Views, Drawing, and Events

Lecture 13
First: Representing Points and Areas
**NSPoint**

```c
typedef struct _NSPoint {
    CGFloat x;
    CGFloat y;
} NSPoint
```

- Pair of $x$, $y$ coordinates
- **NSZeroPoint** constant represents the “origin”
- Create with **NSMakePoint**(x,y)
NSPoint

NSMakePoint(15.0, 40.0)

NSZeroPoint
NSSize

typedef struct _NSSize
{
    CGFloat width; // should be positive
    CGFloat height; // should be positive
} NSSize

- Pair of width, height
- Create with NSMakeSize(width, height)
NSRect

typedef struct _NSRect
{
    NSPoint origin;
    NSSize size;
} NSRect

- Combination of a point and a size
- Represents a rectangular area
- Create with NSMakeRect(x, y, width, height)
NSRect

NSMakeRect(5.0, 5.0, 40.0, 30.0)
C structs

• C structs are not Objective-C classes!

• Declaring variables of structs do NOT need asterisk:
  CGRectGet rect; NOT CGRect *rect;

• As arguments, they are passed by value:
  - (void) drawRect: (CGRect *) rect;
  - (void) drawRect: (CGRect) rect;

• No need to retain/release
Views
**NSView**

- Handles *drawing* and events for a rectangle in a window.
- E.g. buttons and text fields are all subclasses of NSView
- A view can contain subviews
- You subclass NSView to implement your own drawing and/or event handling
The View Hierarchy

NSWindow

NSView

[window contentView]

[contentView subviews]

[box subviews]
Bounds and Frame

- An NSView has two rectangles associated with it: the bounds and the frame.
- The frame is the view’s rectangle inside (relative to) its superview.
- The bounds rectangle defines the coordinates used for drawing and events within a frame.
Bounds and Frame

Now this point is 0,0 when the view is drawing (40, 30)

frame = NSMakeRect(15.0, 15.0, 40.0, 30.0)
bounds = NSMakeRect(0.0, 0.0, 40.0, 30.0)
Bounds and Frame

Changing the size of the bounds scales the coordinates

frame = NSMakeRect(15.0, 15.0, 40.0, 30.0)
bounds = NSMakeRect(0.0, 0.0, 1.0, 1.0)
Drawing
Drawing

- (void)drawRect: (NSRect) rect
  {
    // draw things
  }

• Define it in your NSView subclass

• rect is the subrect of the bounds that needs redrawing: you’re allowed to skip things outside it, but you don’t have to

• You don’t (directly) call it yourself
Redrawing

• Since you can’t call drawRect, you use:

• `[view setNeedsDisplay=YES]`
  or
  `[view setNeedsDisplayInRect:rect]`

to tell Cocoa to redisplay the proper portion of the view
Events
NSApplication

• Manages the main run loop of your application:
  • Waits for events from the mouse, keyboard, etc.
  • Dispatches events to the relevant objects (of class NSResponder)

• Owner of MainMenu.nib
Application Life Cycle

- Application starts
- MainMenu.nib loads
- Run loop:
  - Wait for event
  - Handle event
- Application terminates
Kinds of Events (Mac)

• Some sort of input from the user
  • Mouse button down/up
  • Mouse moved
  • Keyboard key pressed/released
  • Drawing tablet events
Event Dispatch — Mouse Events

- The window server sends the event to the `NSApplication` that owns the window under the cursor.

- In `- (void)sendEvent: (NSEvent *)event`, the `NSApplication` sends the event to the appropriate `NSWindow`.

- That window finds the appropriate view using `hitTest: on its contentView`.
Mouse Events

- Depending on the event type, a different method is called on the NSView
  
  - (void)mouseDown:(NSEvent *)event
  mouseUp:, mouseDragged:, mouseMoved:, mouseEntered:, mouseExited:
Mouse Events

```objective-c
NSPoint windowPos = [event locationInWindow];
NSPoint viewPos = [self convertPoint:windowPos fromView:nil];

if ([event clickCount] == 2) {
    // handle double click
}

drag.x += [event deltaX];
drag.y -= [event deltaY];
```
Event Dispatch — Key Events

• Instead of (clicking) the window under the cursor, keyboard events go to the key window [NSApp keyWindow]

• Each window/view has a first responder [window firstResponder]

• In order for your **View to become the first responder, it must return YES for:
  - (BOOL)acceptsFirstResponder
Key Events

- There are two key event methods for NSResponder:
  - (void)keyUp:(NSEvent *)event
  - (void)keyDown:(NSEvent *)event
Key Events

```c
NSString *chars = [event characters];
unsigned modifiers = [event modifierFlags];

if ((modifiers & NSCommandKeyMask) &&
    [chars isEqualToString:@"s"])
{
    // “command-s” was pressed
}
```
Resources

• **Introduction to View Programming Guide for Cocoa**

• **Working with the View Hierarchy**

• **Introduction to Cocoa Drawing Guide**

• **Programming Mac OS X with Cocoa for Beginners/Wikidraw’s view class**
  • [http://en.wikibooks.org/wiki/Programming_Mac_OS_X_with_Cocoa_for_Beginners/Wikidraw's_view_class](http://en.wikibooks.org/wiki/Programming_Mac_OS_X_with_Cocoa_for_Beginners/Wikidraw's_view_class)