OpenGL 3D graphics in Liberty BASIC

Lesson One: Introduction and Basic Shapes

by Robert McAllister

Introduction

To start with, I have to say that the following is only my understanding of OpenGL graphics and how to use it in Liberty BASIC. I'm by no means an expert when it comes to 3D but I have picked up a basic working knowledge of it and would like to pass it along to my fellow LBer’s. (With a little arm twisting, Thanks Rod)

The first thing you need to know is to not be afraid to experiment with OpenGL, you will not burn up your video card or set your monitor on fire if you make a mistake. Actually, if there is an error in a call to OpenGL, it will just ignore it and keep plugging along.

OpenGL is a “state” environment, much like LB graphics. For example, when you make a call to “glColor4fv”, all objects created thereafter will have that color applied to them, just like the LB graphics command #1, "color red".

Probably one of the most difficult things to understand with 3D is the coordinate system. With LB graphics we are all familiar with X, Y coordinates. X is left to right, Y is up and down. 3D adds one more parameter, Z for depth. With these 3 coordinates we can define a vertex (a point in space) for the objects we want to create. Imagine you have a cereal box in front of you. Each of the 8 corners of the box can be described as a vertex, or a set of X, Y and Z coordinates. One other thing to keep in mind is that the center of the OpenGL window is 0, 0, 0 and we build our scene outward from there.

Basic Shapes

So let’s get started! Create a new folder and then download & extract the following file to it. This file contains an LB .bas file, two DLL’s and a bitmap. The program is set up so you copy & paste any of the code snippets from the lessons into the top section and run them. The SDL.dll (Simple DirectMedia Layer) is from http://www.libsdl.org/ and its sole purpose here is to create an OpenGL window that is Vista friendly. The DevIl.dll (Developers Image Library) is from http://openil.sourceforge.net/ and its sole purpose here is to load bitmaps in a form that can be used by OpenGL.

OpenGL in LB.zip

- Details
- Download
- 475 KB

OK, run the LB program as is...

Wow, a dot! Well this first program is meant to be as simple as possible so I can describe some of the basics about how the 3D works. The first line “GOSUB [Initialize]” sets up the main window and initializes the 3D. If you would like, you can change the size of the main window and/or the 3D window.

```
 CALL
 ClearView eyeX , eyeY , eyeZ , centerX , centerY , centerZ , upX , up
Y , upZ
 CALL glColor4fv 1 , 0 , 0 , 1
 CALL glPointSize 3
 CALL glBegin GL.POINTS
   CALL glVertex 0 , 0 , 0
 CALL glEnd
 CALL RefreshView
 WAIT
```

The next line “CALL ClearView …” clears the buffer we will be drawing to. This implementation of 3D uses two buffers, you can think of them as screens. One is displayed while the other is being created. When you are ready to see what has been created, the buffers are swapped.

The following line “CALL glColor4fv…” sets the R G B colors for the object we are about to create. The first three values can be between 0 and 1. Zero means than none of that color will be used, .5 means that 50% of that color will be used. And 1 tells it to use 100% of the color. The fourth value, alpha, will be discussed later. For now leave it as 1.

“CALL glPointSize 3” tells OpenGL to make points with 3 for the size. The default value is 1.

In the next 3 lines, “CALL glBegin GL.POINTS” lets OpenGL know that we will be creating a point. “CALL glVertex 0 , 0 , 0” specifies the location of the point and “CALL glEnd” tells OpenGL that we are done with this set of commands. Each object must be defined inside a set of glBegin/glEnd commands.

“CALL RefreshView” swaps the buffers so we can see what we've created.

And that’s as simple as it gets. Opengl is only capable of creating 3 basic objects; points, lines and polygons. More complex objects are built up from these 3. Also, OpenGL is optimized for processing triangles. This is due to that fact that triangles are always on a single plane so they don't require complex calculations for rendering.

The following snippets will create a line, a triangle and a square. Note that you can specify a different color for each vertex to create some nice effects.

```
 ' Line
 CALL
 ClearView eyeX , eyeY , eyeZ , centerX , centerY , centerZ , upX , up
Y , upZ
 'CALL glLineWidth 3
 CALL glBegin GL.LINES
   CALL glColor4fv 1 , 0 , 0 , 1
   CALL glVertex -1 , -1 , 0
   CALL glColor4fv 0 , 0 , 1 , 1
   CALL glVertex 1 , -1 , 0
 CALL glEnd
 CALL RefreshView
 WAIT
```

```
 ' Triangle
 CALL
 ClearView eyeX , eyeY , eyeZ , centerX , centerY , centerZ , upX , up
Y , upZ
 CALL glBegin GL.TRIANGLES
   CALL glColor4fv 1 , 0 , 0 , 1
   CALL glVertex -1 , -1 , 0
   CALL glColor4fv 0 , 1 , 0 , 1
   CALL glVertex 0 , 1 , 0
   CALL glColor4fv 0 , 0 , 1 , 1
   CALL glVertex 1 , -1 , 0
 CALL glEnd
 CALL RefreshView
 WAIT
```

```
 ' Square
 CALL
 ClearView eyeX , eyeY , eyeZ , centerX , centerY , centerZ , upX , up
Y , upZ
 CALL glBegin GL.QUADS
```

```
   CALL glColor4fv 1 , 0 , 0 , 1
   CALL glVertex -1 , -1 , 0
   CALL glColor4fv 0 , 1 , 0 , 1
   CALL glVertex -1 , 1 , 0
   CALL glColor4fv 0 , 0 , 1 , 1
   CALL glVertex 1 , 1 , 0
   CALL glColor4fv 0 , 0 , 0 , 1
   CALL glVertex 1 , -1 , 0
 CALL glEnd
 CALL RefreshView
 WAIT
```

Have fun and do some experimenting until the next lesson, Moving objects around the screen