page, aspect ratio = 2 */
bord /* Draw a rectangle around the viewport */

```

(the scene commands are repeated here)

The resulting picture is shown in figure 2. You can create this picture by typing

```
> cplot eg1.dat
```

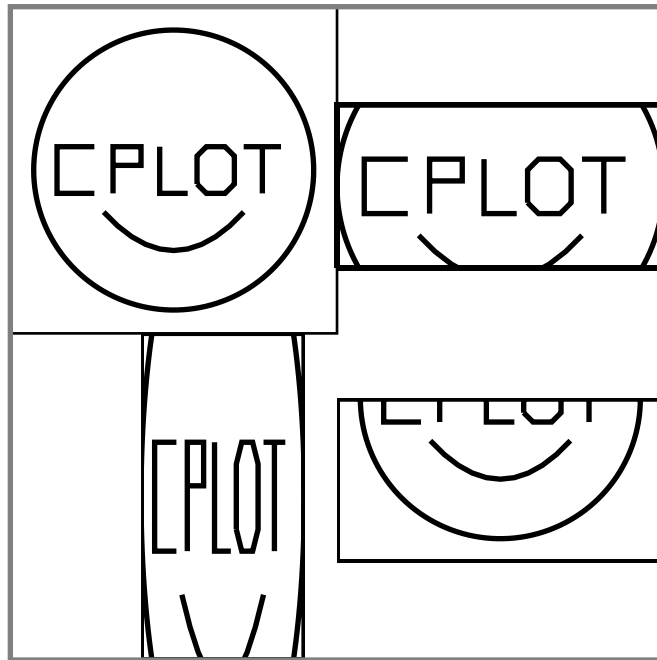


Figure 2: Picture produced from file `eg1.dat`

CPLOT also supports the commands:

/* comment */ C-like comments *between commands* are accepted, but a space has to separate comment marks from text. Thus `*/
Comment */` is legal, but `*/Comment*/` is not legal. *In the FORTRAN version, the begin comment `/*` and end comment `*/` must be on a line by themselves.*

ROTA

angle Rotate all subsequent plotting data the specified angle about the scene origin.

SCAL

sx,sy Scale all subsequent plotting data by the specified sx and sy scale factors.

TRAN

tx,ty Translate all subsequent plotting data by the specified tx and ty offsets in scene coordinates.

IDEN Set the transformation matrix to identity (zero all rotations, scales, and translations).

EXIT Terminate the program.

The effects of rotation, translation, and scaling are cumulative and order dependent.

CPLLOT also works on three dimensional scenes, although we won't be needing this feature until later in the semester. The commands **2d** and **3d**, with no arguments, control whether CPLLOT is working in 2D or 3D mode. The default is 2D. If 3D mode is specified, the scene description commands expect slightly different arguments:

CIRC

$x, y, z, x_n, y_n, z_n, r$ Plot a circle of radius r centered at (x, y, z) and lying in a plane whose normal is (x_n, y_n, z_n) .

LINE

n

x_1, y_1, z_1

x_2, y_2, z_2

...

x_n, y_n, z_n Plot the n line segments $(x_i, y_i, z_i) - (x_{i+1}, y_{i+1}, z_{i+1})$.

STORE

n,ncurve

x_0, y_0, z_0, w_0

x_1, y_1, z_1, w_1

x_2, y_2, z_2, w_2

...

x_n, y_n, z_n, w_n

CPLoT

ncurve

Plot curve # $ncurve$ as a three dimensional Bézier curve. If the curve was in fact stored in 2D mode, all z coordinates are zero.

Viewing in 3D is established using the command

3D

a_x, a_y, a_z

e_x, e_y, e_z, e_w
 u_x, u_y, u_z

where (a_x, a_y, a_z) is the “look-at” point; (e_x, e_y, e_z, e_w) is the vector in homogeneous coordinates which points from the look-at point to the eye position. That is, if $\mathbf{E} = (E_x, E_y, E_z)$ is the location of the eye,

$$(E_x, E_y, E_z) = (a_x, a_y, a_z) + \left(\frac{e_x}{e_w}, \frac{e_y}{e_w}, \frac{e_z}{e_w} \right).$$

(u_x, u_y, u_z) is the “up” vector.

The *projection plane* contains the look-at point and is perpendicular to the to-eye vector. To plot a 3D scene, all entities in that scene are projected onto this plane, and then a conventional window-to-viewport map is performed. The projection of a point \mathbf{P} in 3D onto the projection plane is obtained by intersecting the line \mathbf{P} — \mathbf{E} with the projection plane. The projection of an entire scene is obtained by projecting each point in the scene. Thus, as viewed from \mathbf{E} , the scene and its projection look identical.

The orientation of the window is determined by the up vector: The projection of the up vector onto the projection plane defines the y coordinate direction. A coordinate system is then defined on the projection plane with an origin at the look-at point. The dimensions on this coordinate system are the same as the dimensions in the 3D scene. That is, the distance between two points lying on the projection plane is the same whether based on 3D scene coordinates or 2D projection plane coordinates.

Typically, the eye position and view plane are thought of as forming a pyramid with the eye at the apex of the pyramid, and a rectangle centered at the look-at point and lying on the projection plane defining the base of the pyramid. This pyramid is referred to as the *view pyramid*, or *pyramid of vision*.

The output from data file `p1.dat` is shown in figure 3. The top right picture shows three circles in 3D, a view pyramid, and the circles projected onto the projection plane. Labelled are the eye position, the look-at point, the to-eye vector, the up vector, and the projection plane. The bottom-right picture shows the coordinate system that is imposed on the projection plane. These two pictures also show the window that is defined on the projection plane. The left hand pictures show what is actually seen from the vantage point of the respective eye positions.

Note that in the top-right case, the scene lies between the projection plane and the eye, while in the lower-right case the plane lies between the scene and the eye. Actually, any projection plane perpendicular to a given to-eye vector will produce identical pictures (if the respective windows are projections of each other).

The right hand pictures were created using a special CPLOT command. The 3D3D command allows us to view the scene, pyramid, and projected scene, from another vantage point.

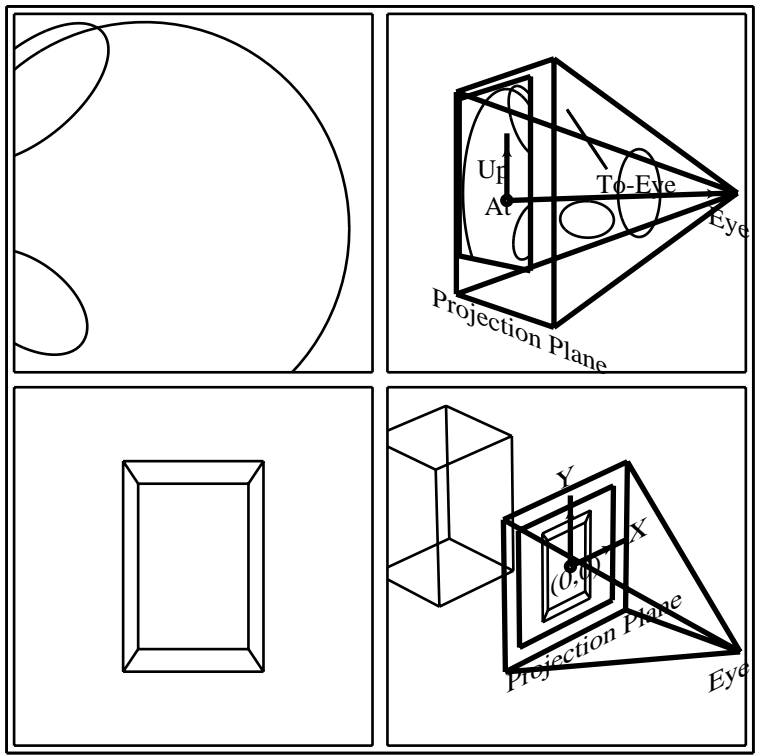


Figure 3: Picture produced from file p1.dat

3D3D

a_x, a_y, a_z

e_x, e_y, e_z, e_w /* at, to-eye, and up for pyramid being looked at */

u_x, u_y, u_z

wh, wv /* Pyramid width and height */ wcx, wcy, wx, wy /* Window drawn on projection plane (center and half-widths) */ $clip, pyr$ at_x, at_y, at_z /* View pyramid being used to view the first view pyramid */

E_x, E_y, E_z, E_w

up_x, up_y, up_z

where

clip = 1 to clip the projected scene to the window

pyr = 1 to draw the pyramid

pyr = 2 to draw only the view plane

pyr = 0; don't draw either the pyramid or plane