Input and Interaction

CS 537 Interactive Computer Graphics
Prof. David E. Breen
Department of Computer Science

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

Physical Devices

- Mouse
- Trackball
- Light pen
- Data tablet
- Joy stick
- Space ball

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

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
    - Trackball
  - Logical Properties
    - What is returned to program via API
      - A position
      - An object identifier
      - A scalar value
• Modes
  - How and when input is obtained
    - Request or event
### 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:
  ```c
  glutMouseFunc(mymouse)
  ```

GLUT callbacks

GLUT recognizes a subset of the events recognized by most/all window systems (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`
  ```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
**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 displays
  - See the program `single_double.c` for an example with a rotating cube

**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(GLUT_RGB | GLUT_DOUBLE)`
  - At the end of the display callback buffers are swapped
    ```c
    void mydisplay()
    {
        glClear();
        /* 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
    ```c
    void myIdle() {
        /* change something */
        t += dt;
        glutPostRedisplay();
    }
    ```

**Using globals**
- The form/interface of all GLUT callbacks is fixed
  - `void myDisplay()`
  - `void myMouse(GLint button, GLint state, GLint x, GLint y)`
- Must use globals to pass information to callbacks
  ```c
  float t; /*global */
  void myDisplay()
  {
      /* draw something that depends on t */
      glutSwapBuffers();
  }
  ```

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

```c
void mymousie(GLint button, GLint state, GLInt x, GLInt y)
```

- Is called with the following information
  - which button (GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, GLUT_RIGHT_BUTTON) caused event
  - state of that button (GLUT_UP, GLUT_DOWN)
  - Position in window

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);
}
```

Drawing squares at cursor location

```c
void drawSquare(int x, int y)
{
  y=h-y; /* invert y position */
  points[i++] = point2(x+size, y+size);
  points[i++] = point2(x-size, y+size);
  points[i++] = point2(x-size, y-size);
  points[i++] = 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)
• Calls drawSquare if mouse is moving in window and any button is depressed
• Function is called with mouse’s (x, y) location at the time of the event

Using the motion callback

• We can draw squares without depressing a button using the passive motion callback
  - glutPassiveMotionFunc(drawSquare)
• The magnitude of motion that triggers this event is system dependent

The entry callback

• The cursor generates an entry event whenever it enters or leaves the OpenGL window
• The callback for this event is registered with glutEntryFunc()
  void glutEntryFunc(void (*f) (int state))
• Event specifies state of entry
  - (GLUT.ENTERED, GLUT_LEFT)

Using the keyboard

glutKeyboardFunc(mykey)
void mykey(unsigned char key,
           int x, int y)
  - Is called with ASCII code of key depressed and mouse location
void mykey(...) {
    if(key == 'q' || key == 'Q')
      exit(0);
}

Special 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')......
• glutSpecialFunc(myspecial) specifies the callback function that is called when a special key (i.e., a function or arrow key) is depressed

Special 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')......
• glutSpecialFunc(myspecial) specifies the callback function that is called when a special key (i.e., a function or arrow key) is depressed

Modalier Keys

• Can also check if one of the modifiers
  - GLUT_ACTIVE_SHIFT
  - GLUT_ACTIVE_CTRL
  - GLUT_ACTIVE_ALT
  is depressed with glutGetModifiers()

if(glutGetModifiers() == GLUT_ACTIVE_CTRL &&
   (key == 'c' || (key == 'C')))
  exit(0);

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

Example Reshape

```c
// We will revisit this once we have learned about viewing
void myReshape(int w, int h)
{
    glViewport(0, 0, w, h); /* Make viewport and window the same size */
    // More to come
}
```

Timers

```c
// change color each second
glutTimerFunc(1000, timerColor, 0);
// change the shape after five seconds
glutTimerFunc(5000, timerShape, 0);

void timerColor(int value){
    // get new color or a value in [0,1]
    r = (1.0*(random()%256))/256.0;
    g = (1.0*(random()%256))/256.0;
    b = (1.0*(random()%256))/256.0;
    // draw it + reinitialize timer
    glutPostRedisplay();
    glutTimerFunc(1000, timerColor, 0);
}
```

Redefining Callbacks

- Callback functions can be redefined
- Change binding during program execution
- Callbacks can be undefined
  - `glutReshapeFunc(NULL);`

```c

```
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

Defining a simple menu

```c
#include <glut/glut.h>

int main(int argc, char **argv) {
  glutInit(&argc, argv);
  glutCreateWindow("Simple Menu Example");

  menu_id = glutCreateMenu(mymenu);
  glutAddMenuEntry("Clear Screen", 1);
  glutAddMenuEntry("Exit", 2);
  glutAttachMenu(GLUT_RIGHT_BUTTON);

  glutMainLoop();
  return 0;
}
```

Entries that appear when right button depressed include:
- Identifiers: clear screen, exit

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

Submenu example

```c
void createGLUTMenus()
{
  int menu,submenu;

  submenu = glutCreateMenu(processMenuEvents);
  glutAddMenuEntry("Red", RED);
  glutAddMenuEntry("Blue", BLUE);
  glutAddMenuEntry("Green", GREEN);

  menu = glutCreateMenu(processMenuEvents);
  glutAddMenuEntry("White", WHITE);
  glutAddSubMenu("RGB Menu", submenu);
  glutAttachMenu(GLUT_RIGHT_BUTTON);
}
```

Submenu example

```c
void processMenuEvents(int option)
{
  switch (option)
  {
    case RED : red = 1.0; green = 0.0; blue = 0.0; break;
    case GREEN : red = 0.0; green = 1.0; blue = 0.0; break;
    case BLUE : red = 0.0; green = 0.0; blue = 1.0; break;
    case WHITE : red = 1.0; green = 1.0; blue = 1.0; break;
  }
  glutPostRedisplay();
}
```

http://www.lighthouse3d.com/opengl/glut
Dynamic, Multiple Windows and Subwindows

```c
int glutCreateWindow(char *name)
void glutDestroyWindow(int id)
int glutGetWindowId()  // returns id of current window
void glutSetWindow(int id)
    // Set current window. This lets you change its properties, e.g. size
    // OpenGL renders to the current window!
void glutCreateSubWindow(int parent, int x, int y, int width, int height)
    // Posts a redisplay for a particular window
    // Menus are defined for specific windows
```

Subwindow Example

```c
int mainWindow, subWindow1;
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
    glutInitWindowPosition(100, 100);
    mainWindow = glutCreateWindow("SnowMen from 3D-Tech");
    glutKeyboardFunc(processNormalKeys);
    glutReshapeFunc(changeSize);
    glutDisplayFunc(renderScene);
    glutIdleFunc(renderSceneAll);
    subWindow1 = glutCreateSubWindow(mainWindow, 0, 0, SUB_WIDTH, SUB_HEIGHT);
    glutDisplayFunc(renderScene1);
}
```

Displaying in Multiple Windows

- Each window has its own graphics context
- For each window set up a separate VAO, buffers and compiled shader programs
- Creating a window / setting its ID makes it current
- Subsequent OpenGL structures will be associated with current window
- Be sure that the correct window is current before executing its display function

Suggestions for HW3

**Structure of main()**
```c
glutInit()  // Setup glut
    // Specify callback functions for Window 1
    // Specify menus for Window 1
    // Specify callback functions for Window 2
    // Specify menus for subwindows
    glutMainLoop();
```

**Structure of each init()**
```c
    // Compute geometric data needed for this window
    // Create a vertex array object
    glGenVertexArrays( 1, &vao[0] );
    glBindVertexArray( vao[0] );
    // Create and initialize a buffer object
    glEnableVertexAttribArray(0); // vertex data
    glEnableVertexAttribArray(1); // color data
    // Load shaders and use the resulting shader program
    // Initialize the vertex position attribute from the vertex shader
    // Initialize the vertex color attribute from the vertex shader
    glClearColor( 0, 0, 0, 1.0 ); // black background
```

**Suggestions for HW3**

**Structure of display callback function**
```c
glClear( GL_COLOR_BUFFER_BIT );     // clear the window
    // update triangle vertices for animation
    // update circle vertices for animation
    // send vertex data to VBO
    glVertexAttribPointer(1, windowVertices);
    // draw the triangle
    glVertexAttribPointer(2, windowCircles);
    // draw the circle
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, windowVertices);
```

Suggestions for HW3

5/25/17
Suggestion for HW3

• Structure of idle function

// Do calculations needed to animate objects
// e.g. increment angle or time

glutPostWindowRedisplay(mainWindow);
glutPostWindowRedisplay(subWindow);
glutPostWindowRedisplay(window2);