Input and Interaction

CS 432 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

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
      - A scalar value
• Modes
  - How and when input is obtained
    • Request or event

Physical 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

Incremental (Relative) Devices
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)
  ```
  - mouse callback function

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

Using globals

- The form 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:
  ```
  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

`glutMouseFunc(mymouse)`

`void mymousse(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

```
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 mymousse(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=h-y; /* invert y position */
    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)`

- 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

- Mouse 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 returns state of entry
  
  - (GLUT_ENTERED, GLUT_LEFT)

Using the keyboard

`glutKeyboardFunc(mykey)`

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

- Returns ASCII code of key depressed and mouse location

```c
void mykey()
{
  if(key == b'Q' || key == b'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

Using the keyboard

```c
void glutKeyboardFunc(mykey)
{
  if key is depressed
    return ASCII code of depressed key and mouse location
}
```

```c
void mykey()
{
  if(key == GLUT_KEY_F1)
    exit(0);
}
```

Modifier 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)
    {
      if(key == 'c') || (key == 'C'))
      exit(0);
    }

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

Angel: Interactive Computer Graphics 5E © Addison-Wesley 2009
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)`

```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
void myReshape(int w, int h)
{
    glViewport(0, 0, w, h); /* Make viewport and window the same size */
    // More to come
}
```

Timers

- Callback function that is triggered after a specified number of milliseconds

```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);`

Example

```c
void myReshape(int w, int h)
{
    glViewport(0, 0, w, h);
    // More to come
}
```
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
#define GLUT_MAINMENU

void main() {
  int menu_id = glutCreateMenu(mymenu);
  glutAddmenuEntry("Clear Screen", 1);
  glutAddmenuEntry("Exit", 2);
  glutAttachMenu(GLUT_RIGHT_BUTTON);
}
```

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 WHITE : red = 1.0; green = 1.0; blue = 1.0; break;
  }
  glutPostRedisplay();
}
```

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

- Set current window. This lets you change its properties, e.g. size
-Menus are defined for specific windows

Subwindow Example

```c
int mainWindow, subWindow;
int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
    glutInitWindowSize(100, 100);
    mainWindow = glutCreateWindow("SnowMen from 3D-Tech");
    // keyboard stuff
    glutKeyboardFunc(processNormalKeys);
    // reshape function
    glutReshapeFunc(changeSize);
    // display and idle function
    glutDisplayFunc(renderScene);
    glutIdleFunc(renderSceneAll);
    subWindow = glutCreateSubWindow(mainWindow,
        border, border, w-2*border, h/2 - border*3/2);
    // Must register a display function for each window
    glutDisplayFunc(renderScene1);
}
```