
CIS 454 Computer Graphics

Lecture 8, 09/28/2006

Li Shen
Computer and Information Science
UMass Dartmouth

Notes

- HW1 due
 - HW2 out
-

Input and Interaction

- Introduce the basic input devices
 - Physical Devices
 - Logical Devices
 - Input Modes
 - Event-driven input
 - Introduce double buffering for smooth animations
 - Programming event input with GLUT
-

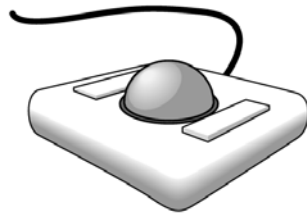
Graphical Input

- 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
-

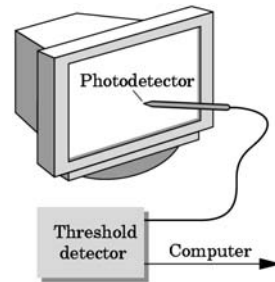
Physical Devices



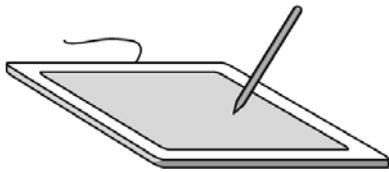
mouse



trackball



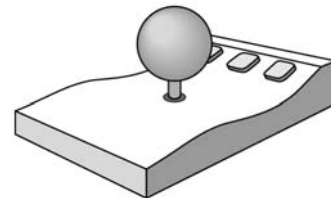
light pen



data tablet



joystick



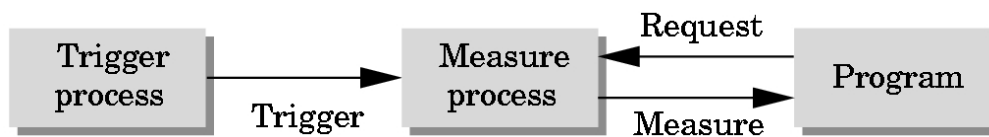
space ball

Graphical Logical Devices

- Graphical input is more varied than input to standard programs which is usually numbers, characters, or bits
- Two older APIs (GKS, PHIGS) defined six types of logical 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

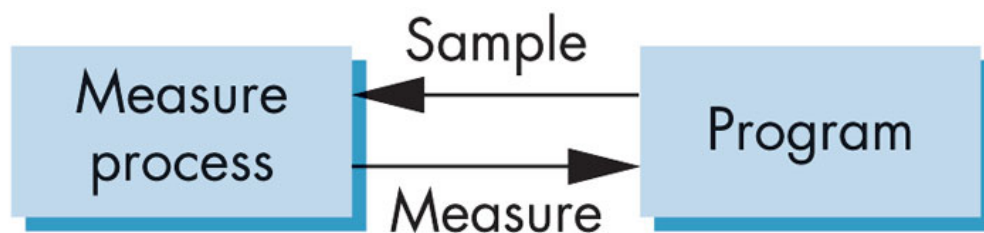
Request Mode

- 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



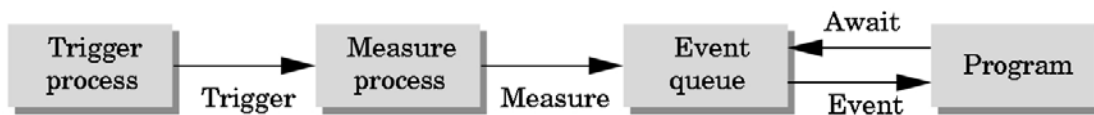
Sample Mode

- Input is immediate
- No trigger needed
- Function call \leq measure



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
-

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`
-

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()
}
```

Working with Callbacks

Objectives

- Learn to build interactive programs using GLUT callbacks
 - Mouse
 - Keyboard
 - Reshape
 - Introduce menus in GLUT
-

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

[square.c](#)

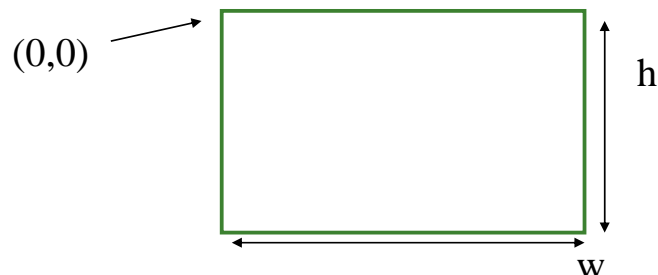
Positioning

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
-

Terminating a program

- In our original programs, there was no way to terminate them through OpenGL
- We can use the simple mouse callback

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

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);
}
```

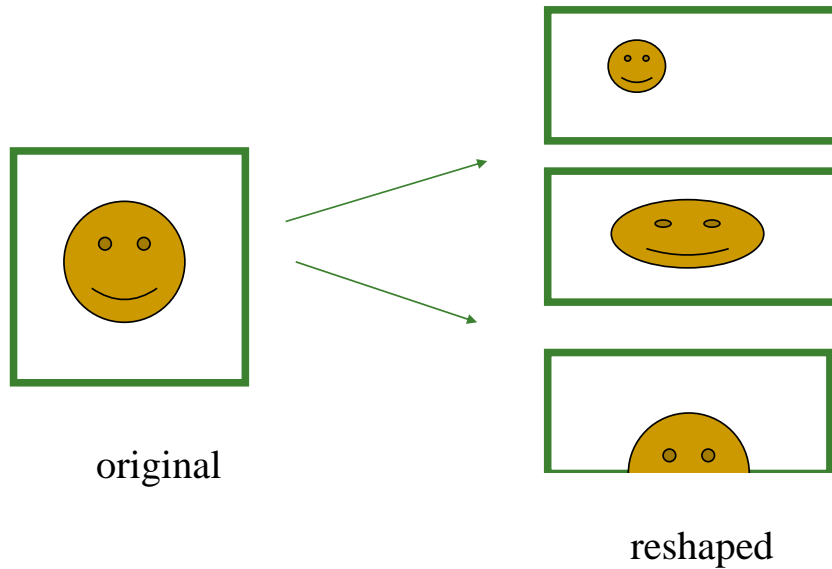
Special and Modifier Keys

- 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
-

Reshaping the window

- We can reshape and resize the OpenGL display window by pulling the corner of the window
 - What happens to the display?
 - 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 possibilities



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 is good place to put viewing functions because it is invoked when the window is first opened

Example Reshape

- This reshape preserves shapes by making the viewport and world window have the same aspect ratio

```
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 */
}
```

square.c: squares of the same size are drawn, regardless of the size or shape of the window.

Toolkits and Widgets

- Most window systems provide a toolkit or library of functions for building user interfaces that use special types of windows called *widgets*
 - Widget sets include tools such as
 - Menus
 - Slidebars
 - Dials
 - Input boxes
 - But toolkits tend to be platform dependent
 - GLUT provides a few widgets including menus
-

Menus

- GLUT supports pop-up menus
 - A menu can have submenus
- Three steps
 - Define entries for the menu
 - Define action for each menu item
 - Action carried out if entry selected
 - Attach menu to a mouse button

[paint.c](#)

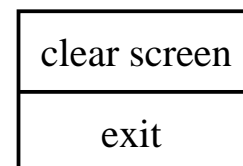
Defining a simple menu

- In `main.c`

```
menu_id = glutCreateMenu(mymenu);
glutAddmenuEntry("clear Screen", 1);

gluAddMenuEntry("exit", 2);

glutAttachMenu(GLUT_RIGHT_BUTTON);
```



entries that appear when
right button depressed

identifiers

[paint.c](#)

Menu actions

- Menu callback

```
void mymenu(int id)
{
    if(id == 1) glClear();
    if(id == 2) exit(0);
}
```

- Note each menu has an id that is returned when it is created
- Add submenus by

```
glutAddSubMenu(char *submenu_name, submenu id)
```

 entry in parent menu

Other functions in GLUT

- Dynamic Windows
 - Create and destroy during execution
 - Subwindows
 - Multiple Windows
 - Changing callbacks during execution
 - Timers
 - Portable fonts
 - `glutBitmapCharacter`
 - `glutStrokeCharacter`
-

After Class

- Read Chapter 3
- Work on HW2

