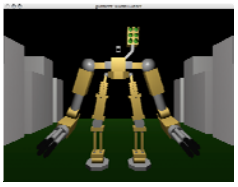
COMP30019 Graphics and Interaction

Introduction to OpenGL

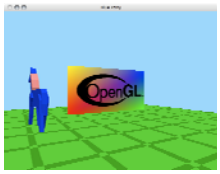
Outline

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

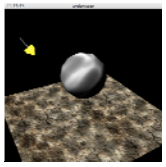


OpenGL Background and History

- 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 Technology



- Low Level Graphics
- OpenGL
- Scene Graphs, BSPs
  - OpenSceneGraph, Java3D, VRML, PLIB
- DirectX (Direct3D)
- Can mix some DirectX with OpenGL (e.g OpenGL and DirectInput in Quake III)

Platform Specifics

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

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 will clear the colour and depth buffers.

```
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], with each value being in the range of 0.0 to 1.0.
- There are many variants of the glColor command

```
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

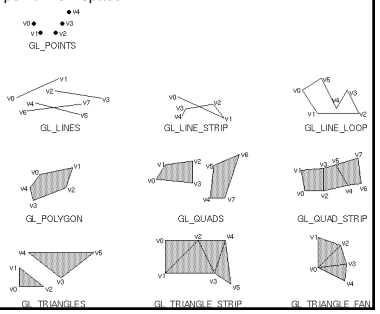
```
glFlush();
```

- For more information see Chapter of the Red Book:
  http://fly.cc.fer.hr/~unreal/theredbook/chapter02.html
  or
  http://www.gamedev.net/download/redbook.pdf

Drawing in OpenGL

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

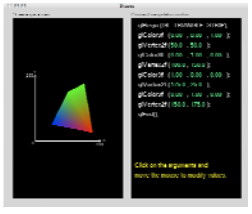
```
glBegin(GL_POLYGON);
glVertex3f(0.0, 0.0, 0.0);
glVertex3f(0.0, 3.0, 1.0);
glVertex3f(3.0, 3.0, 3.0);
glVertex3f(4.0, 1.5, 1.0);
glVertex3f(3.0, 0.0, 0.0);
glEnd();
```



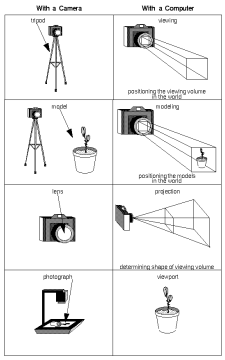
Mixing Geometry with Colour

- 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

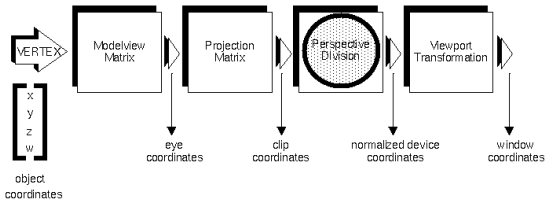
$$v = \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

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

- For further information on homogenous coordinate systems as used in projective geometry and in OpenGL, see Appendix G of the Red Book http://fly.cc.fer.hr/~unreal/theredbook/appendixg.html

13

Vertex Transformation



$$\mathbf{M} = \begin{bmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m_{12} & m_{16} \end{bmatrix} \qquad v' = Mv$$

14

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)

15

The Projection Matrix

- `glMatrixMode(GL_PROJECTION);`
- Specifying the Projection matrix is like choosing a lens for a camera.
- It lets you specify field of view and other parameters.

16

OpenGL Matrix Operations

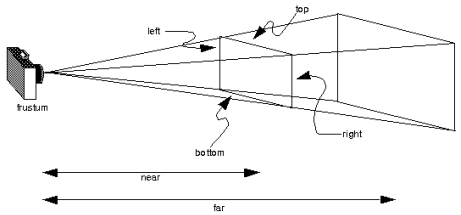
- `glMatrixMode(`*mode*`);`
- `glLoadIdentity();`
- `glMultMatrix();`
- `glLoadMatrix();`

Identity Matrix

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

17

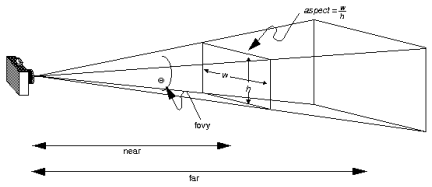
Perspective Projection (glFrustrum)




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

18

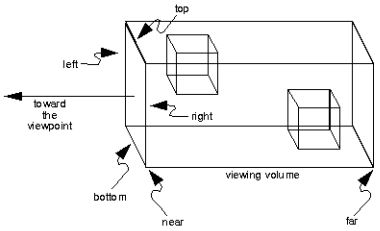
Perspective Projection (gluPerspective) from GLU




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

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

Orthographic (Parallel) Projection (glOrtho)



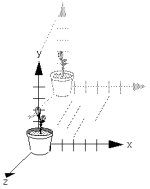

```
glOrtho(left, right, bottom, top, near, far);
```


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);
```

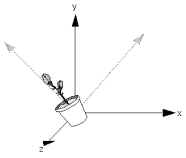
Translation Transformation



$$T = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and } T^{-1} = \begin{bmatrix} 1 & 0 & 0 & -x \\ 0 & 1 & 0 & -y \\ 0 & 0 & 1 & -z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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


Rotation Transformations



$$\text{glRotate*}(\alpha, 1, 0, 0): \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

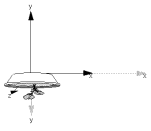
$$\text{glRotate*}(\alpha, 0, 1, 0): \begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{glRotate*}(\alpha, 0, 0, 1): \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

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

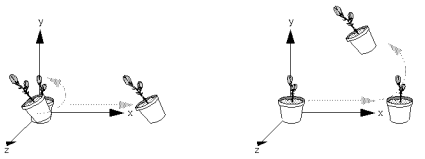
Scaling Transformations



$$S = \begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and } S^{-1} = \begin{bmatrix} \frac{1}{x} & 0 & 0 & 0 \\ 0 & \frac{1}{y} & 0 & 0 \\ 0 & 0 & \frac{1}{z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

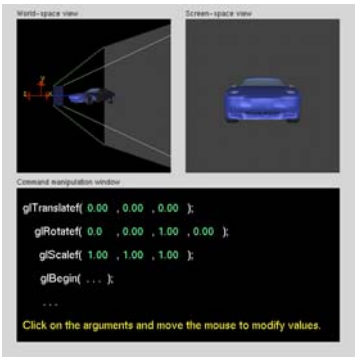
```
glScale(x,y,z);
```

Order of Transformations



25

Transformations in Action

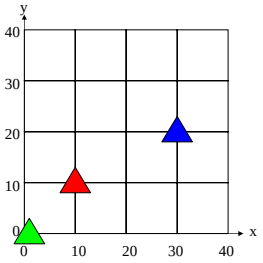


26

The Matrix Stack

- glPushMatrix() and glPopMatrix()

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



27

OpenGL Lighting

- Eight available lights (individually toggled)
- Lighting types
  - Emitted
  - Ambient
  - Diffuse
  - Specular
- Material Colours and Properties
- Lighting Demo

28

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);
```

29

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

Table 6-1 : Default Values for pname Parameter of glLight*()

30

Material Properties

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

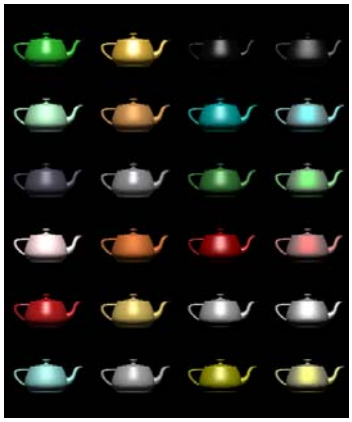
```
GLfloat material[] = {0.1, 0.5, 0.8, 1.0 };
glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE,
                                material);

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);
```

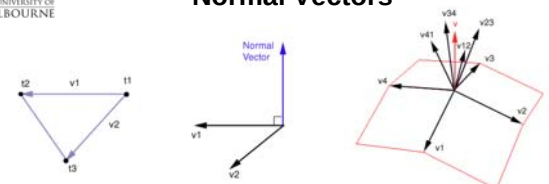

glMaterial Default Parameters

Parameter Name	Default Value	Meaning
GL_AMBIENT	(0.2, 0.2, 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

Table 6-2 : Default Values for pname Parameter of glMaterial*()

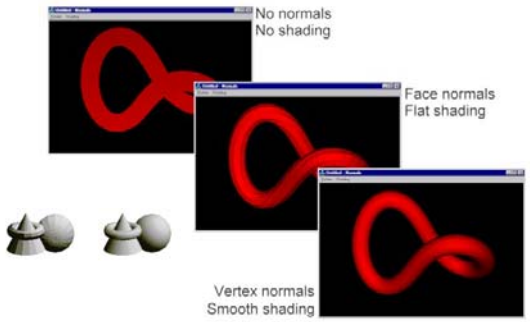


Normal Vectors




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


Normal Vectors (2)



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 Utility Library
- 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);
   glutMotionFunc(motion);
   InitGL();

   glutMainLoop();
}
```


Typical InitGL() Function

```
void InitGL(void)
{
    float ambientLight[] = {0.5, 0.5, 0.5, 0.0};
    float diffuseLight[] = {0.5, 0.0, 0.5, 0.0};
    float lightPosition[] = {1.0, 1.0, 1.0, 0.0};

    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);

    glLightfv(GL_LIGHT0, GL_AMBIENT,  ambientLight);
    glLightfv(GL_LIGHT0, GL_DIFFUSE,  diffuseLight);
    glLightfv(GL_LIGHT0, GL_POSITION, lightPosition);
    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 (cmd line)

```
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 (cmd line)

```
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: 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
  (see PDF File)

Project Preparation

- Read Chapters 1-6 of OpenGL Red Book
- Familiarise yourself with OpenGL Blue Book
- Play with OpenGL Tutors
- Learn about GLUT
- Do NeHe Tutorial Lessons 1-5 (with GLUT)