INTERACTION

Slides modified from Angel book 6e

Objectives

- 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

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

- Incremental (Relative) Devices

- 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 Devices

- Consider the C and C++ code
  - C++: `cin >> x;
  - C: `scanf`("%d", &x);
- What is the input device?
  - Can’t tell from the code
  - Could be keyboard, file, output from another program
- The code provides logical input
  - A number (an int) is returned to the program regardless of the physical device

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

X Window Input

- The X Window System introduced a client-server model for a network of workstations
  - Client: OpenGL program
  - Graphics Server: bitmap display with a pointing device and a keyboard

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)`

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

- Recall that the last line in `main.c` for a program using GLUT must be `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
- In `main.c`
  - `glutDisplayFunc(mydisplay)` identifies the function to be executed
  - Every GLUT program must have a display callback

Posting redisplay

- 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
WORKING WITH CALLBACKS

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

The mouse callback

```c
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

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

```c
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

```cpp
def mydisplay():
    ...
```

Drawing squares at cursor location

```cpp
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 */
    points[i] = point2(x+size, y+size);
    points[i+1] = point2(x-size, y+size);
    points[i+2] = point2(x-size, y-size);
    points[i+3] = point2 x+size, y-size);
    i+=4
}
```

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

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

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

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

Defining a simple menu

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

menu_id = glutCreateMenu(mymenu);
glutAddMenuEntry("clear Screen", 1);
gluAddMenuEntry("exit", 2);
glutAttachMenu(GLUT_RIGHT_BUTTON);
```

entry in parent menu

Menu actions

- Menu callback
  ```c
  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)`
Other functions in GLUT

- Dynamic Windows
- Create and destroy during execution
- Subwindows
- Multiple Windows
- Changing callbacks during execution
- Timers
- Portable fonts (deprecated)