Text layout with Core Text

Jjgod Jiang <gzjjgod@gmail.com>

What is text layout?

- Convert a large piece of text into glyphs, and position them into correct place.
- Simple explanation: “Width in, heights out.”

- Input: text (in Unicode), font (platform dependent or independent), box (or other shapes), line height, tab width, etc.
- Output: glyphs (in glyph index of a specific font), origin points in the box

What is Core Text?

- Mac OS X text layout framework in Core Foundation (C) API
- Existed since Tiger, used as private framework for Cocoa and Apple apps
- Publicly available since Leopard
- No visible changes in Snow Leopard (There could be more Cocoa text system stuff transition to Core Text)

Why Core Text?

A tale of two Apples

- The original Apple development team created “Classic” APIs like QuickDraw and MLTE
- The NeXT team created NS* APIs which we later called “Cocoa”
- Since the merge of the two company, solutions like Carbon and Quartz were created to close the gap
- For modern, Unicode-aware text layout, there is ATSUI, which is still a popular choice for now

More about ATSUI

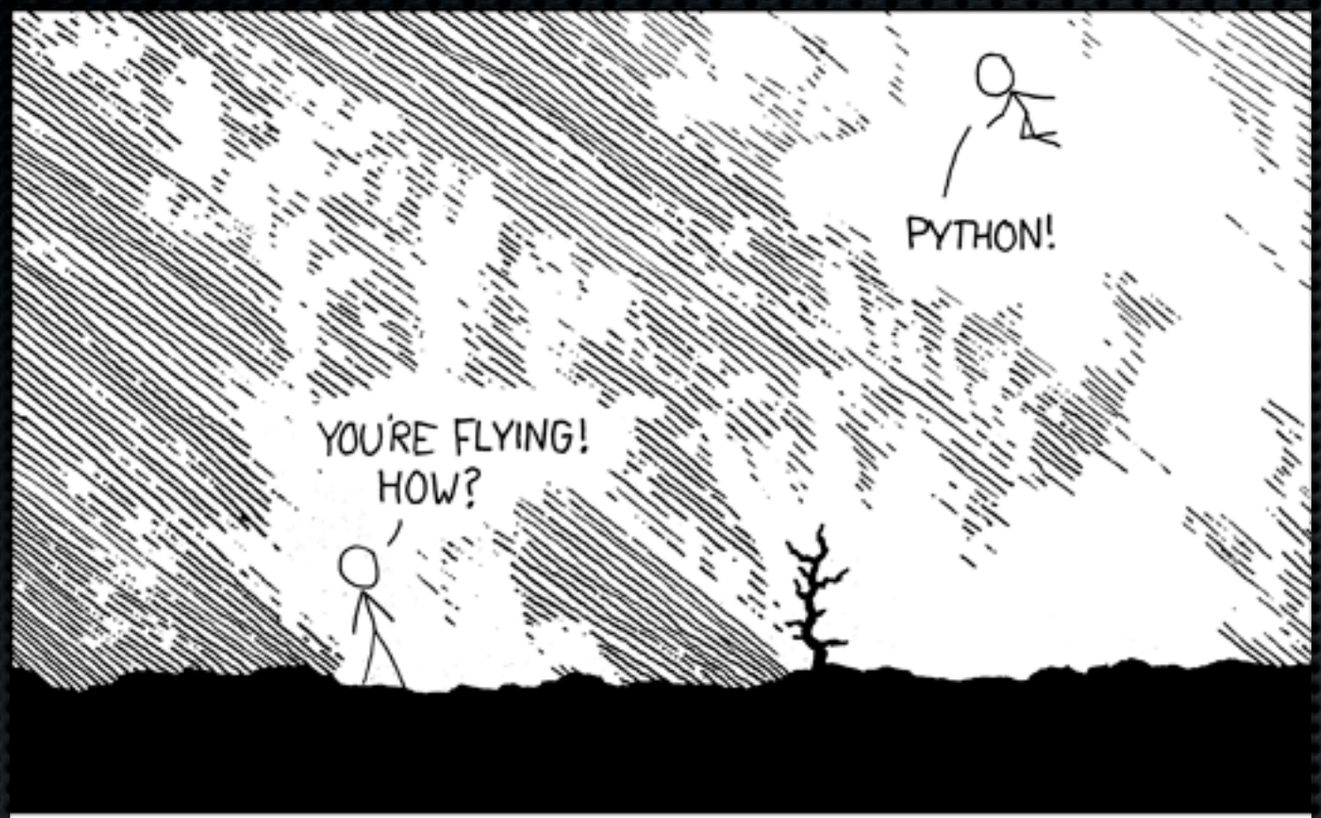
- Short for "Apple Type Services for Unicode Imaging"
- Carbon like API (`OSErr`, `Fixed`, Pascal String, etc.)
- Fast, robust, powerful, but the API is a bit complicate to use
  (Story: it took me a week to fix the original ATSUI-based implementation in Vim, but only one day to create a Core Text based one in vim-cocoa)
- ATS (for font enumeration and selection) still exists, but ATSUI is now almost deprecated, no 64-bit support

Core Text was created to replace ATSUI.

Why Core Text (again)?

- Simpler, Core Foundation style API, much easier to call from Cocoa apps
- Minimize conversion between different data formats, improve performance (up to 2 times faster according to Apple)
- Supported behavior (like font traits) closely matches with Cocoa Text System

Do I need Core Text?



Use APIs as high level as possible

Courtesy of http://xkcd.com/353/

Use APIs as high level as possible

- Read the "Text" section of Cocoa Drawing Guide
- Use `NSString`/`NSAttributedString` or create a `NSTextField` for simple drawing in UI
- For more sophisticated text, the best (and easiest) choice is to use high level control like `NSTextView` or `WebView`. They are well optimized.
- If the function is too limited, try Cocoa Text System, utilize its low level plumbers (`NSLayoutManager`, `NSTypesetter`, `NSTextContainer`, etc.)

If none of the above work well enough...

Case study: Textus

URL: http://www.jjgod.org/projects/textus
Icon courtesy of: chumsdock@newsmth

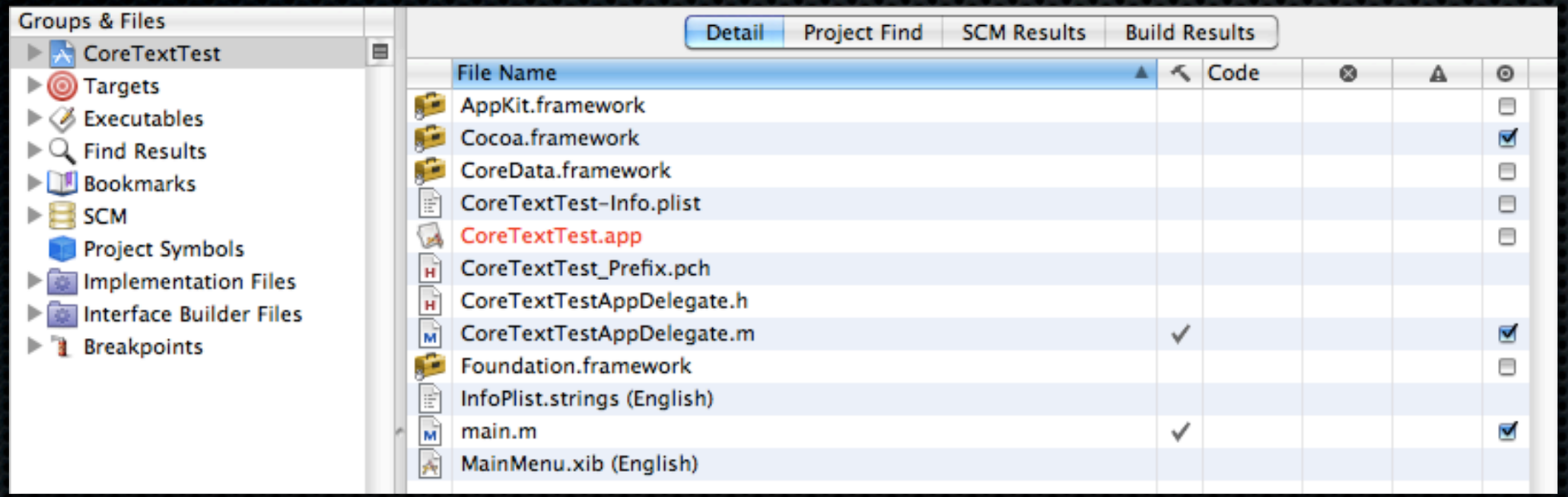
Case study: Textus

- Aim to be a fast, lightweight (plain text) ebook reader.
- `WebView` is too heavyweight for our purpose. So I went for `NSTextView`, customize NSTypesetter, NSTextContainer to control the output.
- Cocoa Text System sometimes is just too unpredictable, produces inconsistent results.
- Took me two weekends to rewrote it with Core Text, the most complex Core Text project I've ever done.

Learn Core Text: Prerequisites

- You need some basic understanding on Core Foundation, `CFString`, `CFDictionary`, memory management, etc.
- You also need some knowledge on Core Graphics, `CGPoint`, `CGRect`, `CGPath`, `CGGlyph`, etc. Notice: most CG-structs are not interchangeable with their NS-correspondence.
- Since we all use Cocoa now, basic knowledge on `NSView` and Cocoa drawing context is needed.

Learn Core Text:
Getting Started

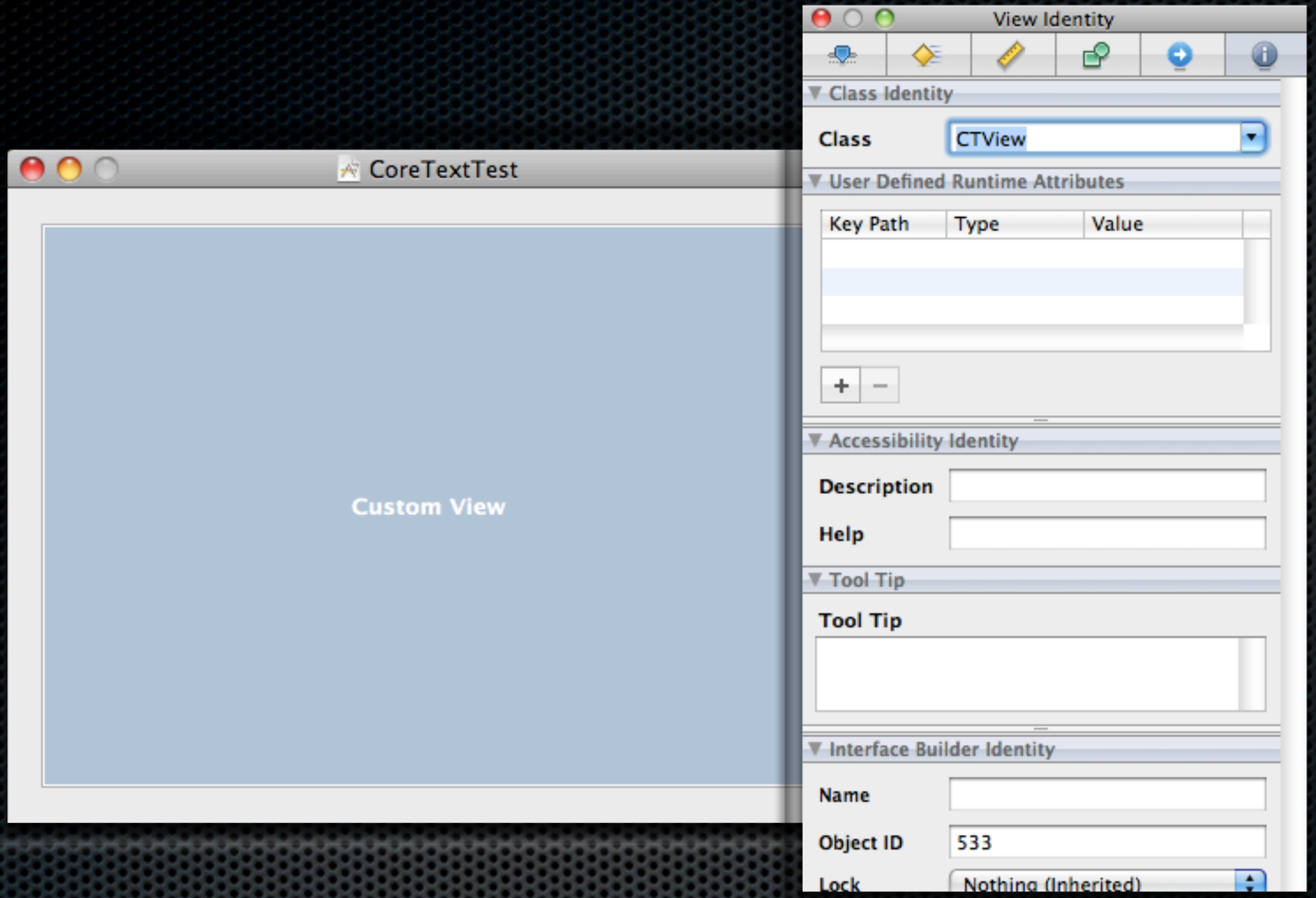


1. Create a new Cocoa Application project called “CoreTextTest” in Xcode.

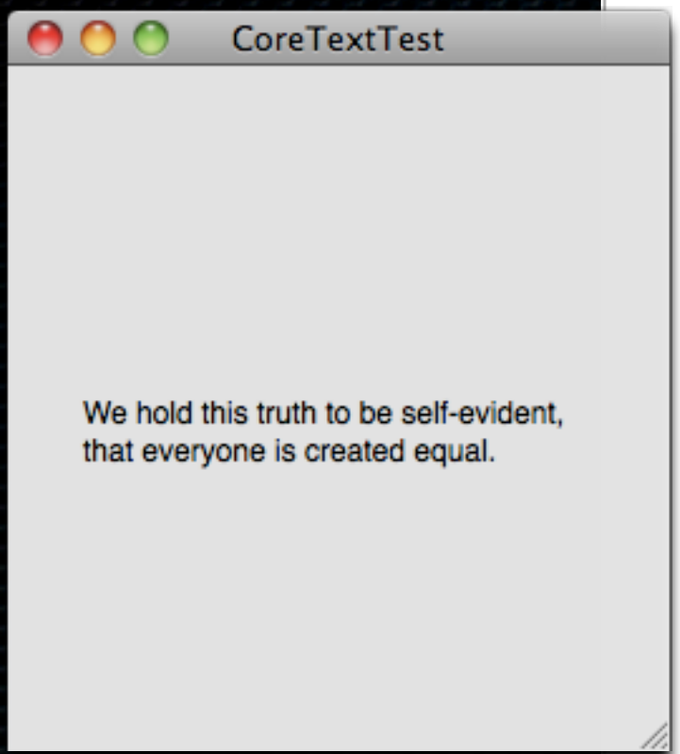
CTView.h:11 @interface CTView

```
1  //
2  //  CTView.h
3  //  CoreTextTest
4  //
5  //  Created by Jiang Jiang on 8/29/09.
6  //
7
8  #import <Cocoa/Cocoa.h>
9
10
11 @interface CTView : NSView {
12
13 }
14
15 @end
16
```

2. In Xcode, create a new class called "`CTView`", make it inherit from `NSView`.



3. Open `MainMenu.xib`, drag a `NSView` to the empty window, in Identity panel, set its class to "`CTView`".




```
- (void) drawRect:(NSRect)dirtyRect
{
    // Initialize a graphics context and set the text matrix to a known value.
    CGContextRef context = (CGContextRef)[[NSGraphicsContext currentContext]
                                                    graphicsPort];
    CGContextSetTextMatrix(context, CGAffineTransformIdentity);

    // Initialize a rectangular path.
    CGMutablePathRef path = CGPathCreateMutable();
    CGRect bounds = CGRectMake(10.0, 10.0, 200.0, 200.0);
    CGPathAddRect(path, NULL, bounds);

    // Initialize an attributed string.
    CFStringRef string = CFSTR("We hold this truth to be self-evident, that "
                               "everyone is created equal.");
    CFMutableAttributedStringRef attrString =
    CFAttributedStringCreateMutable(kCFAllocatorDefault, 0);
    CFAttributedStringReplaceString (attrString,
                                     CFRangeMake(0, 0), string);

    // Create the framesetter with the attributed string.
    CTFramesetterRef framesetter = CTFramesetterCreateWithAttributedString(attrString);
    CFRelease(attrString);

    // Create the frame and draw it into the graphics context
    CTFrameRef frame = CTFramesetterCreateFrame(framesetter,
                                                CFRangeMake(0, 0), path, NULL);
    CFRelease(framesetter);
    CTFrameDraw(frame, context);
    CFRelease(frame);
}
```

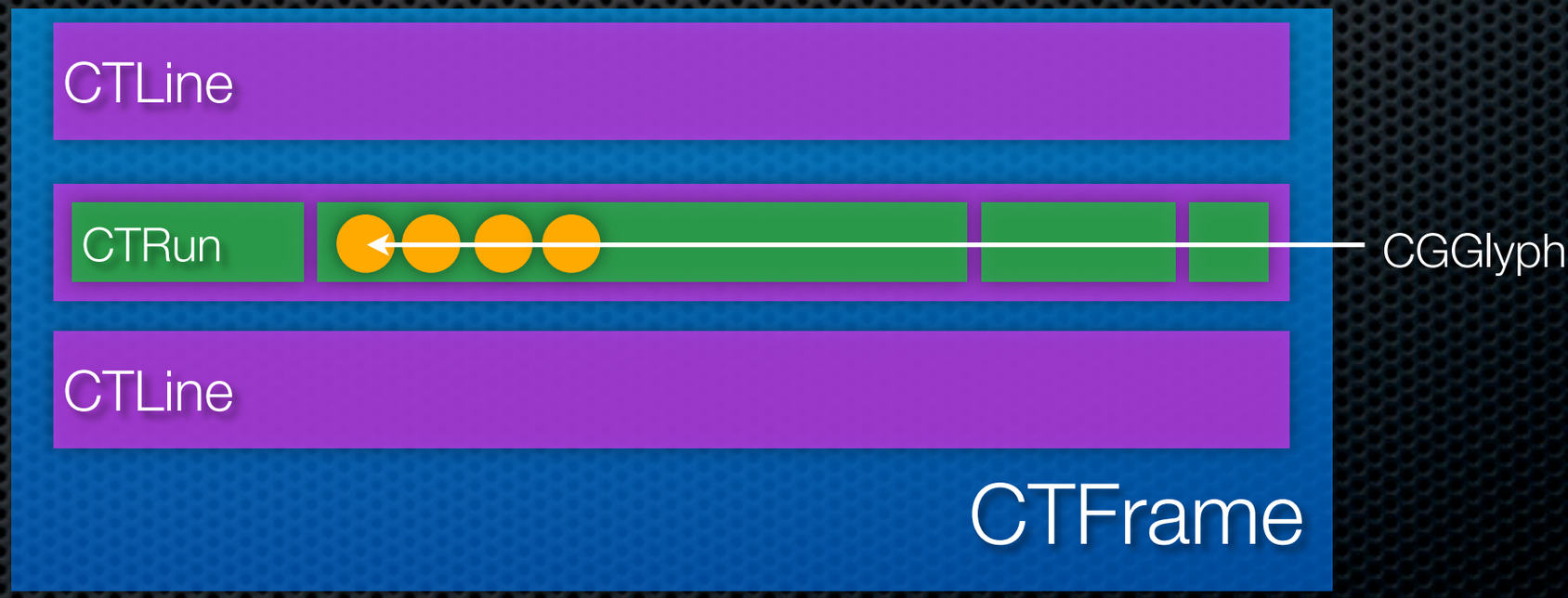
4. Then you can start your Core Text programming in `-drawRect:` of `CTView`, check the result every time you launch it.

Text Layout with Core Text:

Step by step

Basic Concepts

- Every time you try to layout some text, your input is a `CFAttributedString` and a `CGPath`, which describe the text, parameters to layout, and the area containing the resulting output.



Step 1: Prepare Attributed String

- First, you prepare a `CFAttributedString` with all the parameters set (to the whole string or part of the string):

```
NSAttributedString *str = [[NSAttributedString alloc]
    initWithString: @"We hold this truth..."
        attributes: [NSDictionary
    dictionaryWithObjectsAndKeys:
        [NSFont fontWithName: @"Helvetica" size: 12.0],
        (NSString *) kCTFontAttributeName]];

CFAttributedStringRef attrString =
    (CFAttributedStringRef) str;
```

Step 2: Create a CTFramesetter

- Now you can create the central object for Core Text typesetting: `CTFramesetter`, you will use it through the entire layout process:

```
CTFramesetterRef framesetter =
    CTFramesetterCreateWithAttributedString(attrString);
```

Step 3: Create a CGPath

- Third, you add one or more `CGRect` (or other shapes) to form a complete `CGPath`:

```
CGMutablePathRef path = CGPathCreateMutable();

CGRect bounds = CGRectMake(10.0, 10.0, 200.0, 200.0);

CGPathAddRect(path, NULL, bounds);
```

Step 4: Get the frame!

- Use the `CGPath` and `CTFramesetter` to create `CTFrame` (s), then draw it in current context!

```
CTFrameRef frame =
    CTFramesetterCreateFrame(framesetter,
                             CFRangeMake(0, 0),
                             path, NULL);

CTFrameDraw(frame, context);
```

Step 5: Don’t forget to release them

- Release everything you created with `CFRelease`.

```
CFRelease(attrString);
CFRelease(framesetter);
CFRelease(frame);
CFRelease(path);
```

Learn Core Text:
Techniques and Gotchas

Text Matrix

- Always remember to reset the text matrix before drawing, otherwise the result will be totally unpredictable, like using uninitialized memory:

```
CGContextSetTextMatrix(context, CGAffineTransformIdentity);
```

- How to draw the text from top to bottom? use a up-side-down transform:

```
CGContextSetTextMatrix(context,
                       CGAffineTransformMakeScale(1, -1));
```

Breaking layout into parts (1)

- Reading files into `CFAttributedString` is fast, but laying out them is much slower, but you don't need to layout them all at once!
- Set a fixed rectangle for the `CGPath`, say, one or two pages, and initialize a range to create the `CTFrame`
- Each time a `CTFrame` is created, you can get the text range within this frame with `CTFrameGetVisibleStringRange(frame)`

Breaking layout into parts (2)

- Update the range for text not typesetted yet, continue creating `CTFrame`s until there is no more
- Release the `CTFrame`s each time you finish drawing them

Breaking layout (code)

```
CFRange fullRange = CFRangeMake(0, text.length);
CGRect frameRect = CGRectMake(0, 0, 100, 1000);
for (range = frameRange = CFRangeMake(0, 0);
     range.location < fullRange.length;
     range.location += frameRange.length)
{
    CGMutablePathRef path = CGPathCreateMutable();
    CGPathAddRect(path, NULL, frameRect);
    CTFrameRef frame =
        CTFramesetterCreateFrame(framesetter,
                                 range, path, NULL);
    CTFrameDraw(frame);
    frameRange = CTFrameGetVisibleStringRange(frame);
    frameRect.origin.y += frameRect.size.height;

    CFRelease(path);
    CFRelease(frame);
}
```

Flipped drawing (1)

- Cocoa (and Core Graphics) used a PostScript-like, up-side-down drawing model, `(0, 0)` is at bottom left.
- It is extremely inconvenient for text layout, because you can't tell the `y` coordinates until you know the maximum height, but you don't know the height until all the text is typesetted
- The coordinates associated `CTFrame` is up-side-down, so we need to break frame into `CTLine`s with `CTFrameGetLines(frame)`

Flipped drawing (2)

- Retrieve the origins with `CTFrameGetLineOrigins(frame)`
- Calculate the line origin with coordinates flipped back
- Move to calculated coordinates, draw each line with `CTLineDraw(line)`

Flipped Drawing (code)

```
CFArrayRef lines = CTFrameGetLines(frame);
CFIndex i, total = CFArrayGetCount(lines);
CGFloat y;
CTFrameGetLineOrigins(frame,
                      CFRangeMake(0, total), origins);

for (i = 0; i < total; i++)
{
    CTLineRef line =
        (CTLineRef) CFArrayGetValueAtIndex(lines, i);
    y = frameRect.origin.y +
        frameRect.size.height - origins[i].y;
    CGContextSetTextPosition(context,
                             frameRect.origin.x +
                             origins[i].x, y);
    CTLineDraw(line, context)
}
```

Reference Reading

- "Core Text Programming Guide" from Apple is a must read
- Core Text session video/slides in WWDC 2008
- Text Rendering: an Excursion: http://jjgod.org/docs/slides/text-rendering-tech.pdf
- State of Text Rendering: http://behdad.org/text/

Reference Codes

- Core Text sample code from Apple
  A step by step tutorial.
- Textus: http://github.com/jjgod/textus
  Complete demo of multipart layout and flipped drawing
- Neuro: http://github.com/ishikawa/neuro
  Vertical layout and text editing support.
- vim-cocoa: http://github.com/jjgod/vim-cocoa
  Low-level drawing with `CTLine` and `CGGlyph`.

That’s all. Thanks.