Interactive Techniques

Ben Fry’s Zipdecode

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

Responding to Input
- Polling vs. interrupts
- Event queues

Building blocks for interaction
- Picking and tracking
- Dragging
- Interactive techniques

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

Polling and Interrupts

Polling – Regularly read the state of the device
  Good for devices that constantly change
  - Mouse sends messages
    - 100 times per second
    - State of the buttons
    - Relative motion of the mouse ($\Delta x$, $\Delta y$)
  - Joystick/Gamepad interface
    - glutJoyStick( int buttonmask, int x, int y, int z )
    - STJoystick supports multiple joysticks, axes, balls, hats, and buttons

Interrupts - Send changes when they occur
  - Keyboard sends “make” and “break” messages
Event Queue

OS creates event record and places in a queue
- Time-stamp
- Queues are first-in first-out (FIFO)
- May store state of other devices at time of event
e.g. polled devices like the mouse

Window manager routes events to windows
- Active window normally receives events

Application event manager
- Event priority
  - Reorder events based on priority
e.g. keyboard has priority over PostRedisplay
- Event coalescing
  - Combine mouse motions so current position is returned

Demo of input.py

Interacting with Objects
Browser Document Object Model (DOM)

```html
<html>
  <head>
    <script>:js</script>
  </head>
  <body>
    <div>
      <p>text</p>
      <img src="url" />
    </div>
    ...
    ...
  </body>
</html>
```
Flash Movies

<table>
<thead>
<tr>
<th>Movie</th>
<th>Instances have attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MovieClip (Sprite)</td>
<td>name</td>
</tr>
<tr>
<td>Text</td>
<td>position, rotation, scale</td>
</tr>
<tr>
<td>Shape</td>
<td>color and transparency</td>
</tr>
<tr>
<td>Image</td>
<td>“actions”</td>
</tr>
<tr>
<td>Video</td>
<td></td>
</tr>
<tr>
<td>Sound</td>
<td></td>
</tr>
<tr>
<td>Button</td>
<td></td>
</tr>
<tr>
<td>Action (Javascript)</td>
<td></td>
</tr>
<tr>
<td>MovieClip</td>
<td></td>
</tr>
</tbody>
</table>

Routing Mouse Events to Objects

Three issues

- What objects overlap the cursor?
  - Check whether cursor overlaps bounding rectangle
  - Check whether cursor overlaps the pixels in the object

- 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?
Flash Button Tracking

Tracking: Convert in/out state to enter/leave events

Flash MenuButton Tracking

Tracking: Convert in/out state to enter/leave events
Draggling

```javascript
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();
```

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
Sketchpad Video

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
Basic Application Patterns

Games
Game Controller Design Pattern

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

Position Control

Direct:
- \( \text{obj.pos} = \text{device.value} \)

1\(^{st}\), 2\(^{nd}\) derivatives
- “Asteroids” program
### Input Filtering

Character rotation over time.

Character speed based on controller input.

"Classic Super Mario 64 Third-Person Control and Animation"
Steve Rabin in Game Programming Gems 2

### Animation Graphs

```
CS148 Lecture 6  Pat Hanrahan, Fall 2009
```
Model-View-Controller Design Pattern

```java
m = new Model();
m.addView(v1);
m.addView(v2);

v1 = new ViewA(m, c);
c = new Controller(m);
v2 = new ViewB(m, c);
```
Model-View-Controller Design Pattern

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

Model-View-Controller Design Pattern

```javascript
v.onchange = function() {
  v.c.setValue(v.value);
}
```
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);
}
```
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
- Game controller pattern
- Model-view-controller (MVC) pattern