Open GL Basics

Thursday, January 15, 2009

Table of Contents

What is Open GL?

- A software API consisting of around several hundred functions that allow you to talk to your graphics hardware. It is cross-platform and the most commonly used in professional graphics applications.
- We'll be using OpenGL this class as well as:
  - OpenGL Utility Library (GLU): a library of functions useful for drawing and transforming objects
  - OpenGL Utility Toolkit (GLUT) : a library of utility functions that perform system and I/O functions.

Some Basic Syntax

- In order to write our first OpenGL program, there are some things that we should know.
- All functions in OpenGL use the prefix "gl"
  - Functions from GLU and GLUT have the prefixes "glu" and "glut" respectively.
- All constants in OpenGL use the prefix "GL_"
  - i.e. glBegin(GL_POLYGON);

Code Example

```
//Snippet of OpenGL Code...
int main(int argc, char ** argv )
{
    glutInit(&argc, argv); //process arguments
    glutInitDisplayMode( GLUT_RGBA | GLUT_DOUBLE ); //designates buffers

    glutInitWindowPosition( 0, 0 ); //sets some initial stuff
    glutInitWindowSize(100, 100);
    static int window = glutCreateWindow( "OpenGL!"); //creates window

    glutIdleFunc(idle_callback); // maps sample callback functions
    glutDisplayFunc(display_callback);

    glutMainLoop(); //enters main processing loop

    exit(0);
}
```

Code Example

```
//Sample drawing function
void display_callback(){
    glClearColor(0.0, 0.0, 0.0); //clears our buffers
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //perform any other matrix transformations here

    //DRAW OBJECTS HERE!

    glFlush; //ensures our objects are drawn right away
    glutSwapBuffers(); //if we are using double buffering
}
```

A Bit on Buffers

- Pixels are drawn to the screen using the information stored in the buffers.
- Each time we update, we have to clear and rewrite information to the buffers.
- OpenGL allows access to modify several buffers such as:
  - Color buffers
    - Contain information about the color stored at each pixel. Double-buffering is the technique of calculating one while displaying the other.
  - Depth buffer (z-buffer)
    - Stores depth information at each pixel, allows closer objects to be drawn on top of ones further away

Drawing 101

- All geometric objects can basically be represented as a set of vertices in 2-D or 3-D space.
- We draw objects by designating vertices and deriving primitives, or basic shapes from them.
- Every vertex can be declared using the function: glVertex*()
  - Here, * is simply an expression that consists of the number of dimensions our vertex is in and the type of parameter we are passing to the function.
  - For example: glVertex3f(0.0, 0.0, 1.0) denotes a vertex in 3-D space with floats passed in as parameters.

Geometric Primitives

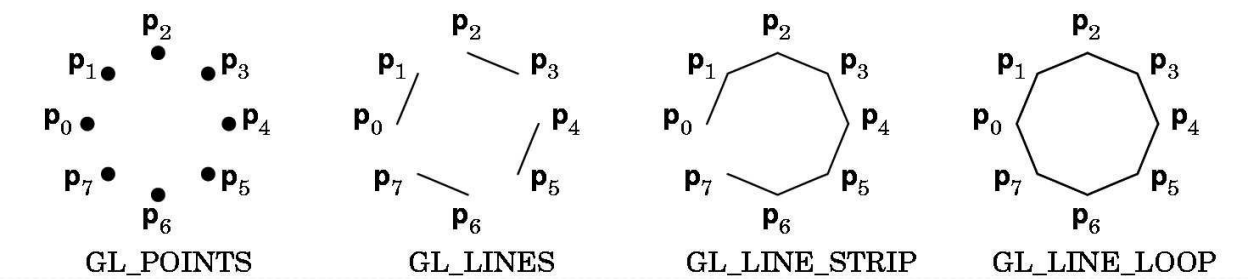
- In OpenGL, primitives are nothing more than points, lines, and polygons that make up larger objects.
- Declare what kind of primitive we are drawing
  - glBegin(Glenum *mode*);
    - *mode* is basically the type of primitive we're drawing
- Declare our vertices
  - i.e. glVertex2f(float *x*, float *y*);
- Declare that we are finished
  - glEnd();

Code Example

```
//Sample drawing function
void display_callback(){
    ...
    glBegin(GL_QUADS);
            glColor3f(0.0, 0.0, 1.0); //sets color to blue
            glVertex2f(0.0, 0.0);
            glVertex2f(0.0, 1.0);
            glVertex2f(1.0, 1.0);
            glVertex2f(1.0, 0.0);
    glEnd();
    ...
}
```

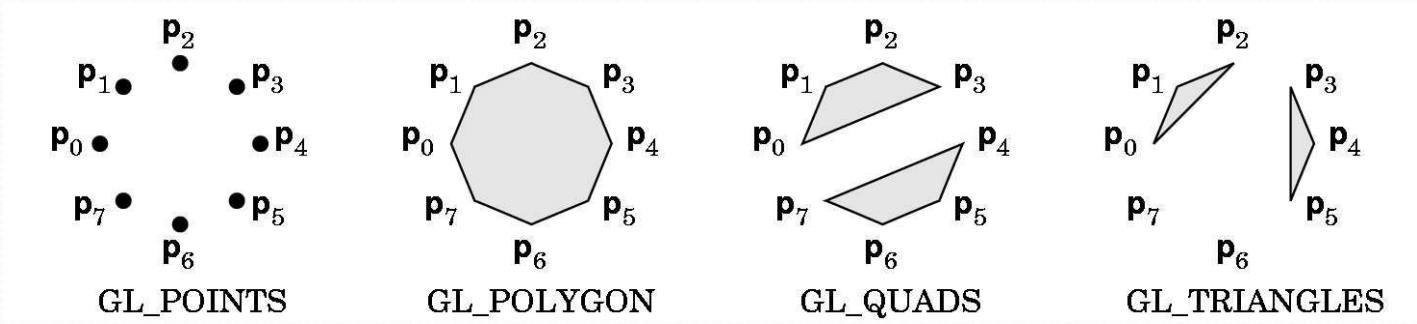
Types of Primitives

- GL_POINTS
  - Simply draws the points in the order you pass them in.
- GL_LINES
  - Takes pairs of vertices and draws lines between them
- GL_LINE_STRIP
  - Takes any number of vertices and draws a series of connected line segments
- GL_LINE_LOOP
  - Same as GL_LINE_STRIP, except that it connects the first and last vertices.



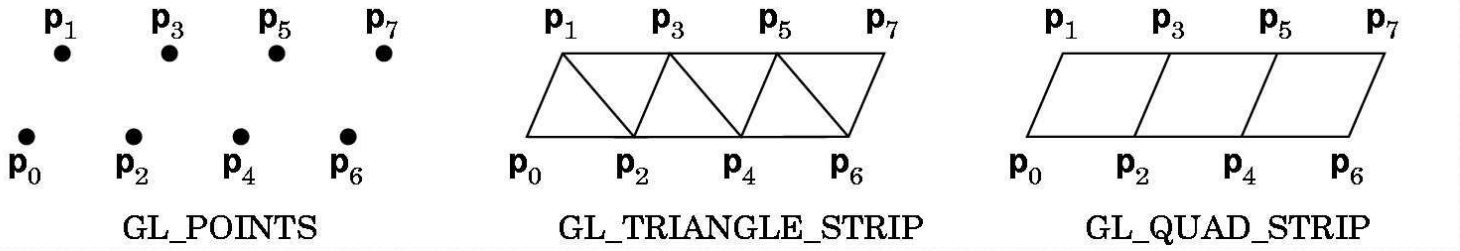
Types of Primitives

- GL_TRIANGLES
  - Takes vertices in triples and draws them as them as triangles.
- GL_QUADS
  - Takes vertices in quadruples and draws them as four-sided-polygons.
- GL_POLYGON
  - Takes any number of vertices and draws the boundary of the convex polygon they form.
  - Note: Order of vertices here is important. All polygons must be convex and their edges cannot intersect.



Types of Primitives

- GL_TRIANGLE_STRIP
  - Takes any number of vertices and draws a strip of triangles.
  - Here: p0, p1, p2 followed by p2, p1, p3 followed by p2, p3, p4…
- GL_QUAD_STRIP
  - Takes any number of vertices and draws a strip of quads .
  - Here: p0, p1, p3, p2 followed by p2, p3, p5, p4 …
- GL_TRIANGLE_FAN
  - Takes any number of vertices and draws a circular fan of triangles starting from the first vertex.
  - Would draw: p0, p1, p2 followed by p0, p2, p3 followed by p0, p3, p4 …



Display Lists

- One thing to note about drawing primitives is that we will often draw them many times.
- One way to optimize drawing objects is to store them in an object called a display list.
- Basically, a display list provides a way for OpenGL to remember the exact way something is drawn and then redraw it again on the fly.

Display Lists II

- Declare a new list
  - glNewList(GLuint *list*, GLenum *mode);*
    - *list* is the name of our list
    - *mode* is our compilation mode (either GL_COMPILE or GL_COMPILE_AND_EXECUTE)
- Draw an object in between.
  - i.e. glBegin(GL_POLYGONS); …; glEnd();
- Declare the end of the list
  - glEndList();
- Anytime you want to draw your object, call
  - glCallList(GLuint *list*);

Code Example

```
//Some method called to initialize objects at start
void buildObject{
    glNewList(MY_SQUARE_LIST, GL_COMPILE);
            glBegin(GL_QUADS);
                    glVertex2f(0.0, 0.0);
                    glVertex2f(0.0, 1.0);
                    glVertex2f(1.0, 1.0);
                    glVertex2f(1.0, 0.0);
            glEnd();
    glEndList();
}

//Sample drawing function
void display_callback(){
    …
    //Draw objects here
    glCallList(MY_SQUARE_LIST);
    …
}
```