

---

# CIS 454 Computer Graphics

## Lecture 4, 09/14/2006

---

Li Shen  
Computer and Information Science  
UMass Dartmouth

---

## Notes

- HW1
    - Learning portal
  - TA
-

## Models and Architectures

- Pipeline architecture
- Programmer's interface: API

## Practical Approach

- Process objects one at a time in the order they are generated by the application
  - Can consider only local lighting
- Pipeline architecture



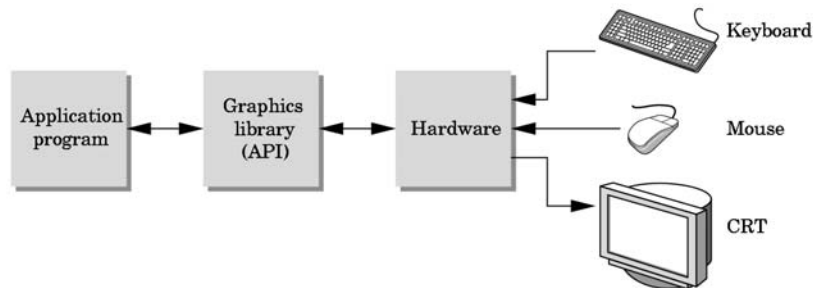
application  
program

display

- All steps can be implemented in hardware on the graphics card

## The Programmer's Interface

- Programmer sees the graphics system through a software interface: the Application Programmer Interface (API)



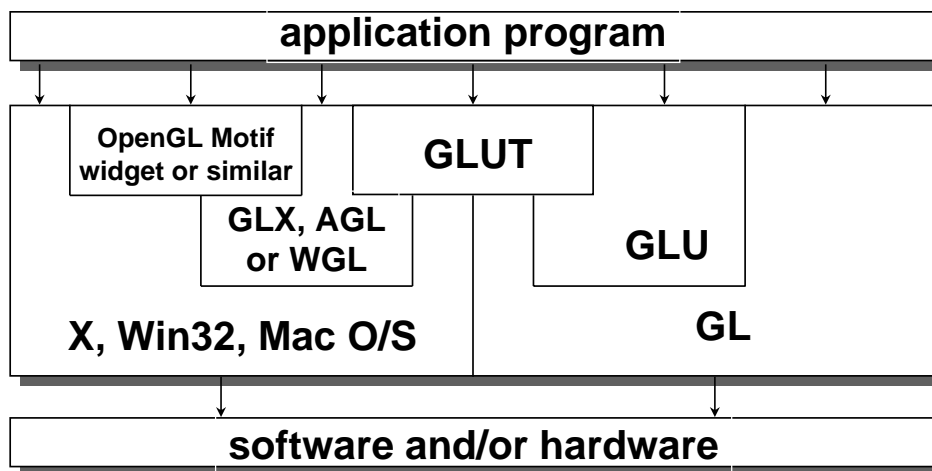
## API Contents

- Functions that specify what we need to form an image
  - Objects
  - Viewer
  - Light Source(s)
  - Materials
- Other information
  - Input from devices such as mouse and keyboard
  - Capabilities of system

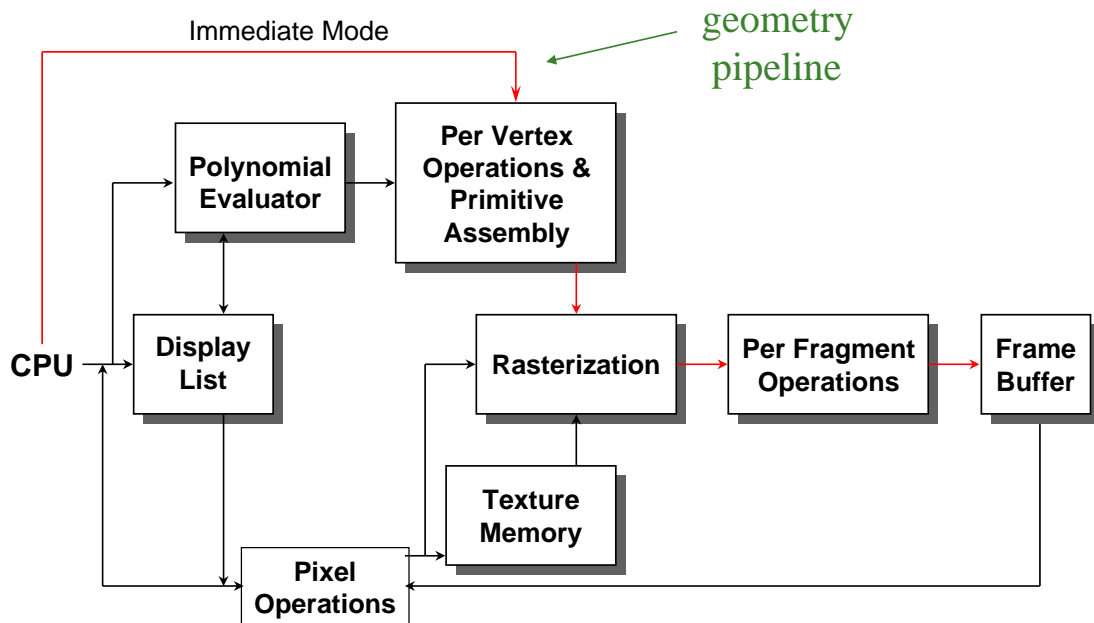
# Programming with OpenGL

## Part 1: Background

## Software Organization



# OpenGL Architecture



<http://windowssdk.msdn.microsoft.com/en-us/library/ms537479.aspx>

# OpenGL Functions

- Primitives
  - Points
  - Line Segments
  - Polygons
- Attributes
- Transformations
  - Viewing
  - Modeling
- Control (GLUT)
- Input (GLUT)
- Query

---

## simple.c

```
#include <GL/glut.h>
void mydisplay(){
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
        glVertex2f(-0.5, -0.5);
        glVertex2f(-0.5, 0.5);
        glVertex2f(0.5, 0.5);
        glVertex2f(0.5, -0.5);
    glEnd();
    glFlush();
}
int main(int argc, char** argv){
    glutCreateWindow("simple");
    glutDisplayFunc(mydisplay);
    glutMainLoop();
}
```

The OpenGL  
Reference Manual  
<http://www.rush3d.com/reference/opengl-bluebook-1.0/>

GLUT  
<http://www.opengl.org/resources/libraries/glut/spec3/spec3.html>

---

# Programming with OpenGL

## Part 2: Complete Programs

---

---

## Objectives

- Refine the first program
    - Alter the default values
    - Introduce a standard program structure
  - Simple viewing
    - Two-dimensional viewing as a special case of three-dimensional viewing
  - Fundamental OpenGL primitives
  - Attributes
- 

---

## Program Structure

- Most OpenGL programs have a similar structure that consists of the following functions
    - **main()**:
      - defines the callback functions
      - opens one or more windows with the required properties
      - enters event loop (last executable statement)
    - **init()**: sets the state variables
      - Viewing
      - Attributes
    - callbacks
      - Display function
      - Input and window functions
-

## simple.c revisited

- In this version, we shall see the same output but we have defined all the relevant state values through function calls using the default values
- In particular, we set
  - Colors
  - Viewing conditions
  - Window properties

## main.c

OpenGL <http://www.rush3d.com/reference/opengl-bluebook-1.0/>  
GLUT <http://www.opengl.org/resources/libraries/glut/spec3/spec3.html>

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

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("simple");
    glutDisplayFunc(mydisplay);

    init();

    glutMainLoop();
}
```

includes **gl.h**

define window properties

display callback

set OpenGL state

enter event loop

# GLUT functions

- `glutInit` allows application to get command line arguments and initializes system
- `glutInitDisplayMode` requests properties for the window (the *rendering context*)
  - RGB color
  - Single buffering
  - Properties logically OR'ed together
- `glutWindowSize` in pixels
- `glutWindowPosition` from top-left corner of display
- `glutCreateWindow` create window with title "simple"
- `glutDisplayFunc` display callback
- `glutMainLoop` enter infinite event loop

## init.c

OpenGL <http://www.rush3d.com/reference/opengl-bluebook-1.0/>  
GLUT <http://www.opengl.org/resources/libraries/glut/spec3/spec3.html>

```
void init()
{
    glClearColor (0.0, 0.0, 0.0, 1.0);
    glColor3f(1.0, 1.0, 1.0);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
}
```

black clear color

opaque window

fill/draw with white

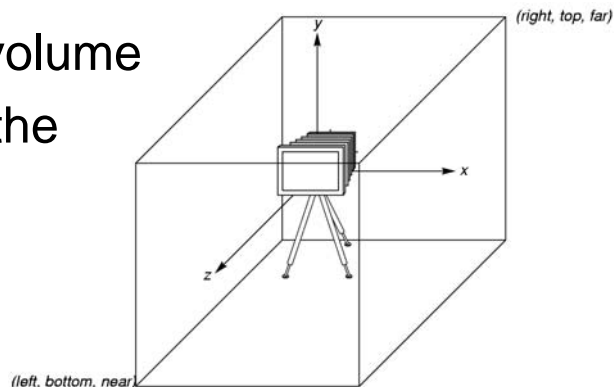
viewing volume

## Coordinate Systems

- The units in `glVertex` are determined by the application and are called *object* or *problem coordinates*
- The viewing specifications are also in object coordinates and it is the size of the viewing volume that determines what will appear in the image
- Internally, OpenGL will convert to *camera (eye) coordinates* and later to *screen coordinates*
- OpenGL also uses some internal representations that usually are not visible to the application

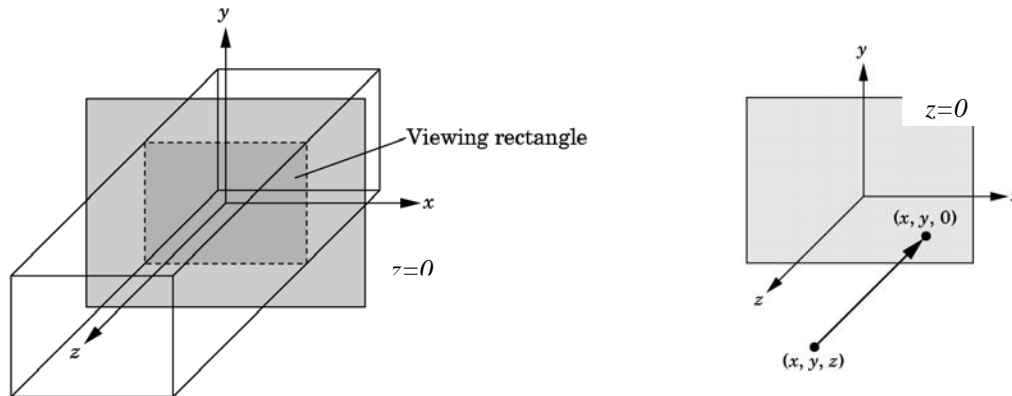
## OpenGL Camera

- OpenGL places a camera at the origin in object space pointing in the negative  $z$  direction
- The default viewing volume is a box centered at the origin with a side of length 2



## Orthographic Viewing

In the default orthographic view, points are projected forward along the  $z$  axis onto the plane  $z=0$



## Transformations and Viewing

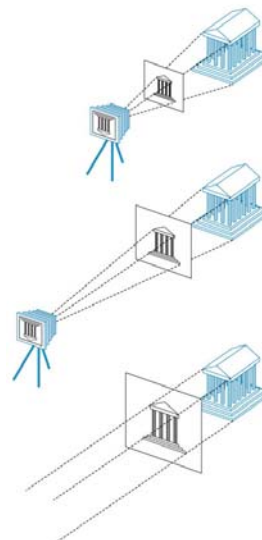
- In OpenGL, projection is carried out by a projection matrix (transformation)
- There is only one set of transformation functions so we must set the matrix mode first
- Transformation functions are incremental so we start with an identity matrix and alter it with a projection matrix that gives the view volume

```
glMatrixMode(GL_PROJECTION);  
glLoadIdentity();  
glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
```

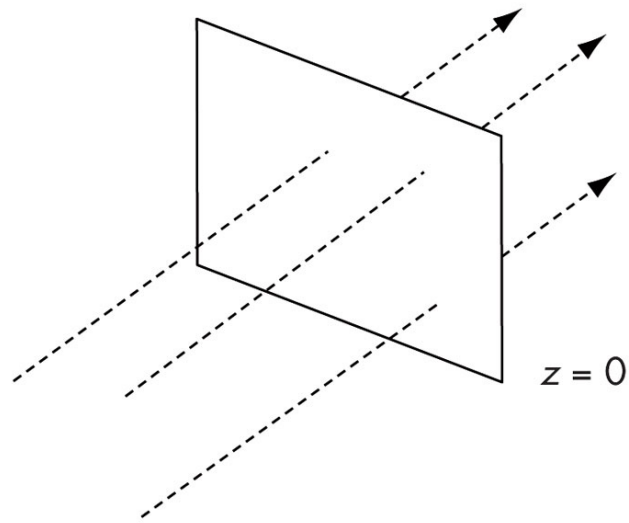
## Two- and Three-dimensional Viewing

- In `glOrtho(left, right, bottom, top, near, far)` the near and far distances are measured from the camera
- Two-dimensional vertex commands place all vertices in the plane  $z=0$
- If the application is in two dimensions, we can use the function  
`gluOrtho2D(left, right, bottom, top)`
- In two dimensions, the view or clipping volume becomes a *clipping window*

**FIGURE 2.27** Creating an orthographic view by moving the camera away from the projection plane.



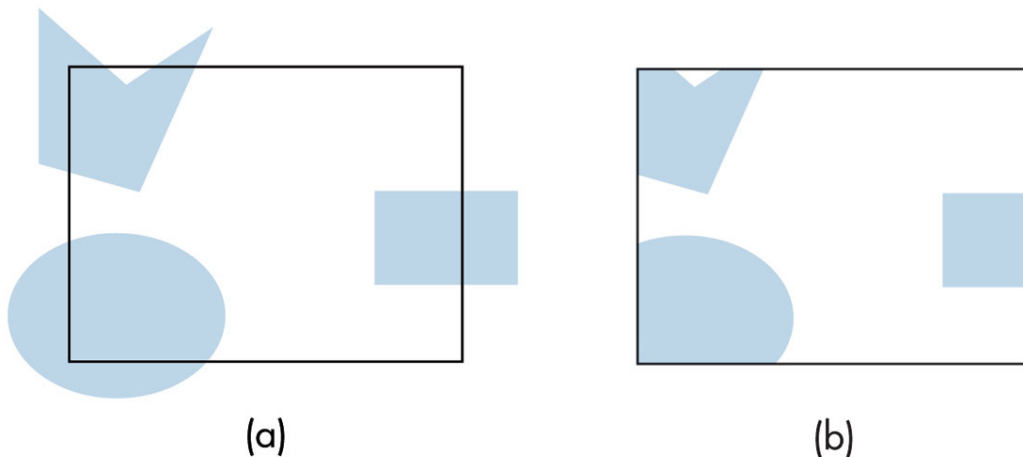
**FIGURE 2.28** Orthographic projectors with projection plane  $z = 0$ .



**FIGURE 2.32** Two-dimensional viewing.

(a) Objects before clipping.

(b) Image after clipping.



---

## After Class

- Read Chapter 2
  - Work on HW1
-