Getting Started with OpenGL Graphics Programming in C/C++

Mike Bailey
mjb@cs.oregonstate.edu

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

The Basic Computer Graphics Pipeline

Geometry vs. Topology

3D Coordinate Systems

Homer Simpson uses Right-handed Coordinates. So, we will too.
Right-handed Coordinate System

Right-handed Positive Rotations

OpenGL Topologies

OpenGL Topologies -- Polygon Requirements

OpenGL Topologies -- Orientation

Drawing in 3D

```c
# Set any display characteristics state that you want to have in effect when you do the drawing.
setColor(x, y, z);

# Begin the drawing. Use the current state's display characteristics. Here is the topology to be used with these vertices.
begin([GL_LINE_STRIP, GL_LINE_LOOP, GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_QUADS, GL_POLYGON]);

# This is a wonderfully understandable way to start with 3D graphics – it is like holding a marker in your hand and sweeping out linework in the 3D air in front of you!

# But it is also incredibly internally inefficient! We'll talk about that later and what to do about it...
```

OpenGL Topologies

OpenGL Topologies -- Polygon Requirements

OpenGL Topologies -- Orientation
OpenGL Topologies – Vertex Order Matters

GL_LINE_LOOP

V0 V1 V2 V3

GL_LINE_LOOP

V0 V2 V1

Probably what you meant to do

Probably not what you meant to do

This disease is referred to as “The Bowtie”

What does “Convex Polygon” Mean?

We can go all mathematical here, but let’s go visual instead. In a convex polygon, a line between any two points inside the polygon never leaves the inside of the polygon.

Convex

Stays within the polygon

Not Convex

Leaves the polygon

Why is there a Requirement for Polygons to be Convex?

Graphics polygon-filling hardware can be highly optimized if you know that, no matter what direction you fill the polygon in, there will be two and only two intersections between the scanline and the polygon’s edges.

Graphics polygon-filling hardware can be highly optimized if you know that, no matter what direction you fill the polygon in, there will be two and only two intersections between the scanline and the polygon’s edges.

Convex

2 edge intersections

Not Convex

2 edge intersections

What if you need to display Polygons that are not Convex?

There are two good solutions I know of (and there are probably more):

1. OpenGL’s utility (gluXxx) library has a built-in tessellation capability to break a non-convex polygon into convex polygons.
2. There is an open source library to break a non-convex polygon into convex polygons. It is called Polypartition, and the source code can be found here:
   https://github.com/ivanfratric/polypartition

If you ever need to do this, contact me. I have working code for each approach…

OpenGL Drawing Can Be Done Procedurally

Listing a lot of vertices explicitly gets old in a hurry

glColor3f( r, g, b );
gBegin( GL_LINE_LOOP );
gVertex3f( x0, y0, 0. );
. . .
gEnd();

Coloring a lot of vertices explicitly gets old in a hurry

gColor3f( r, g, b );
float dang = 2. * M_PI / (float)( NUMSEGS – 1 );
float ang = 0.;
gBegin( GL_LINE_LOOP );
for( int i = 0; i < NUMSEGS; i++ )
{
    glVertex3f( RADIUS*cos(ang), RADIUS*sin(ang), 0. );
    ang += dang;
}
gEnd();

Why is there a Requirement for Polygons to be Planar?

Graphics hardware assumes that a polygon has a definite front and a definite back, and that you can only see one of them at a time

OK

OK

Not OK

OpenGL Topologies – Vertex Order Matters

GL_LINE_LOOP

V0 V1 V2 V3

GL_LINE_LOOP

V0 V2 V1

Probably what you meant to do

Probably not what you meant to do

This disease is referred to as “The Bowtie”

What does “Convex Polygon” Mean?

We can go all mathematical here, but let’s go visual instead. In a convex polygon, a line between any two points inside the polygon never leaves the inside of the polygon.

Convex

Stays within the polygon

Not Convex

Leaves the polygon

Why is there a Requirement for Polygons to be Convex?

Graphics polygon-filling hardware can be highly optimized if you know that, no matter what direction you fill the polygon in, there will be two and only two intersections between the scanline and the polygon’s edges.

Convex

2 edge intersections

Not Convex

2 edge intersections

What if you need to display Polygons that are not Convex?

There are two good solutions I know of (and there are probably more):

1. OpenGL’s utility (gluXxx) library has a built-in tessellation capability to break a non-convex polygon into convex polygons.
2. There is an open source library to break a non-convex polygon into convex polygons. It is called Polypartition, and the source code can be found here:
   https://github.com/ivanfratric/polypartition

If you ever need to do this, contact me. I have working code for each approach…

OpenGL Drawing Can Be Done Procedurally

Listing a lot of vertices explicitly gets old in a hurry

glColor3f( r, g, b );
gBegin( GL_LINE_LOOP );
gVertex3f( x0, y0, 0. );
. . .
gEnd();

Coloring a lot of vertices explicitly gets old in a hurry

gColor3f( r, g, b );
float dang = 2. * M_PI / (float)( NUMSEGS – 1 );
float ang = 0.;
gBegin( GL_LINE_LOOP );
for( int i = 0; i < NUMSEGS; i++ )
{
    glVertex3f( RADIUS*cos(ang), RADIUS*sin(ang), 0. );
    ang += dang;
}
gEnd();

Why is there a Requirement for Polygons to be Planar?

Graphics hardware assumes that a polygon has a definite front and a definite back, and that you can only see one of them at a time

OK

OK

Not OK
Color

Color

Cyan = Green + Blue
Magenta = Red + Blue
Yellow = Red + Green
White = Red + Green + Blue

This is referred to as “Additive Color”

OpenGL Transformations

Single Transformations

Why do the Compound Transformations Take Effect in Reverse Order?

Envision fully-parenthesizing what is going on. In that case, it makes perfect sense that the most recently-set transformation would take effect first.
Order Matters! Compound Transformations are Not Commutative

Rotate, then translate.

Translate, then rotate.

The OpenGL Drawing State

The designers of OpenGL could have put lots and lots of arguments on the glVertex3f call to totally define the appearance of your drawing, like this:

```
glVertex3f(x, y, z, r, g, b, m00, ..., m33, s, t, nx, ny, nz, linewidth, ...);
```

Yuck! That would have been ugly. Instead, they decided to let you create a “current drawing state”. You set all of these characteristics first, then they take effect when you do the drawing. They continue to remain in effect for future drawing calls, until you change them.

Set the state first.

Draw with the state second.

Projecting an Object from 3D into 2D

Orthographic (or Parallel) projection

```
glOrtho(xl, xr, yb, yt, zn, zf);
```

Perspective projection

```
gluPerspective(fovy, aspect, zn, zf);
```

Parallel lines remain parallel.

Parallel lines appear to converge.

"Vanishing Point".

OpenGL Projection Functions

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
```

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

```
gluLookAt(ex, ey, ez, lx, ly, lz, ux, uy, uz);
```

```
glTranslatef(tx, ty, tz);
```

```
glRotatef(degrees, ax, ay, az);
```

```
glScalef(sx, sy, sz);
```

```
glColor3f(r, g, b);
```

```
gBegin(GL_LINE_STRIP);
```

```
gVertex3f(x0, y0, z0);
gVertex3f(x1, y1, z1);
gVertex3f(x2, y2, z2);
gVertex3f(x3, y3, z3);
gVertex3f(x4, y4, z4);
```

```
gEnd();
```

How the Viewing Volumes Look from the Outside

```
gluPerspective(fovy, aspect, zn, zf);
```
The Perspective Viewing Frustum

\[ \text{gluPerspective}( \text{fovy}, \text{aspect}, \text{zn}, \text{zf}); \]

- \text{fovy} = \text{vertical field of view angle (degrees)}
- \text{good values are} 50-100°

\[ \text{aspect} = \frac{\text{DX}}{\text{DY}} \]

Arbitrary Viewing

\[ \text{glMatrixMode}( \text{GL_PROJECTION}); \]
\[ \text{glLoadIdentity}(); \]
\[ \text{gluPerspective}( \text{fovy}, \text{aspect}, \text{zn}, \text{zf}); \]
\[ \text{glMatrixMode}( \text{GL_MODELVIEW}); \]
\[ \text{glLoadIdentity}(); \]
\[ \text{gluLookAt}( \text{ex}, \text{ey}, \text{ez}, \text{lx}, \text{ly}, \text{lz}, \text{ux}, \text{uy}, \text{uz}); \]
\[ \text{glTranslatef}( \text{tx}, \text{ty}, \text{tz}); \]
\[ \text{glRotatef}( \text{degrees}, \text{ax}, \text{ay}, \text{az}); \]
\[ \text{glScalef}( \text{sx}, \text{sy}, \text{sz}); \]
\[ \text{glColor3f}( \text{r}, \text{g}, \text{b}); \]
\[ \text{glBegin}( \text{GL_LINE_STRIP}); \]
\[ \text{glVertex3f}( \text{x0}, \text{y0}, \text{z0}); \]
\[ \text{glVertex3f}( \text{x1}, \text{y1}, \text{z1}); \]
\[ \text{glVertex3f}( \text{x2}, \text{y2}, \text{z2}); \]
\[ \text{glVertex3f}( \text{x3}, \text{y3}, \text{z3}); \]
\[ \text{glVertex3f}( \text{x4}, \text{y4}, \text{z4}); \]
\[ \text{glEnd}(); \]

How Can You Be Sure You See Your Scene?

\[ \text{gluPerspective}( \text{fovy}, \text{aspect}, \text{zn}, \text{zf}); \]
\[ \text{gluLookAt}( \text{ex}, \text{ey}, \text{ez}, \text{lx}, \text{ly}, \text{lz}, \text{ux}, \text{uy}, \text{uz}); \]

Here’s a good way to start:
1. Set \( \text{lx,ly,lz} \) to be the average of all the vertices
2. Set \( \text{ux,uy,uz} \) to be 0.,1.,0.
3. Set \( \text{ex=lx} \) and \( \text{ey=ly} \)
4. Now, you change \( \Delta E \) or \( \text{fovy} \) so that the object fits in the viewing volume:

\[ \Delta E = \frac{H}{2 \tan \left( \frac{\text{fovy}}{2} \right)} \]

\[ \Delta \text{fovy} = 2 \arctan \left( \frac{H}{2 \Delta E} \right) \]

Giving:

\[ \text{fovy} = \text{ZeroValue} \]

or:

\[ \Delta \text{E} = \frac{H}{2 \tan \left( \frac{\text{fovy}}{2} \right)} \]

Specifying a Viewport

\[ \text{glViewport}( \text{ixl}, \text{iyb}, \text{idx}, \text{idy}); \]

Viewports use the upper-left corner as (0,0) and their Y goes down.

Note: setting the viewport is not part of setting either the Modelview or the Projection transformations.

Saving and Restoring the Current Transformation

\[ \text{glViewport}( \text{ixl}, \text{iyb}, \text{idx}, \text{idy}); \]
\[ \text{glMatrixMode}( \text{GL_PROJECTION}); \]
\[ \text{glLoadIdentity}(); \]
\[ \text{gluPerspective}( \text{fovy}, \text{aspect}, \text{zn}, \text{zf}); \]
\[ \text{glMatrixMode}( \text{GL_MODELVIEW}); \]
\[ \text{glLoadIdentity}(); \]
\[ \text{gluLookAt}( \text{ex}, \text{ey}, \text{ez}, \text{lx}, \text{ly}, \text{lz}, \text{ux}, \text{uy}, \text{uz}); \]
\[ \text{glTranslatef}( \text{tx}, \text{ty}, \text{tz}); \]
\[ \text{glRotatef}( \text{degrees}, \text{ax}, \text{ay}, \text{az}); \]
\[ \text{glScalef}( \text{sx}, \text{sy}, \text{sz}); \]
\[ \text{glColor3f}( \text{r}, \text{g}, \text{b}); \]
\[ \text{glBegin}( \text{GL_LINE_STRIP}); \]
\[ \text{glVertex3f}( \text{x0}, \text{y0}, \text{z0}); \]
\[ \text{glVertex3f}( \text{x1}, \text{y1}, \text{z1}); \]
\[ \text{glVertex3f}( \text{x2}, \text{y2}, \text{z2}); \]
\[ \text{glVertex3f}( \text{x3}, \text{y3}, \text{z3}); \]
\[ \text{glVertex3f}( \text{x4}, \text{y4}, \text{z4}); \]
\[ \text{glEnd}(); \]
\[ \text{glPopMatrix}(); \]

...
sample.cpp Program Structure

- #includes
- Consts and #defines
- Global variables
- Function prototypes
- Main program
- InitGraphics function
- Display callback
- Keyboard callback

includes

```cpp
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#define _USE_MATH_DEFINES
#include <math.h>
#ifdef WIN32
#include <windows.h>
#pragma warning(disable:4996)
#include "glew.h"
#endif
#include <GL/gl.h>
#include <GL/glu.h>
#include "glut.h"
```

consts and #defines

```cpp
const char *WINDOWTITLE = { "OpenGL / GLUT Sample -- Joe Graphics" };
const char *GLUITITLE = { "User Interface Window" };
const int GLUITRUE = { true };
const int GLUIFALSE = { false };
define ESCAPE 0x1b
const int INIT_WINDOW_SIZE = { 600 };
const float BOXSIZE = { 2.f };
const float ANGFACT = { 1. };
const float SCLFACT = { 0.005f };
const float MINSCALE = { 0.05f };
const int LEFT = { 4 };
const int MIDDLE = { 2 };
const int RIGHT = { 1 };
enum Projections { ORTHO, PERSP };
enum ButtonVals { RESET, QUIT };
enum Colors { RED, YELLOW, GREEN, CYAN, BLUE, MAGENTA, WHITE, BLACK };
```

const and #defines are always preferred over #defines. But, Visual Studio does not allow consts to be used in case statements or as array sizes.

Initialized Global Variables

```cpp
cconst GLutButtonCallback *GLUT_BUTTON_CALLBACKS[ ] = { DoAxesMenu, DoColorMenu, DoDepthMenu, DoDebugMenu, DoMainMenu, DoProjectMenu, DoRasterString, DoStrokeString, ElapsedSeconds, InitGraphics, InitLists, InitMenus, Keyboard, MouseButton, MouseMotion, Reset, Resize, Visibility, Axes, HsvRgb };```

Global Variables

```cpp
int ActiveButton; // current button that is down
GLuint AxesList; // list to hold the axes
int AxesOn; // != 0 means to draw the axes
int DebugOn; // != 0 means to print debugging info
int DepthCueOn; // != 0 means to use intensity depth cueing
GLuint BoxList; // object display list
int MainMenu; // window id for main graphics window
float Scale; // scaling factor
int WhichColor; // index into Colors[]
int WhichProjection; // ORTHO or PERSP
int Xmouse, Ymouse; // mouse values
float Xrot, Yrot; // rotation angles in degrees
```

Function Prototypes

```cpp
void Animate( );
void Display( );
void DoAxesMenu( int );
void DoColorMenu( int );
void DoDepthMenu( int );
void DoDebugMenu( int );
void DoMainMenu( int );
void DoProjectMenu( int );
void DoRasterString( float, float, float, char * );
void DoStrokeString( float, float, float, float, char * );
float ElapsedSeconds( );
void InitGraphics( );
void InitLists( );
void InitMenus( );
void Keyboard( unsigned char, int, int );
void MouseButton( int, int, int, int );
void MouseMotion( int, int );
void Reset( );
void Resize( int, int );
void Visibility( int );
void Axes( float );
void HsvRgb( float[3], float[3] );
```
Main Program

```c
main(int argc, char *argv[]) {
    // turn on the glut package:
    // (do this before checking argc and argv since it might
    // pull some command line arguments out)
    glutInit(&argc, argv);
    // setup all the graphics stuff:
    InitGraphics();
    // create the display structures that will not change:
    InitLists();
    // init all the global variables used by Display();
    // this will also post a redisplay
    Reset();
    // setup all the user interface stuff:
    InitMenus();
    // draw the scene once and wait for some interaction:
    // (this will never return)
    glutSetWindow(MainWindow);
    glutMainLoop();
    // this is here to make the compiler happy:
    return 0;
}
```

InitGraphics()

```c
void InitGraphics() {
    // request the display modes:
    // ask for red-green-blue-alpha color, double-buffering, and z-buffering:
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
    // set the initial window configuration:
    glutInitWindowPosition(0, 0);
    glutInitWindowSize(INIT_WINDOW_SIZE, INIT_WINDOW_SIZE);
    // open the window and set its title:
    MainWindow = glutCreateWindow(WINDOWTITLE);
    glutSetWindowTitle(WINDOWTITLE);
    // set the framebuffer clear values:
    glClearColor(BACKCOLOR[0], BACKCOLOR[1], BACKCOLOR[2], BACKCOLOR[3]);
    glutSetWindow(MainWindow);
    glutDisplayFunc(Display);
    glutReshapeFunc(Resize);
    glutKeyboardFunc(Keyboard);
    glutMouseFunc(MouseButton);
    glutMotionFunc(MouseMotion);
    glutTimerFunc(-1, NULL, 0);
    glutIdleFunc(NULL);
}
```

Display()

```c
void Display() {
    // set which window we want to do the graphics into:
    glutSetWindow(MainWindow);
    // erase the background:
    glDrawBuffer(GL_BACK);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);
    // specify shading to be flat:
    glShadeModel(GL_FLAT);
    // set the viewport to a square centered in the window:
    GLsizei vx = glutGet(GLUT_WINDOW_WIDTH);
    GLsizei vy = glutGet(GLUT_WINDOW_HEIGHT);
    GLsizei v = vx < vy ? vx : vy;                  // minimum dimension
    GLint xl = (vx - v) / 2;
    GLint yb = (vy - v) / 2;
    glViewport(xl, yb, v, v);
    // set the viewing volume:
    // remember that the Z clipping  values are actually
    // given as DISTANCES IN FRONT OF THE EYE
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if(WhichProjection == ORTHO)
      glOrtho(-3., 3., -3., 3., 0.1, 1000.);
    else
      gluPerspective(90., 1., 0.1, 1000.);
    // place the objects into the scene:
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    // set the eye position, look-at position, and up-vector:
    gluLookAt(0., 0., 3.,     0., 0., 0.,     0., 1., 0.);
    // rotate the scene:
    glRotatef((GLfloat)Yrot, 0., 1., 0.);
    glRotatef((GLfloat)Xrot, 1., 0., 0.);
    // uniformly scale the scene:
    if(Scale < MINSCALE)
      Scale = MINSCALE;
    glScalef((GLfloat)Scale, (GLfloat)Scale, (GLfloat)Scale);
    // set the fog parameters:
    if(DepthCueOn != 0)
      {
        glFogi(GL_FOG_MODE, FOGMODE);
        glFogfv(GL_FOG_COLOR, FOGCOLOR);
        glFogf(GL_FOG_DENSITY, FOGDENSITY);
        glFogf(GL_FOG_START, FOGSTART);
        glFogf(GL_FOG_END, FOGEND);
        glEnable(GL_FOG);
      }
    else
      {
        glDisable(GL_FOG);
      }
    // possibly draw the axes:
    if(AxesOn != 0)
      {
        glColor3fv(&Colors[WhichColor][0]);
        glCallList(AxesList);
      }
    // draw the current object:
    glCallList(BoxList);
    glFlush();
}
```
// draw some gratuitous text that just rotates on top of the scene

DoRasterString( 0., 1., 0., "Text That Moves" );

// draw some gratuitous text that is fixed on the screen:
// the projection matrix is reset to define a scene whose
// world coordinate system goes from 0-100 in each axis
// this is called "percent units", and is just a convenience
// the modelview matrix is reset to identity as we don't
// want to transform these coordinates

gluOrtho2D( 0., 100.,     0., 100. );

Color3f( 1., 1., 1. );

DoRasterString( 5., 5., 0., "Text That Doesn't" );

// swap the double-buffered framebuffers:

Display( ), IV
There is a piece of hardware called the **Rasterizer**. Its job is to interpolate a line or polygon, defined by vertices, into a collection of **fragments**. Think of it as filling in squares on graph paper.

A fragment is a “pixel-to-be”. In computer graphics, the word “pixel” is defined as having its full RGBA already computed. A fragment does not yet have its final RGBA computed, but all of the information needed to compute the RGBA is available to it.

A fragment is turned into a pixel by the **fragment processing** operation. In CS 457/557, you will do some pretty snazzy things with your own fragment processing code!