CS 4204 Computer Graphics

Window based programming and GLUT

Yong Cao

Virginia Tech

References:

Interactive Computer Graphics, Fourth Edition, Ed Angle

Objectives

Introduce the basic input devices

- Physical Devices
- Input Modes

Event-driven input

Introduce double buffering for smooth animations

Programming event input with GLUT

Project Sketchpad

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

Graphical Input

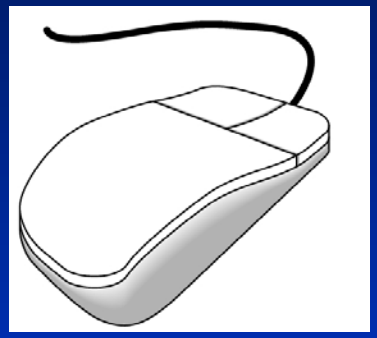
Devices can be described either by

- Physical properties
  - *Mouse*
  - *Keyboard*
  - *Trackball*
- Logical Properties
  - *What is returned to program via API*
    - **A position**
    - **An object identifier**

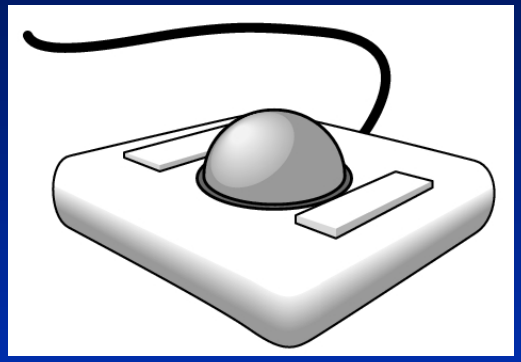
Modes

- How and when input is obtained
  - *Request or event*

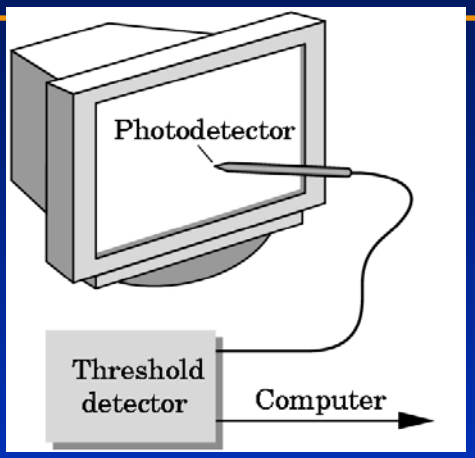
Physical Devices



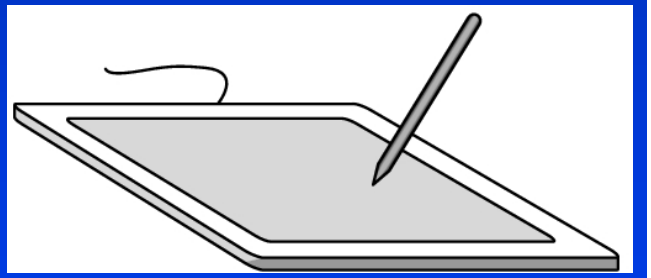
mouse



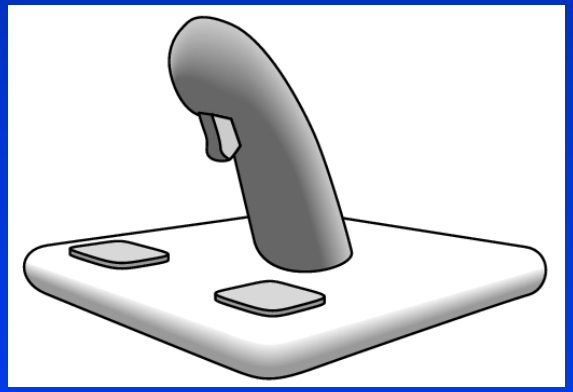
trackball



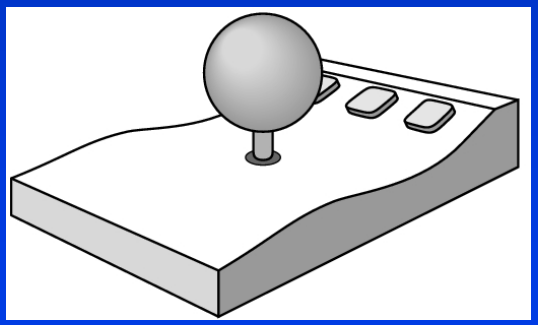
light pen



data tablet



joy stick



space ball

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

Request Mode

Input provided to program only when user triggers the device

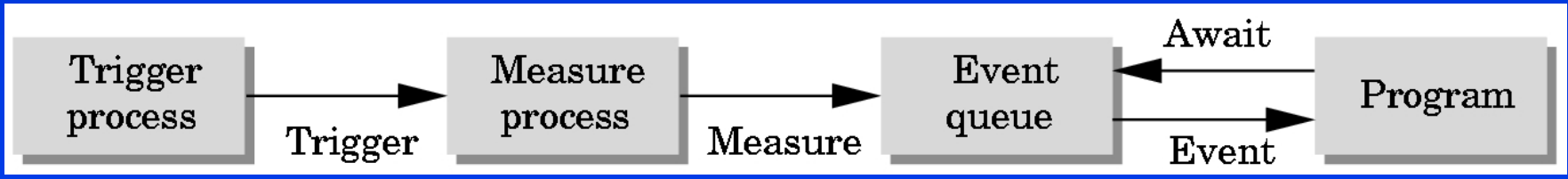
Typical of keyboard input

- Can erase (backspace), edit, correct until enter (return) key (the trigger) is depressed

Event Mode

Most systems have more than one input device, each of which can be triggered at an arbitrary time by a user

Each trigger generates an event *whose measure is put in an* event queue *which can be examined by the user program*



Event Types

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

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)`

mouse callback function

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`

GLUT Event Loop

```
glutMainLoop();
```

which puts the program in an infinite event loop

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

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

- **`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

Animating a Display

When we redraw the display through the display callback, we usually start by clearing the window

- `glClear()`

then draw the altered display

Problem: the drawing of information in the frame buffer is decoupled from the display of its contents

- Graphics systems use dual ported memory

Hence we can see partially drawn display

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)`**

Can use globals to pass information to callbacks

```
float t; /*global */

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

The mouse callback

```
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 event
- state of that button (**GLUT_UP**, **GLUT_DOWN**)
- Position in window

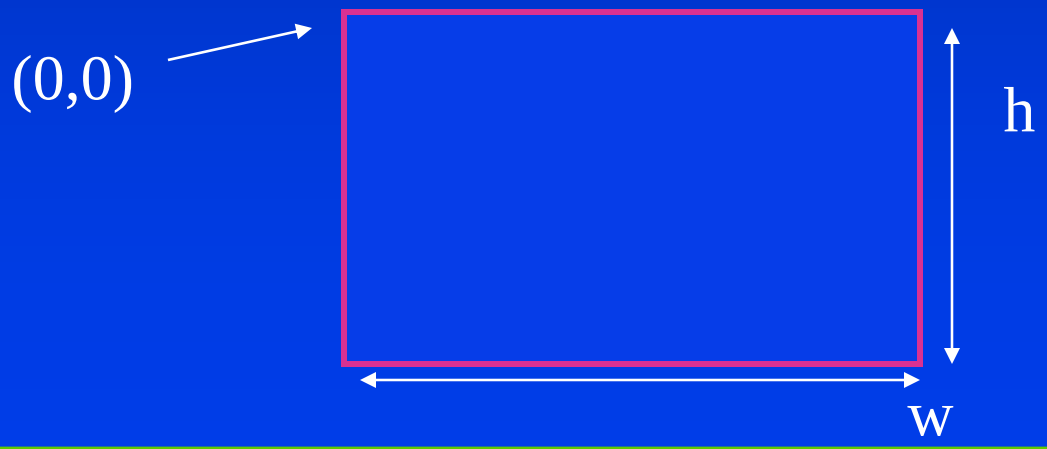
Positioning

The position in the screen window is usually measured in pixels with the origin at the top-left corner

- Consequence of refresh done from top to bottom

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;*



Obtaining the window size

To invert the y position we need the window height

- Height can change during program execution
- Track with a global variable
- New height returned to reshape callback that we will look at in detail soon
- Can also use query functions
  - ***glGetIntv***
  - ***glGetFloatv***

to obtain any value that is part of the state

Using the mouse position

In the next example, we draw a small square at the location of the mouse each time the left mouse button is clicked

This example does not use the display callback but one is required by GLUT; We can use the empty display callback function

```
mydisplay(){}
```

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

GLUT defines the special keys in `glut.h`

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

Can also check of one of the modifiers

- **`GLUT_ACTIVE_SHIFT`**
- **`GLUT_ACTIVE_CTRL`**
- **`GLUT_ACTIVE_ALT`**

is depressed by

`glutGetModifiers()`

- Allows emulation of three-button mouse with one- or two-button mice