Introduction to OpenGL ®

Graphics and Computation

Michael Papasimeon

12 March 2006
11 March 2007
11 March 2008

---

[1] OpenGL and the oval logo are trademarks or registered trademarks of Silicon Graphics, Inc. in the United States and/or other countries worldwide.

Notes

- These slides were presented to a third year computer science class in graphics and computation over a three year period (2006, 2007, 2008). The slides were the practical component introducing the students to OpenGL as part of a longer course in computer graphics. The introductory OpenGL component was taught over two lectures, but the students used OpenGL for the major project.
- There was some variation over the three years in which the material was presented, including one year (2007) where Java was used with JOGL. In 2006 and 2008 the course was taught in C using GLUT.
- The slides were originally done in PowerPoint and they have been convertex to LaTeXand Beamer.

OpenGL Version

Note that the OpenGL presented in these slides is pretty out of date. It represents early OpenGL 1.1-1.4 and does not include anything from OpenGL 2.0 or greater, so this material should **not** be used learn modern OpenGL is primarily provided here for historical reference.

Outline

- OpenGL Background and History
- Other Graphics Technologies
- Drawing
- Viewing and Transformation
- Lighting
- GLUT
- 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();
```
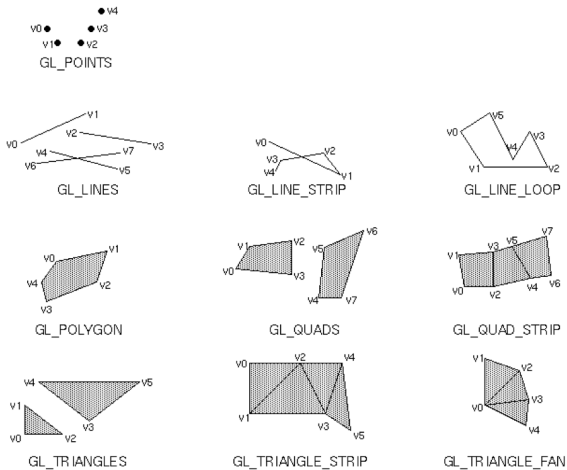
Diagram: Copyright OpenGL Programming Guide. Addison-Wesley.

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();
```
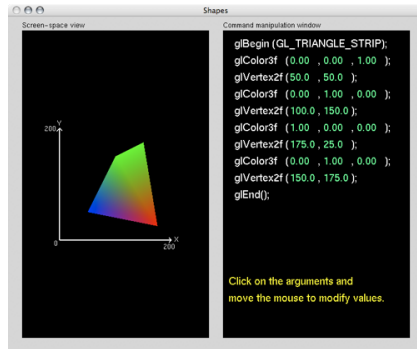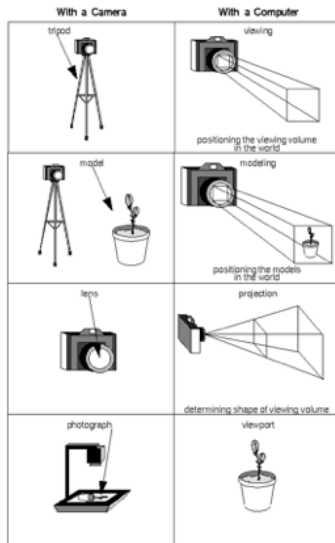
Diagram: Copyright OpenGL Programming Guide. Addison-Wesley.

OpenGL Vertices

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

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

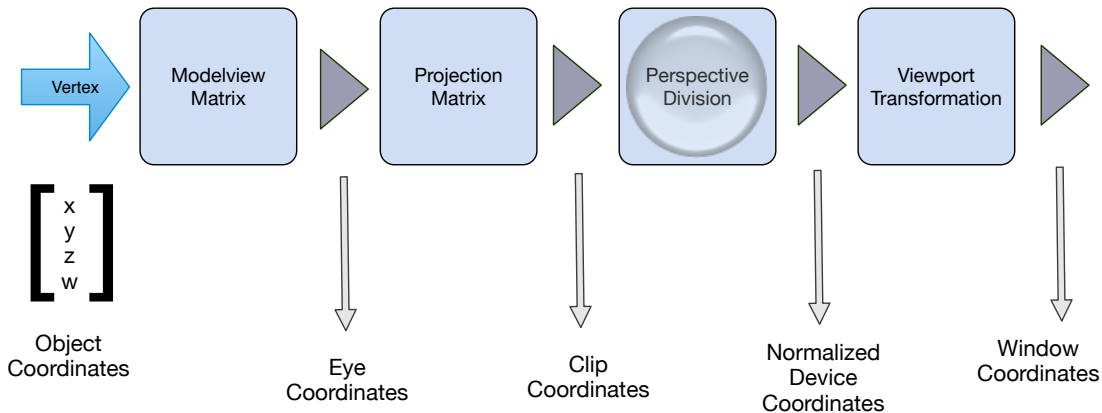[2]For further information on homogenous coordinate systems as used in projective geometry and in OpenGL, see Appendix G of the Red Book `http://www.rush3d.com/reference/opengl-redbook-1.1/appendixg.html`

Vertex Transformations

Vertex → Modelview Matrix → Projection Matrix → Perspective Division → Viewport Transformation

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Object Coordinates

Eye Coordinates

Clip Coordinates

Normalized Device Coordinates

Window Coordinates

Diagram adapted from the OpenGL Programming Guide. Addison-Wesley.

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)

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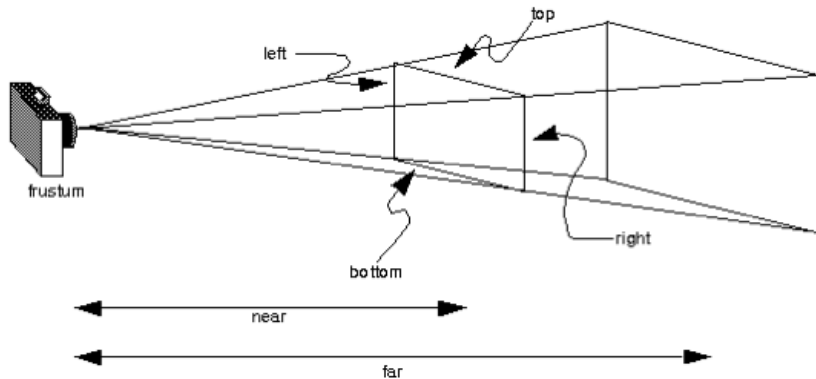

Diagram: Copyright OpenGL Programming Guide. Addison-Wesley.

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)
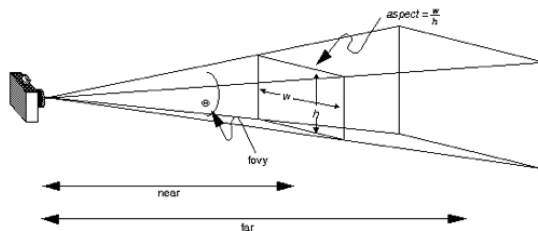
Diagram: Copyright OpenGL Programming Guide. Addison-Wesley.

Orthographics (Parallel) Projection - glOrtho

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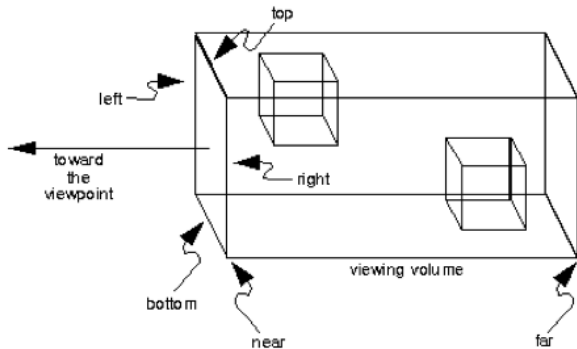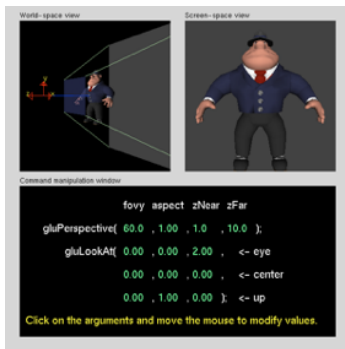

Diagram: Copyright OpenGL Programming Guide. Addison-Wesley.

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);
```
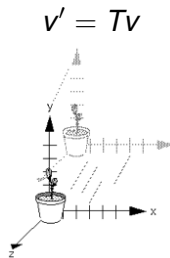
Translation Transformation

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

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

$$v' = Tv$$

Rotation Transformations

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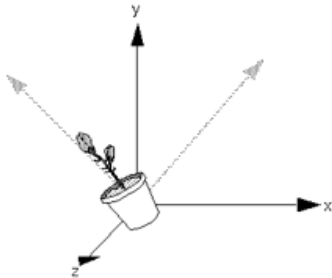
Specifies an angle and an axis (x,y,z) to rotate around.

Diagram: Copyright OpenGL Programming Guide. Addison-Wesley.

Rotation Matrices

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

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

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

Scaling Transformations

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

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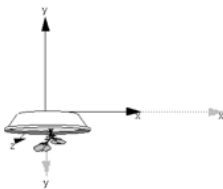
$$v' = Sv$$

Diagram: Copyright OpenGL Programming Guide. Addison-Wesley.

Diagram: Copyright OpenGL Programming Guide. Addison-Wesley.

Transformations in Action
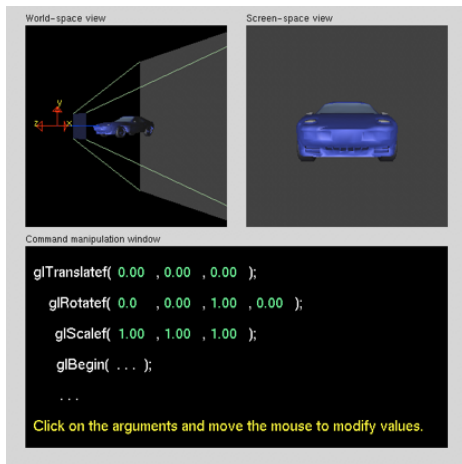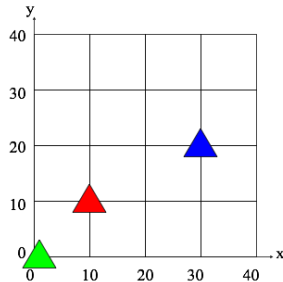

Diagram: Copyright OpenGL Programming Guide. Addison-Wesley.

The Matrix Stack

glPushMatrix() and glPopMatrix()

```
glPushMatrix();
glTranslatef(10, 10, 0);
DrawRedTriangle();
  glPushMatrix();
  glTranslatef(20, 10, 0);
  DrawBlueTriangle();
  glPopMatrix();
glPopMatrix();
DrawGreenTriangle();
```

OpenGL Lighting

- Eight available lights (individually toggled)
- Lighting types:
  - Emitted
  - Ambient
  - Diffues
  - 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)

Parameter Name	Default Value	Meaning
GL_AMBIENT	(0.0, 0.0, 0.0, 1.0)	ambient RGBA intensity of light
GL_DIFFUSE	(1.0, 1.0, 1.0, 1.0)	diffuse RGBA intensity of light
GL_SPECULAR	(1.0, 1.0, 1.0, 1.0)	specular RGBA intensity of light
GL_POSITION	(0.0, 0.0, 1.0, 0.0)	(x, y, z, w) position of light
GL_SPOT_DIRECTION	(0.0, 0.0, -1.0)	(x, y, z) direction of spotlight
GL_SPOT_EXPONENT	0.0	spotlight exponent
GL_SPOT_CUTOFF	180.0	spotlight cutoff angle
GL_CONSTANT_ATTENUATION	1.0	constant attenuation factor
GL_LINEAR_ATTENUATION	0.0	linear attenuation factor
GL_QUADRATIC_ATTENUATION	0.0	quadratic attenuation factor

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);
```

glMaterial Default Parameters

| Parameter Name	Default Value	Meaning
GL_AMBIENT	(0.2, 0.2, 0.2, 1.0)	ambient color of material
GL_DIFFUSE	(0.8, 0.8, 0.8, 1.0)	diffuse color of material
GL_AMBIENT_AND_DIFFUSE		ambient and diffuse color of material
GL_SPECULAR	(0.0, 0.0, 0.0, 1.0)	specular color of material
GL_SHININESS	0.0	specular exponent
GL_EMISSION	(0.0, 0.0, 0.0, 1.0)	emissive color of material
GL_COLOR_INDEXES	(0,1,1)	ambient, diffuse, and specular color indices

Diagram: Copyright OpenGL Programming Guide. Addison-Wesley.

Normal Vectors
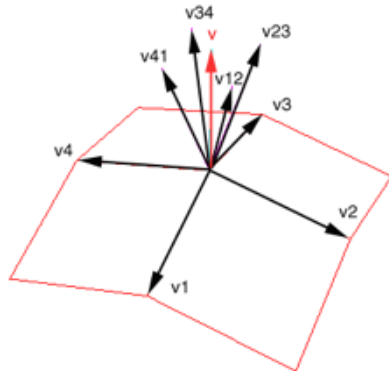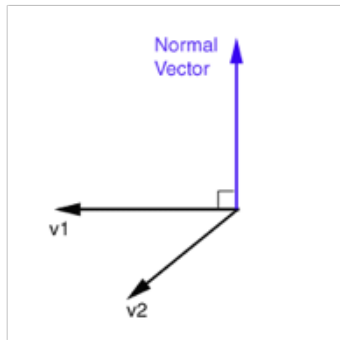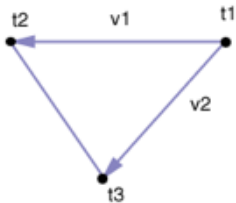
Normal Vectors – `glNormal()`

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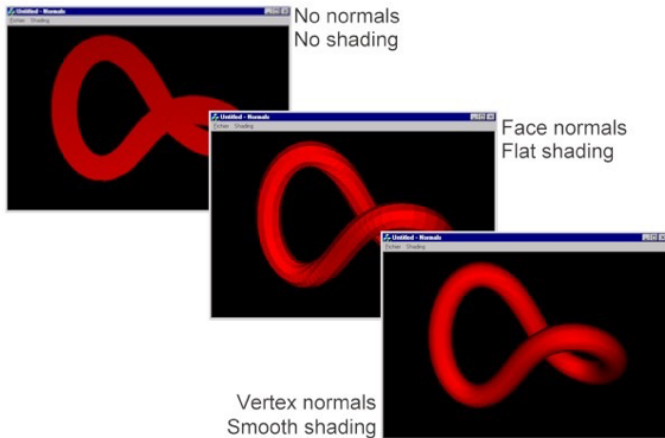
Shading

No normals
No shading

Face normals
Flat shading

Vertex normals
Smooth shading

Diagram from: http://www.codeguru.com/Cpp/G-M/opengl/article.php/c2681/

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

- 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 th eweb use GLUT.
- Default implementation in C (bindings for many languages available: Python, Perl etc)

Example GLUT Program in C

```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);
    glutMotionFunc(motion);
    InitGL();
    glutMainLoop();
}
```

Typical InitGL() Function

```c
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);
    glEnable(GL_LIGHT0);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(fovy, aspect, zNear, zFar);
    glMatrixMode(GL_MODELVIEW);
}
```

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)
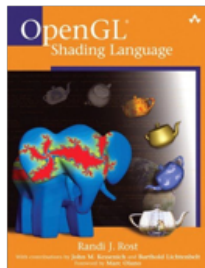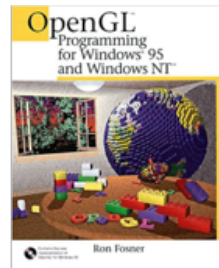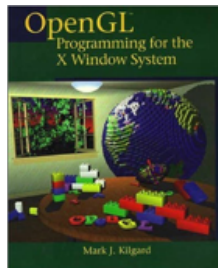
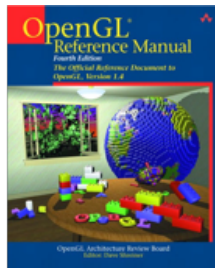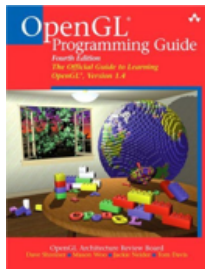## Compiling a GLUT Program on Mac OS X

```
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>
```

OpenGL Books

Resources

- OpenGL Home Page `http://www.opengl.org`
- OpenGL Tutors `http://www.xmission.com/~nate/tutors.html`
- NeHe Tutorials: `http://nehe.gamedev.net`
- OpenGL Red Book (Programming Manual) `http://www.glprogramming.com/red/`
- OpenGL Blue Book (Reference Manual)`http://www.glprogramming.com/blue/`
- Using GLUT with Visual C++ Express Edition `http://www.cs.rit.edu/~wrc/courses/common/cg1/doc/visual/GLUT_with_VCppEE.pdf`