Introduction to OpenGL

Graphics and Computation

Michael Papasimeon

2005, 2006, 2007

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`

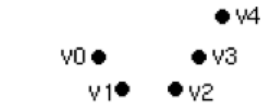
Drawing in OpenGL

- 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.

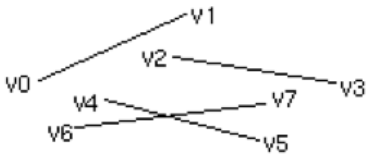
Drawing a Polygon

```
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();
```

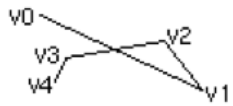
`glBegin` Drawing Modes



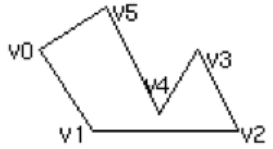
GL_POINTS



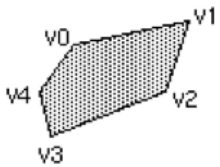
GL_LINES



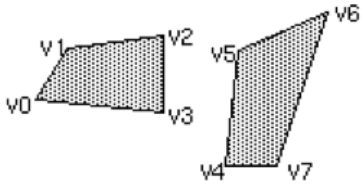
GL_LINE_STRIP



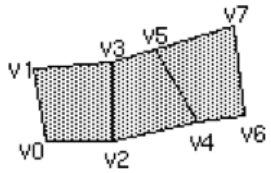
GL_LINE_LOOP



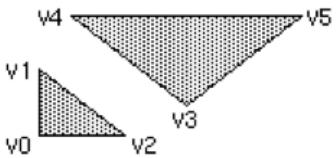
GL_POLYGON



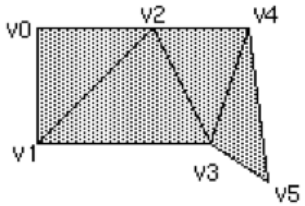
GL_QUADS



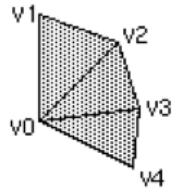
GL_QUAD_STRIP



GL_TRIANGLES



GL_TRIANGLE_STRIP



GL_TRIANGLE_FAN

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 $z = 0$.

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

[1] For 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);
```

- Specifiying the `Projection` matrix is like chosing 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)

Perspective Projection (gluPerspective) from GLU

Orthographics (Parallel) Projection - glOrtho

gluLookAt

Translation Transformation

Rotation Transformations

Scaling Transformations

Order of Transformations

Transformations in Action

The Matrix Stack

OpenGL Lighting

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 Default Parameters

Teapots Materials Example

Normal Vectors

Normal Vectors (2)

Hidden Surface Removal

GLUT

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();
```

JOGL

JOGL Example

JOpenGLDemoClass

JOGL Main Method

JOGL Init Method

JOGL Reshape (Window Resize) Method

JOGL Display Method

JOGL Demo

JOGL on Mac OS X

OpenGL Books

Resources

Project Preparation