Formatted Text in iOS

Richard Warren

About Myself

- Writing freelance articles for MacTech since 2006, many covering iOS Development
- Worked as a scientist for a small R&D Company
  - *Java Development*
  - *Research and Data Analysis*
- Now focusing on full-time iOS contract work

The Problem:

- Need to get information in front of your users.
- Good support for most media:
  - UIImageView for images
  - Core Graphics for graphs and illustrations
  - Multi-media support for video and audio
- But, what if you want to display formatted text?

Why I like this problem

- Seems to be a wide-spread problem, especially for developers moving from Mac OS X
- It doesn't feel like it has a good solution

Formatting Needs

- Formatting that defines function
  Title, Section Headers, Captions, Bullets, etc.
- Formatting that draws **attention**
  Bold, Italic, Underline, Font Color, etc.
- Basic Layouts
  Columns, Sidebars, Pull Quotes, etc.
- Incorporating other media in the text
  Images, Videos, Sound Effects, Interactivity, etc.
- Copy/Paste and Text Editing

Mac OS X Provides Excellent Support

The Macintosh has always been famous for its sophisticated text-handling capabilities

OS X Provides three main layers of support:

- Cocoa Text Systems:
  NSTextView, NSTextField and NSAttributedString
- WebKit
- Core Text

Less Support in iOS

- UITextView and UITextField only support one format at a time.
- Core Text unavailable until iOS 3.2
- NSAttributedString unavailable until iOS 3.2
  It still does not include the extension for building an attributed string from HTML or RTF.

What Are Our Options?

- UIWebView
- Static text layouts using labels and text views.
- UIStringDrawing Extension
- Core Animation's CATextLayer*
- Core Text*
- Core Graphics / Quartz 2D

**Not available until iOS 3.2*

For Most Cases

- Use UIWebView when you can
- Use Core Text (Core Graphics) when you can't
- Reserve other drawing techniques for specialized edge cases:
  - *Only use UIStringDrawing to draw small amounts of text in the current graphics context.*
  - *Only use CATextLayer to draw small amounts of text in Core Animation*

UIWebView

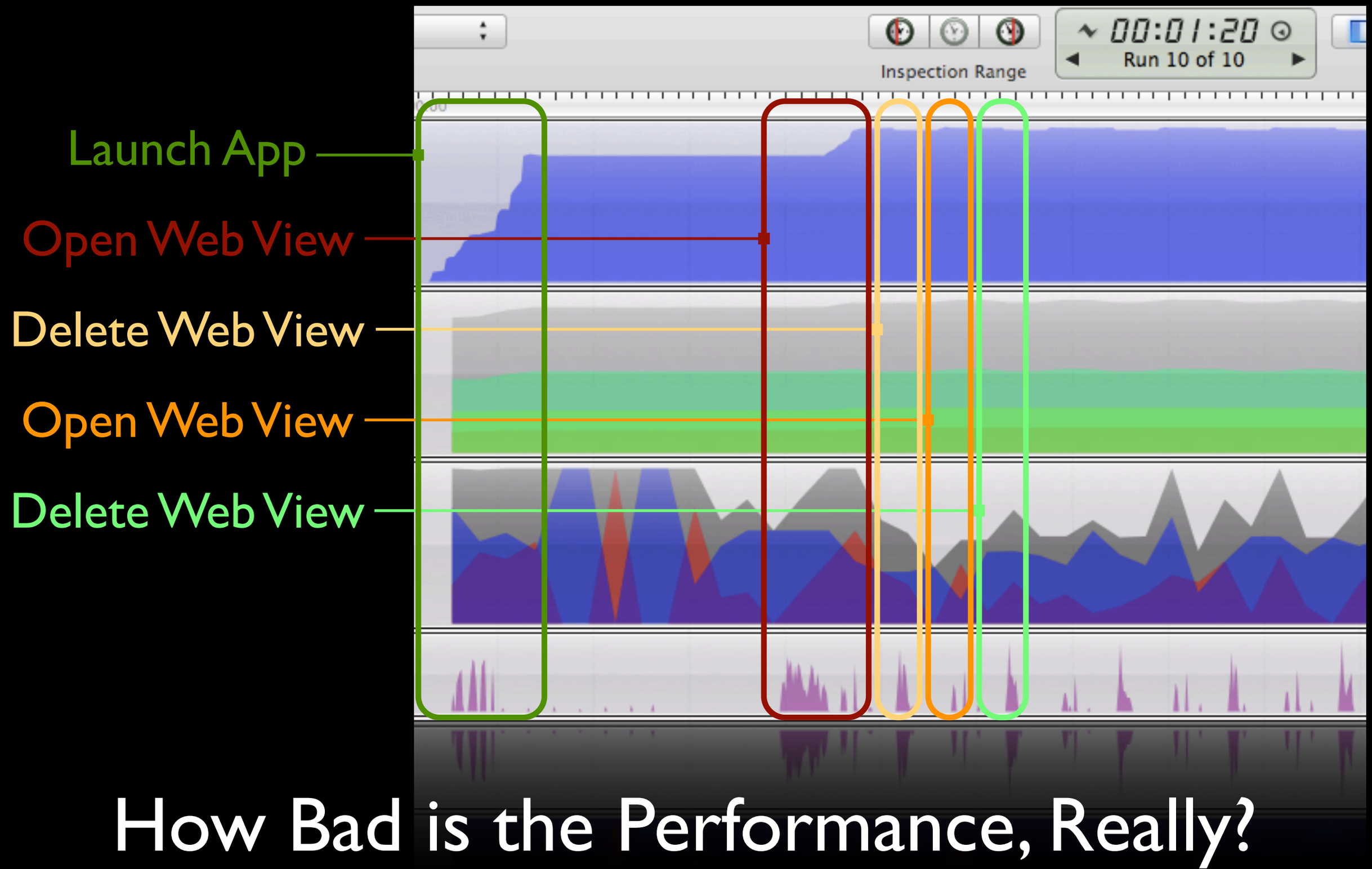
The Old Standby

UIWebView Advantages

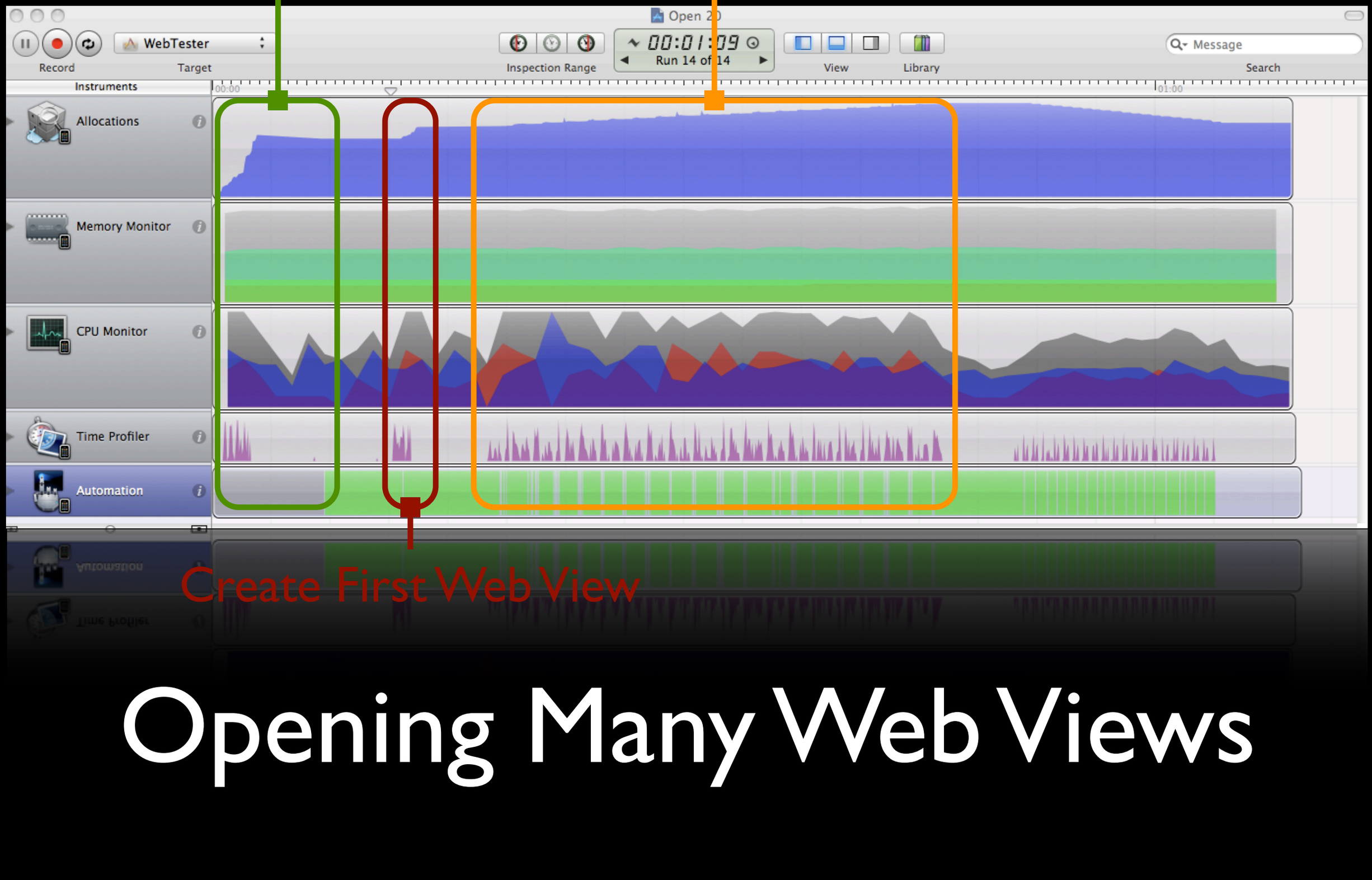
- Provides a complete text presentation system
- Supports (non-Flash) web technologies
- Separating the content (HTML) from the presentation (CSS)
- Allows reuse of existing web design expertise
- Full Copy/Paste support
- CSS can duplicate many of iOS's native controls.
- Easy communication between the web page and Objective-C code
  - *Intercept calls to load pages and call Objective-C Methods instead.*
  - *Objective-C code can call Javascript methods.*

UIWebView Problems

- Performance
  - *Initial launch is noticeably slow*
  - *Can become a memory hog*
- Do we really need full web support?
  - *Sometimes its better to use a scalpel than a jackhammer*
  - *Do we really need JavaScript*
- Very course-grain controls:
  - *Cannot easily change the default white background*
  - *Cannot intercept AJAX calls*
  - *UIWebViewDelegate is only notified when a page is completely loaded.*
  - *Auto-unloading of views may accidentally delete your content.*



How Bad is the Performance, Really?



Opening Many Web Views

For Those Who Like Numbers

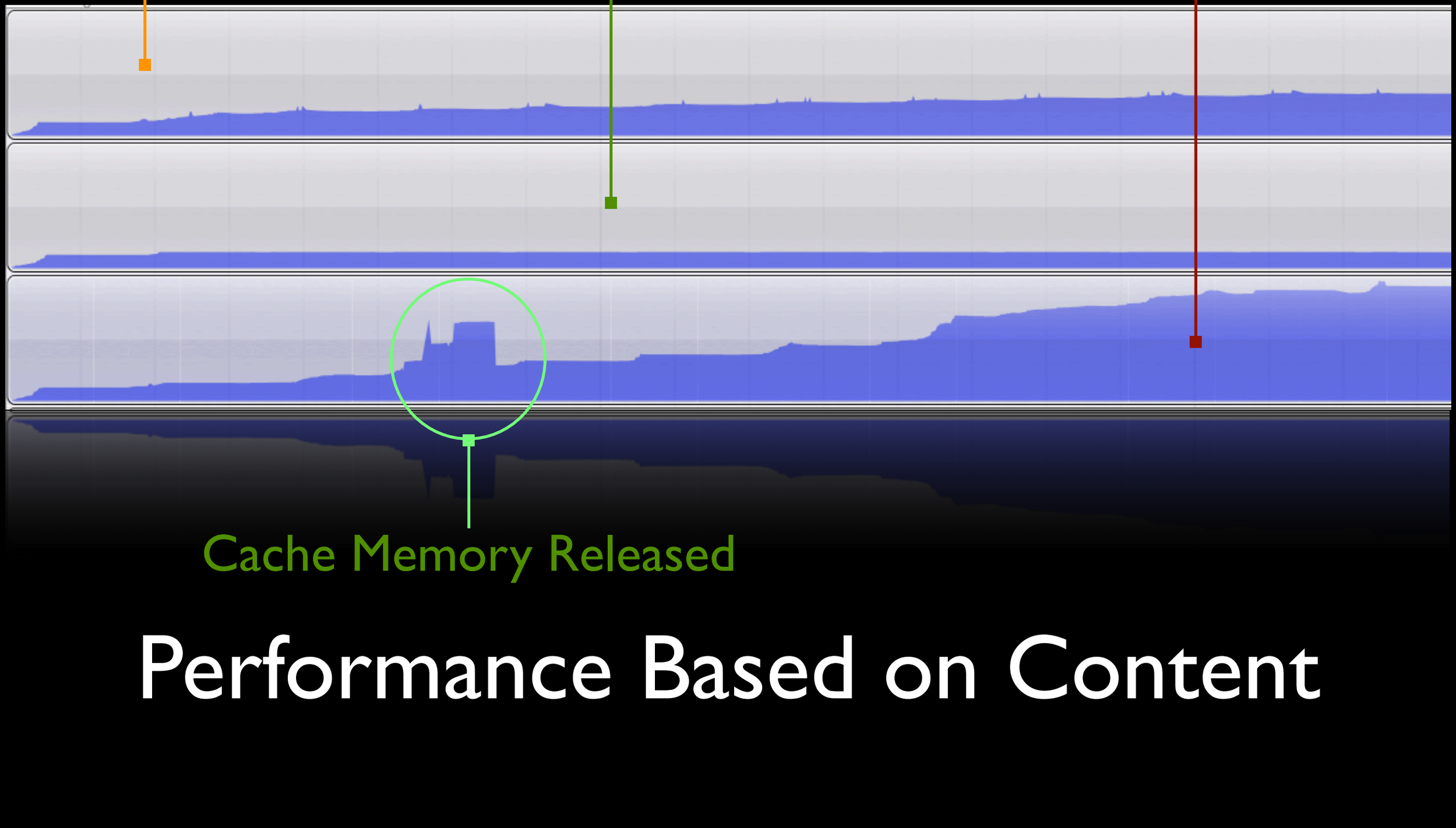
- Creating the Initial Web View: 0.28 MB
- Releasing the Initial Web View: -0.02 MB
- Creating additional Web Views: 0.03 MB

Note: Creating and deleting 20 Web Views resulted in an extra 0.18 MB not being released--probably residual memory in caches.

cnn.com

simple web page

random web pages



Cache Memory Released

Performance Based on Content

UIWebView Conclusions

- Overall the UIWebView is a reasonable approach for most use cases
- Pre-load the initial web view to improve performance
- Don’t be afraid to create multiple web views
- Sometimes, you need clever tricks to work around its idiosyncrasies
- Test on realistic content

Core Text

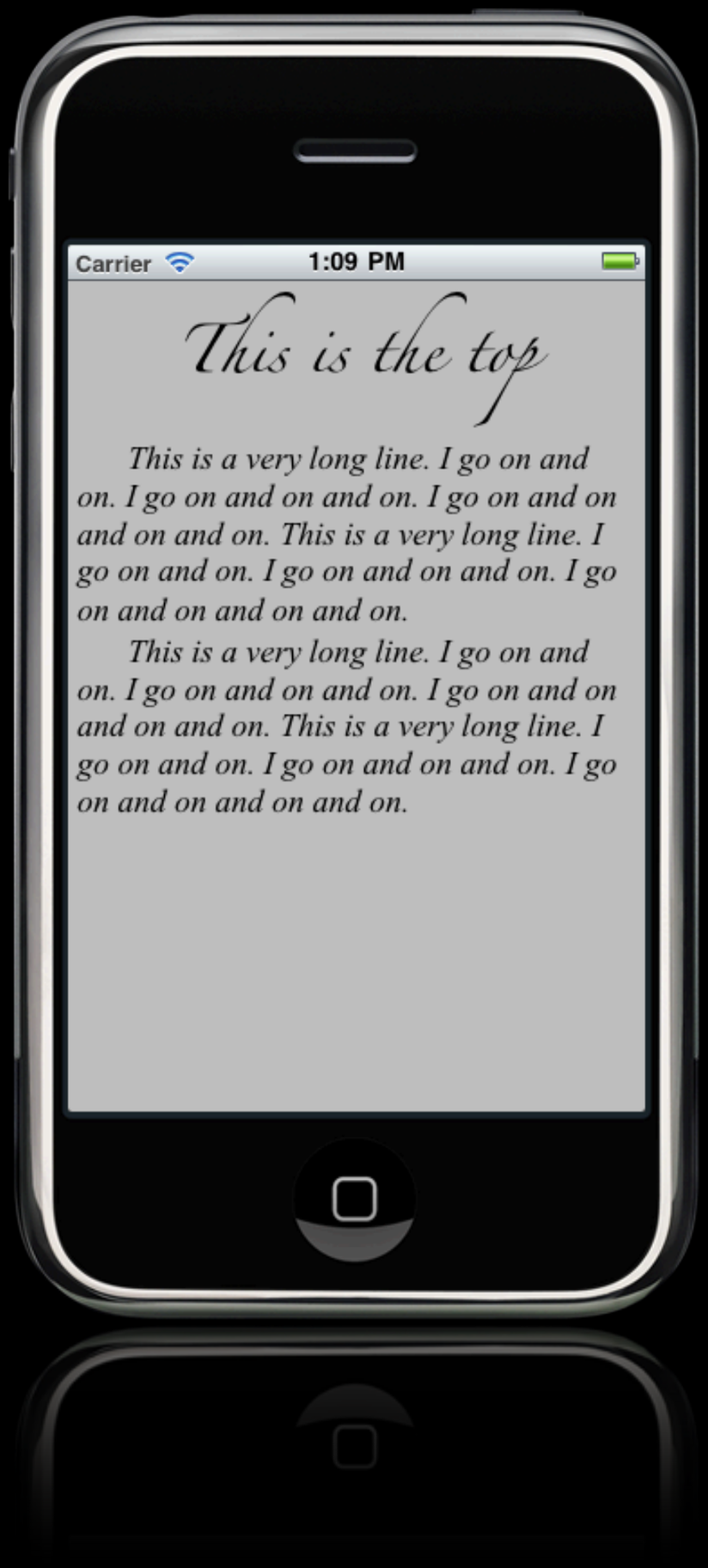
High-Octane Text Rendering

Core Text Advantages

- Designed for High Performance and Ease of Use
- Provides higher-quality typographical control
  - *Kerning*
  - *Ligatures*
- Great for paging apps
- Tight integration with Core Graphics (Quartz)
- Even though Core Text is a C API, it supports many Objective-C Objects through toll-free bridging

Core Text Disadvantages

- Very Low Level
- No support for other media
- No support for separating content and presentation
- No support for copy and paste


Carrier
1:09 PM
This is the top
This is a very long line. I go on and on. I go on and on and on. I go on and on and on and on. This is a very long line. I go on and on. I go on and on and on. I go on and on and on and on.
This is a very long line. I go on and on. I go on and on and on. I go on and on and on and on. This is a very long line. I go on and on. I go on and on and on. I go on and on and on and on.

```
74
75  // This will be the main text-drawing view.
76  - (void)drawRect:(CGRect)rect {
77
78      // setup
79      CGContextRef context = UIGraphicsGetCurrentContext();
80
81      // This alters the orientation for font drawing.
82      CGContextTranslateCTM(context, 0, self.bounds.size.height);
83      CGContextScaleCTM(context, 1.0, -1.0);
84
85      // Create the Frame
86      UIBezierPath* path =
87          [UIBezierPath bezierPathWithRect:[self contentBounds]];
88
89      CTFramesetterRef framesetter =
90          CTFramesetterCreateWithAttributedString((CFAttributedStringRef)self.text);
91
92      CTFrameRef frame =
93          CTFramesetterCreateFrame(framesetter, CFRangeMake(0, 0), path.CGPath, nil);
94
95      // Draw the text
96      CTFrameDraw(frame, context);
97
98
99      // Release the frame
100     CFRelease(framesetter);
101     CFRelease(frame);
102 }
```

Drawing Text

```objc
// Create the string with title text
NSMutableAttributedString* string =
    [[NSMutableAttributedString alloc]
        initWithString:@"This is the top\n"];

// Set the title Font
CTFontRef titleFont = CTFontCreateWithName(CFSTR("Zapfino"), 24, nil);

[string addAttribute:(id)kCTFontAttributeName
               value:(id)titleFont
               range:NSMakeRange(0, 15)];

CFRelease(titleFont);

// Center the title
CTTextAlignment theAlignment = kCTCenterTextAlignment;
CTParagraphStyleSetting alignment;
alignment.spec = kCTParagraphStyleSpecifierAlignment;
alignment.valueSize = sizeof(CTTextAlignment);
alignment.value = &theAlignment;

CTParagraphStyleRef centered = CTParagraphStyleCreate(&alignment, 1);

[string addAttribute:(id)kCTParagraphStyleAttributeName
               value:(id)centered
               range:NSMakeRange(0, 15)];

CFRelease(centered);
```

Building The Title

```
// Now add the body text
for (int i = 0; i < 2; i++) {

    NSAttributedString *addition =
        [[NSAttributedString alloc] initWithString:@"\tThis is a very lon

    [string appendAttributedString:addition];
    [addition release];
}

// Set the body text's font
CTFontRef bodyFont =
    CTFontCreateWithName(CFSTR("Times New Roman"), 18, nil);

CTFontRef bodyItalicFont =
    CTFontCreateCopyWithSymbolicTraits(bodyFont,
                                       0.0,
                                       nil,
                                       kCTFontItalicTrait,
                                       kCTFontItalicTrait);

[string addAttribute:(id)kCTFontAttributeName
               value:(id)bodyItalicFont
               range:NSMakeRange(16, [string length] - 17)];

CFRelease(bodyFont);
CFRelease(bodyItalicFont);
```

Building the Body Text

Core Text Conclusions

- Use Core Text when you need more performance or control than the UIWebView can provide
- Be prepared to do a lot of the work yourself

Questions?

Contact Info:

Rich Warren

email: rikiwarren@me.com

blog: http://freelancemadscience.blogspot.com

twitter:@rikiwarren