

Introduction to OpenGL

Graphics and Computation

Michael Papasimeon

2006, 2007, 2008

Outline

- OpenGL Background and History
- Other Graphics Technologies
- Drawing
- Viewing and Transformation
- Lighting
- GLUT and JOGL
- Resources

OpenGL Background and History

- OpenGL = Open Graphics Library
- Developed at Silicon Graphics (SGI)
- Successor to IrisGL
- Cross Platform (Win32, Mac OS X, Unix, Linux)
- Only does 3D Graphics. No Platform Specifics (Windowing, Fonts, Input, GUI)
- Version 1.4 widely available
- Two Libraries
 - GL (Graphics Library)
 - GLU (Graphics Library Utilities)

Other Graphics Technologies

- Low level graphics
- OpenGL
- Scene Graphs, BSPs
 - OpenSceneGraph, Java3D, VRML, PLIB
- DirectX - Direct3D
- Can mix some DirectX with OpenGL (e.g. in Quake III OpenGL is used w/ DirectInput)

Platform Specifics

- Platform Specific OpenGL Interfaces
 - Windows - WGL
 - XWindows X11 - GLX
 - Mac OS X - CGL/AGL/NSOpenGL
 - Motif - GLwMwidget
 - Qt - QGLWidget, QGLContext
- GLUT - GL Utility Library (C, Python, ...)
- JOGL (Java)

The Drawing Process

```
ClearTheScreen();  
DrawTheScene();  
CompleteDrawing();  
SwapBuffers();
```

- In animation there are usually two buffers.
- Drawing usually occurs on the background buffer.
- When it is complete, it is brought to the front (swapped).
- This gives a smooth animation without the viewer seeing the actual drawing taking place.
- Only the final image is viewed.
- The technique to swap the buffers will depend on which windowing library you are using with OpenGL.

Clearing the Window

```
glClearColor(0.0, 0.0, 0.0, 0.0);  
glClear(GL_COLOR_BUFFER_BIT);
```

Typically you would clear the colour and depth buffers together.

```
glClearColor(0.0, 0.0, 0.0, 0.0);  
glClearDepth(0.0);  
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Setting the Colour

- Colour is specified in (R,G,B,A) form [Red, Green, Blue, Alpha].
- Each value being in the range of 0.0 to 1.0.
- There are many variants of the glColor command.

Specifying Colour with glColor

```
glColor4f(red, green, blue, alpha);  
glColor3f(red, green, blue);  
  
glColor3f(0.0, 0.0, 0.0); /* Black */  
glColor3f(1.0, 0.0, 0.0); /* Red */  
glColor3f(0.0, 1.0, 0.0); /* Green */  
glColor3f(1.0, 1.0, 0.0); /* Yellow */  
glColor3f(1.0, 0.0, 1.0); /* Magenta */  
glColor3f(1.0, 1.0, 1.0); /* White */
```


Complete Drawing the Scene

Need to tell OpenGL you have finished drawing your scene:

```
glFinish();
```

or

```
glFlush();
```

For more information see:

<http://www.rush3d.com/reference/opengl-redbook-1.1/chapter02.html>

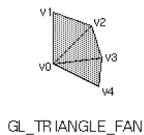
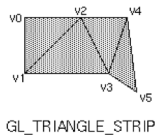
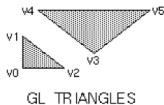
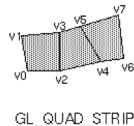
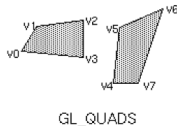
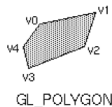
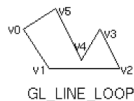
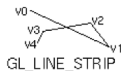
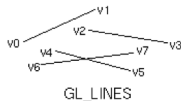
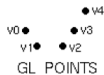
Drawing in OpenGL

- Use `glBegin()` to start drawing and `glEnd()` to stop.
- `glBegin()` can draw in many different styles.
- `glVertex3f(x,y,z)` specifies a point in 3D space.

Drawing a Polygon

```
glBegin(GL_POLYGON);  
glVertex3f(0.0, 0.0, 0.0);  
glVertex3f(0.0, 3.0, 1.0);  
glVertex3f(3.0, 3.0, 3.0);  
glVertex3f(4.0, 1.5, 1.0);  
glVertex3f(3.0, 0.0, 0.0);  
glEnd();
```

glBegin Drawing Modes



Mixing Geometry with Colour

Specifying vertices can be mixed with colour and other types of commands for interesting drawing results.

```
glBegin(GL_POLYGON);  
glColor3f(1.0, 0.0, 0.0);  
glVertex3f(0.0, 0.0, 0.0);  
glColor3f(0.0, 1.0, 0.0);  
glVertex3f(3.0, 1.0, 0.0);  
glColor3f(0.0, 0.0, 1.0);  
glVertex3f(3.0, 0.0, 0.0);  
glEnd();
```

Viewing the Scene: The Camera

OpenGL Vertices

- OpenGL uses a 4-component vector to represent a vertex.
- Known as homogenous coordinate system ¹ where typically $w = 1$.
- In 2D-space $z = 0$.

$$v = \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

¹For further information on homogenous coordinate systems as used in projective geometry and in OpenGL, see Appendix G of the Red Book

<http://www.rush3d.com/reference/opengl-redbook-1.1/appendixg.html>

Vertex Transformations

Vertex Transformation as Matrix Multiplication

$$v' = Mv$$

$$M = \begin{pmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m_{12} & m_{16} \end{pmatrix}$$

The ModelView Matrix

```
glMatrixMode(GL_MODELVIEW);
```

Specifying the ModelView matrix is analogous to:

- Positioning and aiming the camera (viewing transformation)
- Positioning and orienting the model (modeling transformation)

The Projection Matrix

```
glMatrixMode(GL_PROJECTION);
```

- Specifying the Projection matrix is like choosing a lens for a camera.
- It lets you specify parameters such as the field of view.

OpenGL Matrix Operations

Identity Matrix

$$I = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

```
glMatrixMode(mode);  
glLoadIdentity();  
glMultMatrix();  
glLoadMatrix();
```

Perspective Projection (glFrustrum)

```
glFrustrum(left, right, bottom, top, near, far);
```

Perspective Projection (gluPerspective) from GLU

```
gluPerspective(fovy, aspect, near, far);
```

- fovy : field of view in degrees, in the y direction
- aspect : aspect ratio that determines the field of view in the x direction (ratio of width to height)

Orthographics (Parallel) Projection - glOrtho

```
glOrtho(left, right, bottom, top, near, far);
```

gluLookAt

Specifies the camera position, the point where the camera is pointing, and the orientation vector of the camera.

```
gluLookAt(eyex, eyey, eyez,  
          centerx, centery, centerz,  
          upx, upy, upz);
```

Translation Transformation

```
glTranslatef(x, y, z);
```

$$T = \begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$v' = Tv$$

$$T^{-1} = \begin{pmatrix} 1 & 0 & 0 & -x \\ 0 & 1 & 0 & -y \\ 0 & 0 & 1 & -z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation Transformations

```
glRotatef(angle, x, y,z);
```

Specifies an angle and an axis (x,y,z) to rotate around.

Rotation Matrices

$$\text{glRotate3f}(\alpha, 1, 0, 0) : R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\text{glRotate3f}(\alpha, 0, 1, 0) : R_y(\alpha) = \begin{pmatrix} \cos(\alpha) & 0 & \sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\text{glRotate3f}(\alpha, 0, 0, 1) : R_z(\alpha) = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Scaling Transformations

```
glScale(x, y, z);
```

$$S = \begin{pmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$v' = Sv$$

$$S^{-1} = \begin{pmatrix} \frac{1}{x} & 0 & 0 & 0 \\ 0 & \frac{1}{y} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{z} & 1 \end{pmatrix}$$

Order of Transformations

Transformations in Action

The Matrix Stack

OpenGL Lighting

- Eight available lights (individually toggled)
- Lighting types:
 - Emitted
 - Ambient
 - Diffuses
 - Specular
- Material colours and properties
- Lighting demo

Setting up an OpenGL Light

```
GLfloat lightAmbient[] = { 0.4, 0.5, 0.0, 1.0 };
GLfloat lightDiffuse[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat lightSpecular[] = { 1.0, 1.0, 1.0, 1.0};
GLfloat lightPosition[] = {1.0, 1.0, 1.0, 0.0};

glLightfv(GL_LIGHT0, GL_AMBIENT, lightAmbient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, lightDiffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, lightSpecular);
glLightfv(GL_LIGHT0, GL_POSITION, lightPosition);

glEnable(GL_LIGHT0);
glEnable(GL_LIGHTING);
```


`glLight{if}[v](light, pname, param)`

Material Properties

```
glMaterial{if}[v](GLenum face, GLenum pname, TYPE param);
```

```
GLfloat material[] = {0.1, 0.5, 0.8, 1.0 };  
glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, material);  
  
GLfloat matSpecular[] = { 1.0, 1.0, 1.0, 1.0 };  
GLfloat lowShininess[] = { 5.0 };  
  
glMaterialfv(GL_FRONT, GL_SPECULAR, matSpecular);  
glMaterialfv(GL_FRONT, GL_SHININESS, lowShininess);
```

glmMaterial Default Parameters

Teapots Materials Example

Normal Vectors

```
glBegin(GL_POLYGON);  
    glNormal3fv(n0);  
    glVertex3fv(v0);  
    glNormal3fv(n1);  
    glVertex3fv(v1);  
    glNormal3fv(n2);  
    glVertex3fv(v2);  
    glNormal3fv(n3);  
    glVertex3fv(v3);  
glEnd();
```

Normal Vectors (2)

Hidden Surface Removal

In order for OpenGL to remove hidden surfaces, you need to enable depth testing for the depth buffer.

```
glEnable(GL_DEPTH_TEST);
while (1)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    GetCurrentViewingPosition();
    DrawObjectA();
    DrawObjectB();
}
```

- GLUT = GL UtilityLibrary
- Easy, stable, simple to use library for showing OpenGL demos.
- Limited to simple windows, mouse/keyboard input, and some simple 3D shapes.
- Most OpenGL demos and code on the web use GLUT.
- Default implementation in C (bindings for many languages available: Python, Perl etc)

Example GLUT Program in C

```
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>

int main(int argc, char** argv)
{
    glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_DOUBLE);
    glutInitWindowSize(1024, 768);
    glutInitWindowPosition(0, 0);
    glutInit(argc, argv);

    glutCreateWindow("OpenGL_Demo");
    glutReshapeFunc(reshape);
    glutDisplayFunc(display);
    glutIdleFunc(display);
    glutKeyboardFunc(keyboard);
    glutMouseFunc(mouse);
}
```

Typical InitGL() Function

```
void InitGL(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClearDepth(1.0);
    glDepthFunc(GL_LEQUAL);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_COLOR_MATERIAL);
    glShadeModel(GL_SMOOTH);
    glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
    glEnable(GL_BLEND);

    glLightModel(GL_LIGHT_MODEL_AMBIENT, [0.5, 0.5, 0.5, 1.0]);
    glLightfv(GL_LIGHT0, GL_AMBIENT, (0.4, 0.4, 0.4, 1.0));
    glLightfv(GL_LIGHT0, GL_DIFFUSE, (0.4, 0.4, 0.4, 1.0));
    glLightfv(GL_LIGHT0, GL_POSITION, (0.0, 0.0, -100.0, 1.0));
    glEnable(GL_NORMALIZE);
    glEnable(GL_LIGHTING);
}
```

Typical GLUT Reshape Function

```
void reshape(int width, int height);
{
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(fovy, aspect, zNear, zFar);
}
```

Typical GLUT Display Function

```
void display(void)
{
    UpdateWorld();

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(fovy, aspect, zNear, zFar);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(0.0, 0.0, 150.0,
              0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

    RenderScene();
    glutSwapBuffers();
}
```

GLUT on Unix (command line)

Compiling a GLUT Program on Unix/Linux

```
gcc main.c -I/usr/X11R6/include -L/usr/X11R6/lib  
-lGL -lGLU -lglut -lm
```

On Unix or Linux systems includes should be:

```
#include <GL/gl.h>  
#include <GL/glu.h>  
#include <GL/glut.h>
```

GLUT on Mac OS X (command line)

Compiling a GLUT Program on Mac OS X

```
gcc main.c -F/System/Library/Frameworks  
-framework GLUT -framework OpenGL
```

Using frameworks on Mac OS X, includes should be:

```
#include <GLUT/glut.h>  
#include <OpenGL/gl.h>  
#include <OpenGL/glu.h>
```


Resources