Interaction I
Ivan Sutherland (MIT 1963) established the basic interactive paradigm that characterizes interactive computer graphics:

User sees an *object* on the display
User points to (*picks*) the object with an input device (light pen, mouse, trackball)
Object changes (moves, rotates, morphs)
Repeat
Objectives

Event-driven programming
Reshape
Event-Driven Programming

Dominant interaction method in graphical user interfaces

*While* (1)

Look for **events** in the queue

For each event in the queue, if there is a registered **callback**, execute it

otherwise, discard it
Common Event Types

Window: resize, expose, iconify
Mouse: click one or more buttons
Motion: move mouse
Keyboard: press or release a key
Idle: nonevent
Callbacks

Programming interface for event-driven input

Define a *callback function* for each type of event the graphics system recognizes

Callbacks are just functions with the required signature, that are passed as function pointers

GLUT example: `glutMouseFunc (mymouse)`

```c
void glutMouseFunc(void (*func)(int button, int state, int x, int y));
```
GLUT Callbacks

GLUT recognizes a subset of the events recognized by any particular window system (Windows, X, Macintosh)

- glutDisplayFunc
- glutMouseFunc
- glutReshapeFunc
- glutKeyboardFunc
- glutIdleFunc
- glutMotionFunc, glutPassiveMotionFunc
The Display Callback

The display callback is executed whenever GLUT determines that the window should be refreshed, for example:

- When the window is first opened
- When the window is reshaped
- When a window is exposed
- When the user program decides it wants to change the display

In main.c

```c
glutDisplayFunc(mydisplay)
```

Every GLUT program must have a display callback
Post Redisplay

Sometimes you need to signal the application to redraw

```c
    glutPostRedisplay();
```

Sets a flag

GLUT checks to see if the flag is set at the end of the event loop
The idle callback is executed whenever there are no events in the event queue

```
glutIdleFunc(myIdle)
```

The timer callback is executed at specified times

```
glutTimerFunc(33, myTimerCB)
```
 glutMouseFunc(mymouse)

void mymouse(GLint button, GLint state, GLint x, GLint y)

Returns

which button (GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, GLUT_RIGHT_BUTTON) caused the event

state of that button (GLUT_UP, GLUT_DOWN)

position in window
Positioning in Windows

The position in the screen window (from glut or glfw) is measured in pixels with the origin at the top-left corner.

OpenGL uses a coordinate frame for the framebuffer with origin at the bottom left.

Must invert y coordinate returned by callback:

\[ y_{\text{OpenGL}} = h - y_{\text{screen}} \]
glutKeyboardFunc (mykey)

void mykey(unsigned char key, int x, int y)
{
    if (key == ‘Q’ | key == ‘q’)
    exit(0);
}

Windows & Viewports

Do not have to use the entire window to render your scene
Viewport is the part of the window to render to

\texttt{glViewport(x,y,w,h)}
Reshaping the window

Let’s assume the Viewport == Window

We can reshape and resize the OpenGL display window by pulling the corner of the window

What happens to the image?

What should happen to the image?
Default Reshape
Reshape Possibilities

original

reshaped
The Reshape callback

`glutReshapeFunc(myreshape)`
`void myreshape(int w, int h)`

Passed the width and height of new window (in pixels)
A redisplay is posted automatically at end of execution of the callback
GLUT has a default reshape callback to set the viewport to the window size

Called when window is first opened. Often used for camera setup
Default Window Sizing

Resizes the viewport to match the window
Does not change the view volume
Window resizing – Do Nothing

Write a Reshape function, but do nothing in it.

Does not change view volume AND does not change viewport
Better Solution

Make vv aspect ratio match the viewport AR (which matches the window AR)

Square view volume

Resized window and a viewport with no distortion
void myReshape(int w, int h)
{
    glViewport(0,0,w,h);
    float ar = w/h;
    if( ar < startAR) { // (w <= h ){ //taller
        proj = Ortho(vl, vr, vl * (GLfloat) h / (GLfloat) w, vr * (GLfloat) h / (GLfloat) w, 0.1,10.0);
    } else //wider
    {
        proj = Ortho(vb * (GLfloat) w / (GLfloat) h, vt * (GLfloat) w / (GLfloat) h, vb, vt,0.1,10.0);
    }
}