CS193E
Lecture 12

Formatters
Cocoa Text
More View Drawing

Quick Scroll View Demo

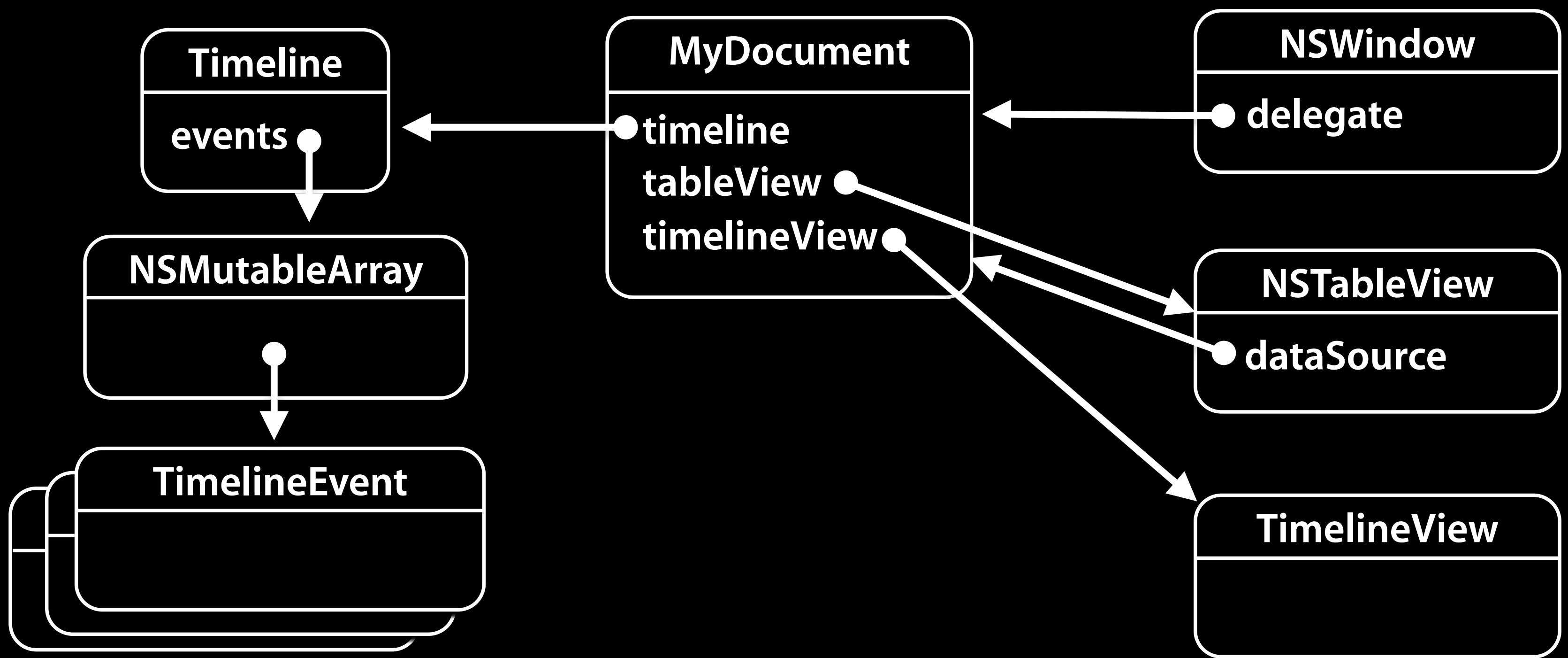
Announcements

- Questions on previous material or assignment?
- If you don’t get a grade by Sunday, please let us know
- Some of today’s content spills into next week’s assignment

Personal Timeline III

No big structural changes

Basic architecture in place, now adding features



Formatters

Formatting Values

- A formatter converts a value to a string and back again
- Text field contents can be formatted using NSFormatters
- Built-in formatters for numbers and dates
- Custom formatters can be written pretty easily
- Easily configured in IB
  - Note: Significant new functionality in 10.4 isn't exposed in IB 2.x and isn't enabled by default, instead configure the formatter manually in code.

NSFormatter

- Abstract class - with concrete subclasses in Foundation
    - NSDateFormatter
    - NSNumberFormatter
- Converts value objects such as NSDates and NSNumbers to strings

```
- (NSString *)stringForObjectValue:(id)obj;
```

- Also can parse strings into objects

```
-(BOOL)getObjectValue:(id *) obj
            forString:(NSString *) string
     errorDescription:(NSString **)errorDesc
```

NSNumberFormatter

- Format specified by a format string which can be localized
- Can control symbols such as decimal point, thousands separator, and positive, negative, and zero formats.

```
NSNumberFormatter *formatter =
    [[NSNumberFormatter alloc] init];

[formatter setFormat:
    @"$#,###.00;0.00;($#,##0.00)"];

[formatter stringForObjectValue:
    [NSNumber numberWithFloat:-12345.6]];
```

```
($12,345.60)
```

Changes in Tiger

- New behavior in 10.4
  - +setDefaultFormatterBehavior:
  - -setFormatterBehavior:
  - NSNumberFormatterBehavior10_4
- By default uses behavior of 10.3 and earlier

Tiger Number Formatter

- Can specify a style which will use formats set by user in International preferences

```
NSNumberFormatterDecimalStyle

NSNumberFormatterCurrencyStyle

NSNumberFormatterPercentStyle

NSNumberFormatterScientificStyle

NSNumberFormatterSpellOutStyle
```

- Can also specifiy new style format string as described in Unicode Technical Standard #35

NSDateFormatter

- Can use a format string like NSNumberFormatter

```
 NSDateFormatter *formatter = [[NSDateFormatter
 alloc]
    initWithDateFormat:@”%A %b %d, %Y”
    allowNaturalLanguage:NO];

[formatter stringForObjectValue:[NSDate date]];
```

```
Tuesday Mar 9, 2004
```

Changes in Tiger

- New behavior in 10.4

```
+setDefaultFormatterBehavior:
-setFormatterBehavior:
NSDateFormatterBehavior10_4
```

- By default uses behavior of 10.3 and earlier

Tiger Date Formatter

- Can specify a style which will use formats set by user in International preferences

```
NSDateFormatterShortStyle
NSDateFormatterMediumStyle
NSDateFormatterLongStyle
NSDateFormatterFullStyle
```

- Can also specifiy new style format string as described in Unicode Technical Standard #35

Using a formatter in a custom view

- Shared formatter for a class uses one formatter

```
static NSDateFormatter *dateFormatter;

@implementation MyClass
+ (NSDateFormatter *)dateFormatter {
   if (!dateFormatter) {
       dateFormatter = [[NSDateFormatter alloc] init];
       [dateFormatter setDateStyle:
                          NSDateFormatterShortStyle];
   }
   return dateFormatter;
}
```

Getting a string from a value object

```
NSDate *someDate = [NSDate date];

NSString *dateString =
    [[MyClass dateFormatter]
                stringForObjectValue:someDate];

// Use string to draw, etc.
```

Cocoa Text System

Cocoa Text System

- One of the richest but most complex APIs
- Numerous ways to interact with text in Cocoa
- Many classes involved, strong MVC design
- We’ll focus on most common text uses
- For much more details, check the “Text System Overview” documentation

Strings & Text

- NSStrings are basis of all text in Cocoa
- You’ve already used them, and we’ve seen examples of how to draw strings

```
NSFont *font = [NSFont fontWithName:@”Helvetica” size:24];

[dict setObject:font forKey:NSFontAttributeName];
[@”Hello World” drawAtPoint:point withAttributes:dict];
```

- Utilities for drawing strings with attributes give a lot of functionality & power
- Notion of “string+attributes” is important...

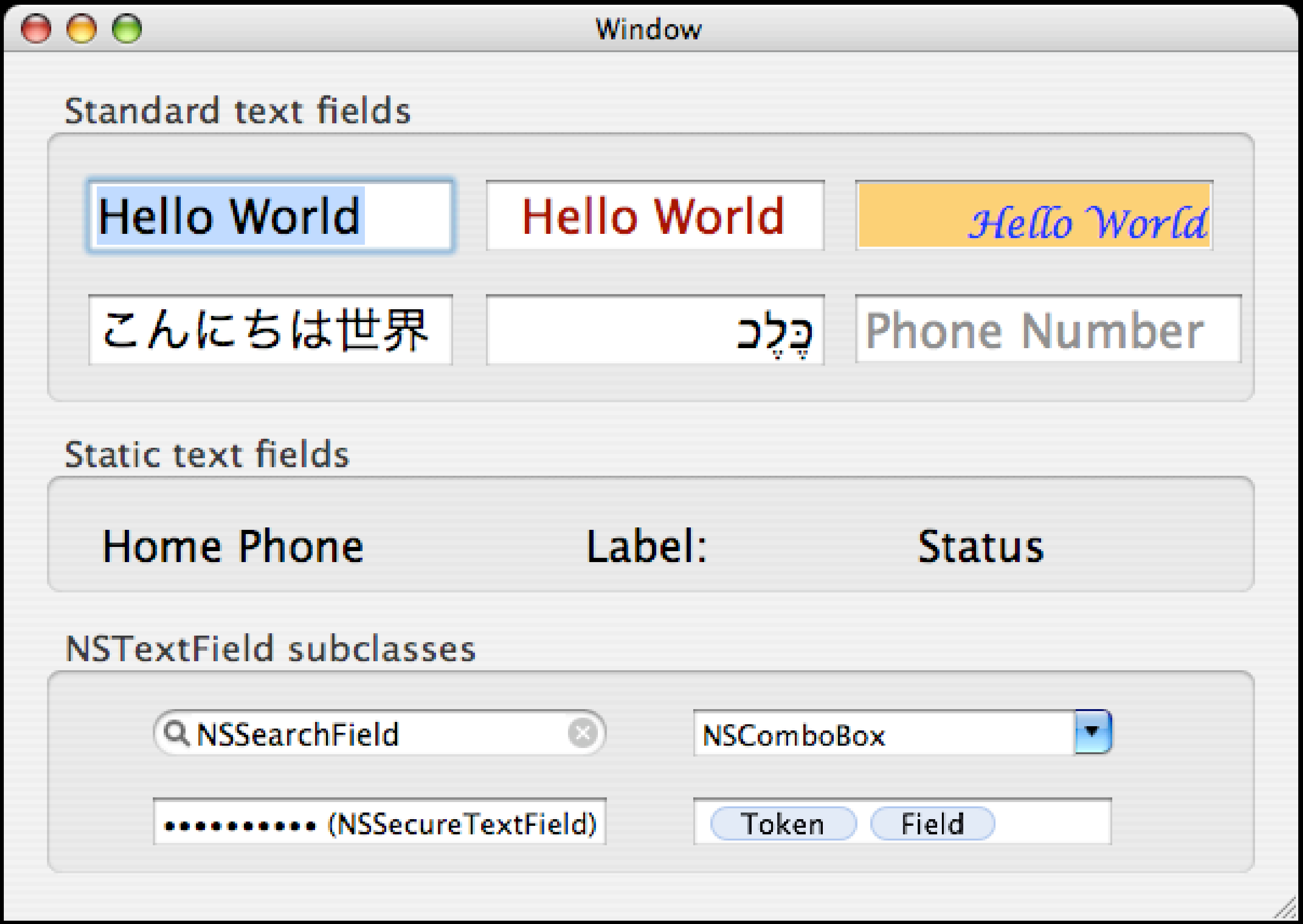
Text Needs

- Drawing text is easy, but most applications want to allow users to see & edit rich text
- Two primary mechanisms in Cocoa, depending on your needs:
  - NSTextField - NSControl subclass for display and edit of small amounts of text
  - NSTextView - Full-blown text editor view

NSTextField

- Standard text field control, you’ve already used it
- Commonly used when an app wants to have an action fired after user edits something
    - Examples: URLs in Safari, Account name in System Preferences
- Primarily for single line, small text entry
- Fairly customizable: bordered, bezeled, text color, background color, font, alignment, etc.

NSTextField Examples



Text Fields

- Since it’s a control, supports target/action
  - Can be configured (in IB) to send action only when return key is hit or when keyboard focus lost (for any reason)
- Can also use via outlet to set/get the value as needed (e.g. a form with multiple fields)
- Allows delegate to fine-tune behavior:

```
- (void)controlTextDidBeginEditing:(NSNotification *)obj;
- (void)controlTextDidEndEditing:(NSNotification *)obj;
- (void)controlTextDidChange:(NSNotification *)obj;

- (BOOL)control:(NSControl *)control
            textShouldBeginEditing:(NSText *)fieldEditor;
- (BOOL)control:(NSControl *)control
            textShouldEndEditing:(NSText *)fieldEditor;
```

NSTextView

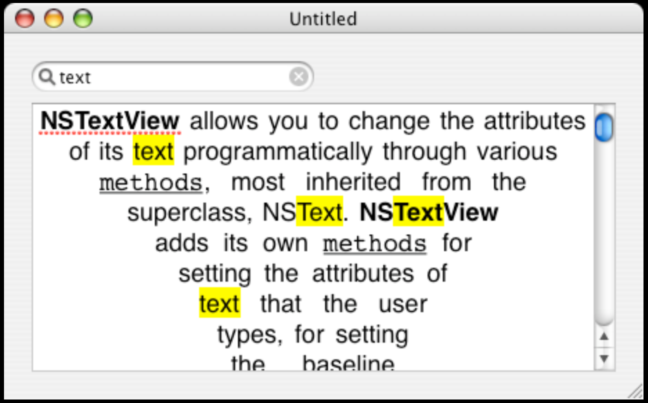
- NSTextField’s big brother - pretty much a full-blown text editor
- Numerous classes involved, you can decide how much of the system you want or need
- /Developer/Examples/AppKit/TextEdit
- “Text Editor in 15 Minutes” section in Text System Overview documentation
- “Best 75,000 lines of code you’ll never write”

Text System & MVC

- Various classes with clearly defined roles
- Model: NSTextStorage (text data)
  NSTextContainer (layout geometry)
- View: NSTextView, presents the text in a
  specific geometry
- Controller: NSLayoutManager, coordinates
  model and view

Common Usage

- Typically you just deal with the NSTextView and NSTextStorage
- Other classes can facilitate special layouts or behavior:



NSTextView allows you to change the attributes of its text programmatically through various methods, most inherited from the superclass, NSText. NSTextView adds its own methods for setting the attributes of text that the user types, for setting the baseline

- But that's beyond the scope of this class!

NSTextView

- Geared for editing significant amounts of text, typically rich text
- Responsible for rendering text and handling user interactions
- Leverages almost everything we’ve learned so far: first responder, copy/paste, drag/drop, delegates, undo, notifications, etc
- Delegate can fine tune text editing and manipulation process

NSTextView Delegates

- The text system has a bunch of delegate callbacks for you to hook into:

```
- (BOOL)textShouldBeginEditing:(NSText *)text;
- (BOOL)textShouldEndEditing:(NSText *)text;

- (void)textDidBeginEditing:(NSNotification *)note;
- (void)textDidChange:(NSNotification *)note;
- (void)textDidEndEditing:(NSNotification *)note;

- (NSRange)textView:(NSTextView *)text
  willChangeSelectionFromCharacterRange:(NSRange)old
  toCharacterRange:(NSRange)new;
- (void)textViewDidChangeSelection:
                           (NSNotification *)note;
```

More NSTextView Delegate

- When user clicked, double clicked or dragged file attachments or hyperlinks
- Provides details on writable pasteboard types
- Providing tooltips for characters with the tooltip attribute set
- Providing completions for words
- Customizing the undo setup for the text view
- Look at the NSTextView class documentation

NSText vs. NSTextView

- NSTextView is a subclass of NSText
- Much of the API is expressed in terms of NSText
- You can generally think of NSText as being the same as NSTextView
    - In practice everything is an NSTextView
- In places where it’s typed (NSText *) you can check the class to see if it’s really an NSTextView

Getting Text In & Out

- Putting a string into a text view is easy:

```
- (NSString *)string;
- (void)setString:(NSString *)string;
- (void)replaceCharactersInRange:(NSRange)range
                      withString:(NSString *)string
```

- Dealing with RTF Data:

```
- (NSData *)RTFFromRange:(NSRange)range;
- (NSData *)RTFDFromRange:(NSRange)range;
- (void)replaceCharactersInRange:(NSRange)range
                         withRTF:(NSData *)rtfData
- (void)replaceCharactersInRange:(NSRange)range
                        withRTFD:(NSData *)rtfData
```

- Use RTF data for pasteboard exchange

Strings + Attributes = ♥

- Underlying the rich text system are NSStrings with associated attributes
- Keeping these separate is very cumbersome
- Welcome: Attributed Strings!
- Encapsulates a string and its attributes in a single object
- Immutable (NSAttributedString) and Mutable (NSMutableAttributedString) flavors

NSAttributedString

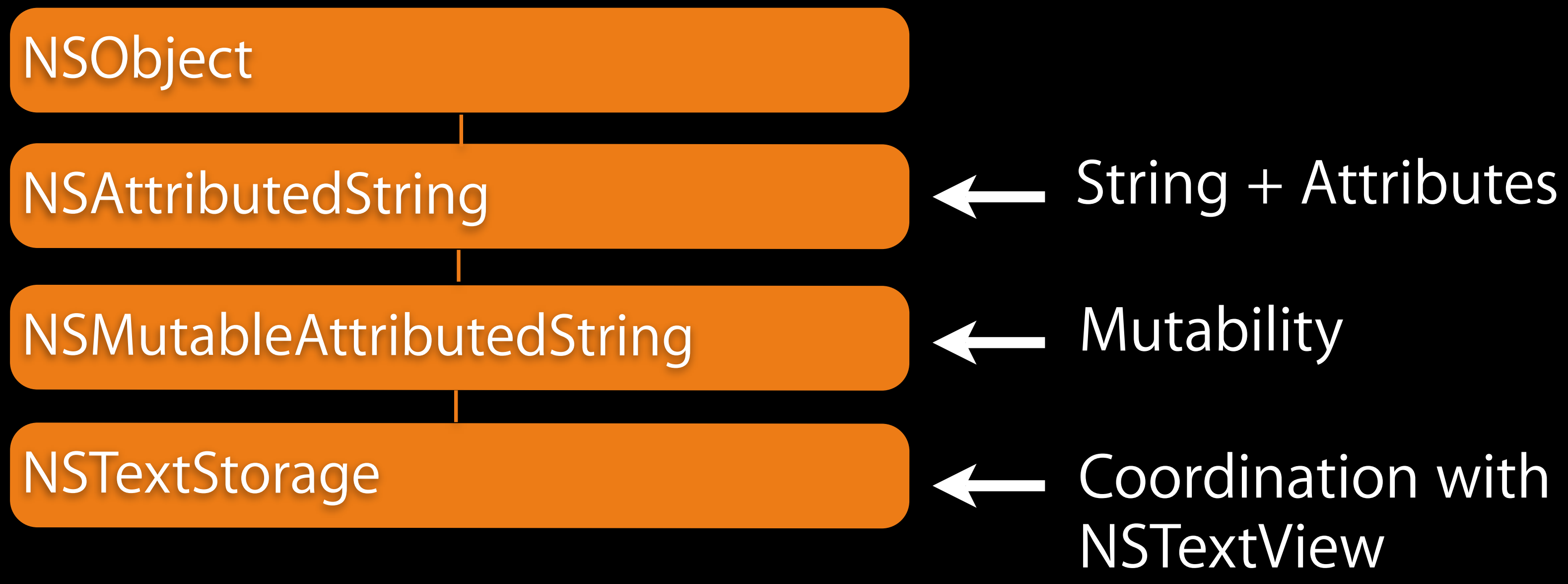
- Has simple drawing API like NSString

```
- (void)drawAtPoint:(NSPoint)point;
- (void)drawInRect:(NSRect)rect;
```

- Cocoa defines all sorts of text attributes:

```
Font name               Paragraph style
Foreground color        Underline
Background color        Stroke color
Shadow                  Cursor
Tooltip                 Link
Attachment              and many more...
```

NSTextStorage



- Contents of NSTextView stored in NSTextStorage
- Subclass of NSMutableAttributedString with added functionality to work with layout managers

Changing The Text Storage

- Instead of using text view API, you can manipulate the text storage directly:

```
NSTextStorage *textStorage = [textView textStorage];

 [textStorage beginEditing];
 [textStorage replaceCharactersInRange:range
              withString:replacementString];
 [textStorage setAttributes:attributes range:range];
 [textStorage endEditing];
```

- You're editing the attributed string directly, the text storage will make sure the view is updated accordingly

NSRange

```
typedef struct _NSRange {
   unsigned int location;
   unsigned int length;
} NSRange;
```

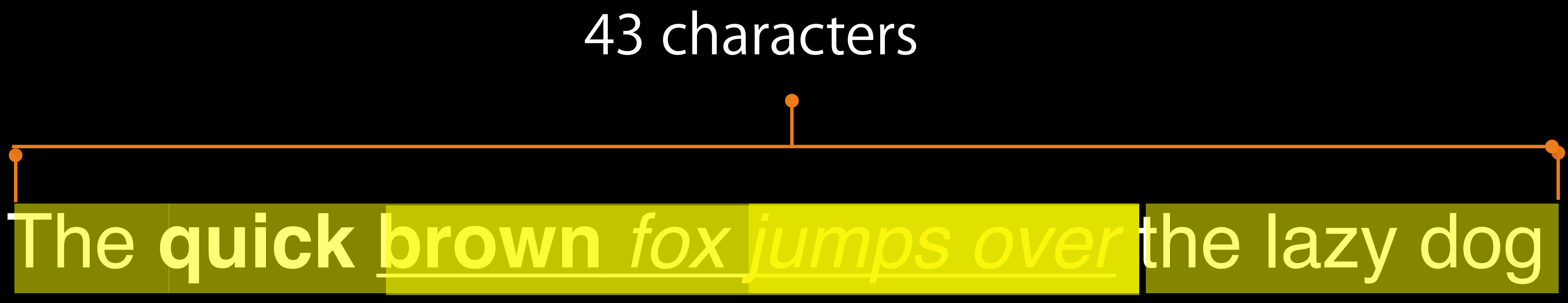
- Data structure contains a location and length

```
NSRange range;
 range.location = 5;  // start at character 5
 range.length = 10;   // for 10 characters
```

- Utilities like rects, points, size, etc

```
NSRange range = NSMakeRange(5, 10);
```

- Used to specify extent of attributes in string of characters



Range	Attribute	Value
{ 0, 4 }	NSFontAttributeName	Helvetica 48pt
{ 4, 12 }	NSFontAttributeName	Helvetica Bold 48pt
{ 10, 20 }	NSUnderlineStyleAttributeName	1
{ 16, 15 }	NSFontAttributeName	Helvetica Italic 48pt
{ 20, 11 }	NSForegroundColorAttributeName	Orange Color
{ 31, 12 }	NSFontAttributeName	Helvetica 48pt

Accessing Attributes

The **quick** **brown** *fox* *jumps over* the lazy dog

On NSAttributedString:

```
-(NSDictionary *)attributesAtIndex:(unsigned)index
                 effectiveRange:(NSRangePointer)aRange;
```

Example:

```
NSAttributedString *attrString; // string from above
NSDictionary *attributes;
NSRange range;


attributes = [attrString attributesAtIndex:25
                         effectiveRange:&range];
```

attributes:	orange color, Helvetica 48pt, italic, underlined
range:	{20, 10}

More Info

- For more details on text system, see the “Text System Overview” document
- If you need to manipulate rich text at the attribute level, see the “Attributed Strings Programming Guide” in the docs
- You can likely just use NSTextFields and NSTextViews as they are through the high level APIs

More View Drawing

Image Drawing
Coordinate Systems and Drawing
Adjusting for String Length

Drawing an Image

- Draw the full image (or part of it):

```
- (void)drawAtPoint:(NSPoint)point
    fromRect:(NSRect)srcRect
    operation:(NSCompositingOperation)op
    fraction:(float)alpha;
```

- Scaling the image:

```
- (void)drawInRect:(NSRect)dstRect
    fromRect:(NSRect)srcRect
    operation:(NSCompositingOperation)op
    fraction:(float)alpha;
```

- “op” is usually NSCompositeSourceOver

Composite Operations

- A composite operation describes how to blend the source (the image) with the background
- Porter-Duff equation (simplified for "source over")

  $\text{color}_{\text{out}} = \alpha_{\text{src}} \cdot \text{color}_{\text{src}} + \text{color}_{\text{bkgd}} \cdot (1 - \alpha_{\text{src}})$
- See also

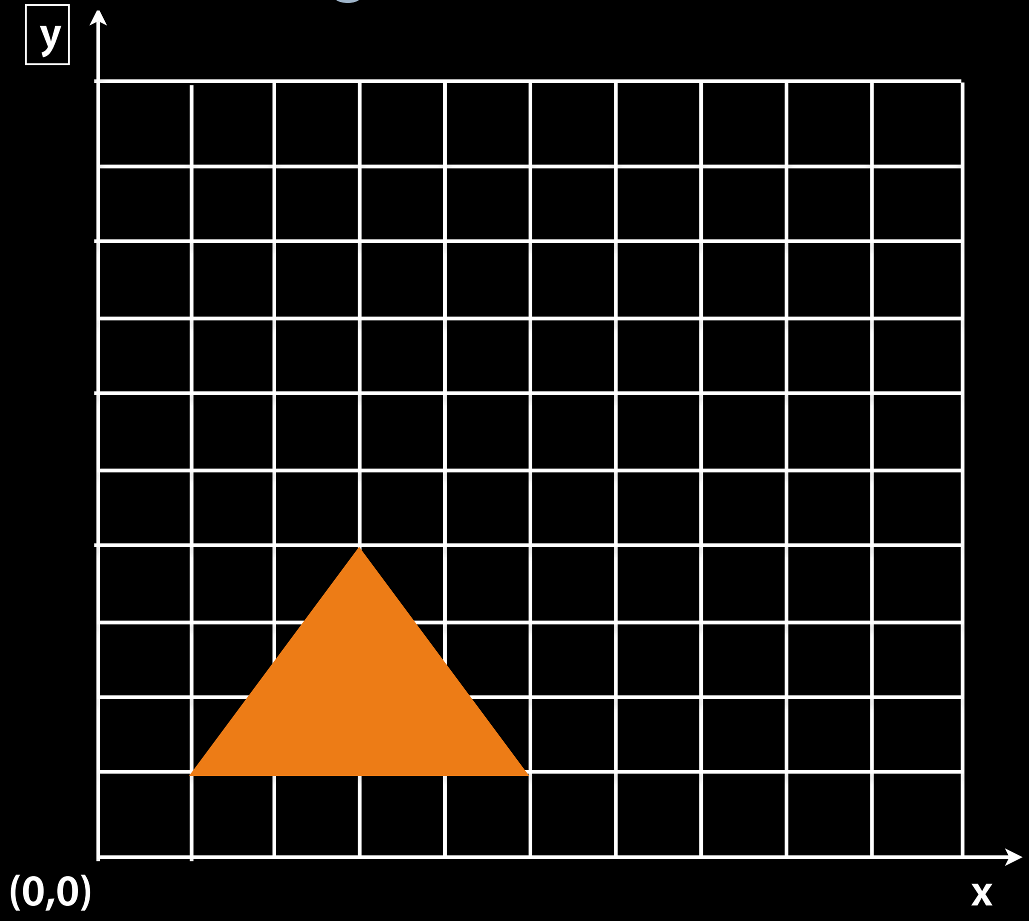
  /Developer/Examples/AppKit/CompositeLab

A little more about -isFlipped

- Returning YES from -isFlipped causes an automatic change
  - Before -drawRect: is called, a transformation is automatically applied to your view’s coordinate system
- Higher level constructs like cells and string drawing check the isFlipped value of a view
- Lower level drawing constructs like bezier paths and images make no direct adjustment

Standard view coordinates

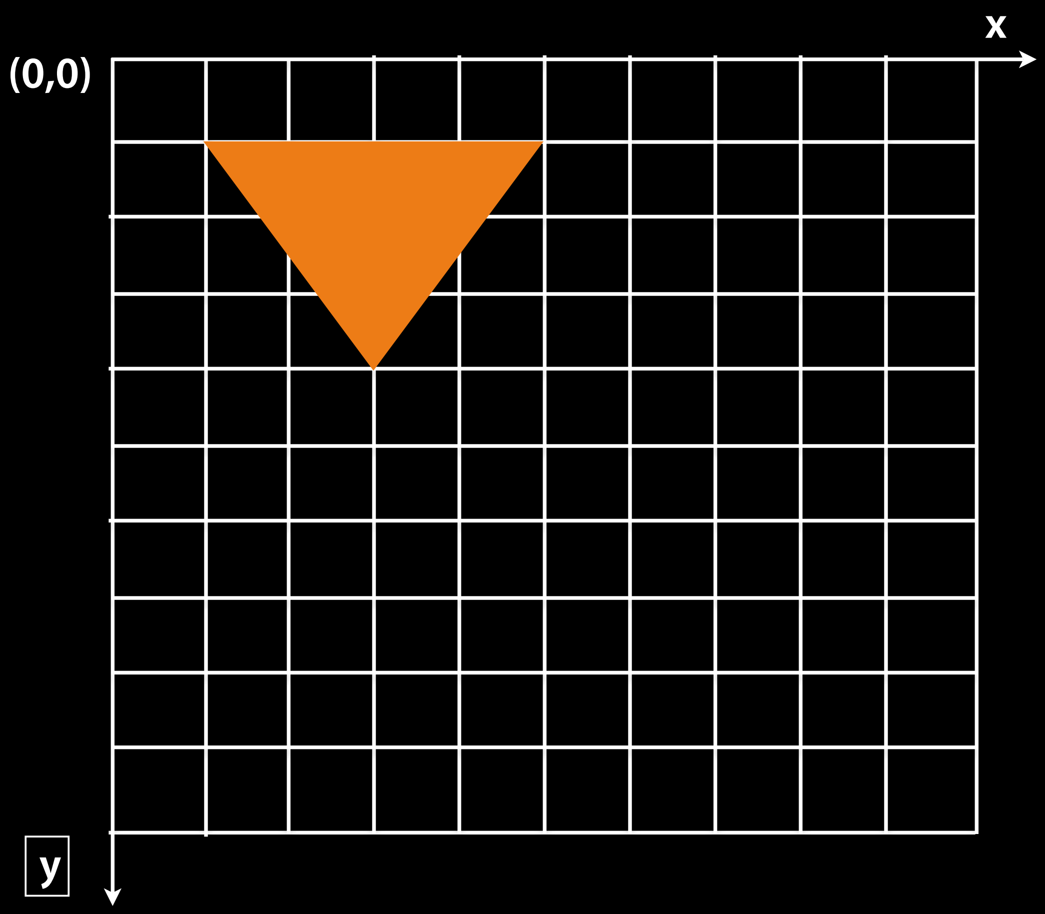
Filling an NSBezierPath



Vertices: (1, 1) (3, 4) (5, 1)

Flipped view coordinates

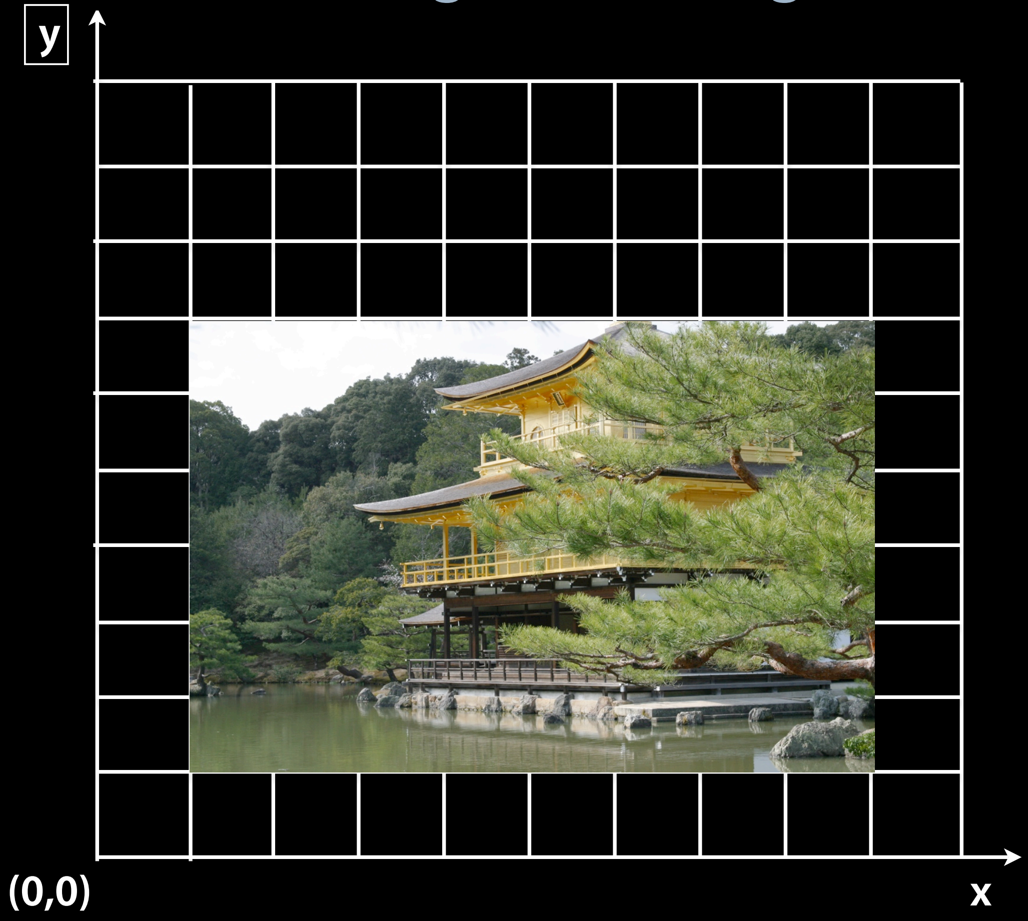
Filling the same NSBezierPath



Vertices: (1, 1) (3, 4) (5, 1)

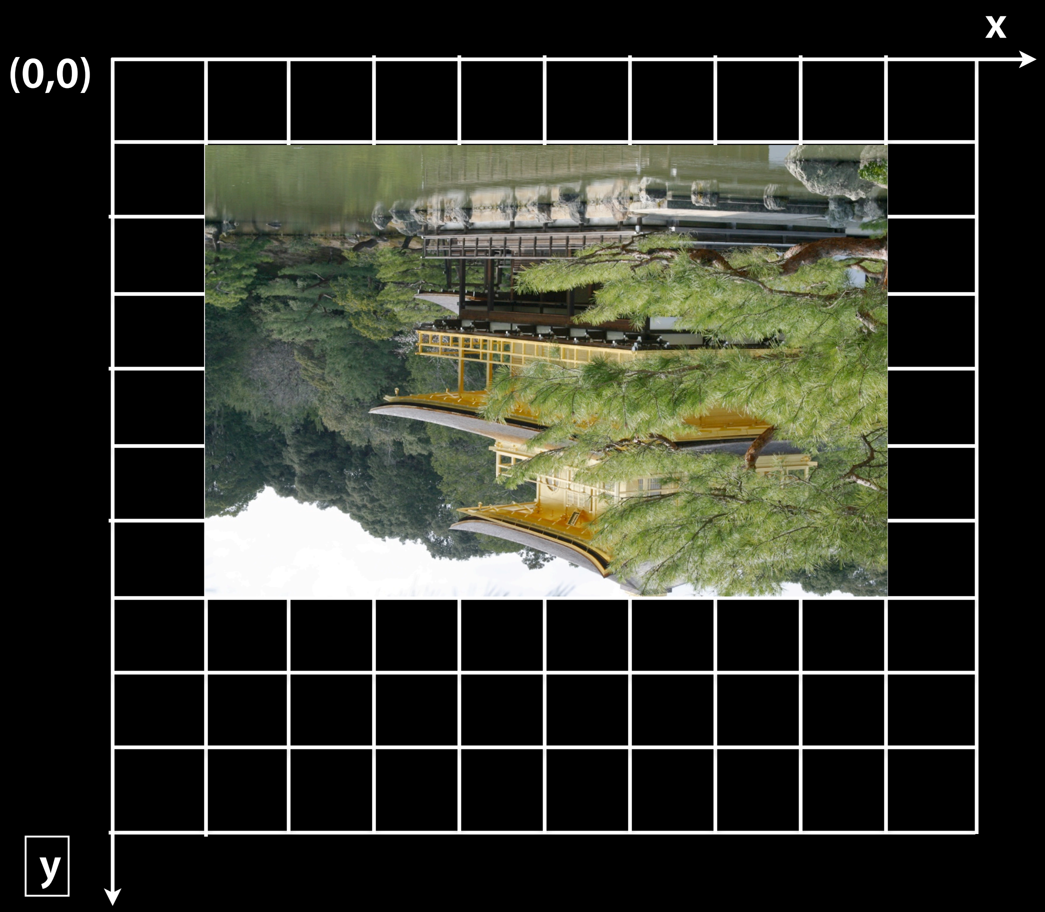
Standard view coordinates

Drawing an NSImage



Flipped view coordinates

Drawing the same NSImage



How to draw right side up images?

- If a view is flipped, just flip the view’s coordinate system back before drawing, then flip it back when done.
- Three tasks
  - Flip the coordinate system
  - Adjust the destination rectangle of the image
  - Flip the coordinate system back

NSAffineTransform

- Object-oriented representation of a transformation matrix
- Can rotate, scale, translate, or set matrix directly
- Use to define an affine transformation then apply it to the coordinate system of the current graphics context

```
NSAffineTransform *transform =
                         [NSAffineTransform transform];
// Flip around the x axis
[transform scaleXBy:1.0 scaleYBy:-1.0];

// concatenate transformation
[transform concat];
```

Useful NSImage category method

```
- (void)my_drawInRect:(NSRect)rect fromRect:(NSRect)fromRect
operation:
(NSCompositingOperation)op fraction:(CGFloat)delta flip:(BOOL)flip
{
    NSAffineTransform *xAxisReflection = nil;
    NSRect destRect = rect;
    if (flip) {
        NSAffineTransform *xAxisReflection = [NSAffineTransform
transform];
        [xAxisReflection scaleXBy:1.0 yBy:-1.0];
        [xAxisReflection concat];
        destRect.origin.y = -rect.origin.y - rect.size.height;
    }
    [self drawInRect:destRect fromRect:fromRect operation:op
fraction:delta];
    if (flip) {
        [xAxisReflection concat];
    }
}
```

Dealing with different string lengths

- How do you deal with string values provided by the user that can have arbitrary length?
- Two main approaches:
    - Extend the area to fit the string
    - Truncate the string in some fashion
- Can also take a hybrid approach
- Usually makes sense to separate sizing logic from drawing logic

Measuring strings

- In a timeline item, you may want to extend the area to fit a very long title
- NSStringDrawing.h provides a handy NSString method

```
-(NSSize)sizeWithAttributes:(NSDictionary *)attributes;
```

- It also provides a handy method for NSAttributedStrings

```
-(NSRect)boundingRectWithSize:(NSSize)size
      (NSStringDrawingOptions)options:options
                 attributes: (NSDictionary *)attributes;
```

Truncating strings

- If a string gets too long, having it truncate with ellipsis is a pleasant and standard user interface
- Make and use a paragraph style for truncation

```
NSParagraphStyle *ps =[[NSParagraphyStyle
                  defaultParagraphStyle] mutableCopy];
[style setLineBreakMode: setLineBreakMode:
                       NSLineBreakByTruncatingMiddle];
NSDictionary *attributes =
     [NSDictionary dictionaryWithObjectAndKey:
        ps, NSParagraphStyleAttributeName];
 [ps release];
```

Truncating strings

```
NSDictionary *attributes; // created on last slide
```

- Now you can:
  - Draw string using those attributes
  - Create an attributed string from that string and attributes
  - Add the attribute to an existing attributed string.

Questions?