Input and Interaction

527970
Fall 2020
9/24/2020
Kyoung Shin Park
Computer Engineering
Dankook University

Overview

- Introduce the basic input devices
  - Physical input devices
    - Mouse, Keyboard, Trackball
  - Logical input devices
    - String, Locator, Pick, Choice, Valuators, Stroke device
- Input modes
  - Request mode
  - Sample mode
  - Event mode
- GLUT Devices & Event-driven programming
  - mouse, keyboard, menu, joystick, tablet, ..

Interaction

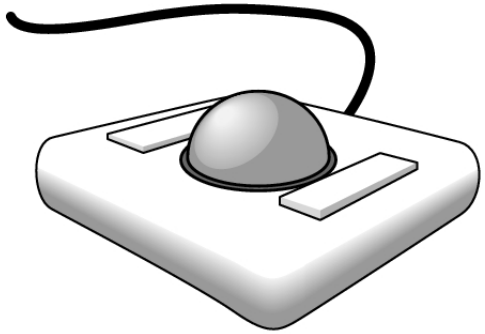
- One of the major advances in computer technology is that users can interact using computer screens.
- Interaction
  - The user takes action through an interactive device such as a mouse.
  - The computer detects user input.
  - The program changes it state in response to this input.
  - The program displays this new status.
  - The users sees the changed display.
  - The processes in which the user reacts to this change are repeated.

Graphical Input

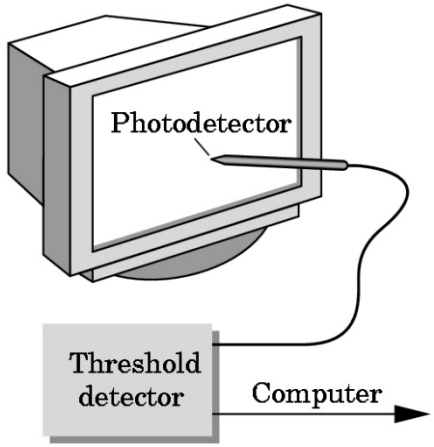
- Input devices can be described either by
  - Physical properties
    - Mouse, Keyboard, Trackball
  - Logical properties
    - Characterized by upper interface with application program, not by physical characteristics
- Input modes
  - The way an input device provides an input to an application program can be described as a **measurement** process and device **trigger**.
    - Request mode
    - Sample mode
    - Event mode

Physical Input Devices

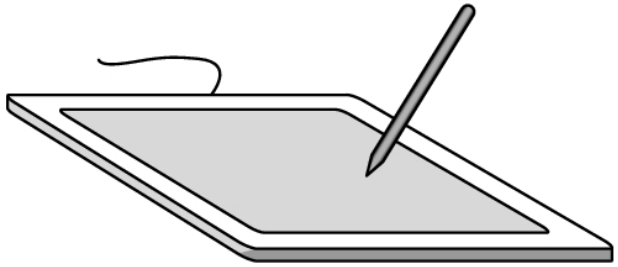
mouse



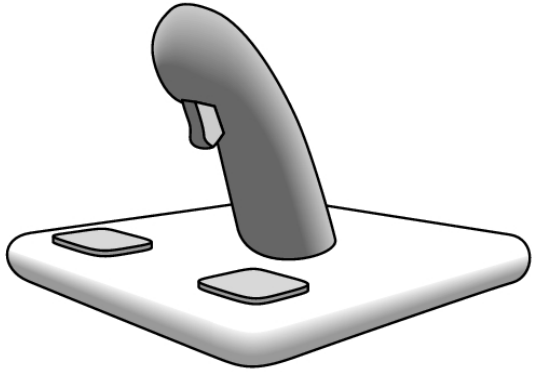
trackball



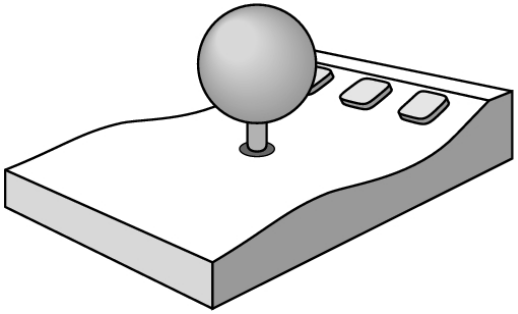
light pen



data tablet



joy stick



space ball

Physical Input Devices

- Physical input devices
  - Pointing devices
    - Allows the user to point to a location on the screen
    - In most cases, the user has more than one button to send a signal or interrupt to the computer.
    - Mouse, trackball, tablet, lightpen, joystick, spaceball
  - Keyboard devices
    - A device that returns a character code to a program
    - Keyboard

Relative Positioning Device

- Devices such as the data tablet return a position directly to the operating system
- Devices such as the mouse, trackball, and joy stick return incremental inputs (or velocities) to the operating system
  - 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 Input Devices

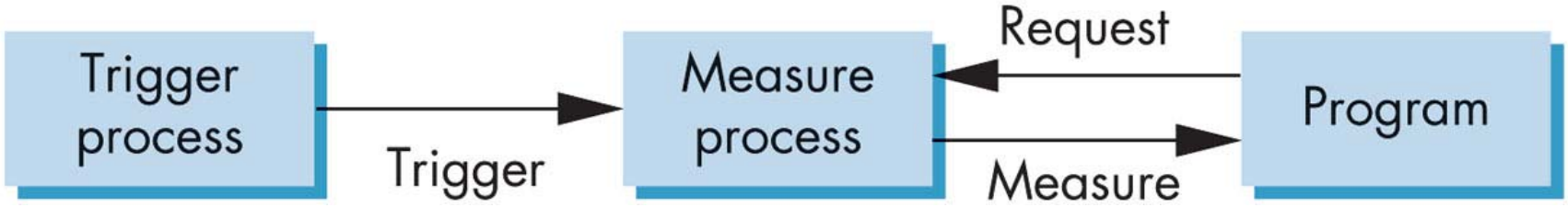
- String device - keyboard
  - Provide ASCII strings of characters to the program
- Locator device – mouse, trackball
  - Provide real world coordinate position to the program
- Pick device – mouse button, gun
  - Return the object's identifier(ID) to the program
- Choice device – widgets, function keys, mouse button
  - Let the user choose one of the options (menu)
- Valuators – slide bars, joystick, dial
  - Provide analog input (range of value) to the program
- Stroke – mouse drag
  - Return array of positions

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

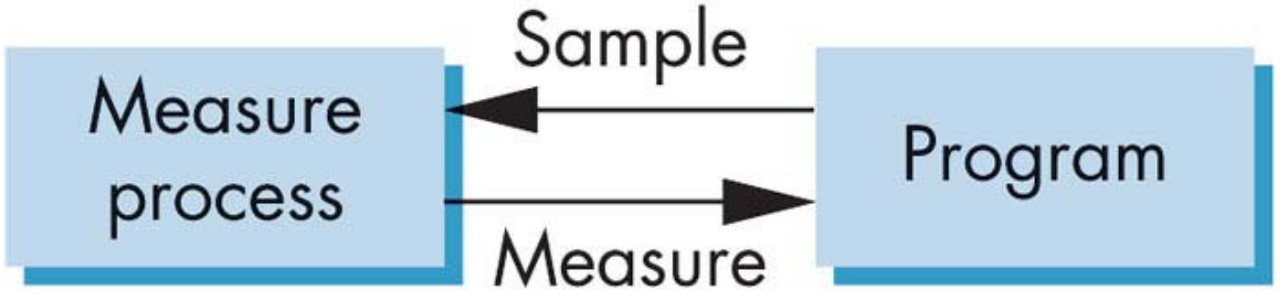
Request Mode

- In request mode, input measurement are not returned to the program until the user triggers the device.
- Standard for typical non-GUI program requiring character input
  - For example, when the C program's scanf function is used, the program stops while waiting for the terminal to type a character. Then, you can type and edit until you hit the enter-key(trigger).



Sample Mode

- Sample mode provides immediate input measures. As soon as the program encounters a function call, the measurement is returned. Therefore, no trigger is required.
- Example: getc function in C program



Event Mode

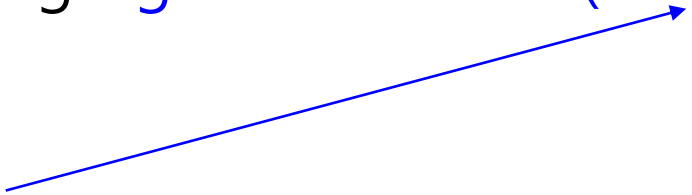
- Most systems have more than one input device, each of which can be triggered at an arbitrary time by a user.
- Each trigger generates an ***event*** whose measure is put in an ***event queue*** which can be examined by the user program.
- Use the callback function for a specific event.

Event Types

- Window – window resize, expose, iconify
- Keyboard – press and release a key
- Mouse – click one or more mouse button
- Motion – move mouse
- Idle – no event (define what should be done if no other event is in queue)

Programming Event-Driven Input

- 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, the callback function for mouse event is specified through glutMouseFunc(mouse) in the main function.



void mouse(int button, int state, int x, int y)

GLUT Devices

- Keyboard
  - "normal" keys
  - "special" keys
- Mouse
  - Position
  - buttons
- Joystick
- Tablet
- Dial/button box
- Spaceball

GLUT *Keyboard* Functions

- glutKeyboardFunc(func)
  - Called when the ACII 'character' key is pressed
- glutSpecialFunc(func)
  - Called when the 'special' key is pressed
- glutKeyboardUpFunc(func)
  - Called when the ACII 'character' key is released
- glutSpecialUpFunc(func)
  - Called when the 'special' key is released
- glutGetModifiers()
  - Indicate the Shift, Control, Alt keys status when an event occurs
- glutIgnoreKeyRepeat(val)
  - Tell GLUT to ignore automatic keyboard repeat

GLUT *Keyboard* Event Callback

- void keyboard(unsigned char key, int x, int y)
  - Specify the handling of keyboard
  - *The key* argument is the designated as ACII character code
  - *The x, y* arguments are the position of the mouse when the key is pressed

```
void keyboard(unsigned char key, int x, int y)  {
  switch (key): /* q-key exits the program */
   {
      case ‘q’:
         exit(0);
   }
}
```

GLUT *Special* Key

- GLUT special key
  - GLUT_KEY_{F1,F2..,F12}
  - GLUT_KEY_{UP,DOWN,LEFT,RIGHT} – arrow key
  - GLUT_KEY_{PAGE_UP,PAGE_DOWN,HOME,END,INSERT}

```
void specialkey(int key, int x, int y)   {
   switch(key) {
     case GLUT_KEY_F1:
                red = 1.0; green = 0.0; blue = 0.0; break;
     case GLUT_KEY_F2:
                ...
   }
```

GLUT *Modifier* Key

- int glutGetModifiers(void) to check if the CTRL, ALT, SHIFT modifier keys are pressed.
  - GLUT_ACTIVE_SHIFT – SHIFT key (or Caps Locked)
  - GLUT_ACTIVE_CTRL
  - GLUT_ACTIVE_ALT

```
void keyboard(unsigned char key, int x, int y) {
    if (key == 27) /* ESC-key exits the program */
      exit(0);
    else if (key ==‘r’) {
      int mod = glutGetModifier();
      if (mod == GLUT_ACTIVE_CTRL)
        red = 0.0;
      else
        red = 1.0;
    }
}
```

GLUT *Mouse* Functions

- glutMouseFunc(void(*func)(int button, int state, int x, int y))
  - Called when the mouse button is pressed
- glutMotionFunc(void(*func)(int x, int y))
  - Called when the mouse moves while the button is pressed
- glutPassiveMotionFunc(void (*func)(int x, int y))
  - Called when the mouse button is moved without being pressed

GLUT *Mouse* Event Callback

- void mouse(int button, int state, int x, int y)
  - *The button* argument is GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, GLUT_RIGHT_BUTTON
  - *The state argument* is GLUT_DOWN (when mouse button is pressed) or GLUT_UP (when mouse button is released)
  - *The x, y* arguments are the position of the mouse when the mouse button is pressed or released (in GLUT window coordinates)

```
void mouse(int button, int state, int x, int y) {
    …
}
```

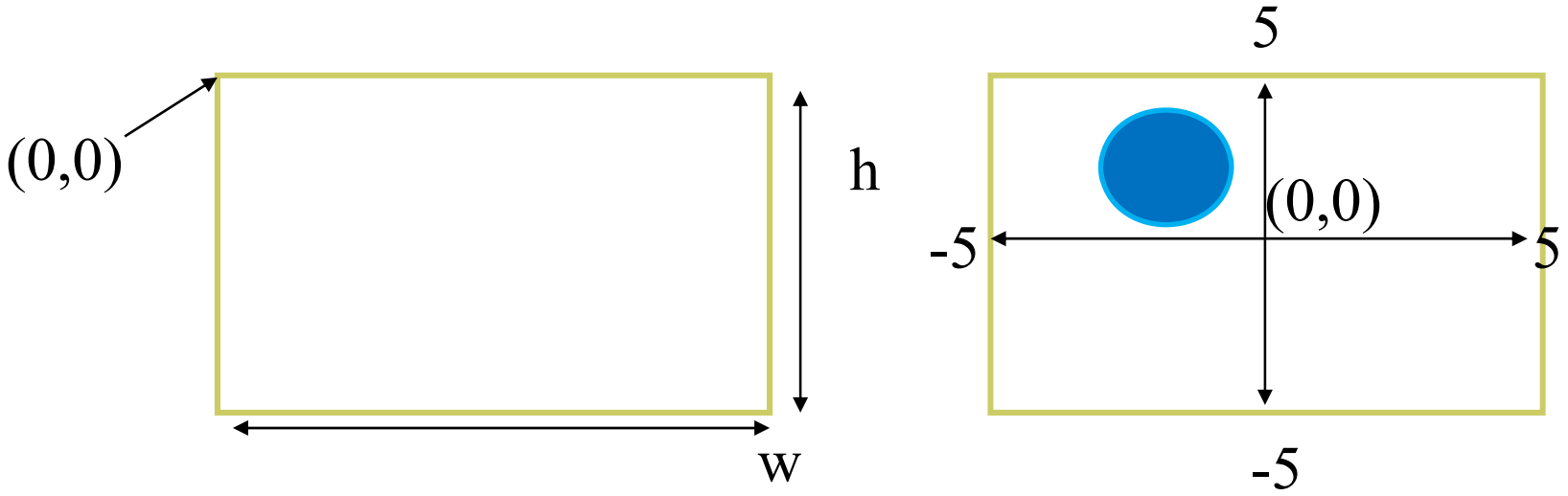
GLUT *Motion* Event Callback

- void motion(int x, int y)
  - *The x, y* arguments are the latest mouse position (in GLUT window coordinates)

```
void motion(int x, int y) {
    ...
}
```

Mouse Positioning

- The GLUT screen coordinate increase the origin to the top-left corner, x+ to the right and y+ to the bottom by 1 pixel.
- In OpenGL, the 2D drawing coordinate has the origin at the bottom-left corner, x+ is increasing to the right, y+ is increasing upwards.



Drawing geo at cursor location

```
void mouse(int button, int state, int x, int y) {
    if(button==GLUT_RIGHT_BUTTON && state==GLUT_DOWN)
        exit(0);
    if(button==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
        g_mousemove = true;
    else if(button==GLUT_LEFT_BUTTON && state==GLUT_UP)
        g_mousemove = false;
}
void motion(int mx, int my) {
    int w = glutGet(GLUT_WINDOW_WIDTH);
    int h = glutGet(GLUT_WINDOW_HEIGHT);
    float x = (float) 10 * (mx - w*0.5) / w; // 0~600(x+right) => -5~5(x+
right)
    float y = (float) 10 * (h*0.5 - my) / h; // 0~600(y+down) => x -5~5(y+ up)
    if (g_mousemove) {
        geo->setPosition(glm::vec3(x, y, 0));
    }
    glutPostRedisplay();
}
```

If both a mouse button and ALT key are pressed

```
void mouse(int button, int state, int x, int y)
{
    specialKey = glutGetModifiers();
    if((state==GLUT_DOWN)&&(specialKey == GLUT_ACTIVE_ALT))
        {
        if (button == GLUT_LEFT_BUTTON) {
                red = 1.0; green = 0.0, blue = 0.0;
        }
        else if (button = GLUT_MIDDLE_BUTTON){
                red = 0.0; green = 1.0, blue = 0.0;
        }
        ...
}
```

Idle Callback

- glutIdleFunc(void (*func)(void)) callback is executed when there is no event.
- Idle is used for animation, *e.g. rotating square*

```
void idle() {
/* change something */
    t += dt
    glutPostRedisplay();
}

void display() {
    glClear();
/* draw something that depends on t */
    glutSwapBuffers();
}
```

- Idle's default callback function is NULL.

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
- Every GLUT program must have **glutDisplayFunc(display)**.

glutPostRedisplay

- 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
- 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 buffering
  - Double buffering initialization
    - glutInitDisplayMode(**GLUT_DOUBLE**| GLUT_RGB)
  - Clear the buffer at the beginning of the display callback
    - glClear(**GL_COLOR_BUFFER_BIT** | ...)
  - Swap the buffer at the end of the display callback
    - **glutSwapBuffers()**

The *Reshape* callback

- **glutReshapeFunc(reshape)** callback reconfigure the window shape.
- void reshape(int w, int h)
  - Return the window width and height.
  - This callback automatically calls redisplay.
- Reshape callback is a good place to put the viewing functions since it is called the first time the window is opened.

Example Reshape

```
void reshape(int w, int h) {
    g_aspectRatio = (float) (w/h);
    g_Projection = glm::perspective(g_fovy, g_aspect, g_near, g_far);

    glViewport(0, 0, w, h);
    glutPostRedisplay();
}
```