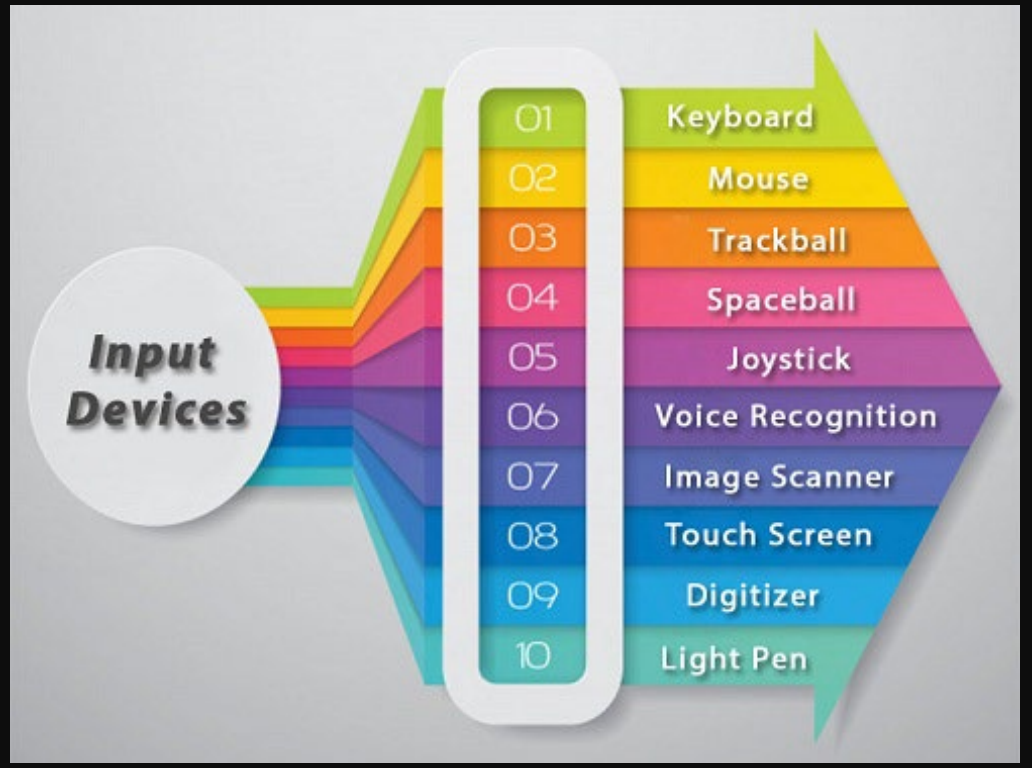
Input and Interaction

Hamzah Asyrani Sulaiman



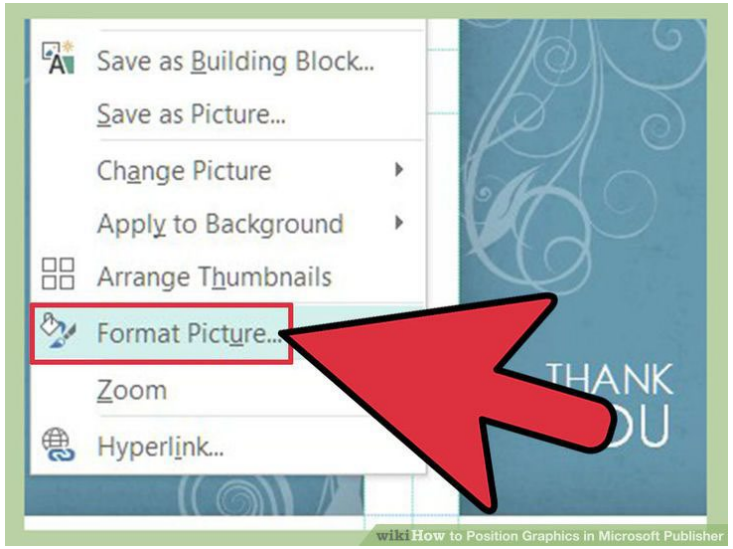
Physical Device

Incremental (Relative) Devices

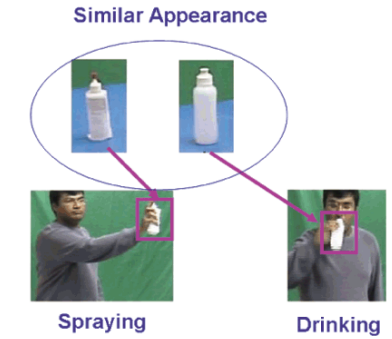
Must integrate these inputs to obtain an absolute position

- Rotation of cylinders in mouse
- Roll of trackball
- Difficult to obtain absolute position
- Can get variable sensitivity

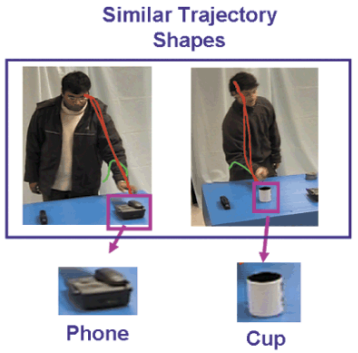
Logical Properties



- What is returned to program via API
  - A position
  - An object identifier



Contextual cues from human interactions aids in object recognition



Contextual cues from object aids in action recognition

Logical Properties

The code provides *logical input*

A number (an **int**) is returned to the program regardless of the physical device

Consider the C and C++ code

C++: **cin >> x;**

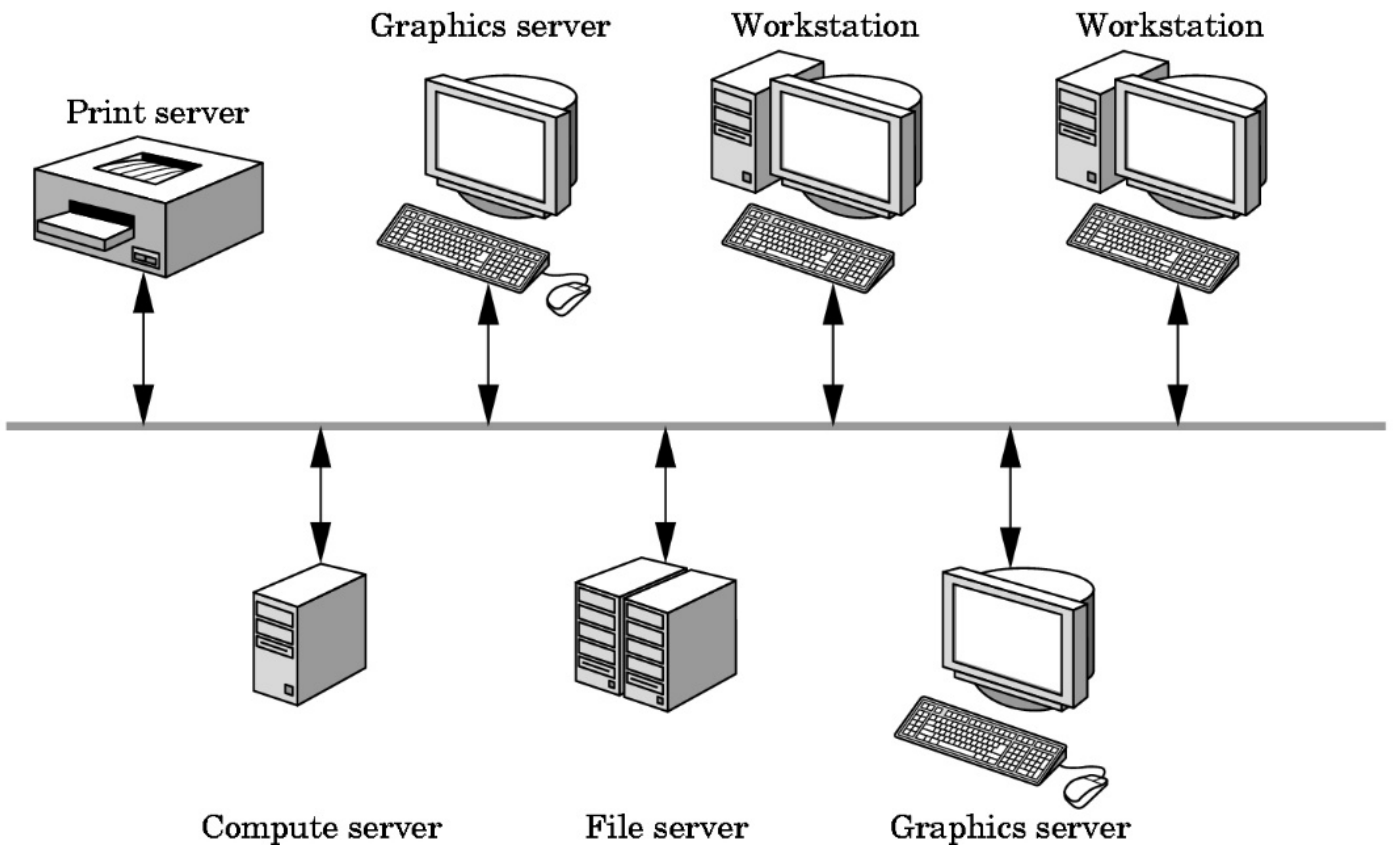
C: **scanf (“%d”, &x);**

Logical Properties (Graphical Input)

- **Locator**: return a position
- **Pick**: return ID of an object
- **Keyboard**: return strings of characters
- **Stroke**: return array of positions
- **Valuator**: return floating point number
- **Choice**: return one of n items

Input Modes

- How and when input is obtained
    - Request or event



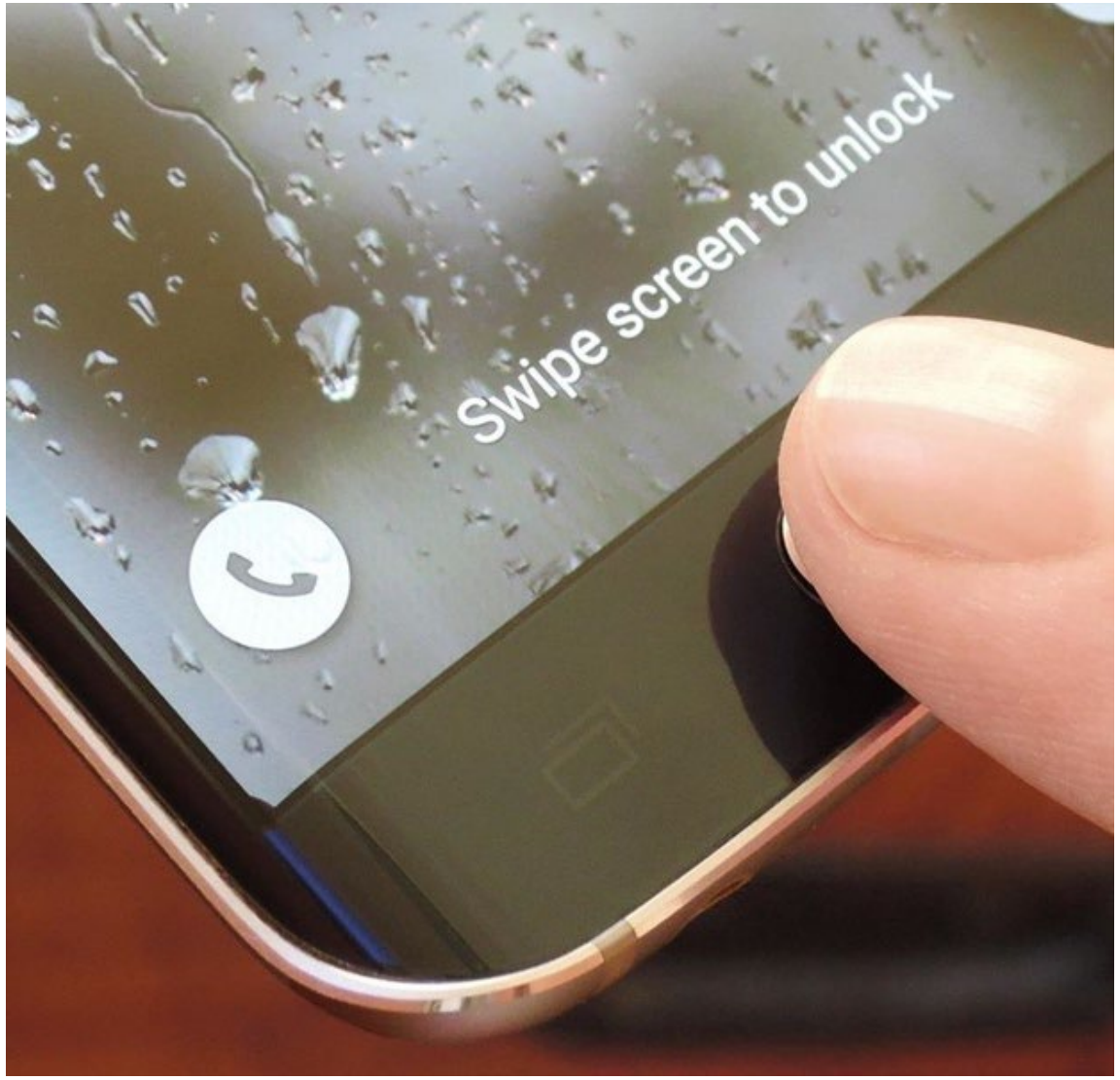
Input Modes

Input devices contain a *trigger* which can be used to send a signal to the operating system

- Button on mouse
- Pressing or releasing a key

When triggered, input devices return information (their *measure*) to the system

- Mouse returns position information
- Keyboard returns ASCII code



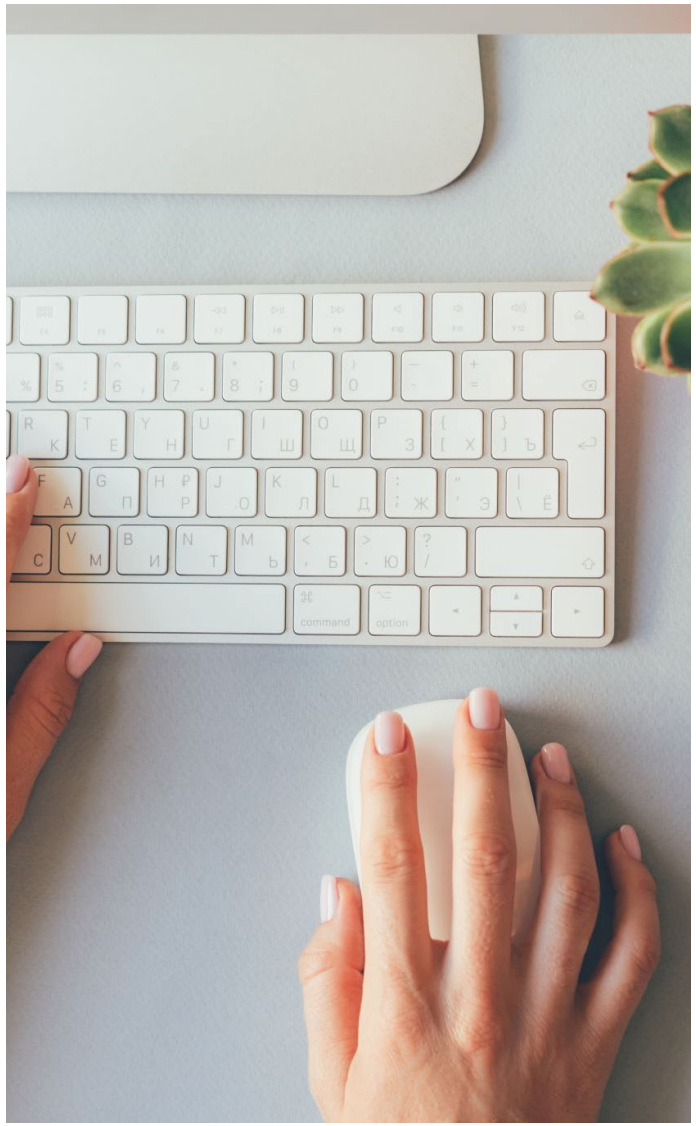
Terminology

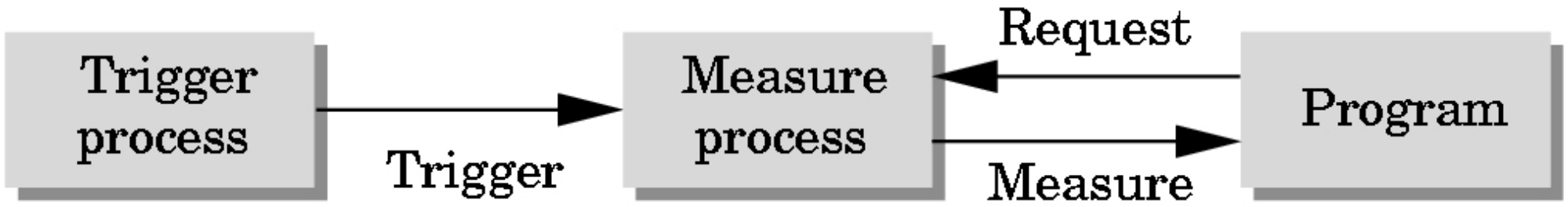
measure - information returned to user program

- one or more characters from a keyboard
- position for a locator

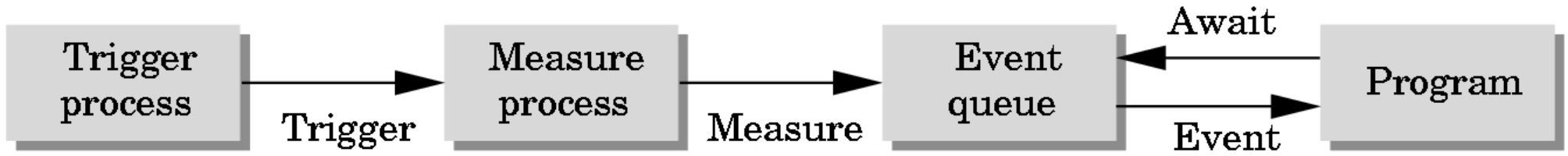
trigger - way device user can signal the computer that input is available

- **Enter** key
- button on locator





Request Mode


Trigger process
Trigger
Measure process
Measure
Event queue
Await
Event
Program

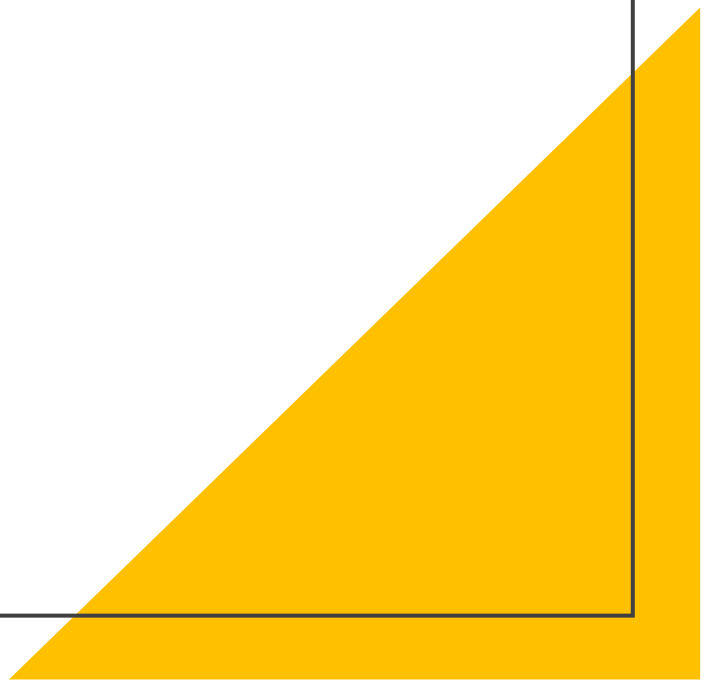
Event Mode

Event Mode

- Window: resize, expose, iconify
- Mouse: click one or more buttons
- Motion: move mouse
- Keyboard: press or release a key
- Idle: nonevent
  - Define what should be done if no other event is in queue

GLUT callbacks

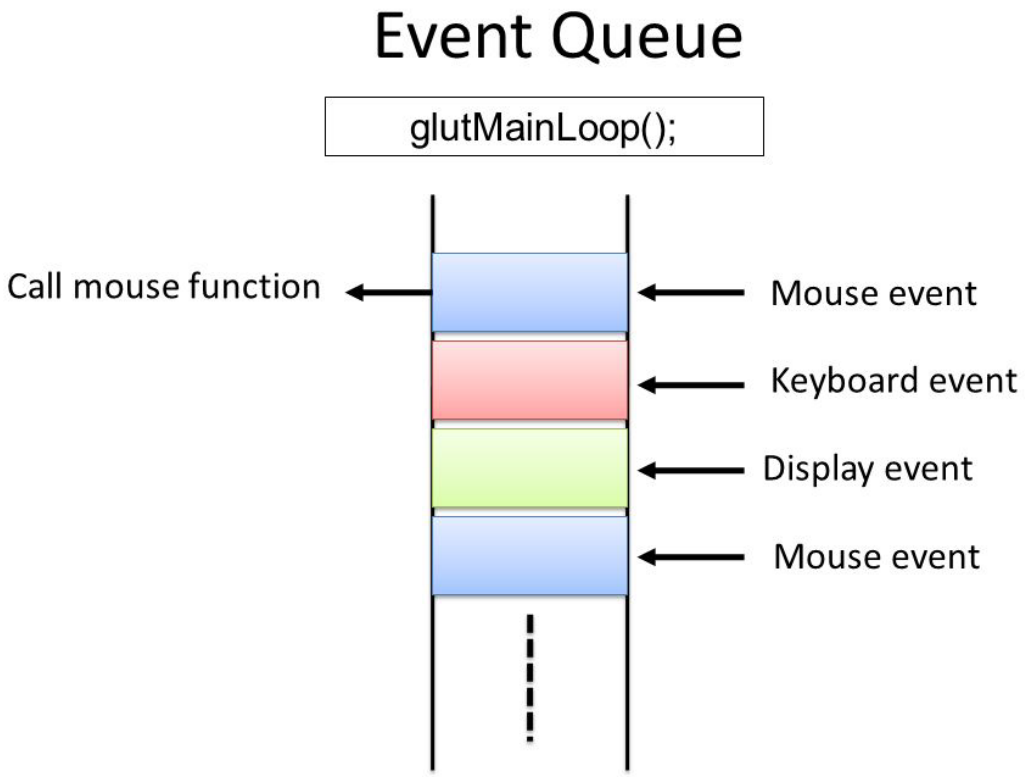
- **glutDisplayFunc**
- **glutMouseFunc**
- **glutReshapeFunc**
- **glutKeyboardFunc**
- **glutIdleFunc**
- **glutMotionFunc, glutPassiveMotionFunc**



GLUT Event Loop

```
glutMainLoop();
```

which puts the program in an infinite event loop



The display callback

```
glutDisplayFunc(mydisplay)
```

identifies the function to be executed

- 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

```
glutPostRedisplay();
```

GLUT checks to see if the flag is set at the end of the event loop

13

glutDisplayFunc is called whenever your window must be redrawn. This includes the time when one calls glutPostRedisplay :)

When does a window need to be redrawn?

- When its size changes
- when it becomes visible
- when some parts of it become visible
- when it is moved
- etc

But what if your display function paints a triangle at position x;y where x;y; are determined by the mouse position? In this case you must ask the system to redraw the window whenever the mouse is moved right? That's why you'll call glutPostRedisplay from MouseFunc(). Actually when you call glutPostRedisplay, the redraw event is queued along with other window-events, like mouse click ets. Essentially what your mainLoop does it pick events from that queue and call their handlers

share improve this answer

answered Nov 17 '10 at 16:15

Armen Tsirunyan

87.4k ● 40 ● 258 ● 379

https://stackoverflow.com/questions/4206472/understanding-the-relationship-between-glutdisplayfunc-and-glutpostredisplay/4206529

Animating a Display

`glClear()` clearing the window

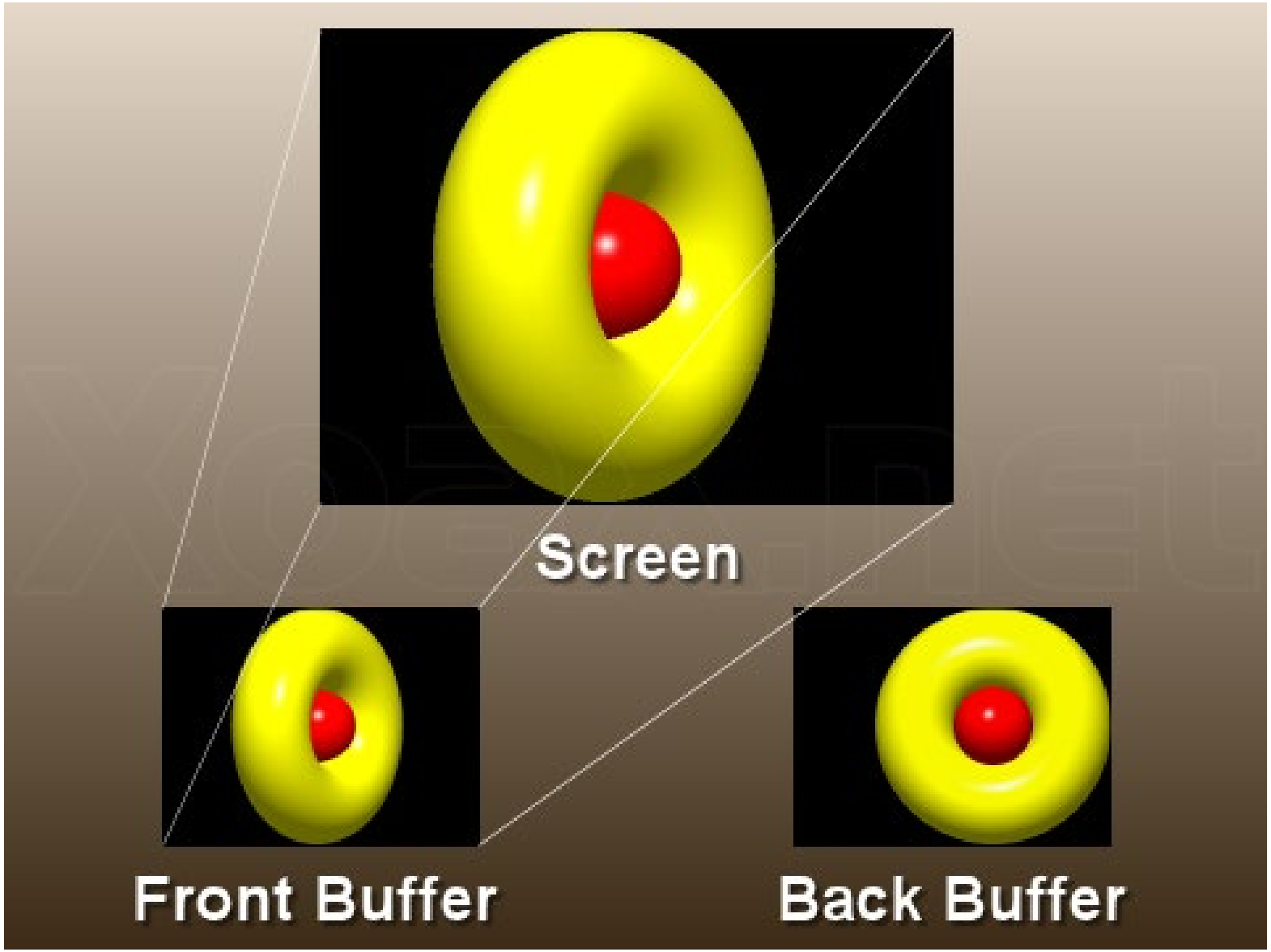
```
glutInitDisplayMode(GL_RGB | GL_DOUBLE)
```

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

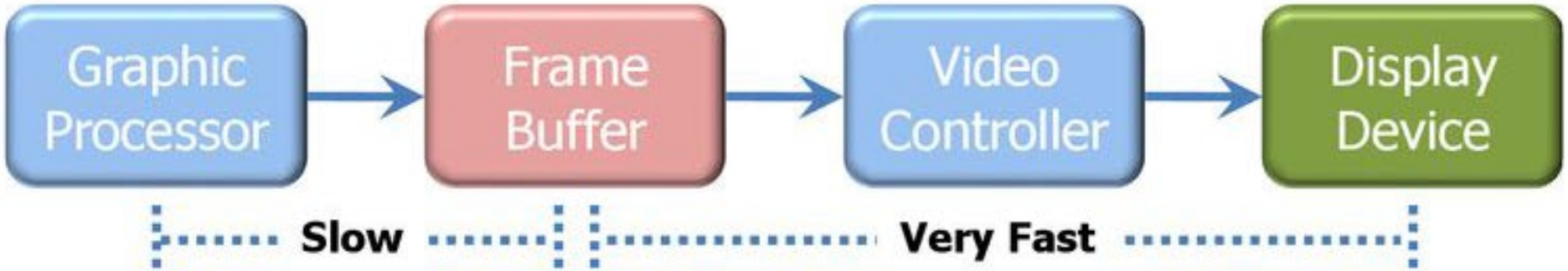
Difference between single buffered(GLUT_SINGLE) and double buffered drawing(GLUT_DOUBLE)

- When using GL_SINGLE, you can picture your code drawing directly to the display.

- When using GL_DOUBLE, you can picture having two buffers. One of them is always visible, the other one is not. You always render to the buffer that is **not** currently visible. When you're done rendering the frame, you swap the two buffers, making the one you just rendered visible. The one that was previously visible is now invisible, and you use it for rendering the next frame. So the role of the two buffers is reversed each frame.

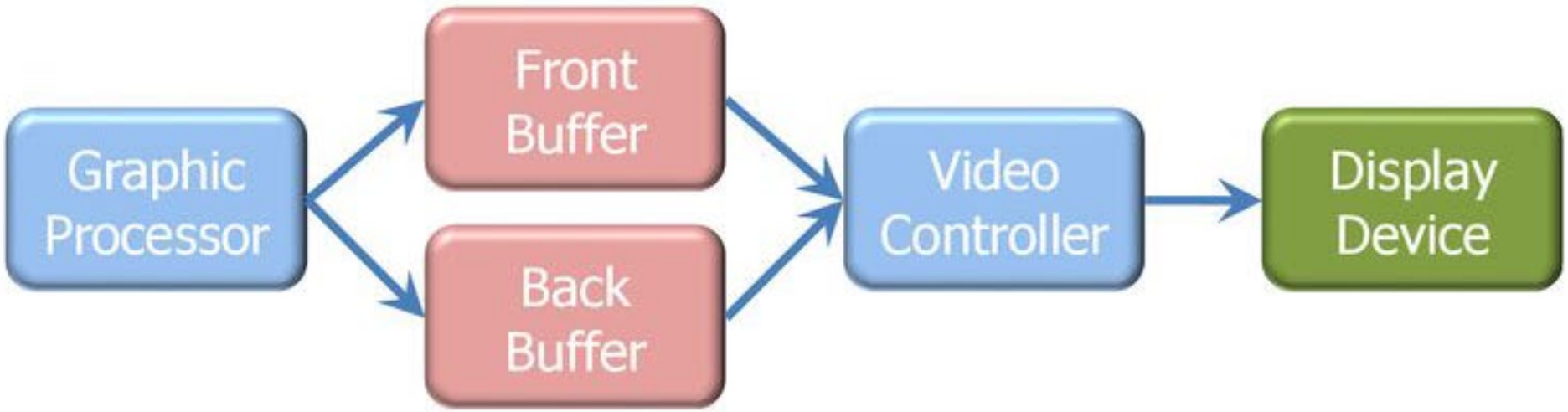

Screen
Front Buffer
Back Buffer

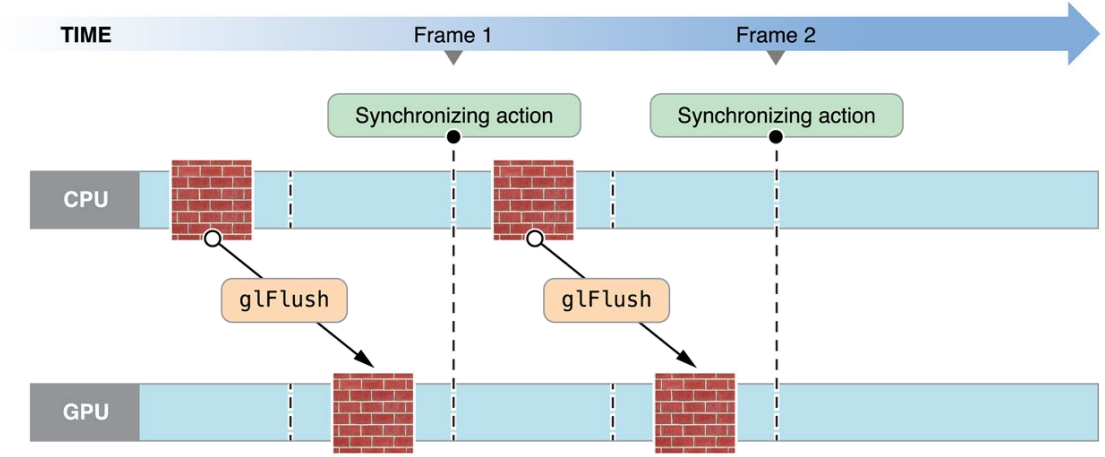
- **Single Buffering (GLUT_SINGLE, Default)**

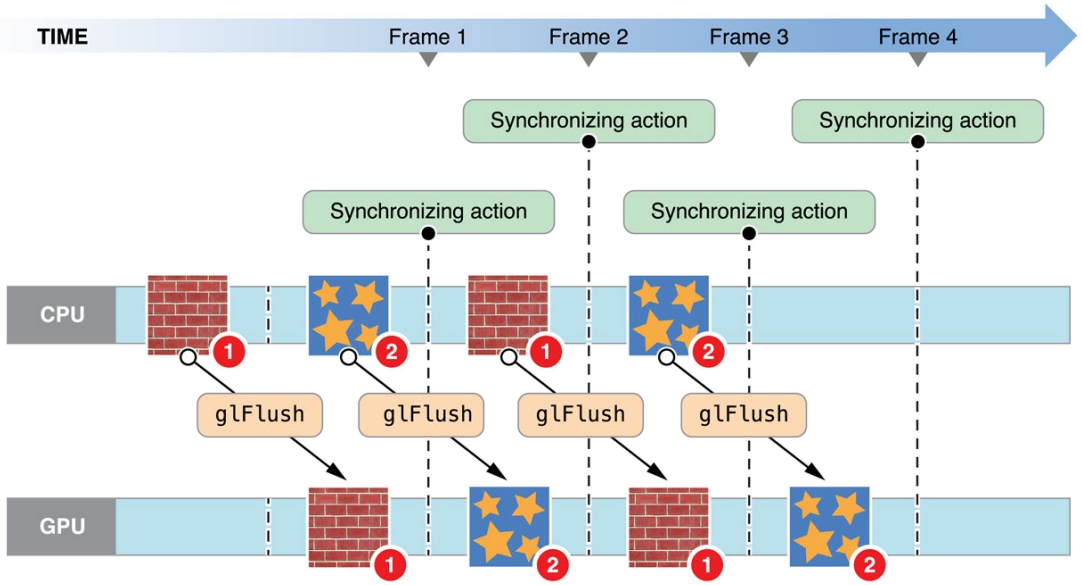


- **Double Buffering (GLUT_DOUBLE)**

  cf.) triple buffering




TIME
Frame 1
Frame 2
Synchronizing action
Synchronizing action
CPU
glFlush
glFlush
GPU


TIME
Frame 1
Frame 2
Frame 3
Frame 4
Synchronizing action
Synchronizing action
Synchronizing action
Synchronizing action
CPU
1
2
1
2
glFlush
glFlush
glFlush
glFlush
GPU
1
2
1
2

In single buffer mode, you call this at the end:

glFlush();

In double buffer mode, you call:

glutSwapBuffers();

You must read this slide together with the text book or Angel Powerpoint slides

Working with Callbacks

Hamzah Asyrani Sulaiman

Using the idle callback

```
glutIdleFunc(myidle)
```

The idle callback is executed whenever there are no events in the event queue

- Use for animation and continuous update

```
glutIdleFunc( idle );
void idle( void )
{
    t +=dt;
    glutPostRedisplay();
}
```

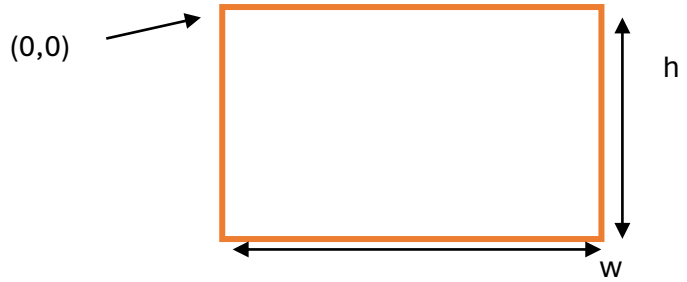
The mouse callback

- **glutMouseFunc(mymouse)**
- **void mymouse(GLint button, GLint state, GLint x, GLint y)**

Positioning

OpenGL uses a world coordinate system with origin at the bottom left

- Must invert *y* coordinate returned by callback by height of window
- *y = h – y;*



Terminating a program

```
void mouse(int btn, int state, int x, int y)
{
    if(btn==GLUT_RIGHT_BUTTON &&
state==GLUT_DOWN)
        exit(0);
}
```

Drawing squares at cursor location

```
void mymouse(int btn, int state, int x, int y)
{
   if(btn==GLUT_RIGHT_BUTTON && state==GLUT_DOWN)
       exit(0);
    if(btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
       drawSquare(x, y);
}
void drawSquare(int x, int y)
{
    y=w-y; /* invert y position */
    glColor3ub( (char) rand()%256, (char) rand )%256,        (char)
  rand()%256); /* a random color */
    glBegin(GL_POLYGON);
        glVertex2f(x+size, y+size);
        glVertex2f(x-size, y+size);
        glVertex2f(x-size, y-size);
        glVertex2f(x+size, y-size);
     glEnd();
}
```

Using the motion callback

We can draw squares (or anything else) continuously as long as a mouse button is depressed by using the motion callback

glutMotionFunc(drawSquare)

We can draw squares without depressing a button using the passive motion callback

glutPassiveMotionFunc(drawSquare)

Using the keyboard

```
glutKeyboardFunc(mykey)

void mykey(unsigned char key,int x, int y)
```

Returns ASCII code of key depressed and mouse location

```
void mykey()
{
      if(key == 'Q' | key == 'q')
            exit(0);
}
```

Special and Modifier Keys

```
Function key 1: GLUT_KEY_F1
Up arrow key: GLUT_KEY_UP
if(key == 'GLUT_KEY_F1' ……
```

Returns ASCII code of key depressed and mouse location

Can also check of one of the modifiers

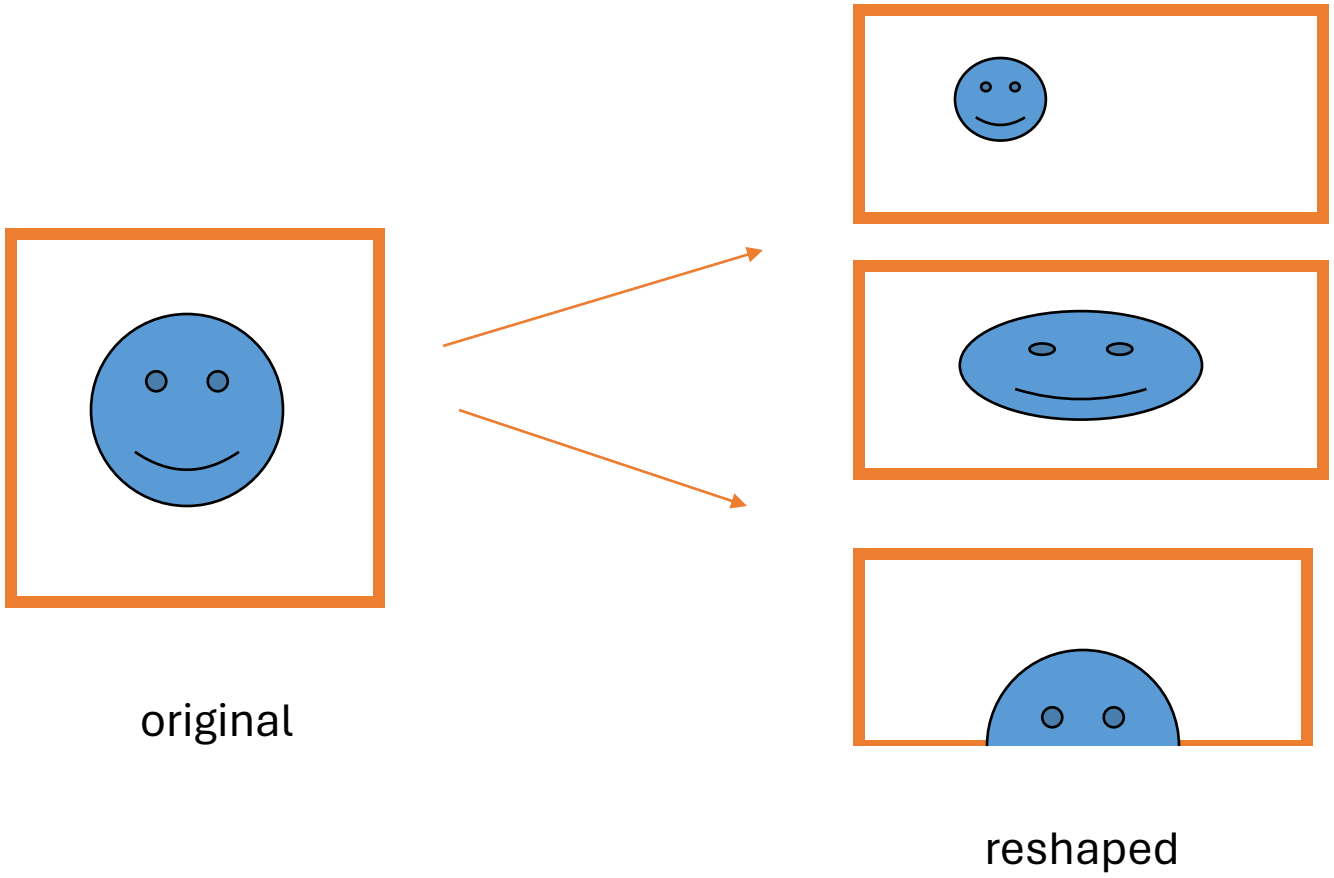
```
GLUT_ACTIVE_SHIFT
GLUT_ACTIVE_CTRL
GLUT_ACTIVE_ALT
```

Reshaping the window

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

- Must redraw from application
- Two possibilities
  - ❑ Display part of world
  - ❑ Display whole world but force to fit in new window
    - ✓ Can alter aspect ratio

Reshape possiblities



The Reshape callback

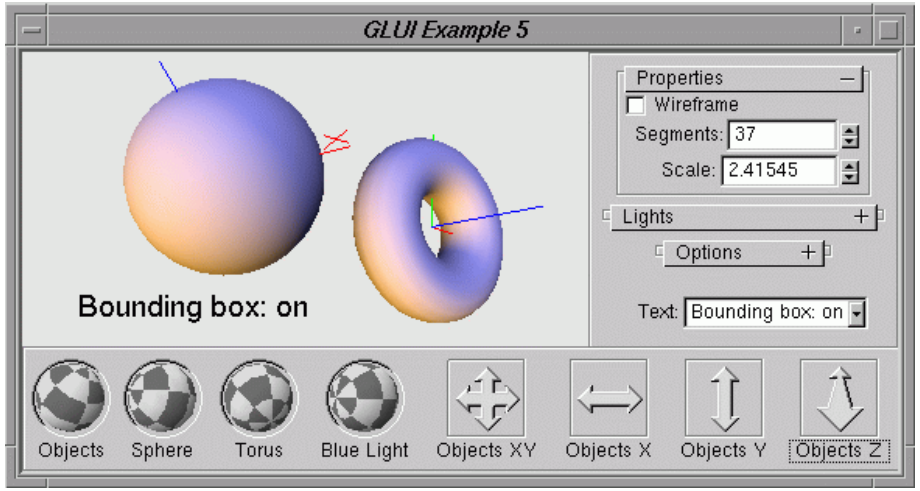
```
glutReshapeFunc(myreshape)
void myreshape( int w, int h)
```

- Returns 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 but you probably want to define your own

The Reshape callback

```
void myReshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION); /* switch matrix mode */
    glLoadIdentity();
    if (w <= h)
        gluOrtho2D(-2.0, 2.0, -2.0 * (GLfloat) h / (GLfloat) w,
            2.0 * (GLfloat) h / (GLfloat) w);
    else  gluOrtho2D(-2.0 * (GLfloat) w / (GLfloat) h, 2.0 *
            (GLfloat) w / (GLfloat) h, -2.0, 2.0);
    glMatrixMode(GL_MODELVIEW); /* return to modelview mode */
}
```

Toolkits and Widgets



- Menus
- Slidebars
- Dials
- Input boxes

https://sourceforge.net/projects/glui/

End for Input and Interaction