Introduction to OpenGL

EE4702 Fall 2010

Topics

- What is OpenGL
- OpenGL name conventions
- Outlook of an OpenGL program
- Window Management
- Drawing 3D Objects with GLUT
- Important OpenGL operations
- Modeling Scenes
- Mathematics in OpenGL
- OpenGL as a state machine
- OpenGL rendering pipeline

What is OpenGL?

A Standard, hardware-independent interface to Graphics hardware

- **Introduced in 1992**
- **Most widely used 3D graphics API**
- **Portable across a wide array of platforms**

Current version: OpenGL 3.0

- **Older versions: 1.'s and 2.'s**

No commands for windows management

- **Does not create window**
- **Does not take user input (such as mouse click)**

What Is OpenGL?

Provides a powerful but primitive set of rendering commands

- **Points, lines and polygons**

No high-level rendering commands

- **Ultimate control over modeling 3D objects**
- **Assembler language of computer graphics**

Foundations for high-performance graphics

- **Many APIs built on the top of OpenGL**

What Is OpenGL?

GL routine has a prefix gl

- **glColor()**

Head file for GL-library calls

- **#include <GL/gl.h>**

Software information and download

- **http://www.opengl.org**

OpenGL Name Conventions

OpenGL functions

- **Prefix gl and initial capital letters for each word making up the function name**

glVertex() glClearColor()

OpenGL defined constants

- **Begin with GL_, use all capital letters, and use underscore to separate words**

GL_COLOR_BUFFER_BIY GL_TRIANGLES

OpenGL Name Conventions

Suffixes in functions

- **void glVertex{234}{sifd}[v](TYPE coords)**

  2 or 3 or 4 means the # of arguments to be given
  s or i or f or d means date type
  v means a pointer to a vector or array of three values

- **glVertex3f(2.0, 4.0, 1.0);**

  Three floating-point numbers for three arguments

- **Glfloat dvect[3]={2.0, 4.0, 1.0};**

  glVertex3fv(dvect);

  Representation of three arguments by a vector *dvect*

OpenGL Related Libraries

OpenGL Utility Library: GLU

- **Routines for special tasks**

  Matrices for viewing orientations and projections

  Polygon tessellation

  Surfaces Rendering

- **Prefix glu**

#include<GL/glu.h>

OpenGL Related Libraries

OpenGL Utility Toolkit: GLUT

- **Window-system independent**
- **Prefix glut**
- **#include <GL/glut.h>**

Window management

- **Creating window and handling input events**

Modeling 3D objects

- **High-level drawing commands built on top of OpenGL**

Outlook of a OpenGL program

Draws a red sphere in a white window

```
#include <GL/glut.h>
void display (void)
{
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0);
    glutSolidSphere(0.4, 50, 40);
    glFlush();
}
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("A red sphere in a white window");
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Window Management

Initializing and Creating a Window

- **void glutInit(int *argc, char **argv);**

  Initializes the GLUT

  Appears before any other GLUT routine

- **void glutInitDisplay(unsigned int mode);**

  Specifies a display mode(color mode or buffer)

  A double-buffered and RGBA color mode window:

  glutInitDisplay(GLUT_DOUBLE|GLUT_RGBA);

Window Management

void glutInitWindowPosition(int *x*, int *y*);

- **Specifies the location of the upper-left corner of the window**

void glutInitWindowSize(int *width*,int *height*);

- **Specifies window's size in pixels**

void glutCreateWindow(char* name);

- **Opens window with previously set characteristics(display mode, size, etc)**
- **Window is not displayed until glutMainLoop() is called**

Window Management

Handling window and input events

- **Callback functions to specify specific events, e.g. mouse click, keyboard input**
- **Register these functions before entering the main loop**

void glutDisplayFunc(void (*func)(void));

- **Specifies the function that is called whenever the contents of the window need to be redrawn**

Window Management

void glutMouseFunc(void (*func)(int *button*, int *state*, int *x*, int *y*));

- **Specifies the function, *func*, that's called when a mouse button is pressed or released**

void glutMotionFunc(void (*func)(int *x*, int *y*));

- **Specifies the function, *func*, that's called when the mouse pointer moves with the mouse button being pressed**

Window Management

void glutKeyboardFunc(void (*func)(unsigned int *key*, int *x*, int *y*));

- **Specifies the function, *func*, that's called when a key is pressed**

void glutReshapeFunc(void (*func)(int width, int height));

- **Specifies the function that's called whenever the window is resized or moved**
- ***Func* restablishes the rectangular region as a new rendering canvas and adjust coordinate system**

Window Management

Managing a background process

void glutIdleFunc(void (*func)(void));

- **Specifies the function, *func*, to be executed if no other events are pending**
- **If NULL(zero) is passed in, execution of the function is disabled**

void glutPostRedisplayFunc(void);

- **Marks the current window as needing to be redrawn**
- **At the next opportunity, the callback function registered by glutDisplayFunc() is called**

Window Management

Running the program

- **GLUT program enters an“event-processing loop”**

void glutMainLoop(void);

- **Enters the GLUT processing loop, never returns**
- **Registered callback functions will be called when the corresponding events occur**

Drawing 3D Objects with GLUT

GLUT has many high-level drawing routines

Two flavors of model

- **Wireframe without surface normal**

  void glutWireCube(Gldouble *size*);

  void glutWireSphere(Gldouble *radius*, Glint *slices*, Glint *stacks*);

- **Solid with shading and surface normal**

  void glutSolidCube(Gldouble *size*);

  void glutSolidSphere(Gldouble *radius*, Glint *slices*, Glint *stacks*);

- **Other exaples**

  torus, icosahedron, octahedron, cone, teapot

Important OpenGL Operations

Clearing the window

- **Clear the color buffer filled by the last picture before drawing**

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

Specifying a color

- **Set the color to red (RGB mode) before any drawing**

```
glColor3f(1.0, 0.0, 0.0);
```

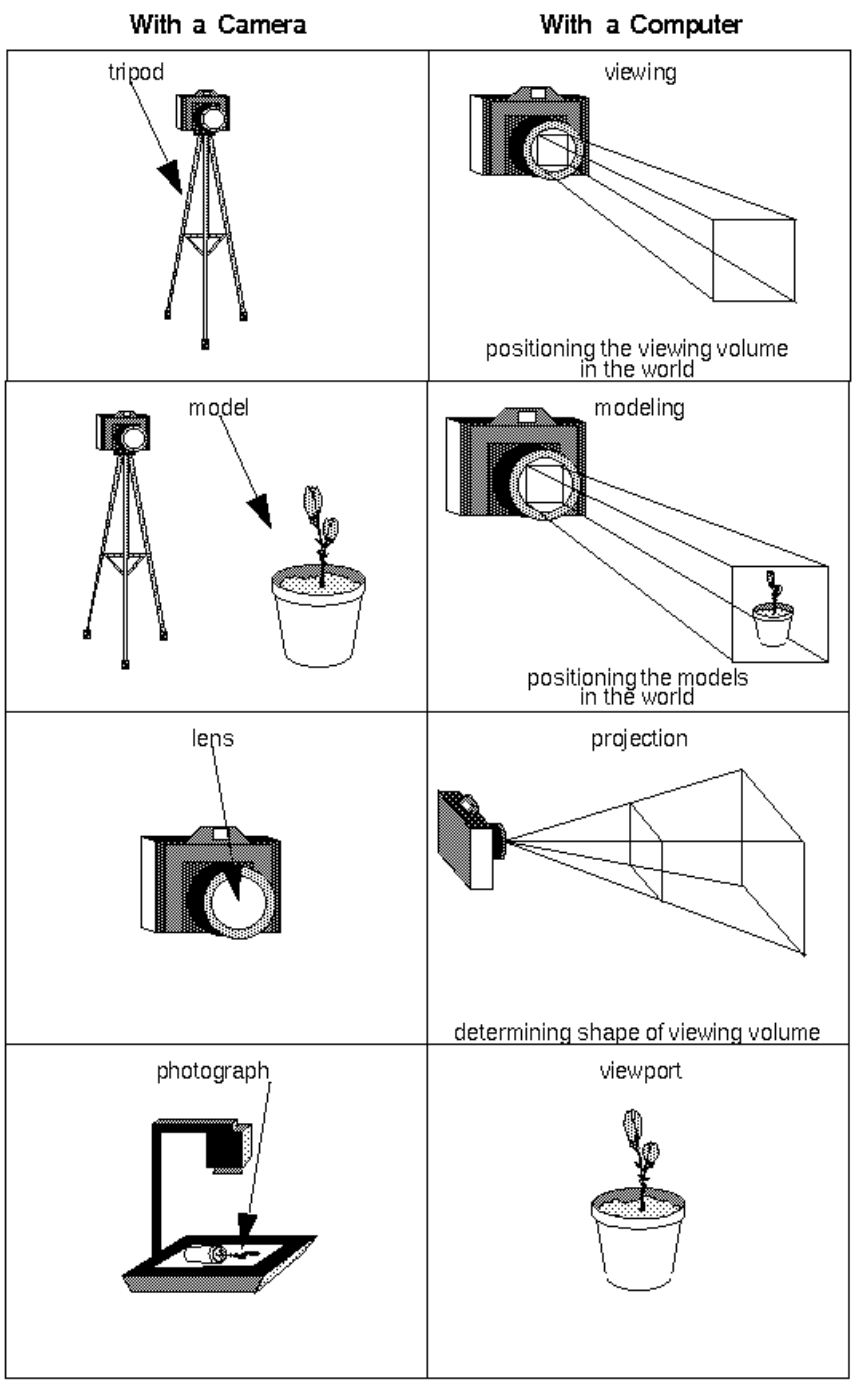
Forcing completion of drawing

- **Force previous commands to begin execution**

```
void glFlush(void);
```

- **Particularly useful in client-server framework**

Modeling


With a Camera
With a Computer
tripod
viewing
positioning the viewing volume in the world
model
modeling
positioning the models in the world
lens
projection
determining shape of viewing volume
photograph
viewport

Modeling

Take real pictures VS“Electronic Pictures”

- **Set up tripod and point at your camera at your scene**
- **Arrange the scene into a desired composition**
- **Choose a lens or adjust zoom**
- **Determine how large you want the final photo to be**

- **Viewing transformation**
- **Modeling transformation**
- **Projection transformation**
- **Viewport transformation**

Mathematics in OpenGL

Homogeneous Coordinates

- **A point (x, y, z) in R3 could be denoted as a 4x1 vector**

  v = (x, y, z, w), in most cases w=1

  details will be covered later.

- **Transformation (translation, rotation, scaling, etc) is denoted as matrix multiplication**

  v'= Mv

  M is a 4x4 matrix, called the transformation matrix

Mathematics in OpenGL

Translation

(x,y,z)->(x+tx, y+ty, z+tz)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

resulting coordinate — 3d translation matrix — original coordinate

Mathematics in O

Rotation

Arbitrary rotation matrix is the concatenation of three rotation matices

Note:

Since matrix multiplication is not commutative, the order of rotation can not be exchanged.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

resulting coordinate — 3d rotation matrix in Z — original coordinate

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

resulting coordinate — 3d rotation matrix in X — original coordinate

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

resulting coordinate — 3d rotation matrix in Y — original coordinate

Mathematics in OpenGL

Scaling

(x, y, z)->

(sx*x, sy*y, sz*z)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

resulting coordinate — 3d scaling matrix — original coordinate

Mathematics in OpenGL

Modeling Transformation

- **void glTranslatef(float x, float y, float z);**
- **void glRotatef(float angle, float x, float y, float z);**
- **void glScalef(float x, float y, float z);**
- **Your own matrix:**

```
float m[]={…};
glMultMatrixf(m)
```

Mathematics in OpenGL

Viewing Transformation

- **void gluLookAt(Gldouble *eyeX*, Gldouble *eyeY*, Gldouble *eyeZ*, Gldouble *centerX*, Gldouble *centerY*, Gldouble *centerZ*, Gldouble *upX*, Gldouble *upY*, Gldouble *upZ*);**

defines a line of sight

encapsulates a series of rotation and translation

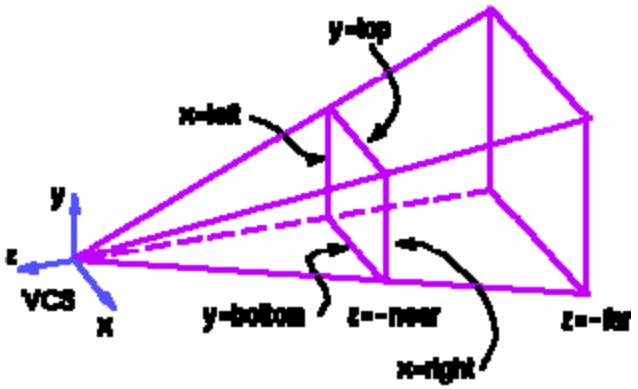
Same effect can be achieved by glTranslate*(), glRotate*(), glScale*()

Mathematics in OpenGL

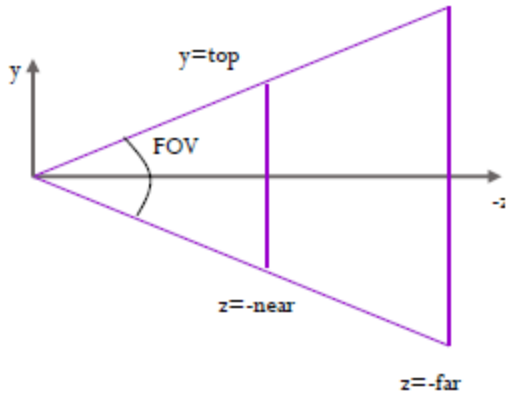
Projection Transformation

➢ Perspective Projection

- void glFrustum(double left, double right, double bottom, double top, double near, double far);



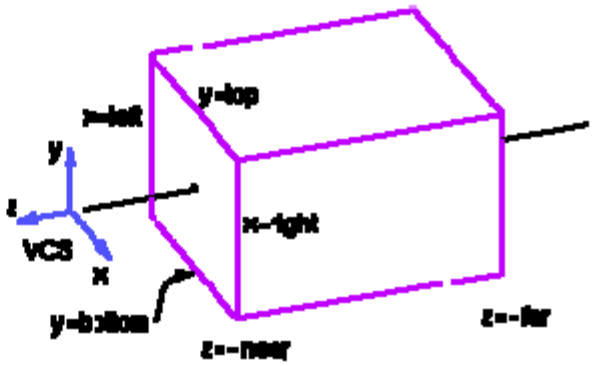
- void gluPerspective(double fovy, double aspect, double near, double far);



Mathematics in OpenGL

Orthographic Projection

- void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble zNear, GLdouble zFar);
- void gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top);



OpenGL as a state machine

Can be put into various states (modes) that remain in effect until they are changed

- **Current color**
- **Current viewing and projection transformations**
- **Position and characteristics of light sources**

State variables are queryable

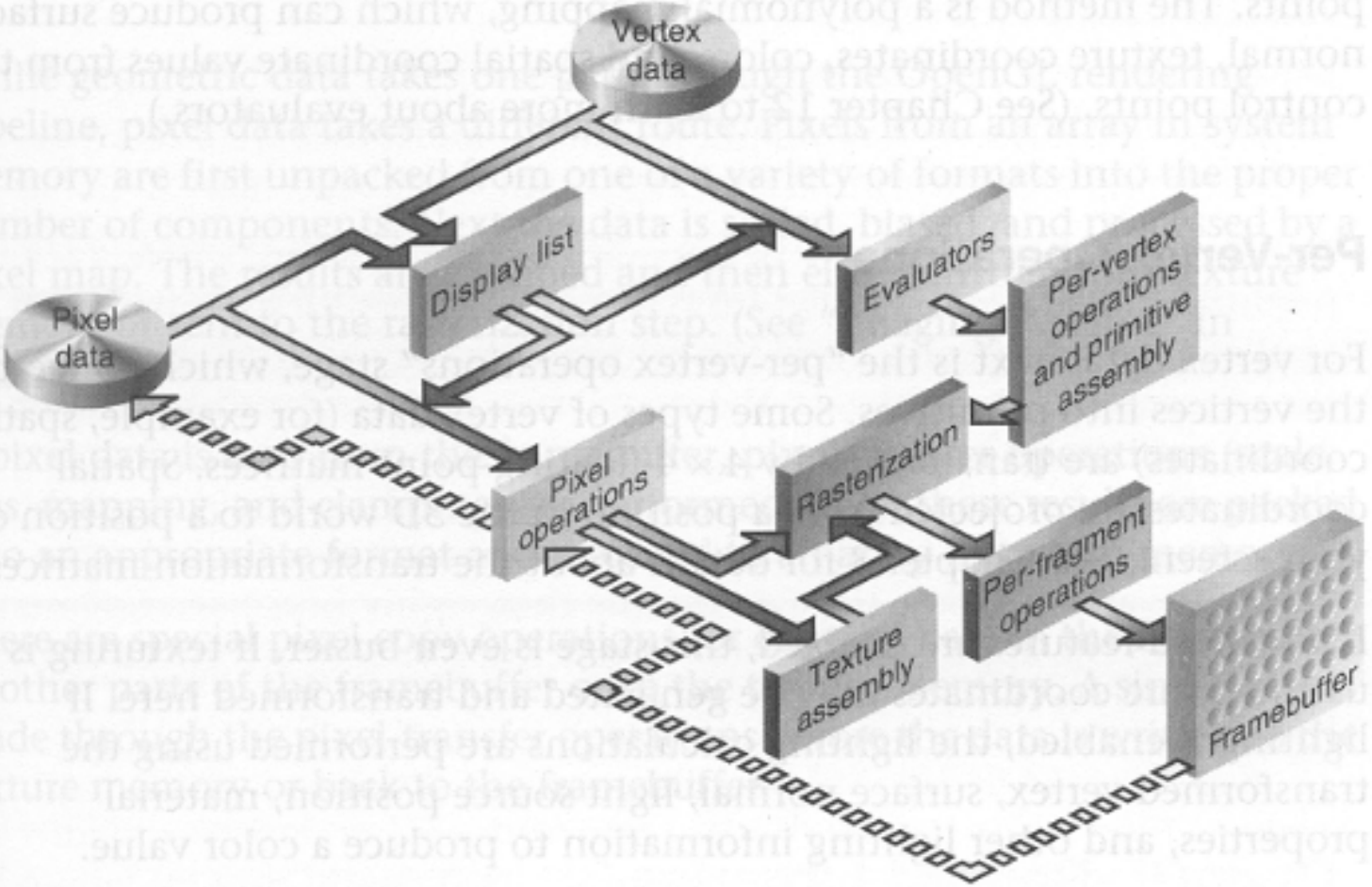
```
glGetFloatv(GL_CURRENT_COLOR, params);
```

By default, these states either have some values or are inactive

Many states can be turned on and off with

```
glEnable() and glDisable()
```

OpenGL Rendering Pipeline



Resources

Reference books(not required):

- **OpenGL Programming Guide (the Red Book)**

http://www.glprogramming.com/red/

- **OpenGL SuperBible: Comprehensive Tutorial and Reference (the Blue book)**
- **OpenGL : A Primer**

Online Tutorials

- **Nate Robin**

http://www.xmission.com/~nate/opengl.html

- **NeHe**

http://nehe.gamedev.net/

- **Jérôme JOUVIE**

http://jerome.jouvie.free.fr/OpenGl/Tutorials1-5.php