CS148: Introduction to Computer Graphics and Imaging

Interactive Techniques

Ben Fry’s Zipdecode

http://acg.media.mit.edu/people/fry/zipdecode/
Topics

Responding to input
- Polling vs. interrupts
- Event queues
- Interaction with the window system

Building blocks for interaction
- Picking and tracking
- Dragging

Basic application patterns
- Model-view-controller (MVC)
- Game input/simulation loop

Polling vs. Interrupts

Polling - Read state of the device directly
- Read when needed
- Read at some regular rate (e.g. 100 Hz)

Interrupt - Notify when device state changes
- Keyboard sends “make” when the key is pressed, and “break” when the key is released
Changes Cause Events

OS places events in a queue with time-stamp

- May track state
  - Keys (SHIFT-A = "A")
  - Mouse double-click

- May store state of other devices at time of event
  - Add continuously changing devices like the mouse

- May reorder events based on priority
  - Input events from the keyboard have priority over PostRedisplay

- Possibly merge events
  - Combine multiple relative mouse motion events so current position is returned

Mouse

Combination of polling and interrupts

Polling: Microcontroller in the mouse reads the position at least 100 times per second (fancy mice for games over 1000 Hz)

Interrupt: Microcontroller sends a message if the mouse moves or button goes up or down

// called when mouse buttons are pressed/released
void glutMouseFunc(
    void (*func)(int button, int state, int x, int y));

// called when the mouse moves with mouse button pressed
void glutMotionFunc(void (*func)(int x, int y));

// called when the mouse moves with no buttons pressed
void glutPassiveMotionFunc(void (*func)(int x, int y));
Gamepads

SONY Playstation 3

Microsoft XBOX 360

CS148 Lecture 6
Pat Hanrahan, Fall 2010

---

Joystick / Gamepad

Polled at some rate ...

```c
// state of joystick
// button state in the bits of "button"
// axis position between -1000 and 1000
void joystick( int button,
               int xaxis, int yaxis, int zaxis );

void glutJoystickFunc(  
    void (*func)(int button, int x, int y, int z),
    int pollrate /* in number of milliseconds */ );

nbbuttons = glutDeviceGet(GLUT_JOYSTICK_BUTTONS);
naxes = glutDeviceGet(GLUT_JOYSTICK_AXIS);

STJoystick interface more powerful: multiple joysticks, buttons, axes, hats, balls
```

CS148 Lecture 6
Pat Hanrahan, Fall 2010
Comparison: Polling vs. Interrupts

Polling - Read state of the device directly
  ■ Good if the device continuously changes state
  ■ Bad because could miss a state change
  ■ Bad because it has high overhead

Interrupt - Notify when a device state changes
  ■ Efficient if the state changes are intermittent
  ■ Need to track state; may lose track of state
  ■ Need to select the events of interest

Window Manager

Manages the windows on the screen
  ■ open, close, quit, active, hide, ...

Output
  ■ Create virtual framebuffer in physical fb
  ■ Only draw in your windows

Input
  ■ Mouse events are routed to the window under the mouse
  ■ Keyboard events are routed to the active window
  ■ Window system will intercept some events
Interacting with Objects

Different Way to Write a GL Program

Immediate mode
- Drawing by calling gl functions
- Procedural description with minimal state
- Unbounded complexity

Retained mode
- What is drawn is stored in a data structure
- Draw by traversing the data structure

Display lists
- Store list of gl commands
Points/Polygons Data Structures

```c
STPoint3 verts[8] = {
    {-1.,-1.,-1.},
    { 1.,-1.,-1.},
    { 1., 1.,-1.},
    {-1., 1.,-1.},
    {-1.,-1., 1.},
    { 1.,-1., 1.},
    { 1., 1., 1.},
    {-1., 1., 1.},
};
```

```c
int polys[6][4] = {
    {0,3,2,1},
    {2,3,7,6},
    {0,4,7,3},
    {1,2,6,5},
    {4,5,6,7},
    {0,1,5,4}
};
```

CS148 Lecture 3
Pat Hanrahan, Fall 2010

---

Drawing Points/Polygons Models

```c
face(int poly[4]) {
    glBegin(GL_POLYGON);
    glVertex3fv(poly[0]);
    glVertex3fv(poly[1]);
    glVertex3fv(poly[2]);
    glVertex3fv(poly[3]);
    glEnd();
}
```

```c
cube() {
    for( int i = 0; i < n; i++ )
        face(polys[i]);
}
```

More efficient to use `glDrawElements()`
Document Object Model (DOM)

```html
<html>
<head>
<script> javascript.js </script>
</head>
<body>
<div>
  <p>Hello</p>
  <img src="bunny.jpg" />
</div>
...
</body>
</html>
```

Hierarchical Tree of Different HTML tags

Hierarchical Scene Graph

Directed acyclic graph (DAG) of Object Instances
Routing Mouse Events to Objects

Three issues

- What objects overlap the cursor?
  - Need to implement method for picking objects
- What happens if cursor overlaps multiple objects?
  - Prioritize objects
  - e.g. the visible object or the object drawn last
- How to handle object hierarchy
  - Send events to enclosing scope
  - Down then up the hierarchy?

Hit Testing / Picking

Methods

- Bounding rectangle
- Hit method for objects
- Pick support during rendering (OpenGL)
- Object tags
  - Render object-id into a secondary framebuffer
  - Read id at the pixel location of the mouse
Two Types of Buttons

Push button
- Press inside and release inside

Menu button
- Press inside or outside and release inside

Flash menu button examples.

Push Button State Transitions

Product of states: Outside vs. Over / Up vs. Down
Three views of the button: Idle, Over, Down

Diagram:

Outside
  Idle ──► Over
  OnRollOver
  OnRollOut
  OnReleaseOutside

Over
  Idle
  OnRollOver ──► Over
  OnRollOut
  OnRelease
  OnDragOver
  OnPress

Up
  Idle
  OnRollOver ──► Over
  OnRollOut
  OnRelease
  OnDragOver
  OnPress

Down
  Over
  OnDragOver ──► Down
  OnDragOut

End of diagram
Push Button State Transitions

Product of states: Outside vs. Over / Up vs. Down
Three views of the button: Idle, Over, Down

```
Outside       Over

Up

Idle     Over
OnRollOver
OnRollOut
OnRelease
OnPress

Down
Idle     Down
OnDragOver
OnDragOut
```

Dragging

```
this.startdrag(true, xmin, xmax, ymin, ymax);
   Attaches current object to the cursor
   Position of object updated when cursor moves
   Constrain to lie within rectangle

stopdrag();
```
Sketchpad Video

Basic Application Patterns
Model-View-Controller

MVC

Model
- The “thing”
  - e.g. a document
  - e.g. an avatar
- Object methods implement abstraction

View
- The visual representation of the “thing”
  - e.g. the slide view vs the presentation view
  - e.g. a world filled with avatars

Controller
- Connect input events and model method calls
  - e.g. drag to select text, press button to speak
Model-View-Controller Design Pattern

\[ m = \text{new Model();} \]
\[ m.\text{addView(v1);} \]
\[ m.\text{addView(v2);} \]

\[ v1 = \text{new ViewA(m,c);} \quad c = \text{new Controller(m);} \]
\[ v2 = \text{new ViewB(m,c);} \]

\[ \text{v.onChange = function()} \{ \]
\[ \quad \text{v.c.setValue(v.value);} \]
\[ \} \]
Model-View-Controller Design Pattern

```javascript
v.onChange() = function() {
  c.setValue = function(s) {
    v.c.setValue(v.value);
    c.m.setValue(s);
    c.m.onUpdate();
  }
}
```

CS148 Lecture 6
Pat Hanrahan, Fall 2010

Model-View-Controller Design Pattern

```javascript
m.onUpdate = function() {
  for v in m.views:
    v.onUpdate();
}
```

```javascript
c.setValue = function(s) {
  c.m.setValue(s);
  c.m.onUpdate();
}
```
Model-View-Controller Design Pattern

```javascript
m.onUpdate = function() {
    for (v in m.views) {
        v.onUpdate();
    }
}
```

```javascript
v.onUpdate = function() {
    m.draw(v);
}
```

CS148 Lecture 6

Pat Hanrahan, Fall 2010

---

Games
Game Controller Design Pattern

forever @ 60 Hz:
readInputDevices()
updateControllers()
processMovement()
handleCollisions()
draw()

Things to Remember

Flow from input to events to method invocation
- Input devices generate events
- Events are routed to objects
- Pointer events require “hit testing”

Interaction “building blocks”
- Filtering/smoothing raw inputs
- Behavioral state-machines
- Tracking, clicking, dragging, selecting, ...

Application design patterns
- Model-view-controller (MVC) pattern
- Game controller pattern
Fragments of Interactive Techniques

Buttons and menus
  Clicking - “Input on output”
  Tracking – temporarily capture the mouse

Selection
  Capture mouse and keyboard
  Set current object for actions

Snapping and dragging
  Handles to move “control” points
  Enforce constraints like snapping to a point