iPhone Programming

CMSC 498i – Spring 2010

Drawing

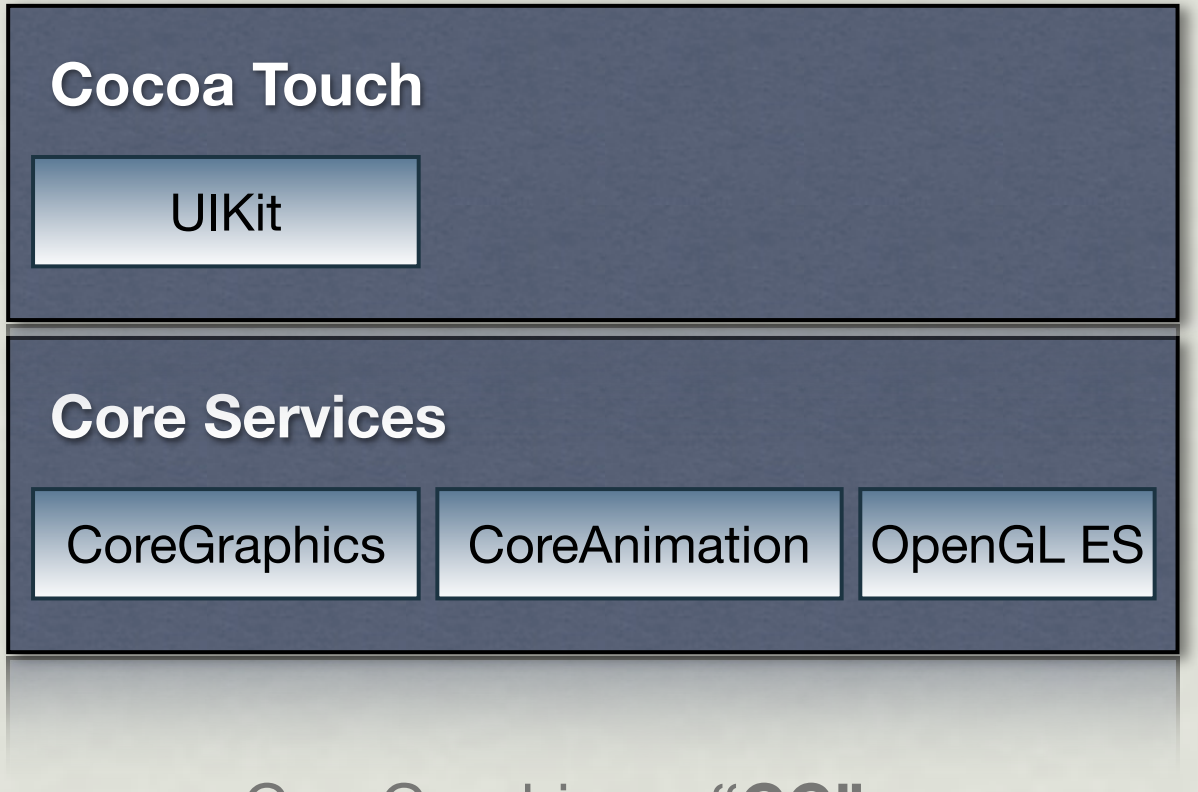
Lecture #12 – Chuck Pisula

Today's Topics

- Where To Draw
  - `UIView`
  - Other destinations
- How To Draw
  - `UIKit` APIs
  - `CoreGraphics` APIs (Quartz 2D)

Overview

System Components



CoreGraphics = **“CG”**

CoreAnimation = **“CA”**

System Components

- UIKit provides some drawing and animation capabilities
- Core Graphics – vector graphics, bitmap and PDF content
  - Provides more powerful drawing options
- Core Animation – high performance compositing / animation
  - Provides more powerful animation option
  - Provides CALayer - the core rendering surface for UIKit, OpenGL
- OpenGL ES – embedded OpenGL APIs

Drawing Destinations

- `UIView`
  - Typically all you will ever need to do
  - We will focus mostly on drawing into `UIViews`

Drawing Destinations

- `UIView`
  - Typically all you will ever need to do
  - We will focus mostly on drawing into `UIViews`
- OpenGLES
  - For 2D / 3D graphics. Will discuss later in the semester if time

Drawing Destinations

- `UIView`
  - Typically all you will ever need to do
  - We will focus mostly on drawing into `UIViews`
- OpenGLES
  - For 2D / 3D graphics. Will discuss later in the semester if time
- Occasionally may draw into other destinations
  - Directly into Core Animation Layer
  - Core Graphics Layer
  - Bitmap “context”

Choosing What To Use

- `UIView` is typically all you need for drawing / animation

- Use CG, CA if you…
  - Need more options
  - Performance
  - Portability with Mac OS X
    - UIKit is iPhone only
    - CG, CA available on both platforms

Drawing System

How it works

Drawing Phases

Drawing Phases

- User interacts with application UI

Drawing Phases

- User interacts with application UI
- Event handled by application code
  - Processing code decides to refresh the UI
  - `-setNeedsDisplay` / `-setNeedsLayout`

Drawing Phases

- User interacts with application UI
- Event handled by application code
  - Processing code decides to refresh the UI
  - `-setNeedsDisplay` / `-setNeedsLayout`
- Update requests coalesced
  - Views needing re-layout asked to `-layoutSubviews`
  - Views needing re-draw asked to `-drawRect:`

Drawing Phases

- User interacts with application UI
- Event handled by application code
  - Processing code decides to refresh the UI
  - `-setNeedsDisplay` / `-setNeedsLayout`
- Update requests coalesced
  - Views needing re-layout asked to `-layoutSubviews`
  - Views needing re-draw asked to `-drawRect:`
- Updates composited with rest of the visible views

Rendering Architecture

- Underneath the hood....

- iPhone (`UIView`, CA) is a compositing engine
  - You draw and provide content to the system
  - System gets content in terms of layers
  - Multiple (overlapping allowed) layers are composited together and rendered into a final scene

Options For Drawing

- By Composing Views
  - Combine and position views within `-layoutSubviews`
  - Up to this point, we have used `UIImageView` to “draw”

Options For Drawing

- By Composing Views
  - Combine and position views within `-layoutSubviews`
  - Up to this point, we have used `UIImageView` to “draw”
- Custom `UIView` drawing using…
  - Draw inside of `-(void)drawRect:(CGRect)rect`
  - `UIImages` directly
  - UIKit drawing primitives
  - CoreGraphics APIs

Composition

- Example: Subclass `UITableViewCell` and compose subviews

Composition

- Example: Subclass `UITableViewCell` and compose subviews

MyTableViewCell

Composition

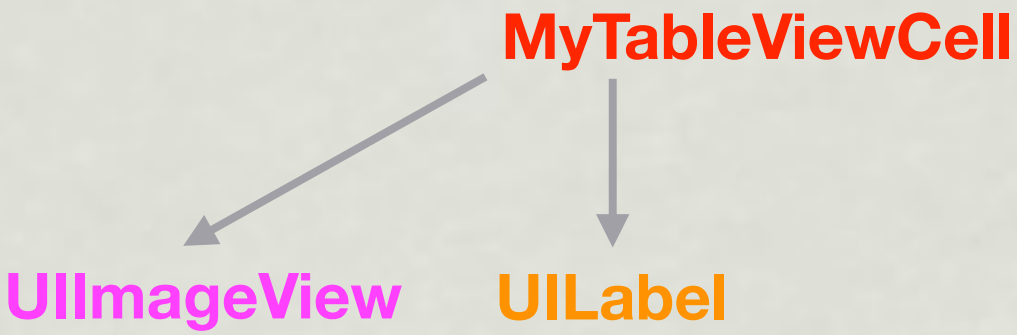
- Example: Subclass `UITableViewCell` and compose subviews

MyTableViewCell

UIImageView

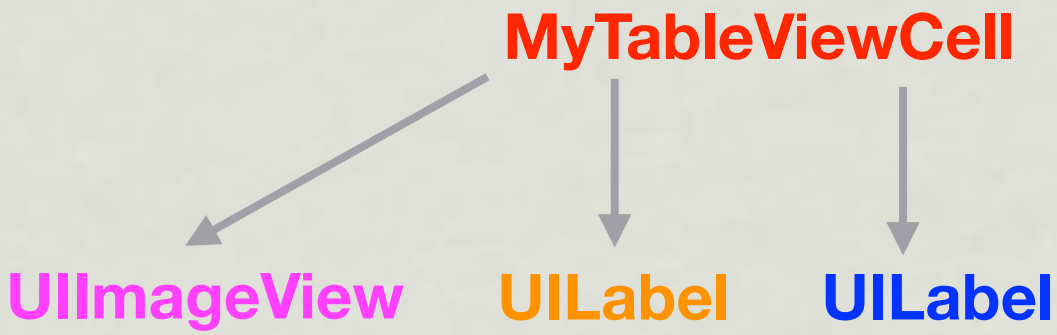
Composition

- Example: Subclass `UITableViewCell` and compose subviews



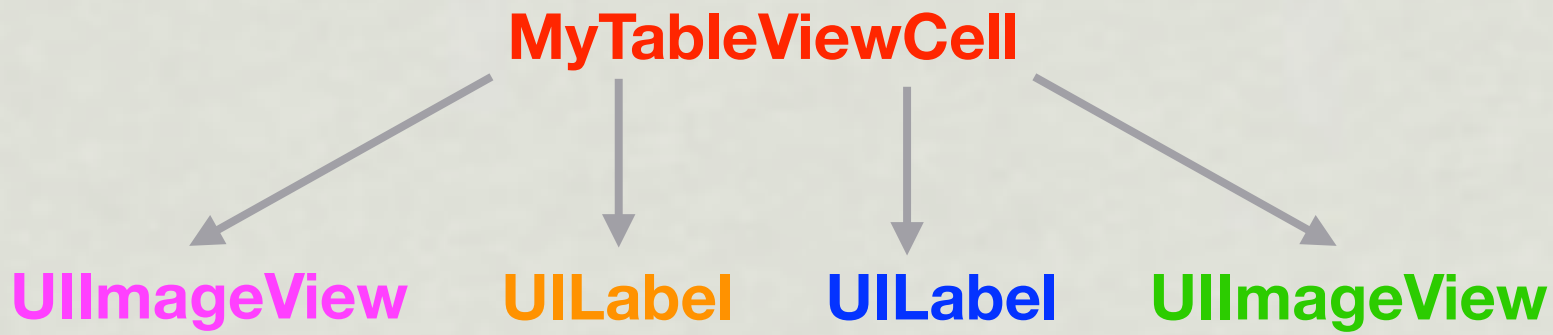
Composition

- Example: Subclass `UITableViewCell` and compose subviews



Composition

- Example: Subclass `UITableViewCell` and compose subviews



Drawing

- Example: Subclass `UITableViewCell` and draw in `-drawRect:`

Drawing

- Example: Subclass `UITableViewCell` and draw in `-drawRect:`

MyTableViewCell

Drawing

- Example: Subclass `UITableViewCell` and draw in `-drawRect:`

MyTableViewCell

- `NSString` drawing API for 2 labels
- `UIImage` drawing API for 2 images

When Drawing Happens

- View composition uses deferred model
  - Call `-setNeedsLayout` to say a view needs to relayout
  - Later, just before display, `-layoutSubviews` called

When Drawing Happens

- View composition uses deferred model
  - Call `-setNeedsLayout` to say a view needs to relayout
  - Later, just before display, `-layoutSubviews` called
- Drawing also uses a deferred model
  - Call `-setNeedsDisplay` to say a view needs to be redrawn
  - `-setNeedsDisplayInRect:` to say just a portion needs redrawing
  - Later, the `-drawRect:` method is called

UIView

```
- (void)drawRect:(CGRect)rect;
```

- UIKit calls `-[view drawRect:]` for each view needing display
- Views should redraw as quickly as possible
- Draw only your content, not subviews
- Draw just the region specified
    - ‘rect’ is not necessarily == view.bounds

Where Drawing Happens

- All drawing requires a **graphics context**
    - Encapsulation of information needed for drawing to a destination

Graphics Context

- Represented as `CGContextRef`
- Encapsulates info need when drawing
  - Drawing properties – Colors, fonts, line styles, stroke width, …
  - CTM (current transform matrix)
  - Clipping Info

Graphics Context

- Represented as `CGContextRef`
- Encapsulates info need when drawing
  - Drawing properties – Colors, fonts, line styles, stroke width, …
  - CTM (current transform matrix)
  - Clipping Info
- Changes to graphics context affect future drawing
  - E.g. Change the current context's stroke color

```
[[UIColor redColor] setStroke];
```

Graphics Context

- Creating a graphics context
  - UIKit automatically configures a context before `-drawRect:`
  - Can create your own (for rending to an image, PDF, etc...)
- Specifying the graphics context
  - UIKit maintains a stack of graphics contexts, uses topmost context
  - Some UIKit API asks you to pass a context as a parameter
  - Most CoreGraphics APIs require context parameter

Graphics Context

- `UIView` provides access to its current graphics context

```
CGContextRef UIGraphicsGetCurrentContext();
```

- Context will be set up and current before `-drawRect:`
- Can save / restore state when making temporary changes

```
void UIGraphicsPushContextContext(CGContextRef context);
void UIGraphicsPopContext(void)
```

- Custom graphics context – more on this later…

Drawing

APIs

Drawing APIs

- Where to Find
  - `UIGraphics.h`
  - `UIStringDrawing.h`
  - `UIImage`
  - `UIColor`, `UIFont`
  - `UIInterface.h` – UI specific values for colors, fonts, ...
  - CoreGraphics framework

UIGraphics

- Simple primitives

```
// Draw a filled rectangle
void UIRectFillUsingBlendMode(CGRect rect, CGBlendMode blendMode);
void UIRectFill(CGRect rect);

// Draw a rectangle frame
void UIRectFrameUsingBlendMode(CGRect rect, CGBlendMode blendMode);
void UIRectFrame(CGRect rect);
```

UIGraphics

- Simple primitives

```
// Draw a filled rectangle
void UIRectFillUsingBlendMode(CGRect rect, CGBlendMode blendMode);
void UIRectFill(CGRect rect);

// Draw a rectangle frame
void UIRectFrameUsingBlendMode(CGRect rect, CGBlendMode blendMode);
void UIRectFrame(CGRect rect);
```

- Example 

```
[[UIColor blueColor] set];
UIRectFill(rect);

[[UIColor redColor] set];
UIRectFrame(rect);
```

String Drawing

- `UIStringDrawing.h` defines a category on `NSString`
  - Draw string at a point, in rect
  - Calculate string size
  - Specify font
  - Color comes from graphics context
  - Options
    - Alignment
    - Truncation Styles

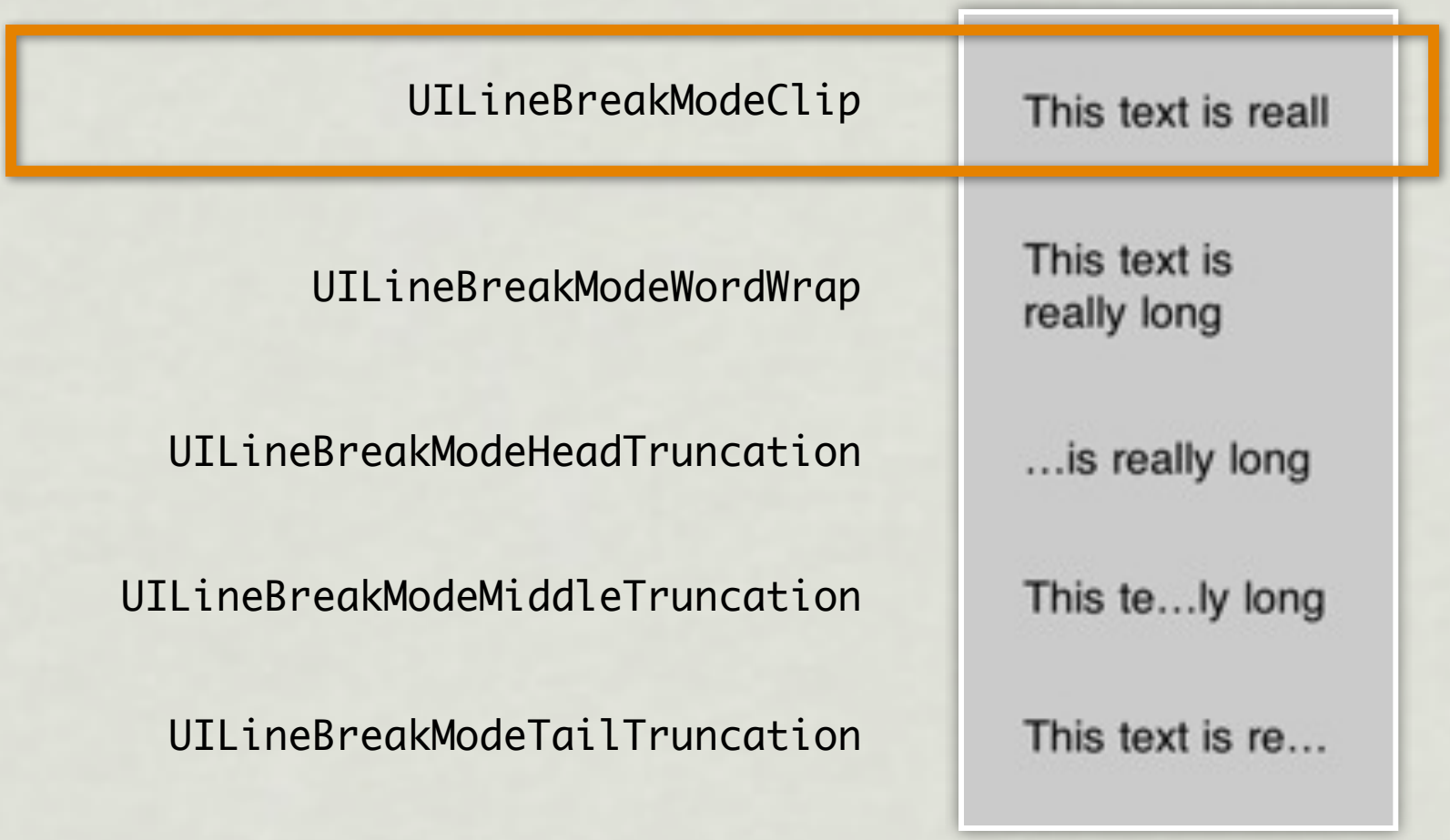
Alignment

"This text is really long"

`UILineBreakModeClip`	This text is reall
`UILineBreakModeWordWrap`	This text is really long
`UILineBreakModeHeadTruncation`	…is really long
`UILineBreakModeMiddleTruncation`	This te…ly long
`UILineBreakModeTailTruncation`	This text is re…

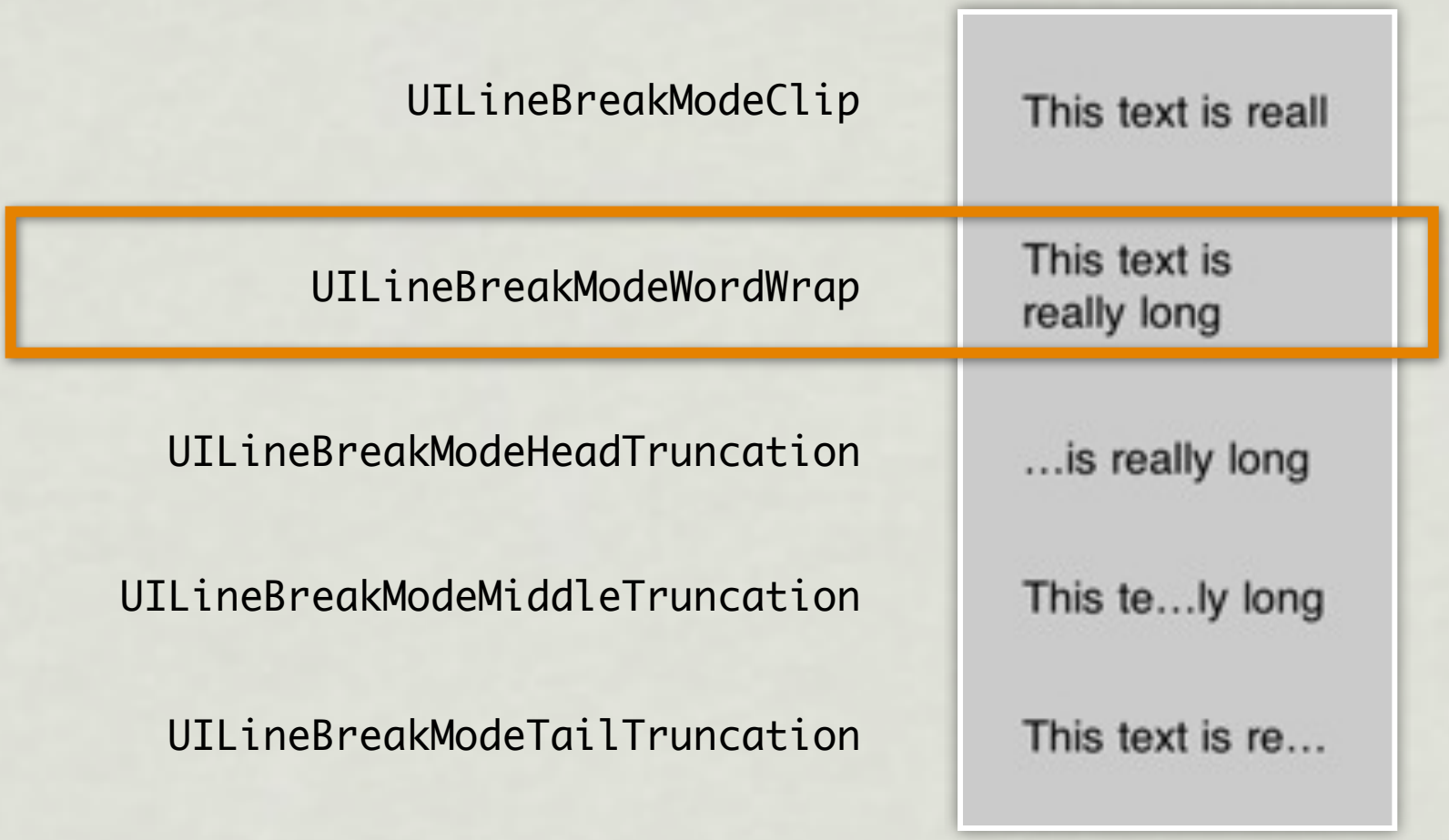
Alignment

"This text is really long"



Alignment

"This text is really long"



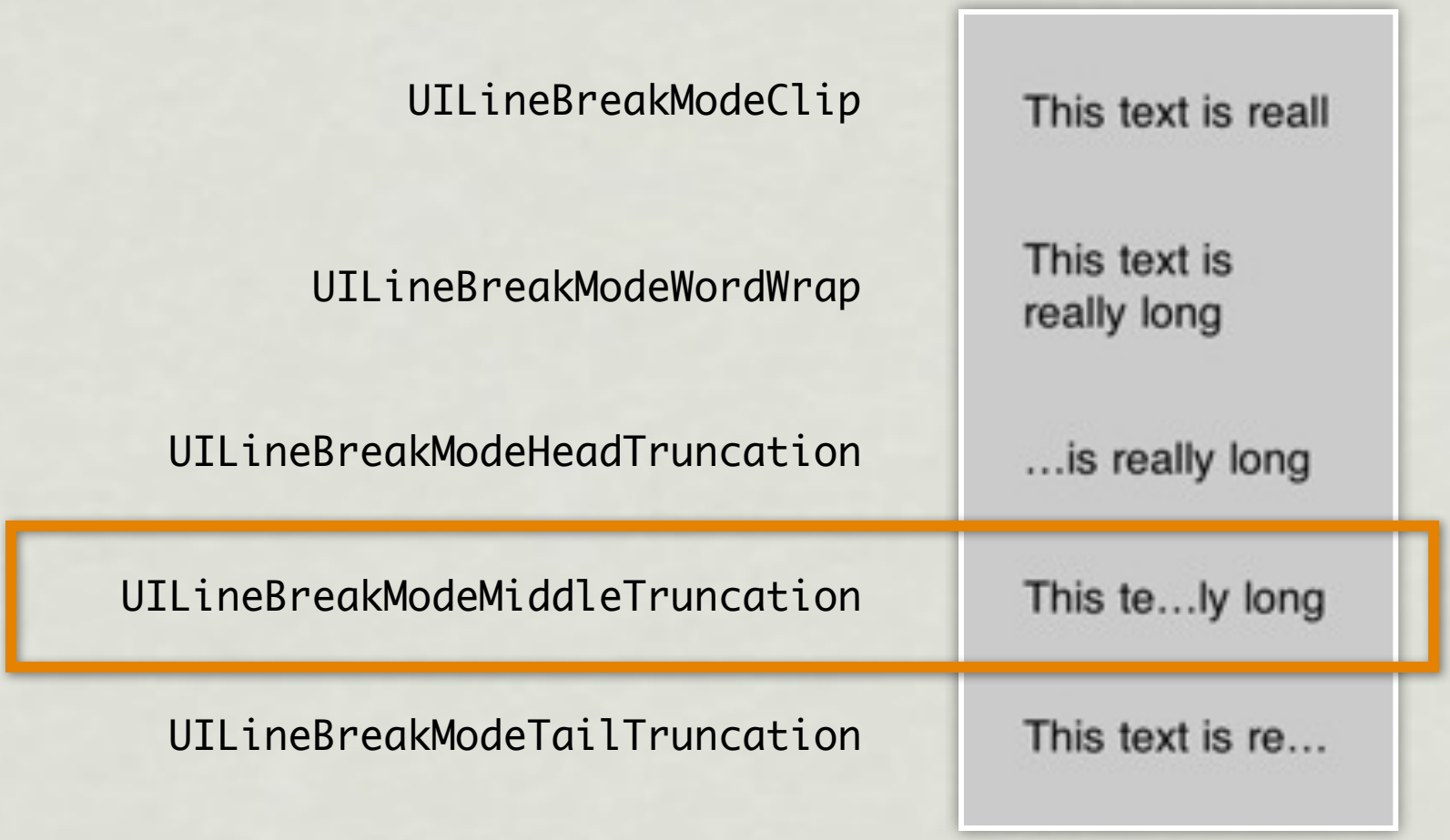
Alignment

"This text is really long"

`UILineBreakModeClip`	This text is reall
`UILineBreakModeWordWrap`	This text is really long
`UILineBreakModeHeadTruncation`	...is really long
`UILineBreakModeMiddleTruncation`	This te...ly long
`UILineBreakModeTailTruncation`	This text is re...

Alignment

"This text is really long"



Alignment

"This text is really long"

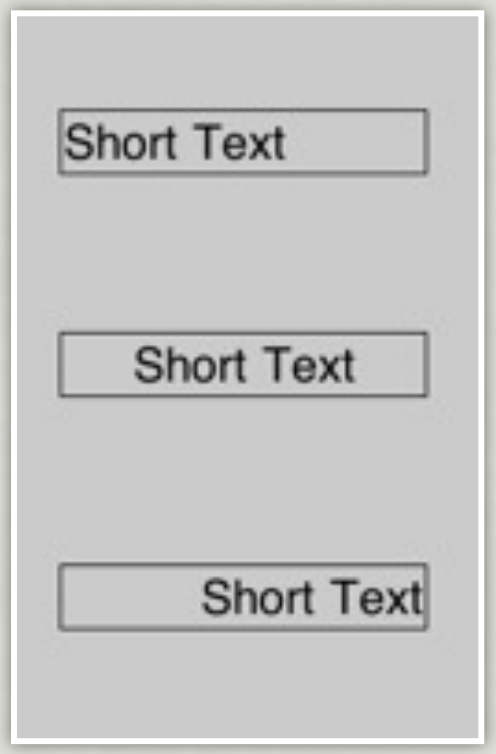
`UILineBreakModeClip`	This text is reall
`UILineBreakModeWordWrap`	This text is really long
`UILineBreakModeHeadTruncation`	…is really long
`UILineBreakModeMiddleTruncation`	This te…ly long
`UILineBreakModeTailTruncation`	This text is re…

Alignment

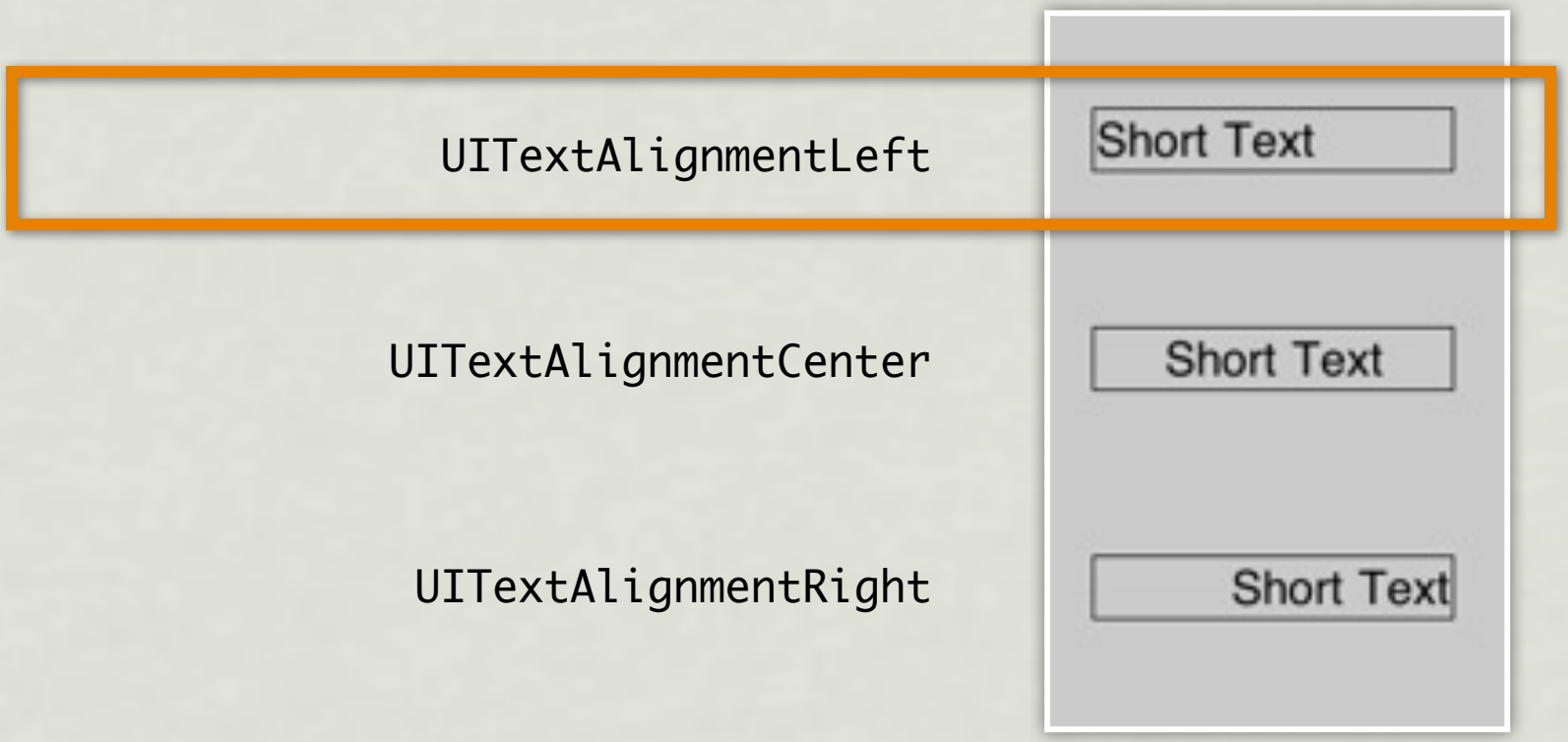
`UITextAlignmentLeft`

`UITextAlignmentCenter`

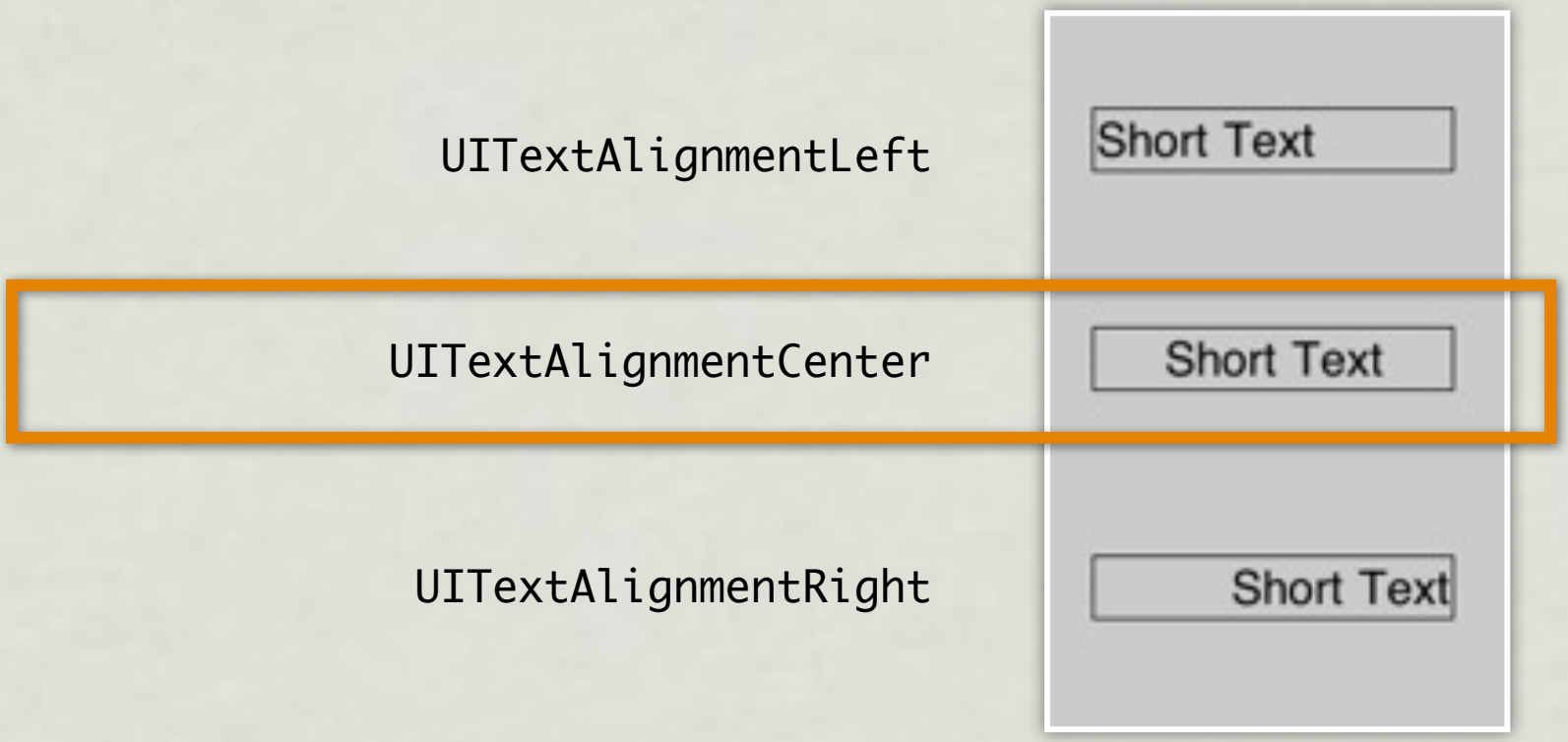
`UITextAlignmentRight`



Alignment


UITextAlignmentLeft
Short Text
UITextAlignmentCenter
Short Text
UITextAlignmentRight
Short Text

Alignment

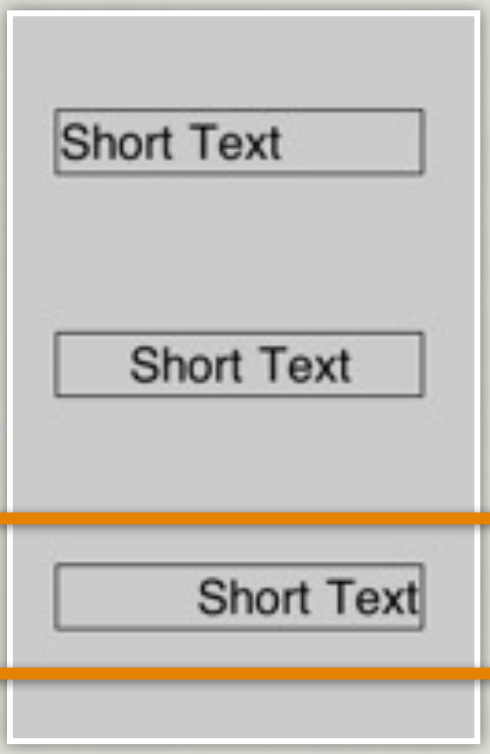

UITextAlignmentLeft
Short Text
UITextAlignmentCenter
Short Text
UITextAlignmentRight
Short Text

Alignment

UITextAlignmentLeft

UITextAlignmentCenter

UITextAlignmentRight



String Drawing

- Sample drawing API from `UIStringDrawing.h`

```
@interface NSString(UIStringDrawing)
```

String Drawing

- Sample drawing API from `UIStringDrawing.h`

```
@interface NSString(UIStringDrawing)

// Simple drawing, with no special options
- (CGSize)drawAtPoint:(CGPoint)point withFont:(UIFont *)font;
```

String Drawing

- Sample drawing API from `UIStringDrawing.h`

```
@interface NSString(UIStringDrawing)

// Simple drawing, with no special options
- (CGSize)drawAtPoint:(CGPoint)point withFont:(UIFont *)font;

// String drawing with options for truncation and alignment
- (CGSize)drawInRect:(CGRect)rect
            withFont:(UIFont *)font
       lineBreakMode:(UILineBreakMode)lineBreakMode
           alignment:(UITextAlignment)alignment;
```

String Drawing

- Sample drawing API from `UIStringDrawing.h`

```
@interface NSString(UIStringDrawing)

// Simple drawing, with no special options
- (CGSize)drawAtPoint:(CGPoint)point withFont:(UIFont *)font;

// String drawing with options for truncation and alignment
- (CGSize)drawInRect:(CGRect)rect
            withFont:(UIFont *)font
       lineBreakMode:(UILineBreakMode)lineBreakMode
           alignment:(UITextAlignment)alignment;

// String drawing with option to auto-shrink font until the text fits...
- (CGSize)drawAtPoint:(CGPoint)point forWidth:(CGFloat)width
              withFont:(UIFont *)font
          minFontSize:(CGFloat)minFontSize
       actualFontSize:(CGFloat *)actualFontSize
        lineBreakMode:(UILineBreakMode)lineBreakMode
   baselineAdjustment:(UIBaselineAdjustment)baselineAdjustment;
```

String Drawing

- Sample Sizing APIs

```
// Calculate full size given font
- (CGSize)sizeWithFont:(UIFont *)font;

// Calculate a constrained size
// Constraint can cause truncation, which would shrink displayed size
- (CGSize)sizeWithFont:(UIFont *)font
     constrainedToSize:(CGSize)size
         lineBreakMode:(UILineBreakMode)lineBreakMode;

// Calculate size assuming auto-shrinking is okay
- (CGSize)sizeWithFont:(UIFont *)font
           minFontSize:(CGFloat)minFontSize
        actualFontSize:(CGFloat *)actualFontSize
              forWidth:(CGFloat)width
         lineBreakMode:(UILineBreakMode)lineBreakMode;
```

UIImage

- Creation Methods
  - Named image file from application bundle
  - Path to image file
  - Using raw data
  - Using `CGImageRef`
- Features
  - Drawing methods
  - Size
  - “Stretchable” images

UIImage APIs

- Creation

```
+ (UIImage *)imageNamed:(NSString *)name;      // load from main bundle

- (id)initWithContentsOfFile:(NSString *)path;
- (id)initWithData:(NSData *)data;
- (id)initWithCGImage:(CGImageRef)imageRef;
```

UIImage APIs

- Creation

```
+ (UIImage *)imageNamed:(NSString *)name;      // load from main bundle

- (id)initWithContentsOfFile:(NSString *)path;
- (id)initWithData:(NSData *)data;
- (id)initWithCGImage:(CGImageRef)imageRef;
```

- Drawing

```
- (void)drawAtPoint:(CGPoint)point blendMode:(CGBlendMode)blendMode
              alpha:(CGFloat)alpha;

- (void)drawInRect:(CGRect)rect blendMode:(CGBlendMode)blendMode
             alpha:(CGFloat)alpha;
```

Stretchable Image

- When drawn in area bigger than image
  - Caps will not be scaled or resized
  - Middle 1-pixel portion repeated to fill up to end cap
  - Often used with resizable buttons

Stretchable Image

- When drawn in area bigger than image
  - Caps will not be scaled or resized
  - Middle 1-pixel portion repeated to fill up to end cap
  - Often used with resizable buttons

Stretchable Image

- When drawn in area bigger than image
  - Caps will not be scaled or resized
  - Middle 1-pixel portion repeated to fill up to end cap
  - Often used with resizable buttons


```
- (UIImage *)stretchableImageWithLeftCapWidth:(NSInteger)leftCapWidth
                                  topCapHeight:(NSInteger)topCapHeight;
```

UIColor

- Represents color and opacity
  - Normal Colors - RGB, HSB, ...
  - Pattern Image “Colors”
- Standard colors – red, orange, brown, …
- Semantic colors - table background, ...
- Modify Graphics Context - set fill, stroke
- Backed by `CGColorRef`
- Immutable

UIColor API

```
// Creation
- (UIColor *)initWithWhite:(CGFloat)white alpha:(CGFloat)alpha;
- (UIColor *)initWithRed:(CGFloat)red
                   green:(CGFloat)green
                  blue:(CGFloat)blue
                 alpha:(CGFloat)alpha;

// Get a color with just a different transparency value
- (UIColor *)colorWithAlphaComponent:(CGFloat)alpha;
```

UIColor API

```
// Creation
- (UIColor *)initWithWhite:(CGFloat)white alpha:(CGFloat)alpha;
- (UIColor *)initWithRed:(CGFloat)red
                   green:(CGFloat)green
                  blue:(CGFloat)blue
                 alpha:(CGFloat)alpha;

// Get a color with just a different transparency value
- (UIColor *)colorWithAlphaComponent:(CGFloat)alpha;
```

```
// Fancy...
- (UIColor *)initWithPatternImage:(UIImage*)image;

// Access the underlying CGColorRef
@property(nonatomic,readonly) CGColorRef CGColor;
```

UIColor API

```
// Standard Colors
+ (UIColor *)whiteColor;      // 1.0 white
+ (UIColor *)blueColor;       // 0.0, 0.0, 1.0 RGB
...
+ (UIColor *)orangeColor;     // 1.0, 0.5, 0.0 RGB
+ (UIColor *)clearColor;      // 0.0 white, 0.0 alpha

// System Colors ( UIInterface.h )
+ (UIColor *)groupTableViewBackgroundColor;
...
```

```
// Modify the current CGContextRef
- (void)set; // set both fill and stroke to the same color
- (void)setFill;
- (void)setStroke;
```

UIFont

- API for getting fonts
  - By name – e.g. “Helvetica”
  - System fonts – e.g. `+boldSystemFontOfSize:`
  - System font sizes – e.g. `+boldSystemFontSize`
- Access to font metrics
  - point size, ascender, descender, …
  - Explanations of what these can be found with UIFont docs…

UIFont API

```
// Access
+ (UIFont *)fontWithName:(NSString *)fontName size:(CGFloat)fontSize;
+ (UIFont *)systemFontOfSize:(CGFloat)fontSize;
+ (UIFont *)boldSystemFontOfSize:(CGFloat)fontSize;

// Get a font with base on this one, with a different size
- (UIFont *)fontWithSize:(CGFloat)fontSize;

// Metrics, Attributes
@property(nonatomic,readonly) CGFloat   pointSize;
@property(nonatomic,readonly) CGFloat   ascender;
@property(nonatomic,readonly) CGFloat   descender;
```

- Notice, no “set” methods to modify the graphics context
  - `NSString` drawing API handle details for you

Drawing APIs

UIKit vs CG

UIKit / CG Counterparts

- UIKit drawing APIs implemented on top of CG
- Can always use CG drawing API counterparts

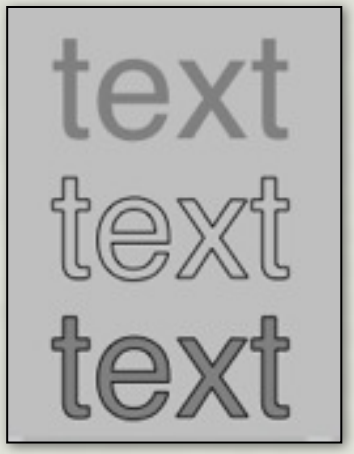
UIKit / CG Counterparts

- UIKit drawing APIs implemented on top of CG
- Can always use CG drawing API counterparts

UIKit	Core Graphics
`UIImage`	`CGImageRef, CGDataProviderRef`
`UIColor`	`CGColorRef, CGPatternRef`
`UIFont`	`CGFontRef, CGContext "text" APIs`
`CGContextRef`	`CGContextRef`

CG – More Options

- Example: CG Text Drawing APIs

CGTextDrawingMode	Result
kCGTextFill	
kCGTextStroke	
kCGTextFillStroke	

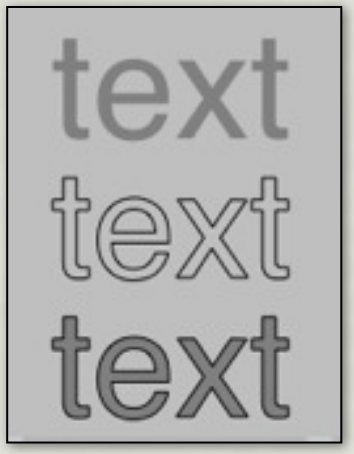
CG – More Options

- Example: CG Text Drawing APIs

CGTextDrawingMode	Result
kCGTextFill	text
kCGTextStroke	text
kCGTextFillStroke	text

CG – More Options

- Example: CG Text Drawing APIs

CGTextDrawingMode	Result
kCGTextFill	
kCGTextStroke	
kCGTextFillStroke	
kCGTextClip	

CG Text Drawing

```
// Text Drawing
void CGContextSetFont(CGContextRef c, CGFontRef font);
void CGContextSetTextDrawingMode(CGContextRef c, CGTextDrawingMode mode);
void CGContextShowGlyphsAtPoint(CGContextRef c,..., const CGGlyph glyphs[]);
```

- Advice – Stick to NSString drawing APIs!

Sampling Of APIs

- Graphics context properties

```
void CGContextSetFillColorWithColor(CGContextRef c,
void CGContextSetStrokeColorWithColor(CGContextRef c,
void CGContextSetShadowWithColor(CGContextRef c, CGSize offset, ...);
void CGContextSetLineWidth(CGContextRef c, CGFloat width)

void CGContextSetLineCap(CGContextRef c, CGLineCap cap)
void CGContextSetLineJoin(CGContextRef c, CGLineJoin join)
void CGContextSetLineDash(CGContextRef c, ...);
```

- Save / restore graphics context

```
// Graphics Context
void CGContextSaveGState(CGContextRef c);
void CGContextRestoreGState(CGContextRef c);
```

Sampling Of APIs

- Drawing

```
void CGContextDrawImage(CGContextRef c, CGRect rect, CGImageRef image);
void CGContextFillRect(CGContextRef c, CGRect rect);
void CGContextStrokeRect(CGContextRef c, CGRect rect);

void CGContextDrawLinearGradient(CGContextRef c, CGGradientRef gradient, ...);
void CGContextDrawShading(CGContextRef c, CGShadingRef shading);
```

UIKit / CG Counterparts

- No UIKit counterpart for some things
  - Path Drawing – `CGPathRef`
  - Shading – `CGShadingRef`, `CGGradientRef`
  - PDF Editing, Parsing, Metadata – many CF types...
- We'll take a quick look at Path Drawing

Path Drawing

- Path defines on or more shapes, or subpaths
- May consist of lines, curves, or both
- Can be as complex as you want

Path Drawing

- Path defines on or more shapes, or subpaths
- May consist of lines, curves, or both
- Can be as complex as you want
- Examples

Path Drawing

- Creation and painting paths are separate tasks
- Can use paths many ways
  - Stroke
  - Fill
  - Use as clipping shape
- Building blocks
  - Points, Lines, Rectangles
  - Arcs, Curves, Ellipses

Working With Paths

- `CGPathRef`
- Create

```
CGMutablePathRef CGPathCreateMutable(void)
```

- Add components to a subpath

```
void CGPathMoveToPoint(CGMutablePathRef path, ... );
void CGPathAddLineToPoint(CGMutablePathRef path, ... );
void CGPathAddRect(CGMutablePathRef path,... );
void CGPathAddArc(CGMutablePathRef path, ...);
void CGPathAddCurveToPoint(CGMutablePathRef path, ...);
```

- Close or open new subpath

```
void CGPathCloseSubpath(CGMutablePathRef path)
```

Working With Paths

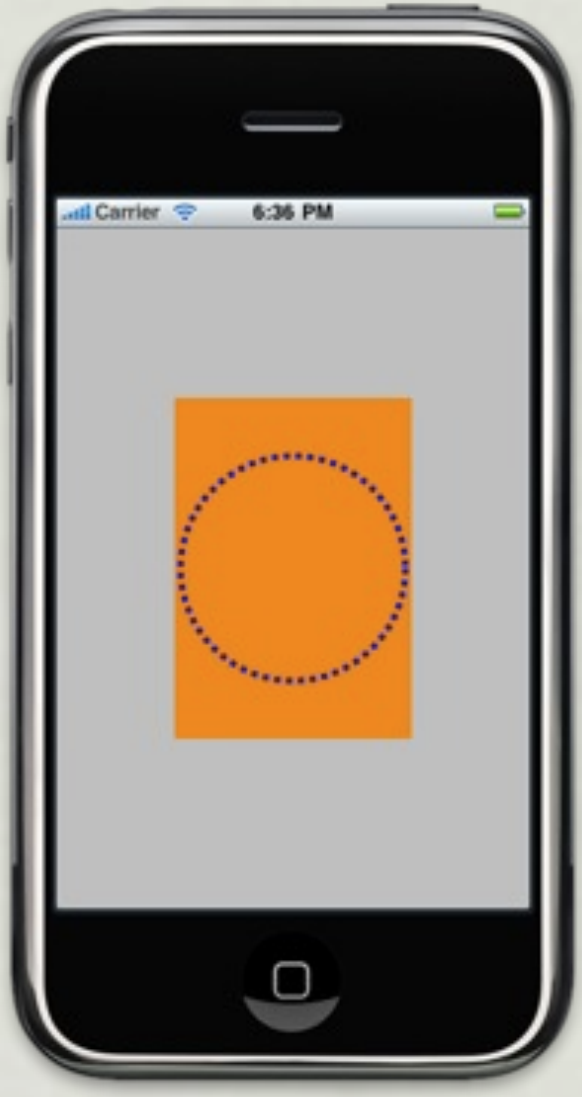
- CGContext API
  - API parallel to `CGPathRef`
  - Operates on path owned by graphics context
- Examples

```
void CGContextBeginPath(CGContextRef c)
void CGContextMoveToPoint(CGContextRef c, CGFloat x, CGFloat y)
void CGContextAddLineToPoint(CGContextRef c, CGFloat x, CGFloat y)
...
```

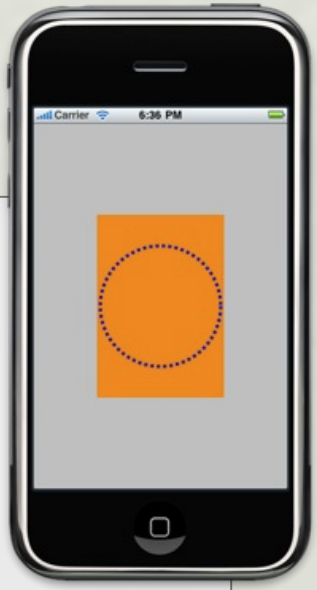
Drawing

Putting it all together…

Sample -drawRect:

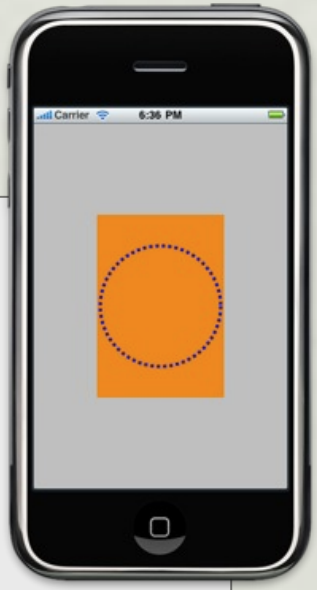


Sample -drawRect:



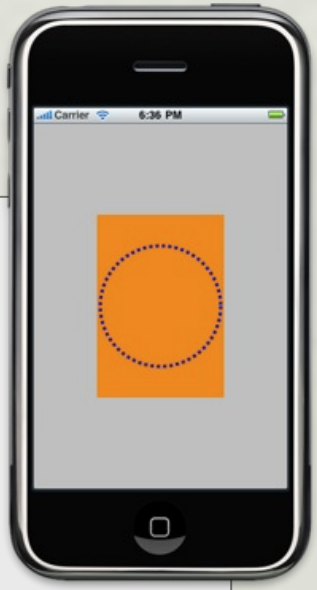

```
- (void)drawRect:(CGRect)rect {
    CGContextRef context = UIGraphicsGetCurrentContext();

    // Modify context to draw with orange
    [[UIColor orangeColor] set];

    // Draw a orange box
    CGFloat boxW = [self bounds].size.width / 2.0;
    CGFloat boxH = [self bounds].size.height / 2.0;
    CGRect boxRect = CGRectMake(boxW / 2.0, boxH / 2.0, boxW, boxH);
    UIRectFill(boxRect);

    // Modify context to draw blue, and use dashed lines
    [[UIColor blueColor] set];
    CGContextSetLineWidth(context, 4.0);
    CGContextSetLineDash(context, 0.0, dashLengths, 1);

    // Draw a blue, dashed circle
    CGFloat radius = MIN(boxW, boxH)/2.0f;
    CGPoint center = CGPointMake(boxW, boxH);

    CGContextAddArc(context, center.x, center.y, radius, 0, 2*M_PI, false);
    CGContextDrawPath(context, kCGPathStroke);
```

Sample -drawRect:



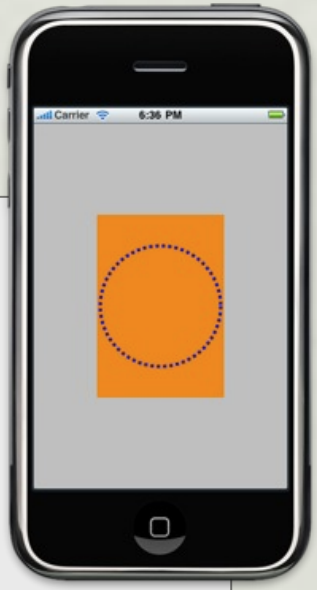

```
- (void)drawRect:(CGRect)rect {
    CGContextRef context = UIGraphicsGetCurrentContext();

    // Modify context to draw with orange
    [[UIColor orangeColor] set];

    // Draw a orange box
    CGFloat boxW = [self bounds].size.width / 2.0;
    CGFloat boxH = [self bounds].size.height / 2.0;
    CGRect boxRect = CGRectMake(boxW / 2.0, boxH / 2.0, boxW, boxH);
    UIRectFill(boxRect);

    // Modify context to draw blue, and use dashed lines
    [[UIColor blueColor] set];
    CGContextSetLineWidth(context, 4.0);
    CGContextSetLineDash(context, 0.0, dashLengths, 1);

    // Draw a blue, dashed circle
    CGFloat radius = MIN(boxW, boxH)/2.0f;
    CGPoint center = CGPointMake(boxW, boxH);

    CGContextAddArc(context, center.x, center.y, radius, 0, 2*M_PI, false);
    CGContextDrawPath(context, kCGPathStroke);
```

Sample -drawRect:



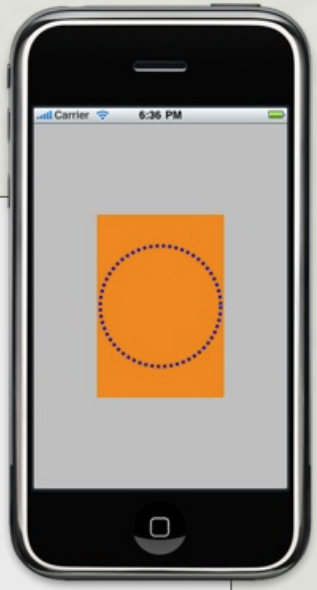

```
- (void)drawRect:(CGRect)rect {
    CGContextRef context = UIGraphicsGetCurrentContext();

    // Modify context to draw with orange
    [[UIColor orangeColor] set];

    // Draw a orange box
    CGFloat boxW = [self bounds].size.width / 2.0;
    CGFloat boxH = [self bounds].size.height / 2.0;
    CGRect boxRect = CGRectMake(boxW / 2.0, boxH / 2.0, boxW, boxH);
    UIRectFill(boxRect);

    // Modify context to draw blue, and use dashed lines
    [[UIColor blueColor] set];
    CGContextSetLineWidth(context, 4.0);
    CGContextSetLineDash(context, 0.0, dashLengths, 1);

    // Draw a blue, dashed circle
    CGFloat radius = MIN(boxW, boxH)/2.0f;
    CGPoint center = CGPointMake(boxW, boxH);

    CGContextAddArc(context, center.x, center.y, radius, 0, 2*M_PI, false);
    CGContextDrawPath(context, kCGPathStroke);
```

Sample -drawRect:



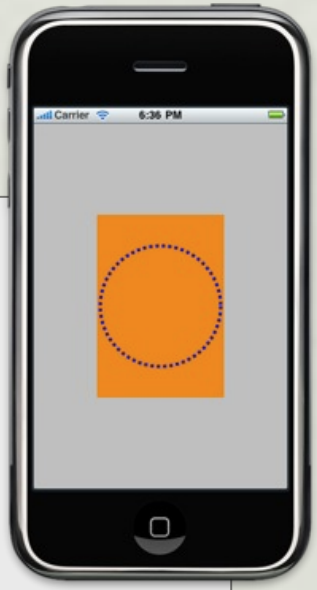

```
- (void)drawRect:(CGRect)rect {
    CGContextRef context = UIGraphicsGetCurrentContext();

    // Modify context to draw with orange
    [[UIColor orangeColor] set];

    // Draw a orange box
    CGFloat boxW = [self bounds].size.width / 2.0;
    CGFloat boxH = [self bounds].size.height / 2.0;
    CGRect boxRect = CGRectMake(boxW / 2.0, boxH / 2.0, boxW, boxH);
    UIRectFill(boxRect);

    // Modify context to draw blue, and use dashed lines
    [[UIColor blueColor] set];
    CGContextSetLineWidth(context, 4.0);
    CGContextSetLineDash(context, 0.0, dashLengths, 1);

    // Draw a blue, dashed circle
    CGFloat radius = MIN(boxW, boxH)/2.0f;
    CGPoint center = CGPointMake(boxW, boxH);

    CGContextAddArc(context, center.x, center.y, radius, 0, 2*M_PI, false);
    CGContextDrawPath(context, kCGPathStroke);
```

Sample -drawRect:




```
- (void)drawRect:(CGRect)rect {
    CGContextRef context = UIGraphicsGetCurrentContext();

    // Modify context to draw with orange
    [[UIColor orangeColor] set];

    // Draw a orange box
    CGFloat boxW = [self bounds].size.width / 2.0;
    CGFloat boxH = [self bounds].size.height / 2.0;
    CGRect boxRect = CGRectMake(boxW / 2.0, boxH / 2.0, boxW, boxH);
    UIRectFill(boxRect);

    // Modify context to draw blue, and use dashed lines
    [[UIColor blueColor] set];
    CGContextSetLineWidth(context, 4.0);
    CGContextSetLineDash(context, 0.0, dashLengths, 1);

    // Draw a blue, dashed circle
    CGFloat radius = MIN(boxW, boxH)/2.0f;
    CGPoint center = CGPointMake(boxW, boxH);

    CGContextAddArc(context, center.x, center.y, radius, 0, 2*M_PI, false);
    CGContextDrawPath(context, kCGPathStroke);
```

Sample -drawRect:




```
- (void)drawRect:(CGRect)rect {
    CGContextRef context = UIGraphicsGetCurrentContext();

    // Modify context to draw with orange
    [[UIColor orangeColor] set];

    // Draw a orange box
    CGFloat boxW = [self bounds].size.width / 2.0;
    CGFloat boxH = [self bounds].size.height / 2.0;
    CGRect boxRect = CGRectMake(boxW / 2.0, boxH / 2.0, boxW, boxH);
    UIRectFill(boxRect);

    // Modify context to draw blue, and use dashed lines
    [[UIColor blueColor] set];
    CGContextSetLineWidth(context, 4.0);
    CGContextSetLineDash(context, 0.0, dashLengths, 1);

    // Draw a blue, dashed circle
    CGFloat radius = MIN(boxW, boxH)/2.0f;
    CGPoint center = CGPointMake(boxW, boxH);

    CGContextAddArc(context, center.x, center.y, radius, 0, 2*M_PI, false);
    CGContextDrawPath(context, kCGPathStroke);
```

Reading

- iPhone Application Programming Guide
  - Layout, Redrawing – p. 71, 72
  - Graphics and Drawing – p.107 - 118
- Quartz 2D Programming Guide
  - Paths – p.47 - 58
- Demo Code
  - UICatalog – available on Apple's developer site
  - Quartz 2D Tutorial – http://www.mactech.com/articles/mactech/Vol.21/21.08/Mineralogy101/

Advanced Drawing

- Typically stay at UIView level
- Drawing to other destinations
  - Into a bitmap context
    - When creating an image for caching, saving
  - CGLayer
    - CGLayer acts as a compositing primitive
    - When high performance “painting” of multiple objects is needed
  - CALayer
    - When high performance drawing is needed within a UIView

Drawing Into Custom Context

- Process
  - Create a graphics context
    - CGContextCreate....
    - UIGraphicsBeginImageContext...
  - Draw
  - Release the graphics context

UIKit Image Context

- Use to capture or cache images
- Easy to use APIs
  - Fewer options than CGContextCreate, but easy work with…
  - Sets up coordinate system to match UIView coordinate system
  - Pushes onto UIKit's graphics context stack
- Create / Release APIs

```
void UIGraphicsBeginImageContext(CGSize size)
void UIGraphicsEndImageContext(void);
```

- When done drawing, extract a UIImage

```
UIImage *UIGraphicsGetImageFromCurrentImageContext(void);
```

CGLayerRef

- Good reasons to use CGLayerRef
  - Offscreen – not in a view hierarchy, no position, location
  - Repeated drawing or Buffering
  - Cross platform – available on Mac OS X and iPhone
  - High-quality drawing, offscreen (not in a hierarchy)
- Use
  - Create CGLayerRef
  - Draw into its CGContextRef
  - Keep the layer around for repeated drawing, or as a cache

CALayer

- Normally just use UIView
- Good reasons to use CALayer
  - Performance – smaller UIView hierarchy can be better
  - For more options
  - Cross platform – available on Mac OS X and iPhone
- CALayer contents providing options
  - delegate callback to -drawLayer:inContext:
  - Subclass CALayer and override -drawInContext:
  - Provide the contents as an image – e.g. layer.contents = image