Introduction to OpenGL

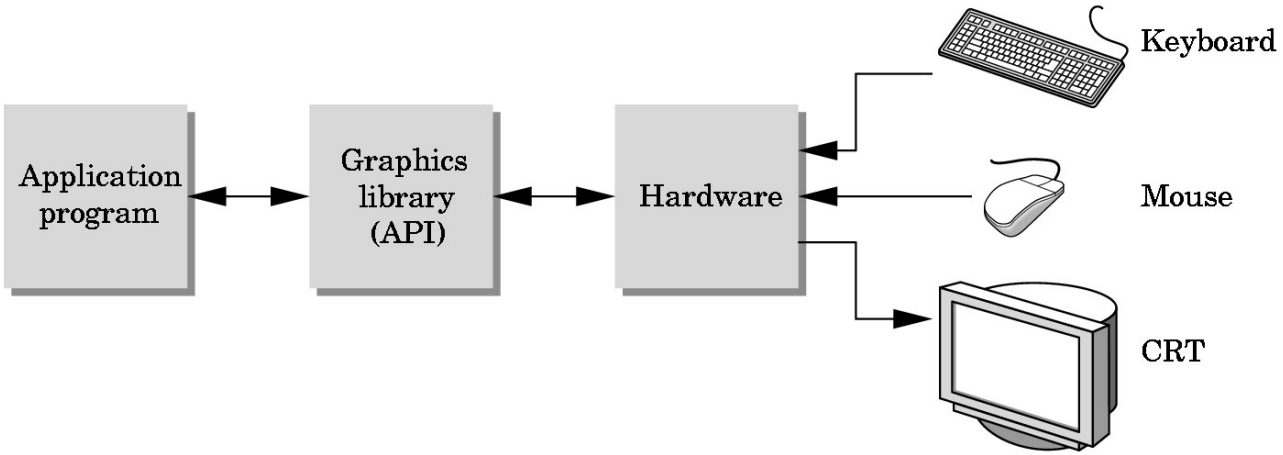
Prof.Dr. E.A.Zanaty
Professor

Acknowledgements

- Most of the material for the slides were adapted from
  - *E. Angel, "Interactive Computer Graphics", 4th edition*
- Some of the slides were taken from
  - *CISC 440/640 Computer Graphics (Spring 2005)*
- Some of the images were taken from
  - *F.S.Hill, "Computer Graphics using OpenGL"*
- Other resources
  - http://www.lighthouse3d.com/opengl/glut/
  - *Jackie Neider, Tom Davis, and Mason Woo, "The OpenGL Programming Guide" (The Red Book)*

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

History of OpenGL

- Silicon Graphics (SGI) revolutionized the graphics workstation by implementing the pipeline in hardware (1982)
- To access the system, application programmers used a library called GL
- With GL, it was relatively simple to program three dimensional interactive applications

OpenGL: What is It?

- The success of GL lead to OpenGL (1992), a platform-independent API that was
  - Easy to use
  - Close enough to the hardware to get excellent performance
  - Focus on rendering
  - Omitted windowing and input to avoid window system dependencies

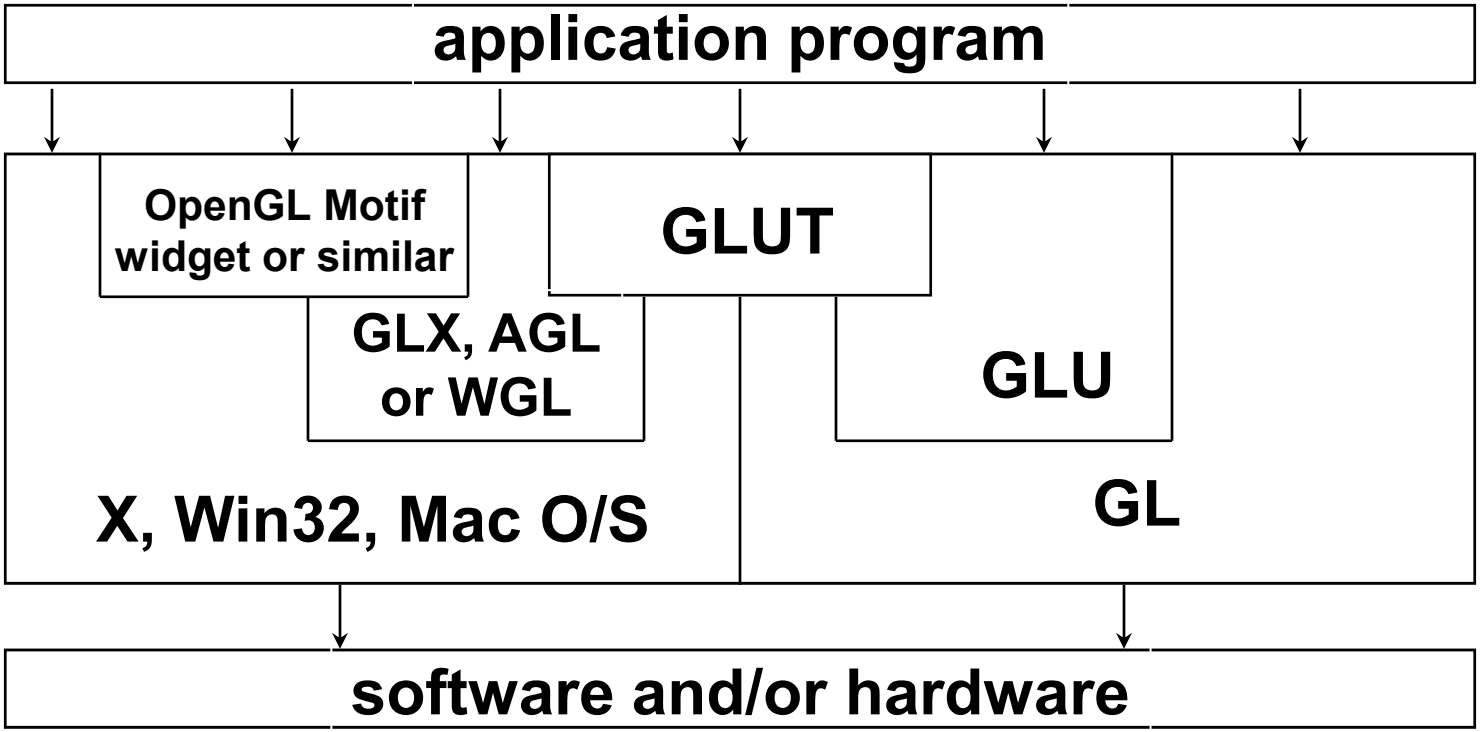
OpenGL Evolution

- Controlled by an Architectural Review Board (ARB)
  - Members include SGI, Microsoft, Nvidia, HP, 3DLabs, IBM,…….
  - Relatively stable (present version 2.0)
    - Evolution reflects new hardware capabilities
      - **3D texture mapping and texture objects**
      - **Vertex programs**
  - Allows for platform specific features through extensions

OpenGL Libraries

- **GL** (**G**raphics **L**ibrary): Library of 2-D, 3-D drawing primitives and operations
  - API for 3-D hardware acceleration
- **GLU** (**GL U**tilities): Miscellaneous functions dealing with camera set-up and higher-level shape descriptions
- **GLUT** (**GL U**tility **T**oolkit): Window-system independent toolkit with numerous utility functions, mostly dealing with user interface

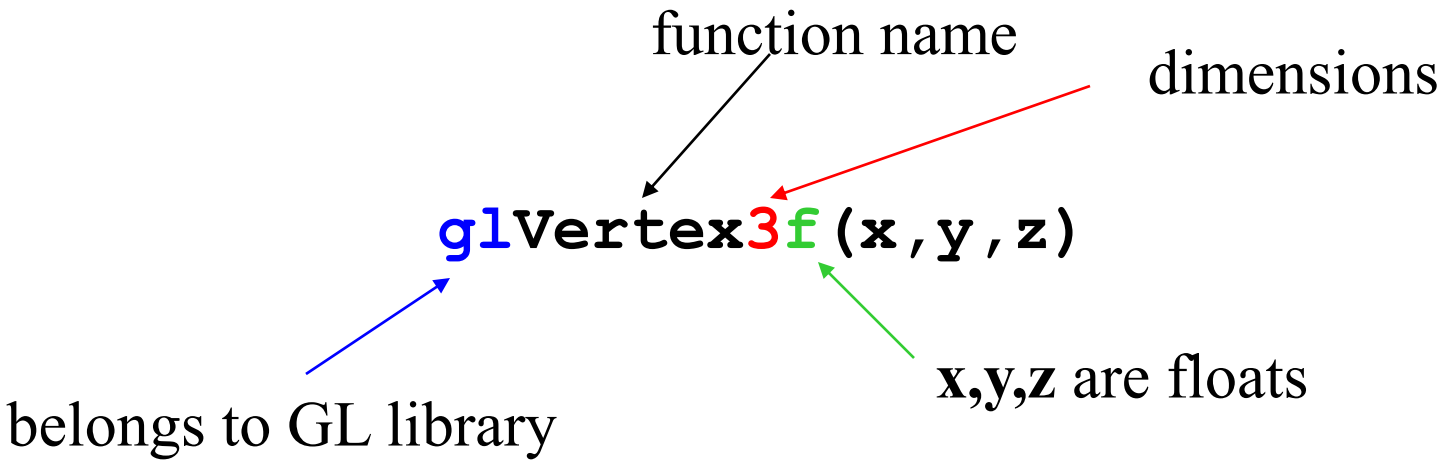
Software Organization



Lack of Object Orientation

- OpenGL is not object oriented so that there are multiple functions for a given logical function
  - `glVertex3f`
  - `glVertex2i`
  - `glVertex3dv`
- Underlying storage mode is the same
- Easy to create overloaded functions in C++ but issue is efficiency

OpenGL function format




```
glVertex3fv(p)
```

p is a pointer to an array

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();
}
```

Event Loop

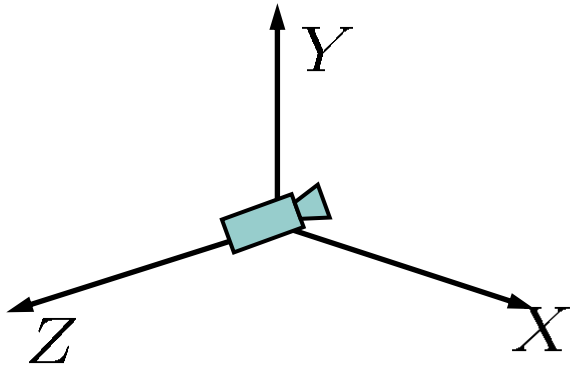
- Note that the program defines a *display callback* function named **`mydisplay`**
  - Every glut program must have a display callback
  - The display callback is executed whenever OpenGL decides the display must be refreshed, for example when the window is opened
  - The **`main`** function ends with the program entering an event loop

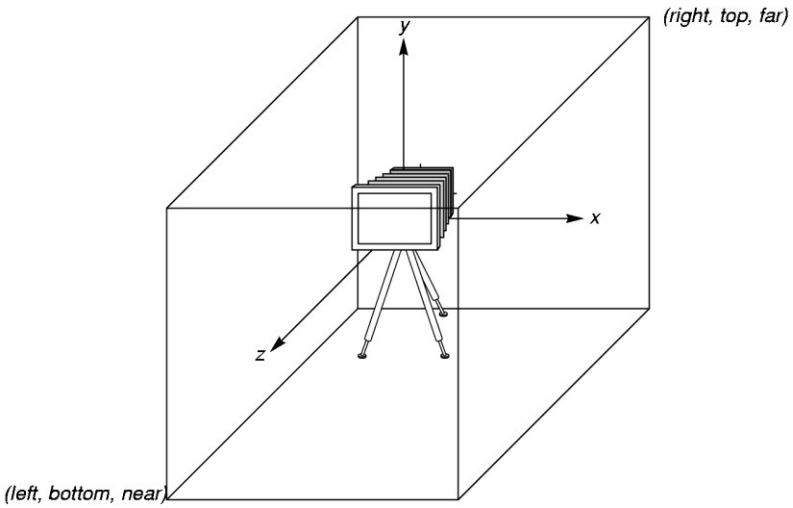
Default parameters

- `simple.c` is too simple
- Makes heavy use of state variable default values for
  - Viewing
  - Colors
  - Window parameters

OpenGL Camera

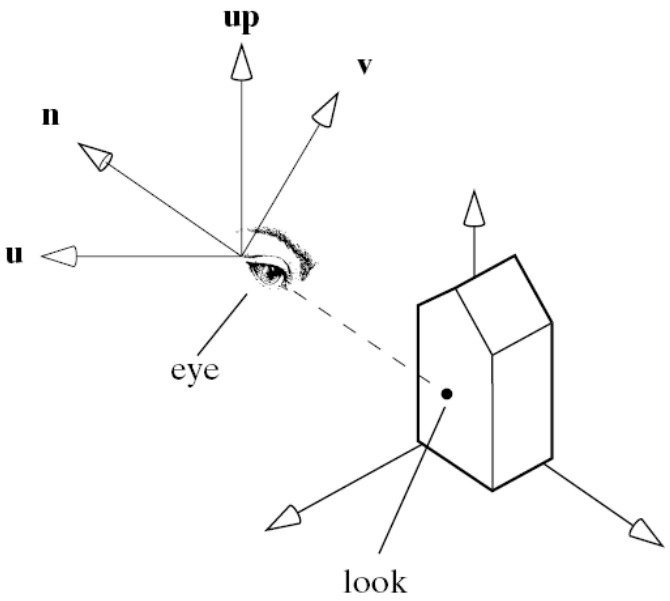
- Right-handed system
- From point of view of camera looking out into scene:
  - OpenGL places a camera at the origin in object space pointing in the negative z direction
- Positive rotations are counterclockwise around axis of rotation





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*



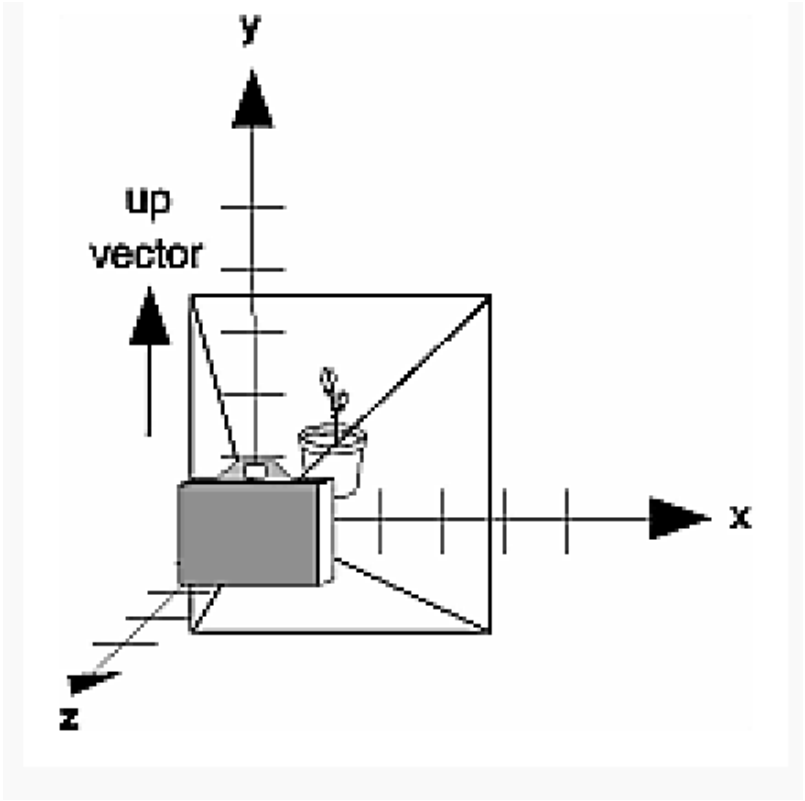
Transformations in OpenGl

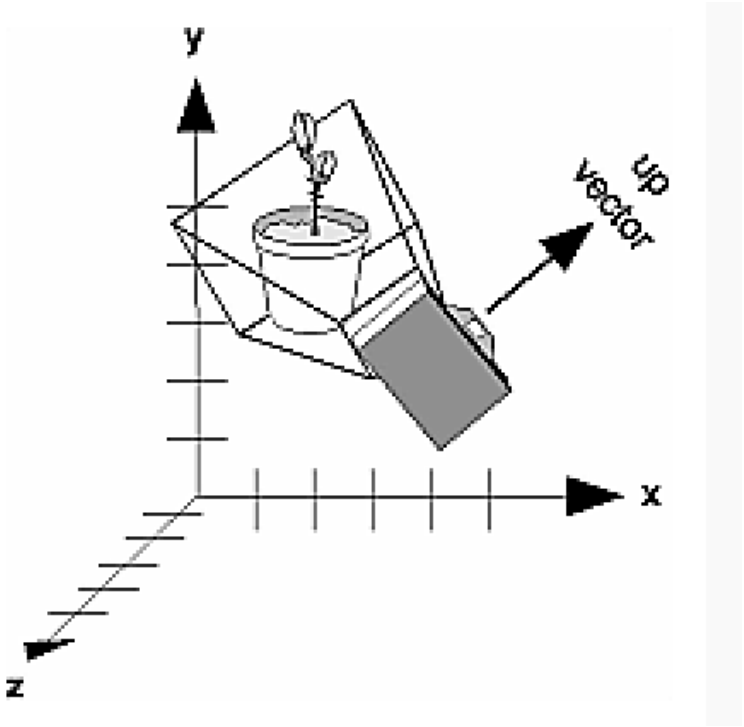
- Modeling transformation
  - Refer to the transformation of models (i.e., the scenes, or objects)
- Viewing transformation
  - Refer to the transformation on the camera
- Projection transformation
  - Refer to the transformation from scene to image

Model/View Transformations

- Model-view transformations are usually visualized as a single entity
  - Before applying modeling or viewing transformations, need to set glMatrixMode(GL_MODELVIEW)
  - Modeling transforms the object
    - Translation: glTranslate(x,y,z)
    - Scale: glScale(sx,sy,sz)
    - Rotation: glRotate(theta, x,y,z)
  - Viewing transfers the object into camera coordinates
    - gluLookAt (eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ)

Model/View transformation



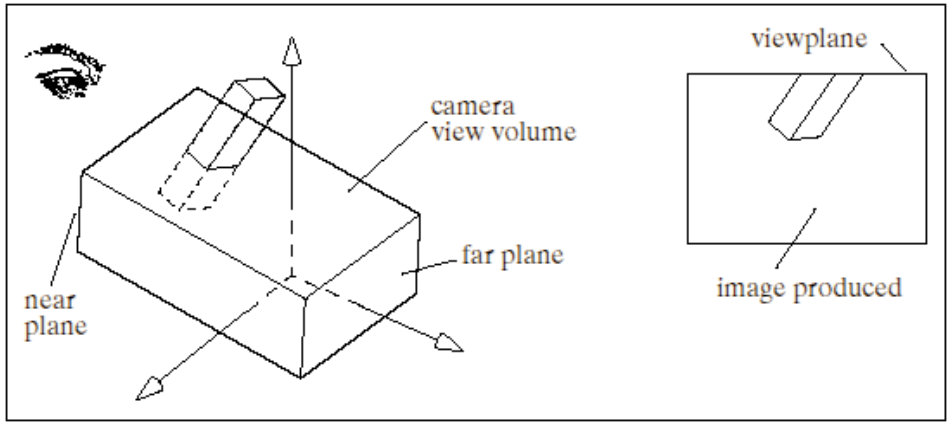


Courtesy: Neider, Davis and Woo, "The OpenGL Programming Guide"

Projection Transformation

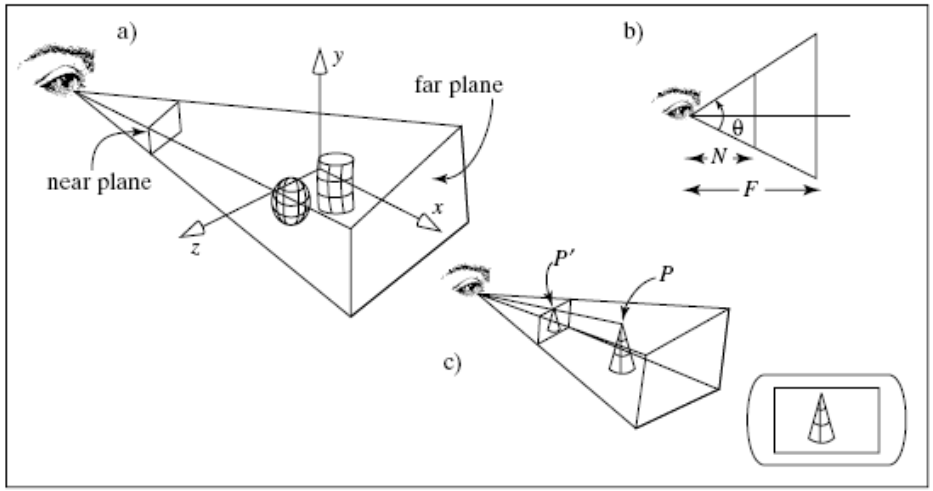
- Transformation of the 3D scene into the 2D rendered image plane
  - Before applying projection transformations, need to set glMatrixMode(GL_PROJECTION)
  - Orthographic projection
    - glOrtho(left, right, bottom, top, near, far)
  - Perspective projection
    - glFrustum (left, right, bottom, top, near, far)

Projection Transformation



Orthographic projection

Perspective projection



F.S.Hill, "Computer Graphics using OpenGL"

Program Structure

- Most OpenGL programs have the following structure
  - **`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

```
#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
- **`gluInitDisplayMode`** requests properties for the window (the *rendering context*)
  - RGB color
  - Single buffering
  - Properties logically ORed 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

Window Initialization

```
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

Display callback function

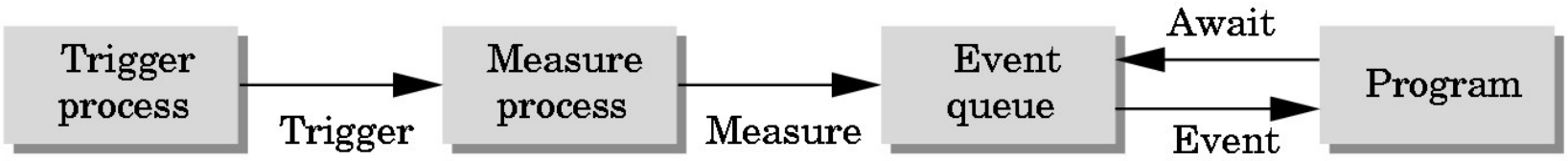
```
void mydisplay()
{
   glClear(GL_COLOR_BUFFER_BIT);

   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();
}
```

Input and Interaction

- Multiple input devices, each of which can send a trigger to the operating system at an arbitrary time by a user
  - Button on mouse
  - Pressing or releasing a key
- Each trigger generates an *event* whose measure is put in an *event queue* which can be examined by the user program



Callbacks

- Programming interface for event-driven input
- Define a *callback function* for each type of event the graphics system recognizes
- This user-supplied function is executed when the event occurs
  - GLUT example:
    `glutMouseFunc(mymouse)`

GLUT event loop

- Last line in **`main.c`** for a program using GLUT is the infinite event loop

  `glutMainLoop();`
- In each pass through the event loop, GLUT
  - looks at the events in the queue
  - for each event in the queue, GLUT executes the appropriate callback function if one is defined
  - if no callback is defined for the event, the event is ignored
- In **`main.c`**
  - **`glutDisplayFunc(mydisplay)`** identifies the function to be executed
  - Every GLUT program must have a display callback

Posting redisplays

- Many events may invoke the display callback function
  - Can lead to multiple executions of the display callback on a single pass through the event loop
- We can avoid this problem by instead using

```
glutPostRedisplay();
```

  which sets a flag.
- GLUT checks to see if the flag is set at the end of the event loop
  - If set then the display callback function is executed

Double Buffering

- Instead of one color buffer, we use two
  - **Front Buffer**: one that is displayed but not written to
  - **Back Buffer**: one that is written to but not displayed
- Program then requests a double buffer in main.c
  - `glutInitDisplayMode(GL_RGB | GL_DOUBLE)`
  - At the end of the display callback buffers are swapped

```
void mydisplay()
{
   glClear(GL_COLOR_BUFFER_BIT|….)
.
/* draw graphics here */
.
   glutSwapBuffers()
}
```

Using the idle callback

- The idle callback is executed whenever there are no events in the event queue
  - **`glutIdleFunc(myidle)`**
  - Useful for animations

```
void myidle() {
/* change something */
   t += dt
   glutPostRedisplay();
}

Void mydisplay() {
   glClear();
/* draw something that depends on t */
   glutSwapBuffers();
}
```

Using globals

- The form of all GLUT callbacks is fixed
  - void **`mydisplay()`**
  - void **`mymouse(GLint button, GLint state, GLint x, GLint y)`**
- Must use globals to pass information to callbacks

```
float t; /*global */

void mydisplay()
{
/* draw something that depends on t
}
```

Other important functions

- glPushMatrix() / glPopMatrix()
  - Pushes/pops the transformation matrix onto the matrix stack
- glLoadIdentity(), glLoadMatrix(), glMultMatrix()
  - Pushes the matrix onto the matrix stack
- Chapter 3 of the “Red Book” gives a detailed explanation of transformations
  - *Jackie Neider, Tom Davis, and Mason Woo, “The OpenGL Programming Guide” (The Red Book)*

Assignment policy

- How to submit
- What to submit
- On late submission

How to submit

- Submit as a tar/zip file
  - Unix:

    > tar -cf username_projectNum_(440|640).tar projectDir

    > gzip username_projectNum_(440|640).tar
  - Windows:
    - Use a zip utility
- Naming convention
  - username_projectNum_(440|640).(tar.gz|zip)
- Submit the tar/zip file through the course web (More details will be announced later)

What to submit

- Must contain
  - Readme
  - Makefile
  - Source codes
  - Output figures (if any)
- Must NOT contain
  - obj intermediate files
  - obj data files

What to submit: Readme

% My name
% My email: myemail@udel.edu
% Project Num

% Part 1: description of this project
This project is to apply xxx algorithm to plot xxx, …

% Part 2: what I did and what I didn't do
I completed all/most/some functionalities required in this project.
The system is robust and the rendering is fairly efficient, …

I didn't do …. The reason is ….

% Part 3: What files contained

% Part 4: How to compile and how to run
The project is developed in windows system and tested in stimpy (strauss) unix system

On late submission

- N * 10 percent of the points you got will be deducted if there are N (<=5) late days (not counting weekends).
- No acceptance for the submission more than 5-day late
- Each student has three free (i.e. without any penalty) late days for entire semester.
  - You should notify the TA the use of free late days ahead

OpenGL: Setup in Unix

Steps to compile the code on Strauss

1. run following command
2. setenv LD_LIBRARY_PATH /home/base/usrb/chandrak/640/OpenGL/Mesa-2.6/lib:/usr/openwin/lib:/opt/gcc/lib (*This is present as a comment in the Makefile*)
3. download Makefile and hello.c
4. compile and run hello.c:

   strauss> gmake -f Makefile_composor
5. run your code (Use ./hello if path not set properly)

   strauss> hello

Steps to compile the code on stimpy

1. run following command
2. setenv LD_LIBRARY_PATH /usr/local/mesa/lib:/usr/openwin/lib
3. download Makefile_stimpy and hello.c
4. compile and run hello.c:

   stimpy> gmake -f Makefile_stimpy
5. run your code (Use ./hello if path not set properly)

   stimpy> hello

OpenGL: Setup in Windows

- Go to the GLUT webpage
  - http://www.opengl.org/resources/libraries/glut.html
- From the bottom of the page, download the following
  - Pre-compiled Win32 for Intel GLUT 3.7 DLLs for Windows 95 & NT
- Follow the instructions in
  - http://www.lighthouse3d.com/opengl/glut/
- When creating the Visual C/C++ project, use the console based setup

Office Hours

- Tuesday 5:30 – 7:30 pm
- Pearson Hall 115B
- Webpage
  - vims.cis.udel.edu/~mani/TA%20Courses/Fall05/graphics/index.html
- Email - mani@udel.edu