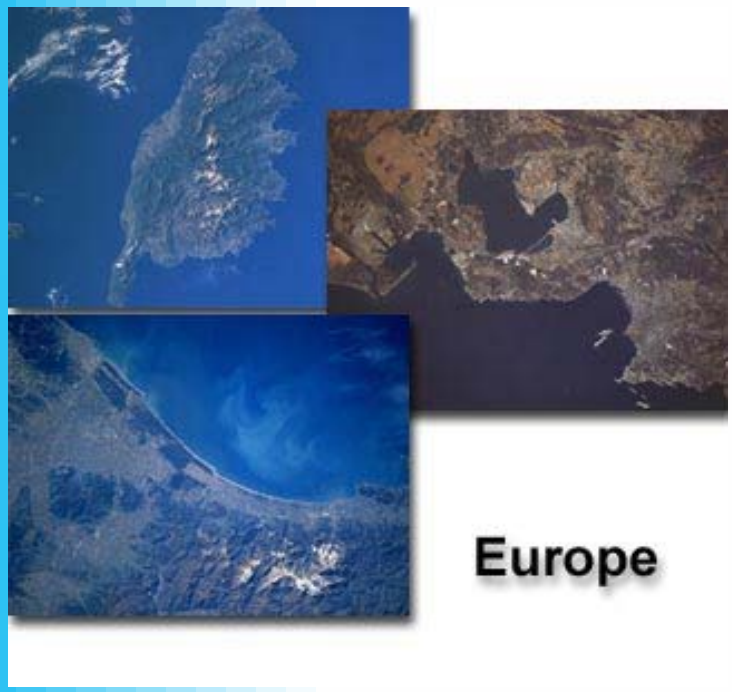


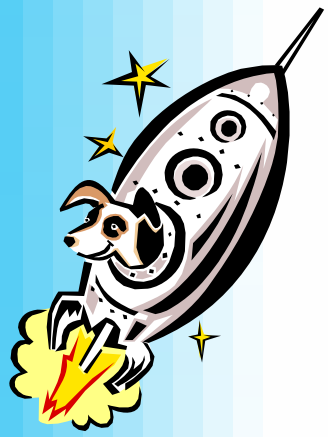
OpenGL and GLUT

Lecture 2
CISC440/640
Spring 2015

COMPUTER GRAPHICS

Department of Computer and Information Science

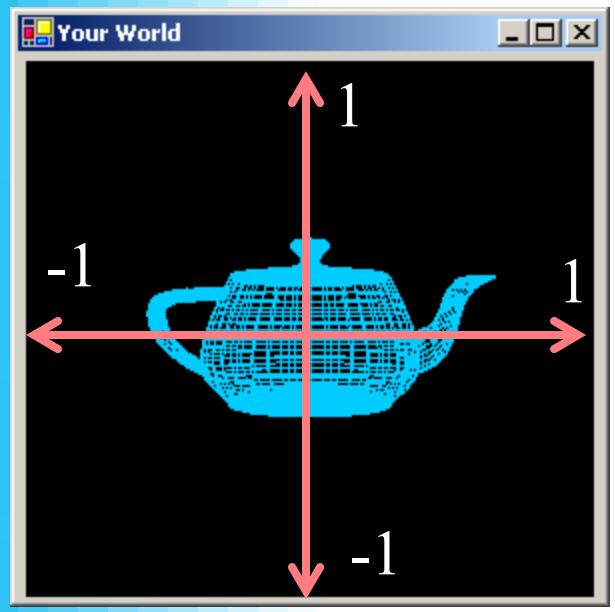
Today's Topic



- The secrets of Glut-tony

So let's do some graphics!

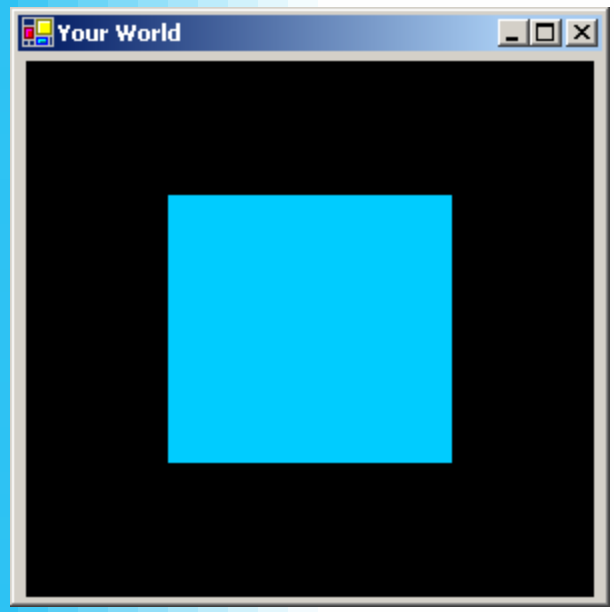
- For the next week or so this is your world:



- A box ranging from
  [-1,1] in x, and
  [-1,1] in y
- We can easily draw in the box with a just few lines of code

So let's do some graphics!

- For the next week or so this is your world:

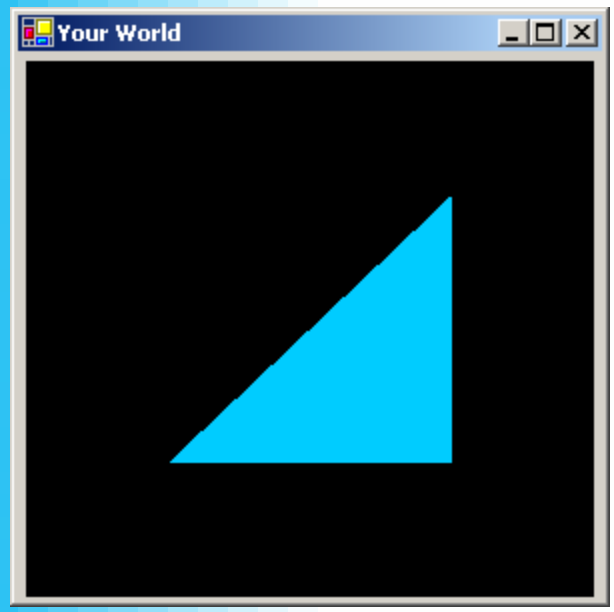



```
void display11() {
    glClearColor(0, 0, 0, 1);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3d(0, 0.8, 1.0);
    glBegin(GL_POLYGON);
      glVertex2d(-0.5, -0.5);
      glVertex2d( 0.5, -0.5);
      glVertex2d( 0.5,  0.5);
      glVertex2d(-0.5,  0.5);
    glEnd();
    glFlush();
}
```

Persistent State

So let's do some graphics!

- Make a small change

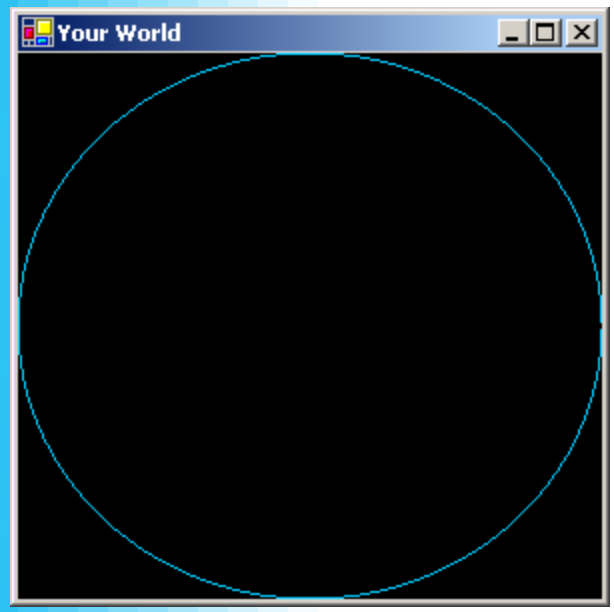



```
void display2() {
    glViewport(0, 0, panelSize.Width, panelSize.Height);
    glClearColor(0, 0, 0, 1);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3d(0, 0.8, 1.0);
    glBegin(GL_POLYGON);
      glVertex2d(-0.5, -0.5);
      glVertex2d( 0.5, -0.5);
      glVertex2d( 0.5,  0.5);

    glEnd();
    glFlush();
}
```

So let's do some graphics!

- Another little tweak




```
void Display3() {
    glClearColor(0, 0, 0, 1);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3d(0, 0.8, 1.0);
    double angle = 0;
    glBegin(GL_LINE_LOOP);
    for (int i = 0; i < 90; i++) {
        glVertex2d(Math.Cos(angle*Math.PI/180.0),
                   Math.Sin(angle*Math.PI/180.0));
        angle += delta;
    }
    glEnd();
    glFlush();
}
```

OpenGL – What is It?

- **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

- Course web page has links to online function references (functions from each library start with library prefix—i.e., **`gl*`**, **`glu*`**, **`glut*`**)

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

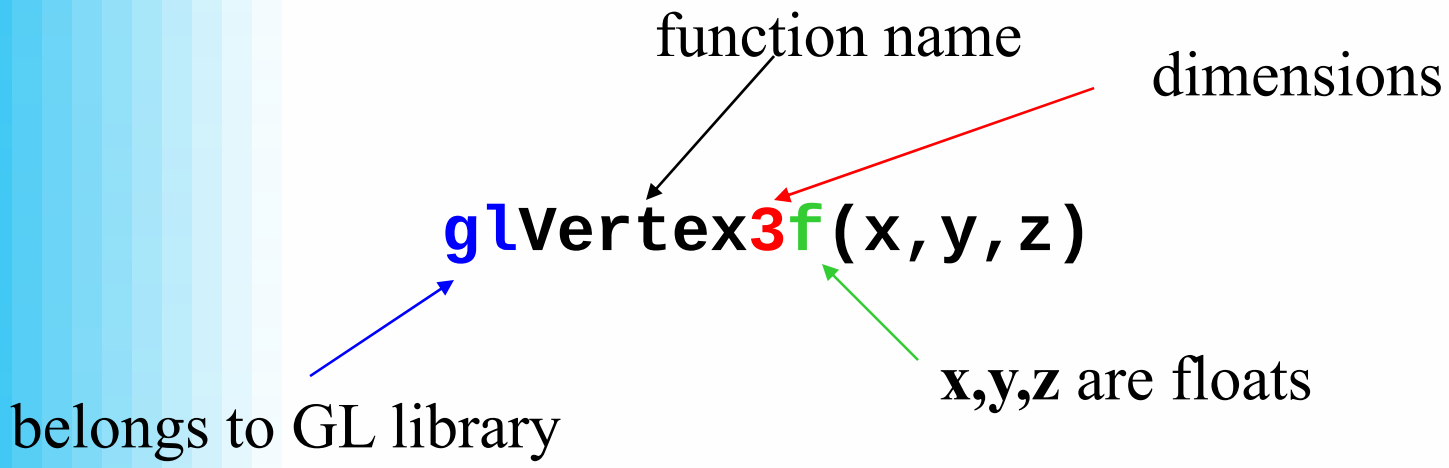
OpenGL Libraries

- **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

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

What is GLUT, Tony?

- A “O/S agnostic” OpenGL environment
- Advantages:
  - Portable: Windows, Cygwin, Linux, Mac-OS
  - minimal-overhead
    (Hides away details of opening windows, etc.)
  - Appeals to C-hackers (console for printf()’s, etc)
- Disadvantages
  - Ugly (lacks look-and-feel of real app, outdated call-back-based event-handling model)
  - Limited Interaction
  - Global variables galore

Event-driven GLUT program structure

1. Configure and open window
2. Initialize OpenGL state, program variables
3. Register callback functions
   - Display (where rendering occurs)
   - Resize
   - User input: keyboard, mouse clicks, motion, etc. (next Tuesday)
4. Enter event processing loop

Portions of some slides adapted from “An Interactive Introduction to OpenGL Programming”, D. Shreiner, E. Angel, V. Shreiner, SIGGRAPH 2001 course

Getting GLUT

- Web site:

  Windows:

  www.xmission.com/~nate/glut.html

  Others:

  www.opengl.org/developers/documentation/glut.html
- Overview:
  Appendix D of OpenGL Programming Guide
- Back to the Code

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 ();
   //glOrtho2D(-1.0, 1.0, 1.0, -1.0);
}
```

black clear color

opaque window

fill/draw with white

viewing volume
i.e., world size

Display callback function

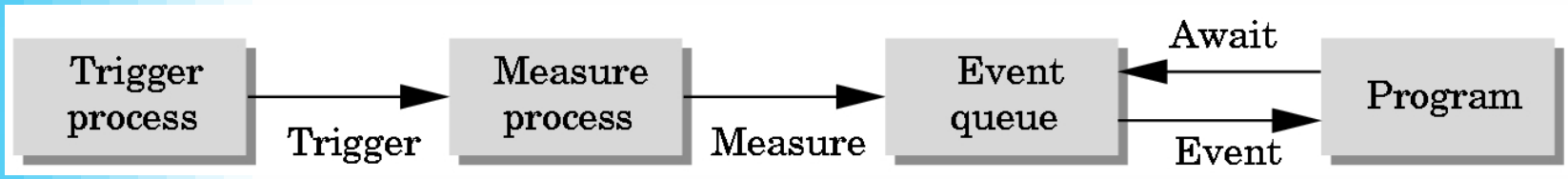
```
void mydisplay()
{
   glClear(GL_COLOR_BUFFER_BIT);

   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)*

Rendering Steps (no animation)

- In function registered with **`glutDisplayFunc()`**:

1. Clear window: **`glClear(GL_COLOR_BUFFER_BIT)`**
2. Draw shapes
   - Set colors, patterns, point/line sizes
   - Specify type of geometric primitive(s) and list vertices
3. Make offscreen draw buffer the display buffer with **`glutSwapBuffers()`**

Specifying Geometric Primitives

- Primitives are specified using

```
glBegin(primType);
...
glEnd();
```

  - *primType* determines how vertices are combined

```
GLfloat red, green, blue;
GLfloat x, y;

glBegin(primType);
for (i = 0; i < nVerts; i++) {
  glColor3f(red, green, blue);
  glVertex2f(x, y);
  ... // change coord. values
}
glEnd();
```

Drawing: Miscellaneous

- **`glColor():`** Range is [0, 1] for each color channel for **`glColor3f()`**; [0, 255] for **`glColor3ub()`**
- **`Can set persistent “pen size” outside of glBegin()/ glEnd()`**
  - **`glPointSize(GLfloat size)`**
  - **`glLineWidth(GLfloat width)`**
- **`glRect(x1, y1, x2, y2)`** specifying opposite corners of rectangle is equivalent to GL_POLYGON with four vertices listed (i.e., filled)

Next Time

- How to display an image
- World and Screen Space