Event-Driven Programming

Reading

Optional:

- Foley et al., Chapter 10

Overview

- Command- vs. event-driven programming
- (Quick) intro to "libui"
- Project 0 starting point

Command-driven programming

In the more "traditional" command-driven programming, the program and input device operate alternately:

```
main()
{
  loop {
    readcommand(cmd);
    if (cmd==cmd1)
      do_cmd1();
    else if (cmd==cmd2)
      do_cmd2();
      .
      .
    endif
  } forever;
}
```

Event-driven programming

In event-driven programming, any new input triggers a "call-back function":

```
main()
{
  GUI_register(event1, func1_callback);
  GUI_register(event2, func2_callback);
  .
  .
  GUI_processEvents();
}

func1_callback() {...}

func2_callback() {...}
```


The user interface library: "libui"

The user interface library, "libui," provides:

- A main application window
- Buttons
- Toggle buttons
- Sliders
- Menus
- Drawing areas
- Text area
- File-selection dialogues
- Event handling

Project 0: Mainline

```
int
main( unsigned int argc, char **argv )
{

     /* Initialize interface */
     init_interface();

     /* Enter the main event loop. */
     /* This routine never returns. */
     uiMainLoop();
}
```


Project 0: Initialization

```
void
init_interface()
{
  /*** set size of various parts of application window ***/
  uiSetControlWidth( CONTROL_WIDTH );
  uiSetControlHeight( CONTROL_HEIGHT );
  uiSetMainWindowWidth( MAIN_WINDOW_WIDTH );
  uiSetMainWindowHeight( MAIN_WINDOW_HEIGHT );

  /*** initialize the interface library ***/
  uiInitWindows( 0, NULL, uiHasControls);

  drawWind = uiAddWindow( GAPSIZE, GAPSIZE, DRAW_WIDTH,
                          DRAW_HEIGHT, exposeCallback,
                          mouseCallback, NULL);

  /* add buttons to control area */
  uiAddButton( "Quit",      quitCallback );
  uiAddButton( "Reset",     resetRange );
  uiAddButton( "Zoom In",   zoomCallback );

  /* possible additions... */
  iterSlider =
    uiAddSlider( "Iterations", 50, 10, 200, 0, sliderCallback );
  cycleToggle =
    uiAddToggleButton( "Cycle Colors", FALSE, toggleCallback );
}
```

Project 0: Callback routines

```
/* Called when the quit button is pressed */
void
quitCallback( void )
{
  uiQuit();
  exit( 0 );
}

/* Called when slider changes */
void
sliderCallback( void )
{
   // change number of iterations
   numIterations = uiGetSliderValue( iterSlider );
   // change the color table
   reloadColorTable( numIterations );
}

/* Called when toggle button changes state */
void
toggleCallback( void )
{
   doCycle = uiGetToggleButtonState( cycleToggle );
   if (doCycle)
      startColorCycling();
}
```


Project 0: Mouse routines

```
/* (abridged) Mouse event handler */
void
mouseCallback( uiWindow *window, int x, int y,
               uiMouseAction action )
{
  uiSetCurrentWindow( window, uiOverlayPlane );
  /* Determine what to do with this mouse action */
  switch( action )
    {
    case uiLeftMouseDown:
      erase_overlay(window);
      /* store coordinates of initial position */
      break;

    case uiLeftMouseUp:
      /* calculations skipped here */
      draw_rubberband(zoomLeft, zoomTop, zoomRight,
                      zoomBottom, zoomColor);
      break;

    case uiLeftMouseDrag:
      erase_rubberband( anchor_x, anchor_y, old_x, old_y );
      /* calculations skipped here */
      draw_rubberband( anchor_x, anchor_y, old_x, old_y,
                       zoomColor);
      break;
    }
}
```


Project 0: Graphics routines

```
/* Draw a rubberband box in the window */
void
draw_rubberband( int x1, int y1, int x2, int y2, int color )
{
  glLineStipple( 1, 0xffff );   /* solid line */
  glPolygonMode( GL_FRONT_AND_BACK, GL_LINE );
  glIndexi( color );
  glRecti( x1, y1, x2, y2 );
}


/* Erase the rubberband box in the window */
void
erase_rubberband( int x1, int y1, int x2, int y2 )
{
  glLineStipple( 1, 0xffff );
  glPolygonMode( GL_FRONT_AND_BACK, GL_LINE );
  glIndexi( uiGetBlackPixel() );
  glRecti( x1, y1, x2, y2 );
}
```


Another possible implementation: XOR mode

```
/* using XOR mode consolidates calls */
void
displayRubberband( int x, int y, int x2, int y2, int color )
{
  glPolygonMode( GL_FRONT_AND_BACK, GL_LINE );
  glIndexi( color );
  glLogicOp( GL_XOR );   // do xor logic when drawing
  glRecti( x1, y1, x2, y2 );
}

mouseCallback( uiWindow *window, int x, int y,
               uiMouseAction action )
    ...
    case uiLeftMouseDown
      displayRubberband( anchor_x, anchor_y, old_x, old_y,
                         zoomColor );
    ...
    case uiLeftMouseDrag:
      /* drawing again will erase this */
      displayRubberband( anchor_x, anchor_y, old_x, old_y,
                         zoomColor );
      /* recalculate values -- this time it will draw */
      displayRubberband( anchor_x, anchor_y, old_x, old_y,
                         zoomColor );
      break;
    ...
```

Advanced topics: idle/timer procedures

Often when doing graphics, you'll find that many procedures are very processor-intensive, and will freeze up the controls while they calculate.

- This is often very frustrating if the user hits the wrong button
- This is also annoying if every button causes calculation to begin

Idle vs. Timer Procedures

- Both are added to and removed from the work queue in similar fashions
- Timer procedures have an additional argument: "interval" (in milliseconds)
- Typically, idle procedures are used by dividing a large task up into smaller ones (e.g., dividing the entire fractal calculation into one row at a time)
- Timer procedures are usually used for very simple, quick tasks (cycling a color table, blinking a cursor, etc.) which necessitate a delay

Project 0 Examples

- Zooming in (allow user to zoom in multiple times quickly)
- Color cycling (use this as a TimerProc to avoid cycling too fast)
- Allow color cycling to continue while fractal is recalculated

```
/* adding an idle/timer procedure */
uiSetIdleCallback( idleCallback );
uiSetTimerCallback( timerCallback, 100 );

/* your callback will look something like what's below.
   when this procedure returns TRUE (i.e. work is done),
   libui will remove this procedure from the event queue.
   When you wish to replace it, you must call
   uiSetIdleCallback again */

uiBool
idleCallback( void )
{
   uiBool calculationDone;
   calculationDone = calculateForAWhile();
   return calculationDone;
}
```


Summary

Here's what you should take home from this lecture:

- Basic differences between command and event-driven programming
- How to add control devices (sliders, toggle buttons, etc.) to your program using "libui" (see online libui documentation for more info)
- Ways of using the overlay plane (XOR mode)
- Advantages of using idle/timer procedures