Custom Views and Drawing

Lecture 5
First - Representing Points and Areas
NSPoint

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

• Pair of x, y coordinates

• NSZeroPoint represents the “origin”

• Create with NSMakePoint(x,y)
CGPoint

typedef struct _CGPoint {
    CFFloat x;
    CFFloat y;
} CGPoint

• Pair of x, y coordinates
• CGPointZero represents the “origin”
• Create with CGPointMake(x,y)
NSPoint

NSMakePoint(15.0, 40.0)

NSZeroPoint
CGPoint

CGPointZero

NSMakePoint(15.0, 40.0)
typedef struct _NSSize {
    CGFloat width; // should be positive
    CGFloat height; // should be positive
} NSSize

- Pair of width, height
- Create with **NSMakeSize**(width, height)
typedef struct __CGSize {
  CGFloat width;  // should be positive
  CGFloat height; // should be positive
} CGSize

- Pair of width, height
- Create with CGSizeMake(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)
CGRect

typedef struct _CGRect {
    CGPoint origin;
    CGSize size;
} CGRect

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

NSMakeRect(5.0, 5.0, 40.0, 30.0)
CGRect

CGRectMake(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:
  `CGRect 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/UIView

• Handles **drawing** and events for a rectangle in a window.

• E.g. buttons and text fields are NS(UI)Views

• A view can contain subviews

• Subclass NS(UI)View to implement your own drawing and/or event handling
The View Hierarchy

NSWindow

NSView

[window contentView]

[contentView subviews]

[box subviews]
Bounds and Frame

- NSView/UIView has two rectangles associated with it: the *bounds* and the *frame*
- The frame is the view’s rectangle inside its superview
- The bounds rectangle defines the coordinates used for drawing and events
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
- (void)drawRect:(NSRect)rect
{
    // draw things
}

• Define it in your NSView/UIView 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

• Don’t call it yourself: it won’t work
Redrawing

- You can’t call `drawRect:` directly
- `[view setNeedsDisplay:YES]`
  or
- `[view setNeedsDisplayInRect:rect]`

to tell Cocoa to redisplay the proper portion of the view
Events
NS(UI)Application

- Manages the main run loop of your application:
  - Waits for events from the mouse, keyboard, etc. (Mac) or touches (iOS)
  - Dispatches events to the relevant objects (of class NSResponder/UIResponder)
- Owner of MainMenu.nib / MainWindow.xib
Application Life Cycle

• Application starts
• MainMenu.nib / MainWindow.xib 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 the window under the cursor, keyboard events go to the `key window` 
  `[NSApp keyWindow]`

• Each window 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 on NSResponder
  - (void)keyUp:(NSEvent *)event
  - (void)keyDown:(NSEvent *)event
Key Events

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

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

• No key or mouse events
• Touch events instead
• `UIEvent` contains a collection of `UITouches`
• Multi-touch support
Touch Events

- (void)touchesBegan:(NSSet *)touches
  withEvent:(UIEvent *)event
- (void)touchesMoved:(NSSet *)touches
  withEvent:(UIEvent *)event
- (void)touchesEnded:(NSSet *)touches
  withEvent:(UIEvent *)event
- (void)touchesCancelled:(NSSet *)touches
  withEvent:(UIEvent *)event
Touch Events

- `(void)touchesBegan:):(NSSet *)touches withEvent:):(UIEvent *)event`

  • `[event allTouches]` returns *all* of the touches in the multi-touch sequence
  • `touches` are *just* the touches associated with this particular call
  • If you’re only expecting one touch, get it with `[touches anyObject]`
UITouch *touch = [touches anyObject];

if ([touch tapCount] == 2)
    // handle double tap

CGPoint a, b;
a = [touch previousLocationInView:self];
b = [touch locationInView:self];

drag.x += b.x - a.x;
drag.y += b.y - a.y;
Turning on Multi-Touch

```swift
self.multipleTouchEnabled = YES;
```