Input and Interaction

Ed Angel
Professor of Computer Science, Electrical and Computer Engineering, and Media Arts
University of New Mexico
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

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 joystick 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:
  \[
  \text{glutMouseFunc(mymouse)}
  \]

  mouse callback function
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 redisplays

• 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
• 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 display
  - 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(GL_RGB | GL_DOUBLE)`
  - At the end of the display callback buffers are swapped

```c
void mydisplay()
{
    glClear(GL_COLOR_BUFFER_BIT | ...);

    /* draw graphics here */

    glutSwapBuffers()
}
```

Angel: Interactive Computer Graphics 4E © Addison-Wesley 2005
Using the idle callback

- The idle callback is executed whenever there are no events in the event queue
  - \texttt{glutIdleFunc(myidle)}
  - Useful for animations

\begin{verbatim}
void myidle() {
  /* change something */
  t += dt
  glutPostRedisplay();
}
\end{verbatim}

\begin{verbatim}
Void mydisplay() {
  glClear();
  /* draw something that depends on t */
  glutSwapBuffers();
}
\end{verbatim}
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 
  }