OpenGL and GLUT Basics

Outline

- OpenGL & GLUT basics
  - User interaction
  - 2-D drawing

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

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.
4. Enter event processing loop

Simple OpenGL program

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

void main(int argc, char** argv)
{
    glutInit(&argc, argv);        // configure and open window
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);
    glutInitWindowSize(100, 100);
    glutCreateWindow("hello");

    init();                       // set OpenGL states, variables

    glutDisplayFunc(display);     // register callback routines
    glutKeyboardFunc(keyboard);


    glutMainLoop();               // enter event-driven loop
}
```

adapted from E. Angel

Configure and open window

- **`glutInit:`** Pass command-line flags on to GLUT
- **`glutInitDisplayMode:`** OR together bit masks to set modes on pixel type (indexed vs. true color), buffering, etc.
- **`glutInitWindowSize, glutCreateWindow:`** Set drawing window attributes, then make it

Initialize OpenGL state

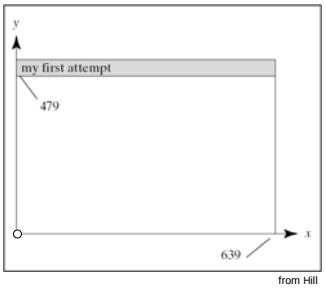
- **`init():`** Set OpenGL state, program variables
  - Use GL types/typedefs **`GLfloat`**, **`GLint`**, **`GL_TRUE`**, **`GL_FALSE`**, etc. for cross-platform compatibility

```
void init() {
   glClearColor(0.0, 0.0, 0.0, 0.0);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(0, right, 0, top);
}
```

sets "units" of subsequent draw commands

OpenGL screen coordinates

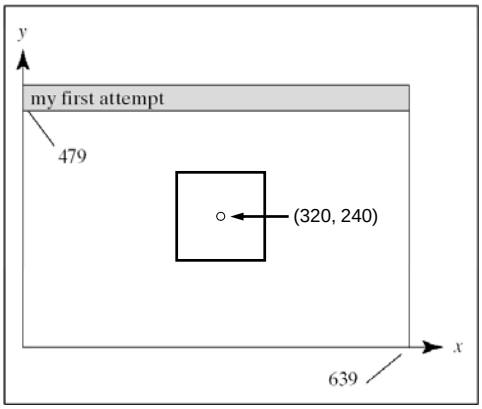
- Bottom left corner is origin
- **`gluOrtho2D()`** sets the units of the screen coordinate system
  - **`gluOrtho2D(0, w, 0, h)`** means the coordinates are in units of pixels
  - **`gluOrtho2D(0, 1, 0, 1)`** means the coordinates are in units of "fractions of window size" (regardless of actual window size)



from Hill

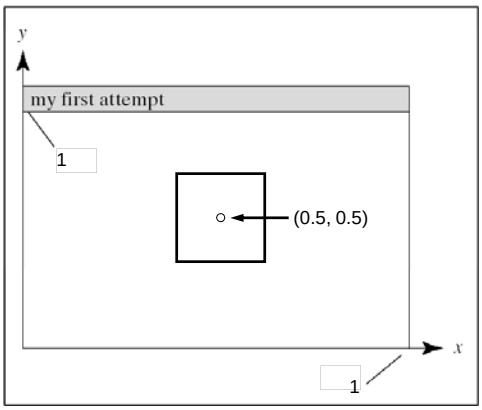
Example: Specifying the center of a square

`gluOrtho2D(0, 640, 0, 480)`



Example: Specifying the center of a square

`gluOrtho2D(0, 1, 0, 1)`



A complete OpenGL program

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

void main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (640, 480);
    glutInitWindowPosition (100, 150);
    glutCreateWindow ("my first attempt");
    glutDisplayFunc(myDisplay);
    myInit ();
    glutMainLoop();
}
```

A complete OpenGL program (cont.)

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

    glColor3f (0.0, 0.0, 0.0);
    glPointSize(4.0);
    glBegin(GL_POINTS);
     glVertex2i(100, 50);
     glVertex2i(100, 130);
     glVertex2i(150, 130);
    glEnd();

    glFlush ();
}
```

A complete OpenGL program (cont.)

```
void myInit (void)
{
  glClearColor(1.0, 1.0, 1.0, 0.0);
  glColor3f(0.0f, 0.0f, 0.0f);
  glPointSize(4.0);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}
```

Rendering Steps

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

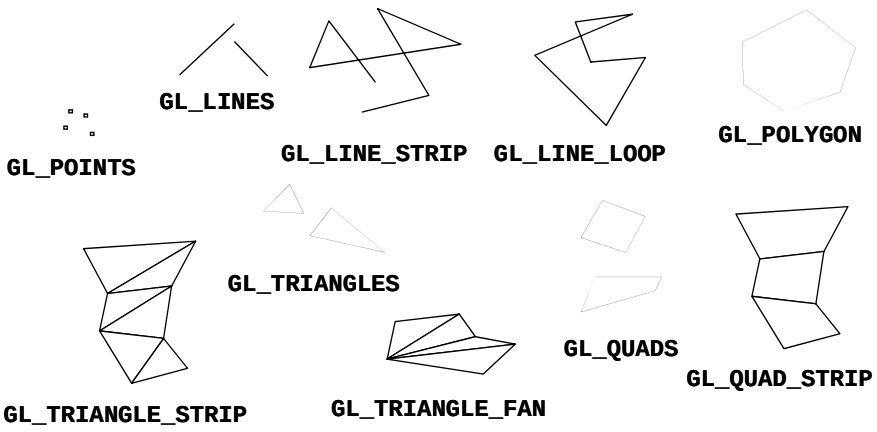
1. Clear window
   - **`glClear(GL_COLOR_BUFFER_BIT);`**
2. Draw shapes
   - Set colors, patterns, point/line sizes
   - Specify type of geometric primitive(s) and list vertices
3. Swap buffers if display mode is **`GLUT_DOUBLE`**
4. Force all operations to complete with **`glFlush()`**

Single- vs. double-buffering

- Single-buffering: Draw directly to screen buffer
- Double-buffering: Draw to offscreen buffer, then make that the screen buffer when done

- For animation, double-buffering is better because it eliminates flickering

OpenGL Geometric Primitives



Specifying Geometric Primitives

- Primitives are specified using

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

  – *primType* determines how vertices are combined

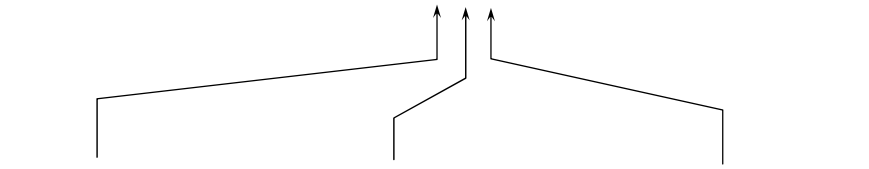
```
GLfloat red, green, blue;
GLfloat x, y;

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

OpenGL Command Formats

glVertex3fv(*v*)

glColor3fv(*v*)



Number of components

```
2 - (x,y)
3 - (x,y,z),
    (r,g,b)
4 - (x,y,z,w),
    (r,g,b,a)
```

Data Type

```
b  - byte
ub - unsigned byte
s  - short
us - unsigned short
i  - int
ui - unsigned int
f  - float
d  - double
```

Vector

omit “v” for scalar form– e.g., glVertex2f(x, y) glColor3f(r, g, b)