Input and Interaction
Overview

- Introduce the basic input devices
- Event-driven input
- Introduce double buffering for smooth animations
- Programming event input with jogl
- Picking
- Rubberbanding
- Display Lists
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 standard input/output (System.out, System.in, System.err)
- What is the input device?
  - Can’t tell from the code
  - Could be keyboard, file, output from another program
- The code provides *logical input*
  - Bytes 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
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
  - Not in jogl
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
Java event listeners
- Objects that implements a specific interface
- Registers interest with an object
  - Observable pattern
- GLEventListener, KeyListener, MouseMotionListener, MouseListener
Redisplay

• From event listeners
  – Call `display()` or `repaint()` (on `GLCanvas` object) at the end

• With an `Animator` or similar external thread
  – Copy the current display state to local variables

• Update and set a flag `or`

• Record changes and set a flag
  – Process in `display()`
Animating a 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

GLCapabilities cap = new GLCapabilities();
cap.setDoubleBuffered(true);
final GLCanvas canvas = GLDrawableFactory.getFactory().createGLCanvas(cap);
The mouse callbacks

MouseListener:
public void mouseExited(MouseEvent e)
public void mouseEntered(MouseEvent e)
public void mousePressed(MouseEvent e)
public void mouseClicked(MouseEvent e)
public void mouseReleased(MouseEvent e)

MouseMotionListener:
public void mouseDragged(MouseEvent e)
public void mouseMoved(MouseEvent e)

MouseWheelListener

e.getX(), e.getY()
The keyboard callbacks

KeyListener:
public void keyPressed(KeyEvent e)
public void keyReleased(KeyEvent e)
public void keyTyped(KeyEvent e)

e.getKeyCode()
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
  – Can also use query functions
    • int viewport[] = new int[4];
    • gl.glGetIntegerv(GL.GL_VIEWPORT, viewport);
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

original

reshaped
Example reshape

• This reshape preserves shapes by making the view port and world window have the same aspect ratio

public void reshape(GLDrawable drawable, int x, int y, int w, int h) {
    GL gl = drawable.getGL();
    double scale = (w <= h) ? (double) h / (double) w :(double) w / (double) h;
    gl.glViewport(x, y, w, h);
    gl.glMatrixMode(GL.GL_PROJECTION); // switch matrix mode
    gl.glLoadIdentity();
    if (w <= h)
        gl.glOrtho(XMIN, XMAX, YMIN*scale, YMAX*scale, -1.0, 1.0);
    else
        gl.glOrtho(XMIN*scale, XMAX*scale, YMIN, YMAX, -1.0, 1.0);
    gl.glMatrixMode(GL.GL_MODELVIEW); // return to modelview mode
}
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

• AWT/Swing in Java
Menus

• AWT supports menus in the menu bar and popup menus
  – Popup menus are hidden functions
  – So, the menu bar is preferred

• Four steps with AWT
  – Define entries for the menu
  – Define action for each menu item
    • Action carried out if entry selected
  – Add menu to menu bar
  – Add menu bar to the frame
Defining a Simple Menu, in *main*()

```java
final static String CMD_CLEAR = "clear",
    CMD_QUIT = "quit";

SimpleRenderer renderer = new SimpleRenderer();
MenuBar mb = new MenuBar();
Menu m = new Menu("Options");
MenuItem clear = new MenuItem("Clear Window");
MenuItem quit = new MenuItem("Quit");
clear.setActionCommand(CMD_CLEAR);
clear.addActionListener(renderer);
quit.setActionCommand(CMD_QUIT);
quit.addActionListener(renderer);
m.add(clear); m.add(quit);
mb.add(m);
simpleFrame.setMenuBar(mb);
...
drawable.addGLEventListener(renderer);
```
static class SimpleRenderer implements GLEventListener, MouseListener, KeyListener, ActionListener {
...
    public void actionPerformed(ActionEvent e) {
        String mc = e.getActionCommand();
        if (mc.equals(CMD_QUIT)) System.exit(0);
        else if (mc.equals(CMD_CLEAR)) {
            cmd = CLEAR;
        }
    }
}
Picking

• Identify a user-defined object on the display
• In principle, it should be simple because the mouse gives the position and we should be able to determine to which object(s) a position corresponds

• Practical difficulties
  – Pipeline architecture is feed forward, hard to go from screen back to world
  – Complicated by screen being 2D, world is 3D
  – How close do we have to come to object to say we selected it?
Three Approaches

• Hit list
  – Most general approach but most difficult to implement
• Use back or some other buffer to store object ids as the objects are rendered
• Rectangular maps
  – Easy to implement for many applications
  – See paint program in text
Rendering Modes

- OpenGL can render in one of three modes selected by `glRenderMode(mode)`
  - `GL_RENDER`: normal rendering to the frame buffer (default)
  - `GL_FEEDBACK`: provides list of primitives rendered but no output to the frame buffer
  - `GL_SELECTION`: Each primitive in the view volume generates a hit record that is placed in a name stack which can be examined later
Selection Mode Functions

- `glSelectBuffer()` : specifies name buffer
- `glInitNames()` : initializes name buffer
- `glPushName(id)` : push id on name buffer
- `glPopName()` : pop top of name buffer
- `glLoadName(id)` : replace top name on buffer

- id is set by application program to identify objects
Using Selection Mode

- Initialize name buffer
- Enter selection mode (using mouse)
- Render scene with user-defined identifiers
- Reenter normal render mode
  - This operation returns number of hits
- Examine contents of name buffer (hit records)
  - Hit records include id and depth information
Selection Mode and Picking

- As we just described it, selection mode won’t work for picking because every primitive in the view volume will generate a hit.
- Change the viewing parameters so that only those primitives near the cursor are in the altered view volume.
  - Use `gluPickMatrix` (see text for details)
Using Regions of the Screen

• Many applications use a simple rectangular arrangement of the screen
  – Example: paint/CAD program

```
+----------------+------------------+
|               | tools            |
|               | drawing area     |
| menus         |                  |
+----------------+------------------+
```

• Easier to look at mouse position and determine which area of screen it is in than using selection mode picking
Using another buffer and colors for picking

• For a small number of objects, we can assign a unique color (often in color index mode) to each object
• We then render the scene to a color buffer other than the front buffer so the results of the rendering are not visible
• We then get the mouse position and use `glReadPixels()` to read the color in the buffer we just wrote at the position of the mouse
• The returned color gives the id of the object
Writing Modes

Source

bitwise logical operation

read_pixel

frame buffer

write_pixel

application
XOR write

• Usual (default) mode: source replaces destination \( (d' = s) \)
  – Cannot write temporary lines this way because we cannot recover what was “under” the line in a fast simple way

• Exclusive OR mode (XOR) \( (d' = d \oplus s) \)
  – \( x \oplus y \oplus x = y \)
  – Hence, if we use XOR mode to write a line, we can draw it a second time and line is erased!
Rubberbanding

• Switch to XOR write mode
• Draw object
  – For line can use first mouse click to fix one endpoint and then use motion callback to continuously update the second endpoint
  – Each time mouse is moved, redraw line which erases it and then draw line from fixed first position to to new second position
  – At end, switch back to normal drawing mode and draw line
  – Works for other objects: rectangles, circles
Rubberband Lines

initial display

mouse moved to new position

draw line with mouse in XOR mode

second point

first point

original line redrawn with XOR

new line drawn with XOR
XOR in OpenGL

• There are 16 possible logical operations between two bits
• All are supported by OpenGL
  – Must first enable logical operations
    • `glEnable(GL_COLOR_LOGIC_OP)`
  – Choose logical operation
    • `glLogicOp(GL_XOR)`
    • `glLogicOp(GL_COPY)` (default)
Immediate and Retained Modes

• Recall that in a standard OpenGL program
  – Once an object is rendered there is no memory of it
  – To redisplay it, we must re-execute the code for it
  – Known as *immediate mode graphics*
  – Can be especially slow if the objects are complex and must be sent over a network

• Alternative is define objects and keep them in some form that can be redisplayed easily
  – *Retained mode graphics*
  – Accomplished in OpenGL via *display lists*
Display Lists

• Conceptually similar to a graphics file
  – Must define (name, create)
  – Add contents
  – Close

• In client-server environment, display list is placed on server
  – Can be redisplayed without sending primitives over network each time
Display List Functions

• Creating a display list

```c
GLuint id;

void init()
{
    id = glGenLists( 1 );
    glNewList( id, GL_COMPILE );
    /* other OpenGL routines */
    glEndList();
}
```

• Call a created list

```c
void display()
{
    glCallList( id );
}
```
Display Lists and State

- Most OpenGL functions can be put in display lists
- State changes made inside a display list persist after the display list is executed
- Can avoid unexpected results by using `glPushMatrix` and `glPushAttrib` upon entering a display list and `glPopAttrib` and `glPopMatrix` before exiting
Hierarchy and Display Lists

• Consider model of a car
  - Create display list for chassis
  - Create display list for wheel

```c
glNewList( CAR, GL_COMPILE );
glCallList( CHASSIS );
glTranslatef( … );
glCallList( WHEEL );
glTranslatef( … );
glCallList( WHEEL );
...
glEndList();
```
Display List alternatives

- Scene graphs
- Vertex Buffer Objects
Example

• Mouse input, animation, picking
Warning: some examples from today might crash the lab computers… (will fix it with Mikael, see course home page)

Next time: Geometry and transformations