# Basic User Interface Concepts

A short outline of input devices and the implementation of a graphical user interface is given:

- Physical input devices used in graphics
- Virtual devices
- Polling is compared to event processing
- UI toolkits are introduced by generalizing event processing

# Physical Devices

Actual, physical input devices include:

- dials (potentiometers)
- selectors
- pushbuttons
- switches
- keyboards (collections of pushbuttons called "keys")
- trackballs (relative motion)
- mice (relative motion)
- joysticks (relative motion, direction)
- tablets (absolute position)
- etc.

# Virtual Devices

Devices can be classified according to the kind of value they return:

- Button: returns a Boolean value; can be *depressed* or *released*.
- Key: returns a character; can also be thought of as a button.
- Selector: returns an integral value (in a given range).
- Valuator: returns a real value (in a given range).
- Locator: returns a position in (2d/3D) space (usually several ganged valuators).

Each of the above is called a *virtual device*.

# Polling and Sampling

In *polling*, the value of an input device is constantly checked in a tight loop:

- To record the motion of a valuator...
- To wait for a change in status...

If a record is taken, this input mode may be called *sampling*.

Generally, polling is inefficient and should be avoided, particularly in time-sharing systems.

# Event Queries

- Device is monitored by an asynchronous process.
- Upon change in status of device, this process places a record into an *event queue*.
- Application can request read-out of queue:
  - number of events
  - 1st waiting event
  - highest priority event
  - 1st event of some category
  - all events
- Application can also
  - specify which events should be placed in queue
  - clear and reset the queue
  - etc.
- Queue reading can be blocking or non-blocking.

- The cursor is usually *bound* to a pair of valuators, typically `MOUSE_X` and `MOUSE_Y`.
- Events can be restricted to particular areas of the screen, based on the cursor position.
- Events can be very general or specific:
  - a mouse button or keyboard key is depressed
  - a mouse button or keyboard key is released
  - the cursor enters a window
  - the cursor has moved more than a certain amount
  - an `Expose` event is triggered under X which a window becomes visible
  - a `Configure` event is triggered when a window is resized
  - a timer event may occur after a certain interval
- Simple event queues just record a code for event (Iris GL).
- Better event queues record extra information such as time stamps (X windows).

# Toolkits

Event-loop processing can be generalized

1. Instead of a `switch`, use table lookup.
2. Each table entry associates an event with a *callback* function.
3. When the event occurs, the *callback* is invoked.
4. Provide an API to make and delete table entries.
5. Divide screen into parcels, and assign different callbacks to different parcels (X windows does this).

Modular UI functionality is provided through a collection of *widgets*.

- Widgets are parcels of the screen that can respond to events.
- A widget has a graphical representation that suggests its function.
- Widgets may respond to events with a change in appearance, as well as issuing callbacks.
- Widgets are arranged in a parent/child hierarchy.
- Widgets may have multiple parts, and in fact may be composed of other widgets in a hierarchy.

Some UI toolkits: Xm, Xt, SUIT, FORMS, Tk, GLUT, GLUI, QT, ...

# Picking and 3D Selection

- *Pick:* Select an object by positioning mouse over it and clicking
- *Question:* How do we decide what was picked?
  - We could do the work ourselves:
    - Map selection point to a ray
    - Intersect with all objects in scene
  - Let OpenGL/graphics hardware do the work
- *Idea:* Draw entire scene, and "pick" anything drawn near the cursor
  - Only "draw" in a small viewport near the cursor
  - Just do clipping, no shading or rasterization
  - Need a method of identifying "hits"
  - OpenGL uses a *name stack* managed by `glInitNames()`, `glLoadName()`, `glPushName()`, and `glPopName()`
  - "Names" are short integers
  - When hit occurs, copy entire contents of stack to output buffer
- *Example:*

```
glSelectBuffer(size, buffer);           /* initialize */
glRenderMode(GL_SELECT);
glInitNames();
glGetIntegerv(GL_VIEWPORT, viewport);  /* set up pick view */
glMatrixMode(GL_PROJECTION);
glPushMatrix();
glIdentity();
gluPickMatrix(x, y, w, h, viewport);
glMatrixMode(GL_MODELVIEW);

ViewMatrix();
glLoadName(1);
Draw1();
glLoadName(2);
Draw2();

glMatrixMode(GL_PROJECTION);
glPopMatrix();
glMatrixMode(GL_MODELVIEW);
hits = glRenderMode(GL_RENDER);
```

- What you get back:
  - If you click on Item 1 only:
    ```
    hits = 1,
    buffer = 1, min(z1), max(z1), 1.
    ```
  - If you click on Item 2 only:
    ```
    hits = 1,
    buffer = 1, min(z2), max(z2), 2.
    ```
  - If you click over both Item 1 and Item 2:
    ```
    hits = 1,
    buffer = 1, min(z1), max(z1), 1, 1, min(z2), max(z2), 2.
    ```

- More complex example:

```
/* initialization stuff goes here */
glPushName(1);
    Draw1();                    /* stack: 1 */
    glPushName(1);
        Draw1_1();              /* stack: 1 1 */
        glPushName(1);
            Draw1_1_1();        /* stack: 1 1 1 */
        glPopName();
        glPushName(2);
            Draw1_1_2();        /* stack: 1 1 2 */
        glPopName();
    glPopName();
    glPushName(2);
        Draw1_2();              /* stack: 1 2 */
    glPopName();
glPopName();
glPushName(2);
    Draw2();                    /* stack: 2 */
```

```
glPopName();
/* wrap-up stuff here */
```

- What you get back:
  - If you click on Item 1:
    `hits = 1,`
    `buffer = 1, min(z1), max(z1), 1.`
  - If you click on Items 1:1:1 and 1:2:
    `hits = 2,`
    `buffer = 3, min(z111), max(z111), 1, 1, 1, 2, min(z12),`
    `max(z12), 1, 2.`
  - If you click on Items 1:1:2, 1:2, and 2:
    `hits = 3,`
    `buffer = 3, min(z112), max(z112), 1, 1, 2, 2, min(z12),`
    `max(z12), 1, 2, 1, min(z2), max(z2), 2.`
- In general, if $h$ is the number hits, the following is returned.
  - `hits` $= h$.
  - $h$ *hit records*, each with four parts:
    1. The number of items $q$ on the name stack at the time of the hit (1 int).
    2. The minimum $z$ value among the primitives hit (1 int).
    3. The maximum $z$ value among the primitives hit (1 int).
    4. The contents of the hit stack, deepest element first ($q$ ints).

- Important Details:
  - Make sure that projection matrix is saved with a `glPushMatrix()` and restored with a `glPopMatrix()`.
  - `glRenderMode(GL_RENDER)` returns negative if buffer not big enough.
  - When a hit occurs, a flag is set.
  - Entry to name stack only made at next `gl*Name(s)` or `glRenderMode` call. So, each draw block can only generate at most one hit.

# Reading Assignment and News

Chapter 3 pages 89 - 127, of Recommended Text.

Please also track the News section of the Course Web Pages for the most recent Announcements related to this course.

(http://www.cs.utexas.edu/users/bajaj/graphics23/cs354/)