Drawing with Quartz on iOS

Bob McCune
http://bobmccune.com

tapharmonic
http://tapharmonic.com

Agenda

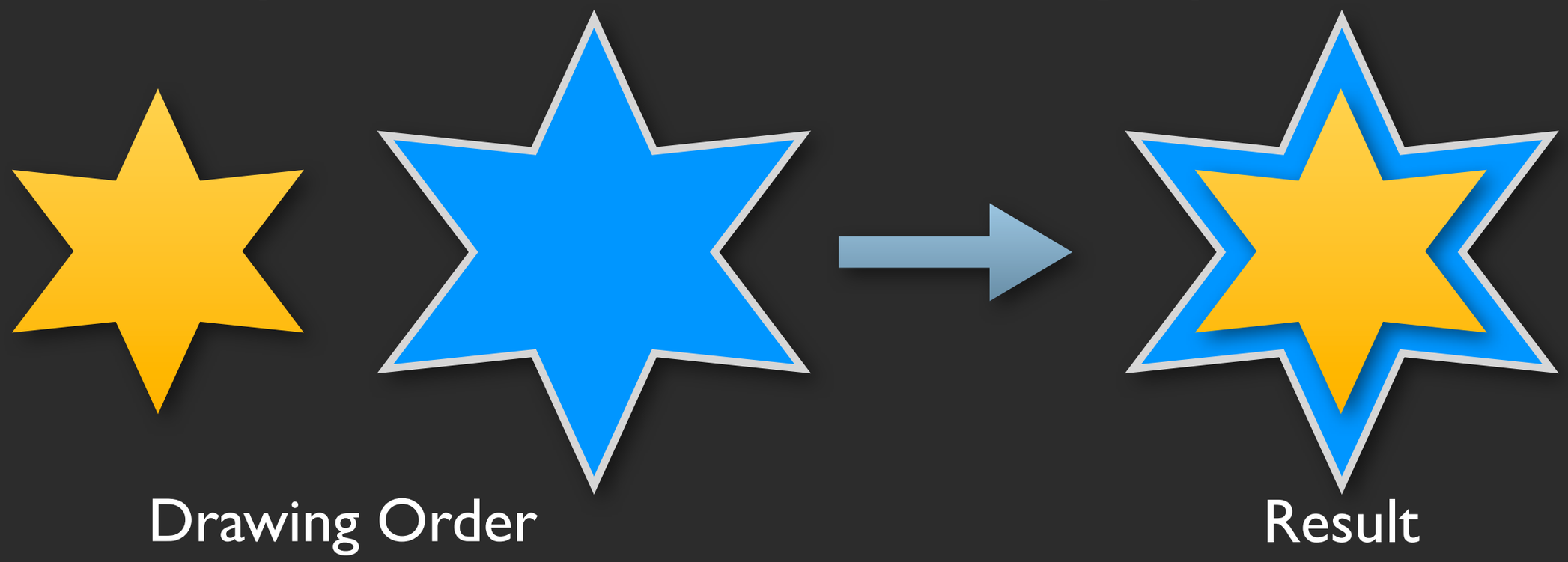
- Overview of the Quartz framework
  - What is it?
  - When do you need it?
- Key framework functionality:
  - Contexts and Paths
  - Drawing lines, curves, and images
  - Applying transformations
- Recipes and Examples

What is Quartz?

What is Quartz?

Wasn't this talk supposed to be about Core Graphics?

- 2D drawing engine for Mac and iOS based on the PDF imaging model
- Graphics API to produce lines, curves, transformations, gradients, etc.
- Uses painters model for drawing operations



When do I need it?

Isn't this what graphic designers are for?

- Always build your UI from images
  - Better performance and caching
  - Easier to create and maintain
  - Keeps your graphic designers employed
- Always, always, always use images...

...until you can't


iPad
11:17 AM
48 %
2,820
240 + 220 + 220 + 140 + 500 + 250 + 500
+/−
y^x
x√y
EE
7
8
9
÷
4
5
6
×
1
2
3
−
0
.
%
+
=
No History.
Perform a calculation and press the equal key to print to tape.


iPad
1:32 PM
36 %
Songs
Capo
A
B
00:20
Van Halen
Eruption
Van Halen
00:18.587
min
sec
SPEED
PITCH
¼
½
¾
1x


STEPS
PADS
MIXER
FX
SONG
DM1
The Drum Machine
SONG MODE
?
ON
OVERDRIVE
DELAY
PHASER
TEXTURIZER
ROBOTIZER
FILTER
COMPRESSOR
DRY/WET:
ON
OVERDRIVE
DELAY
PHASER
TEXTURIZER
ROBOTIZER
FILTER
COMPRESSOR
DRY/WET:


iPad
1:21 PM
80%
Overview
General Electric Company Common
YAHOO! FINANCE
General Electric Company Common
GE
Real Time Price
20.83
Change
0.25
% Change
1.24%
1D 5D 1M 3M 6M 1Y 2Y 5Y Max
Compare to
S&P 500
Nasdaq
Dow
Company or symbol
Add
Reset
50
40
30
20
10
0
1962
1970
1980
1990
2000
2010
200.00M
150.00M
100.00M
50.00M
0.00M
VOLUME
Quotes delayed, except where indicated.

Getting Started

Understanding the Syntax

Dude, where's my objects?

Objective-C

```
CGContext *context = [[CGContext alloc] init];
[context moveToPoint:CGPointMake(12.0f, 12.0f)];
[context addLineToPoint:CGPointMake(24.0f, 24.0f)];
[context fill];
```

Understanding the Syntax

Dude, where's my objects?

Objective-C

```
CGContext *context = [[CGContext alloc] init];
[context moveToPoint:CGPointMake(12.0f, 12.0f)];
[context addLineToPoint:CGPointMake(24.0f, 24.0f)];
[context fill];
```

Quartz is not an Objective-C API

Standard C

```
CGContextRef context = UIGraphicsGetCurrentContext();
CGContextMoveToPoint(context, 12.0f, 12.0f);
CGContextAddLineToPoint(context, 24.0f, 24.0f);
CGContextFillPath(context);
```

Quartz is Consistent

Naming Conventions

Functions always use the following convention:

```
<Type><Action>(<Object>, <Arguments);
```

```
CGContextMoveToPoint(context, 12.0f, 12.0f);
CGPathAddLineToPoint(path, NULL, 22.0f, 100.0f);
CGPDFContextClose(context);
```

Graphics Contexts

Quartz Graphics Contexts

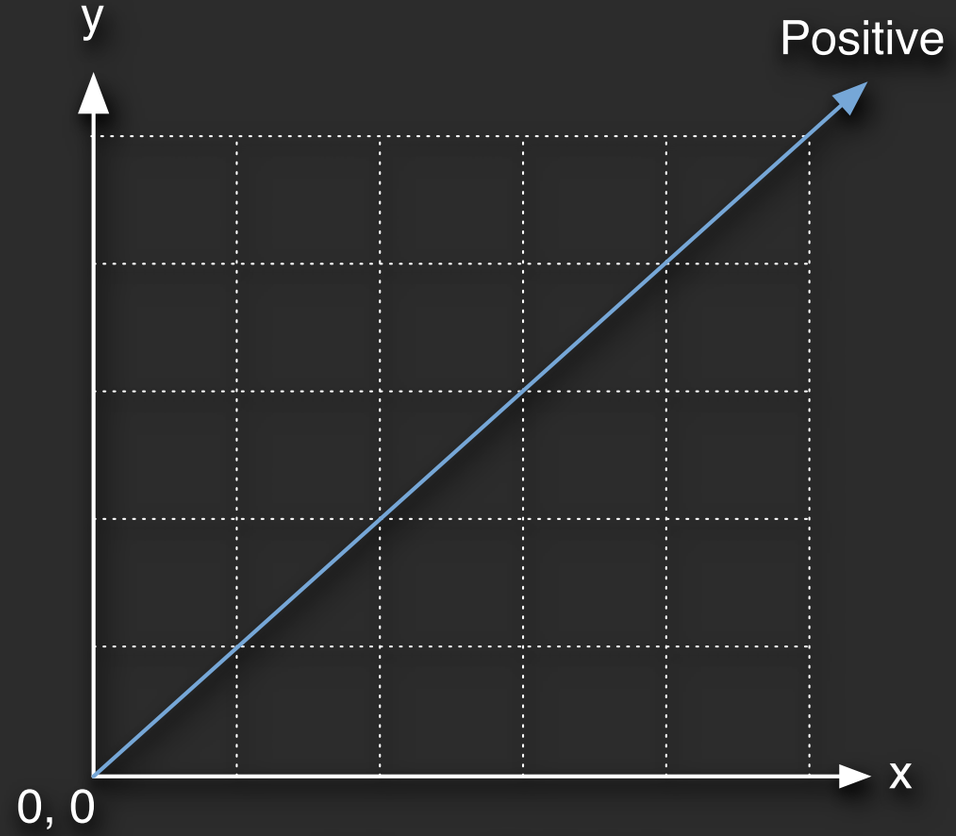
The Canvas

- Graphics context provides the drawing area
- Manages core drawing states:
  - Colors, line widths, transforms, etc.
  - It's a state machine
- Key Quartz Contexts:
  - CGContext
  - CGBitmapContext
  - CGPDFContext

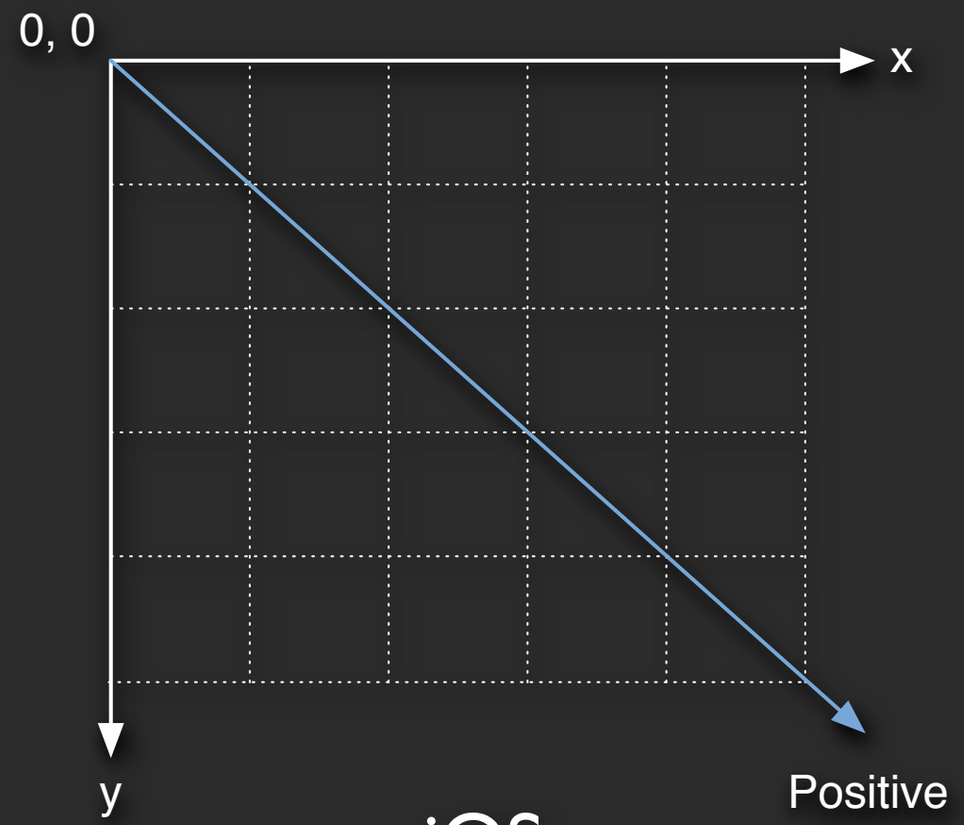
Quartz Coordinate System

Drawing Orientation

- Current Transformation Matrix (CTM) defines the coordinate system
- Coordinate system varies on Mac and iOS



Mac OS X

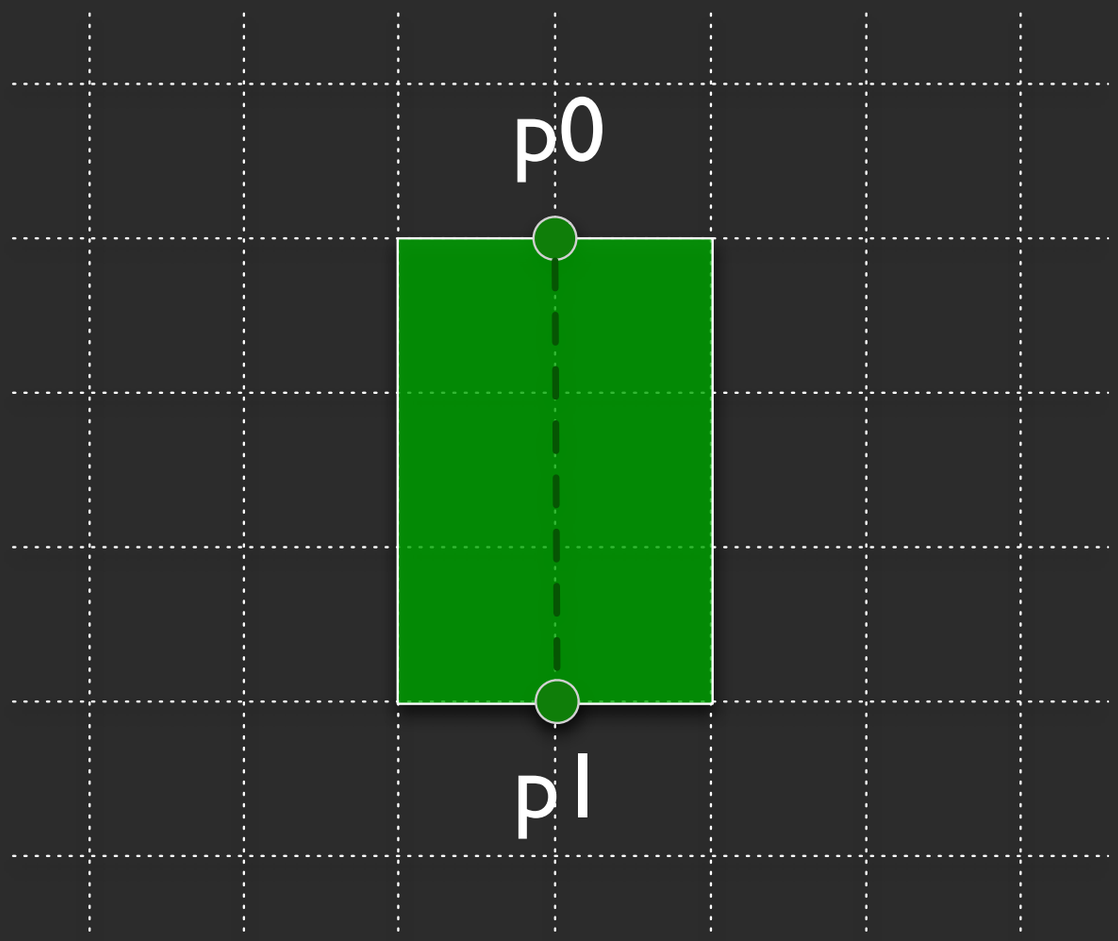


iOS

CGContext Drawing

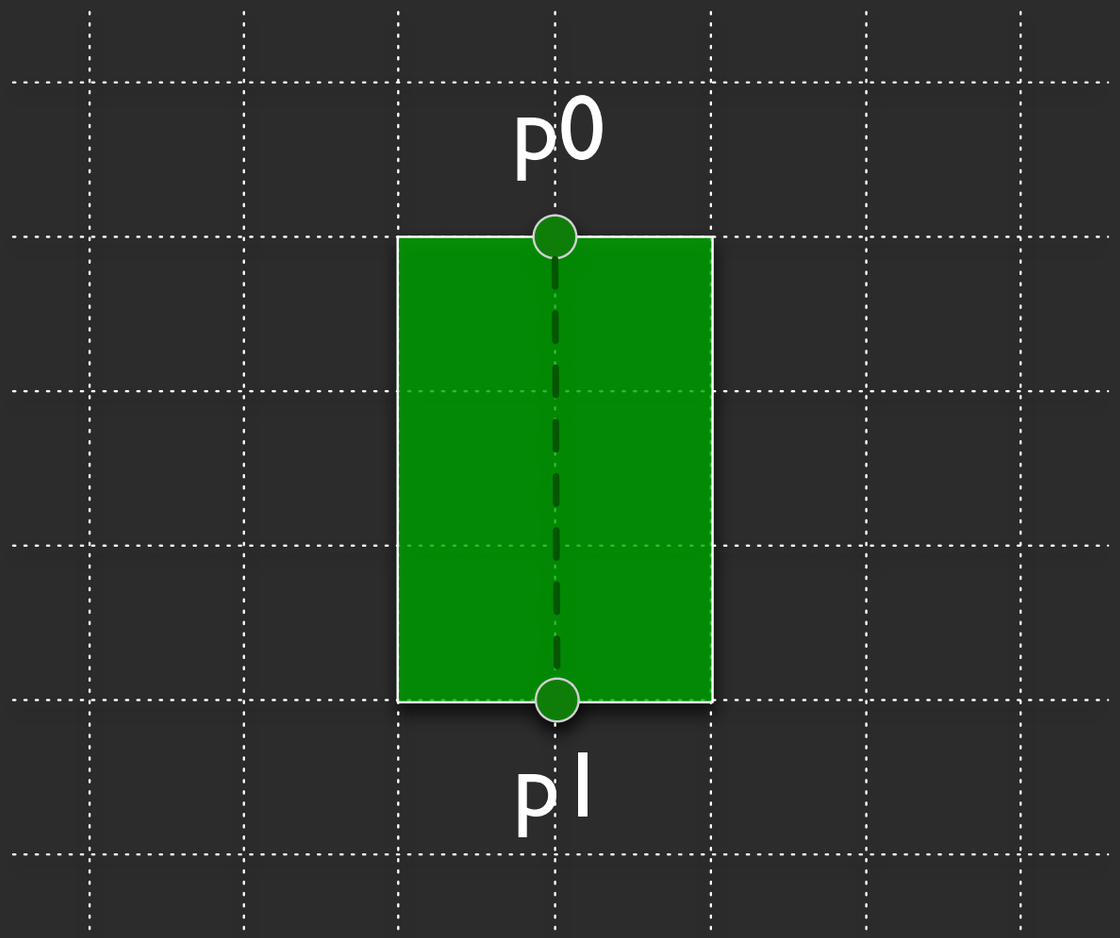
Quartz Points

- Drawing is defined in terms of points
- Points are abstract, infinitely thin
- Points live in *User Space*



Points are Device-Independent

Think in points not pixels



CGContext Drawing

CGContextRef

- CGContextRef is the drawing context
  - Created by UIKit and AppKit
- Get a pointer reference to it:

iOS

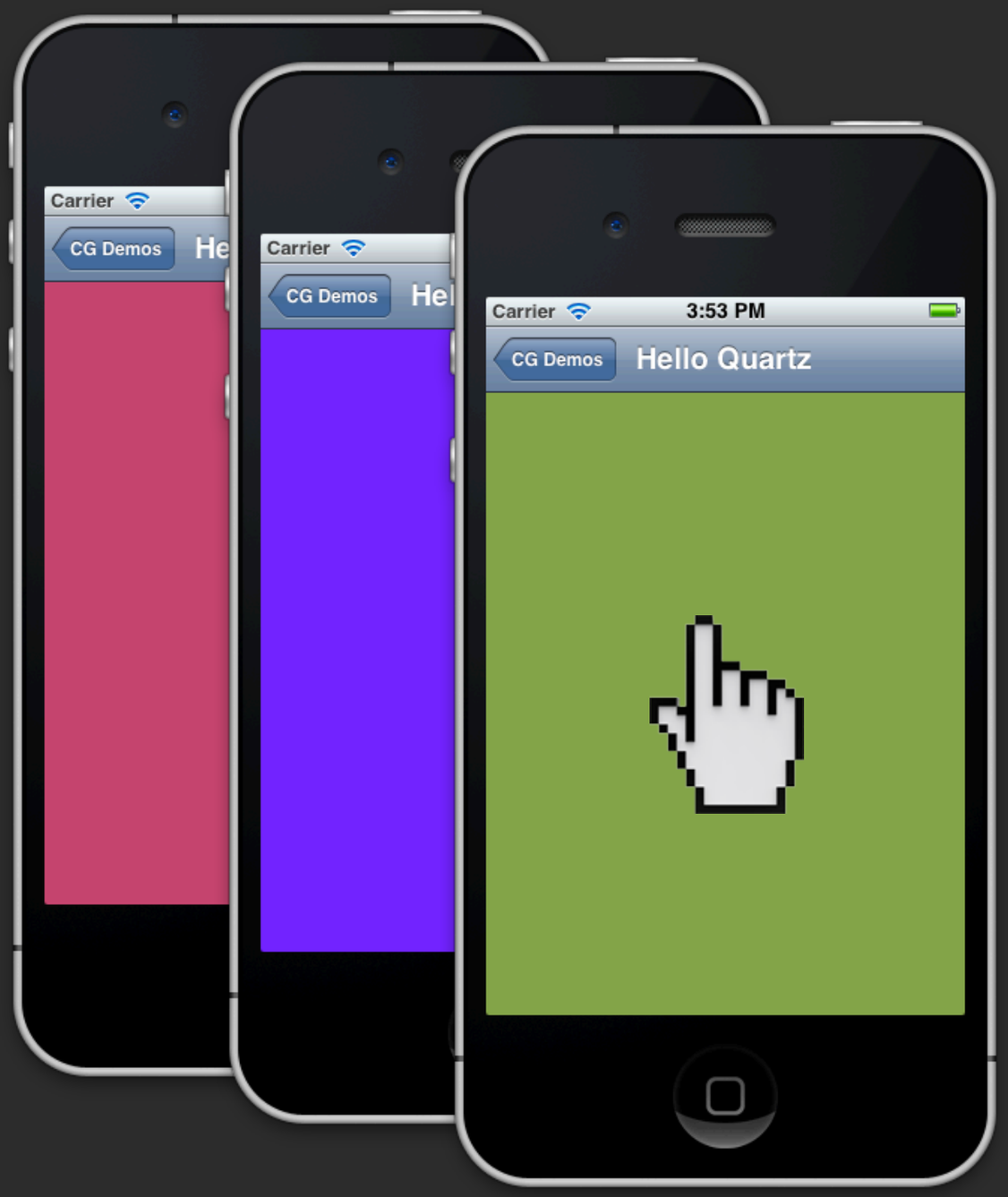
```
CGContextRef context = UIGraphicsGetCurrentContext();
```

Mac

```
NSGraphicsContext *current =
   [NSGraphicsContext currentContext];
CGContextRef context = (CGContextRef)[current graphicsPort];
```

Getting Started

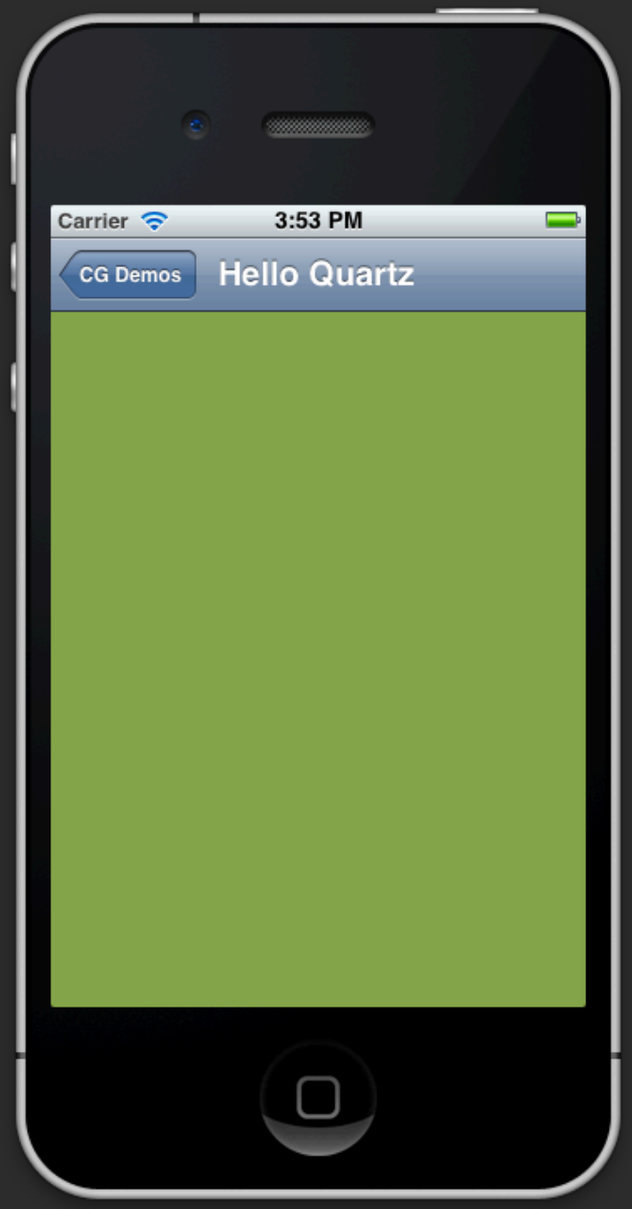
Where do I draw?



Getting Started

Where do I draw?

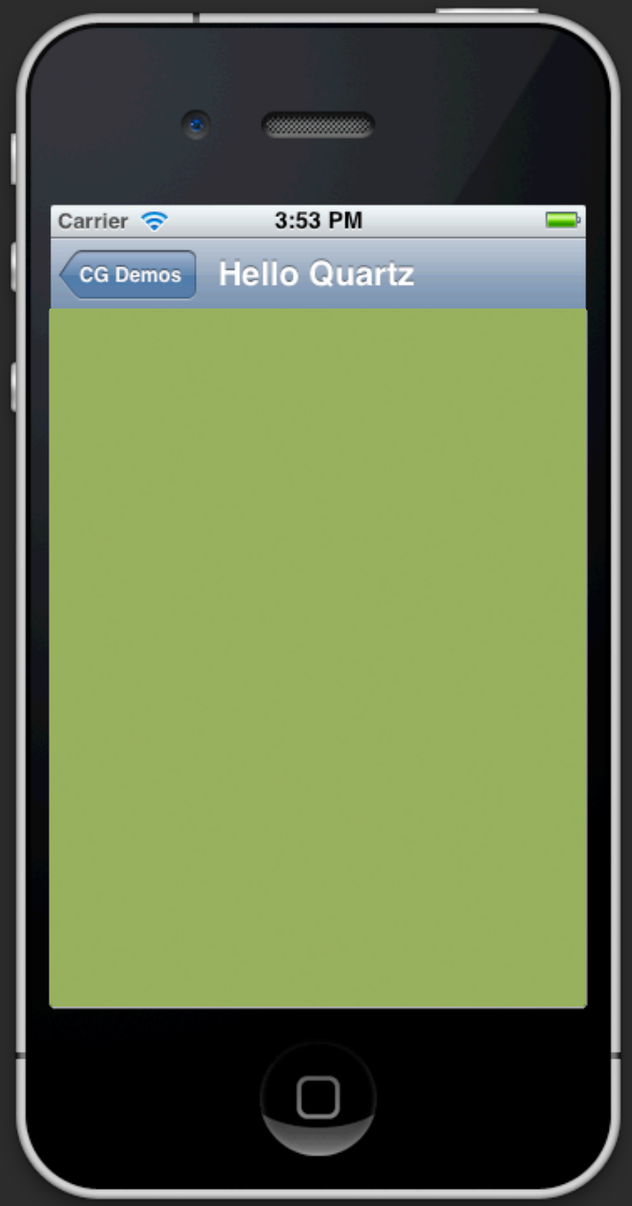
```
@implementation HelloQuartzViewController

- (void)drawBackground:(UIGestureRecognizer *)recognizer {
    CGContextRef context = UIGraphicsGetCurrentContext();
    CGContextSetFillColorWithColor(context, THRandomColor());
    CGContextFillRect(context, self.view.bounds);
}

@end
```



Getting Started

Where do I draw?

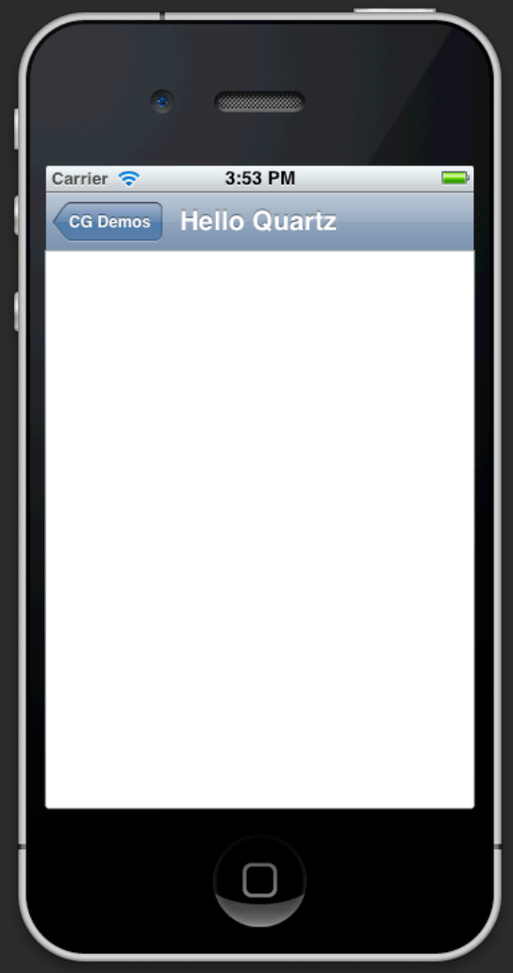
```
@implementation HelloQuartzViewController

- (void)drawBackground:(UIGestureRecognizer *)recognizer {
    CGContextRef context = UIGraphicsGetCurrentContext();
    CGContextSetFillColorWithColor(context, THRandomColor());
    CGContextFillRect(context, self.view.bounds);
}

@end


@implementation HelloQuartzView

- (void)drawRect:(CGRect)rect {


}

@end
```



Getting Started

Where do I draw?

```
@implementation HelloQuartzViewController

- (void)drawBackground:(UIGestureRecognizer *)recognizer {

    // Trigger call to HelloQuartzView's drawRect method (if needed)
    [self.helloView setNeedsDisplay];
}

@end
```

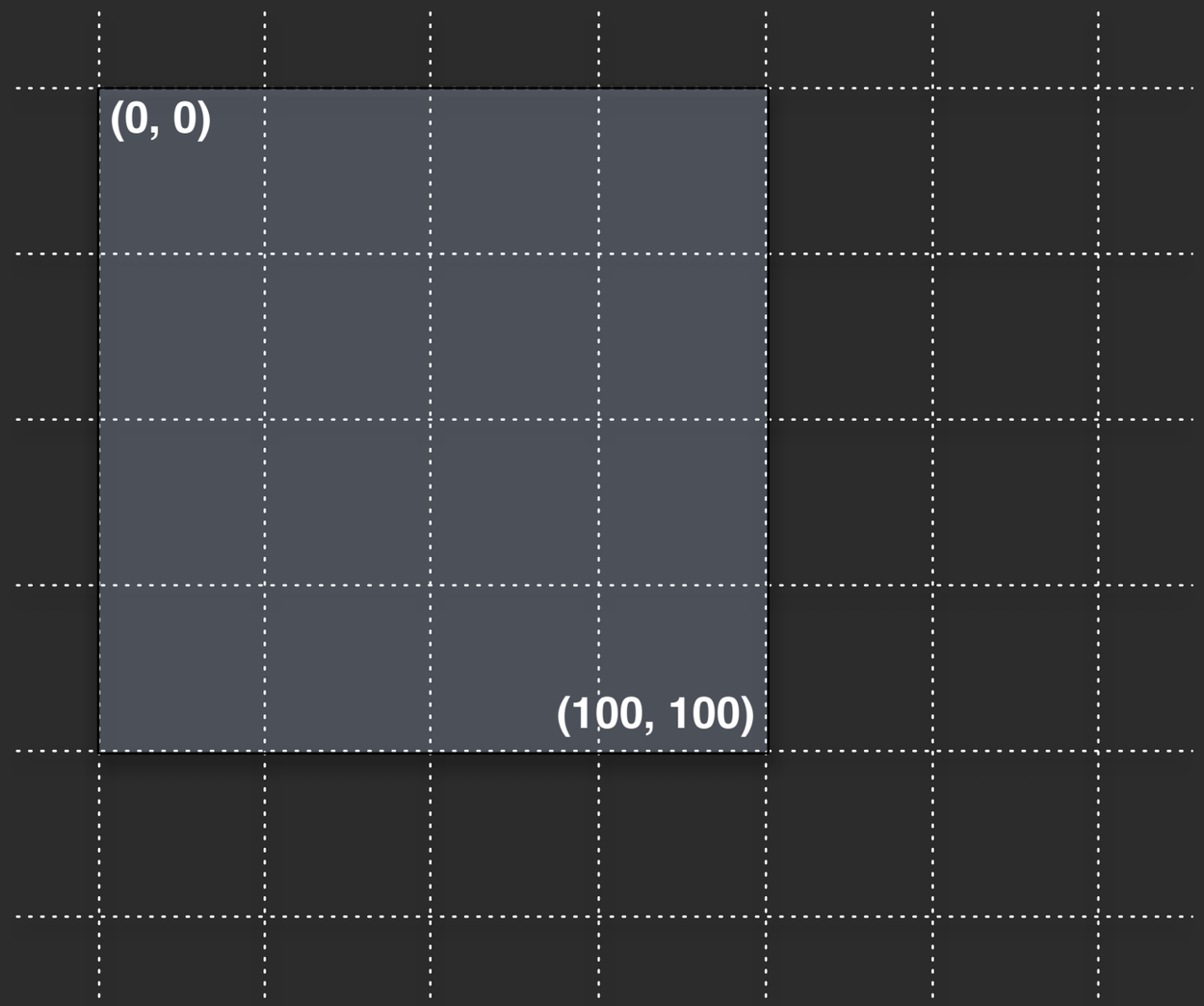



```
@implementation HelloQuartzView

- (void)drawRect:(CGRect)rect {
    CGContextRef context = UIGraphicsGetCurrentContext();
    CGContextSetFillColorWithColor(context, THRandomColor());
    CGContextFillRect(context, rect);
}

@end
```

Basic Paths

CGContext Drawing

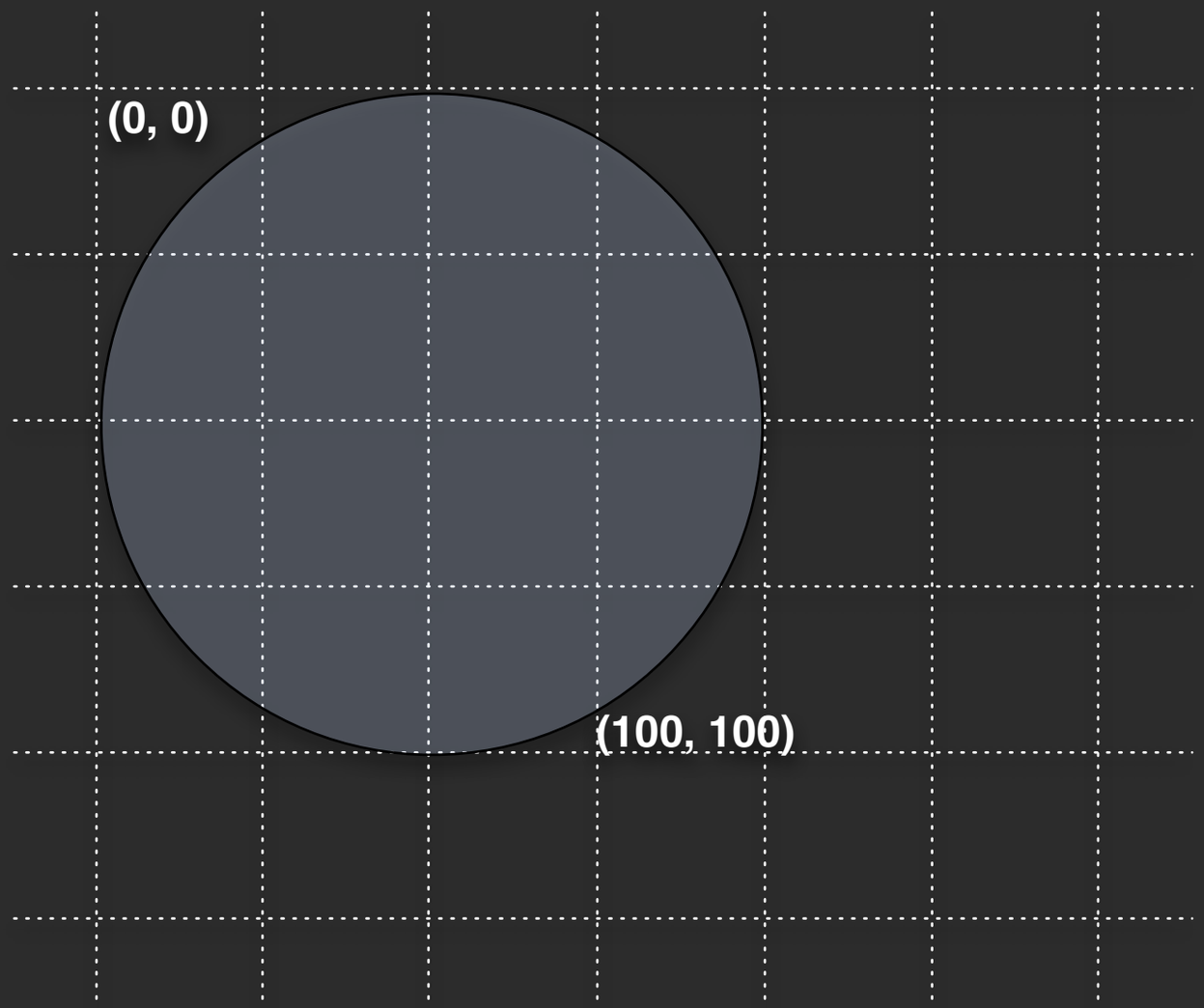
Defining Paths




```
CGRect rect = CGRectMake(0, 0, 100, 100);
CGContextAddRect(context, rect);
```

CGContext Drawing

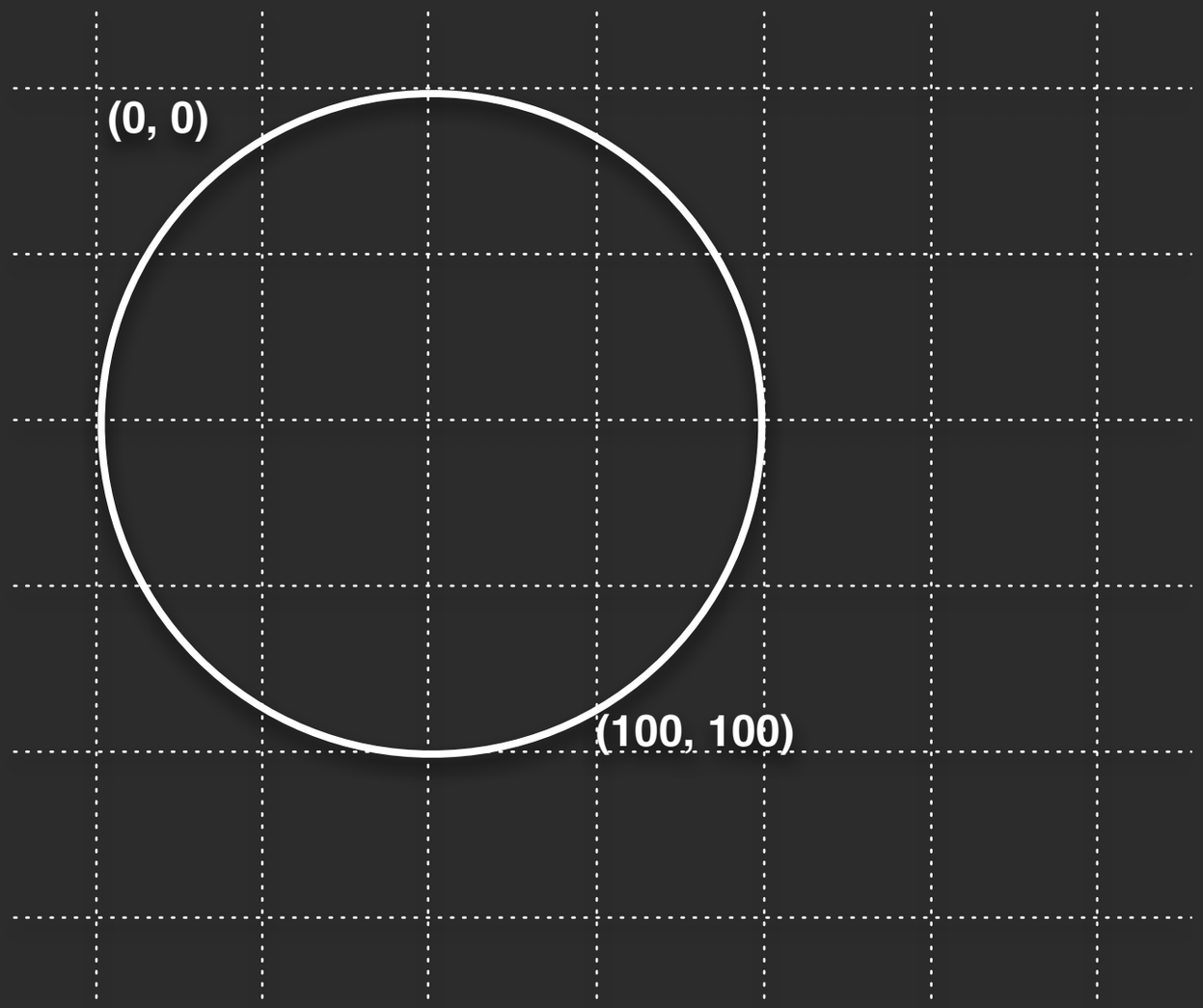
Defining Paths




```
CGRect rect = CGRectMake(0, 0, 100, 100);
CGContextAddEllipseInRect(context, rect);
```

CGContext Drawing

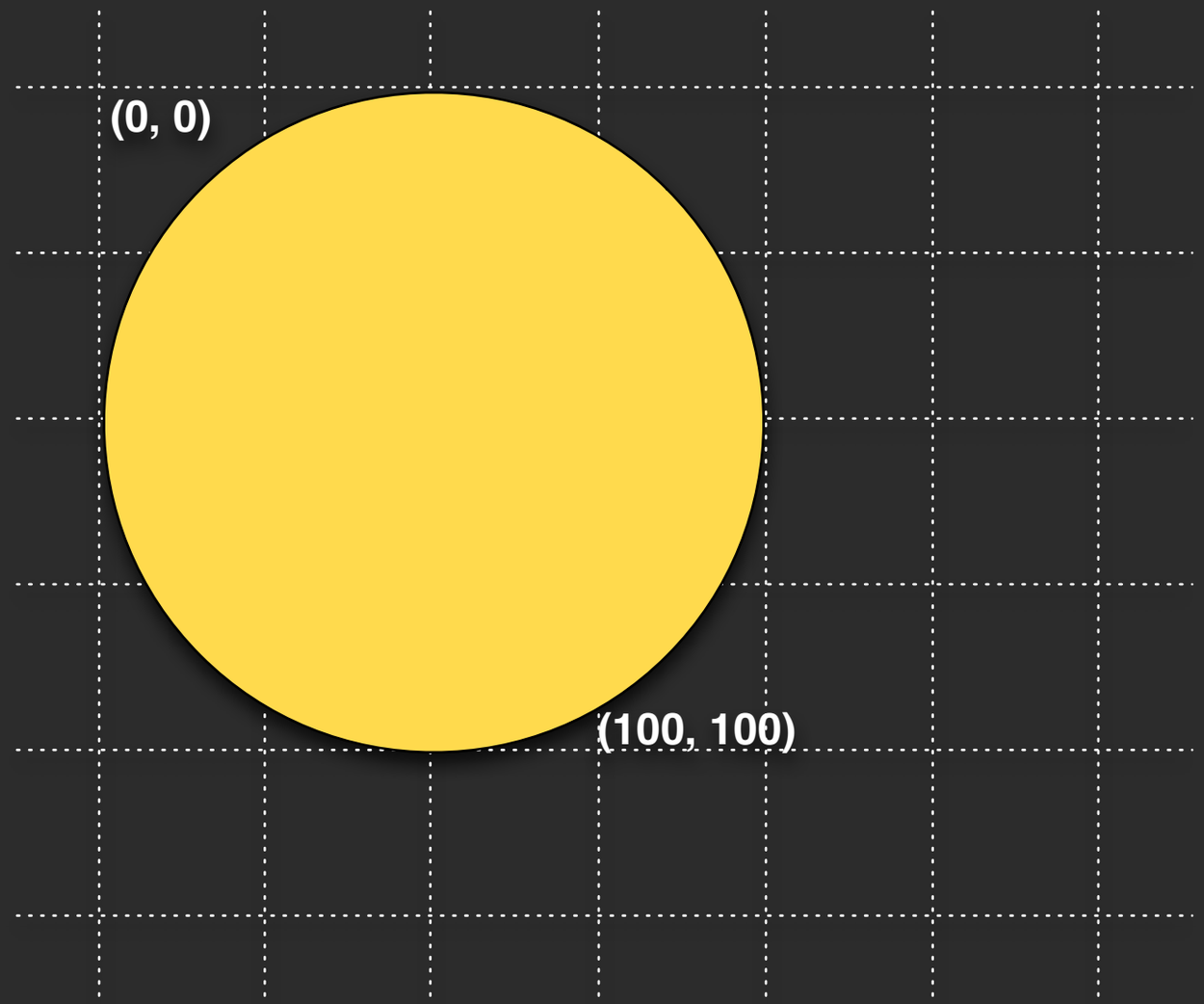
Defining Paths




```
CGContextSetStrokeColorWithColor(context, white);
CGContextDrawPath(context, kCGPathStroke);
```

CGContext Drawing

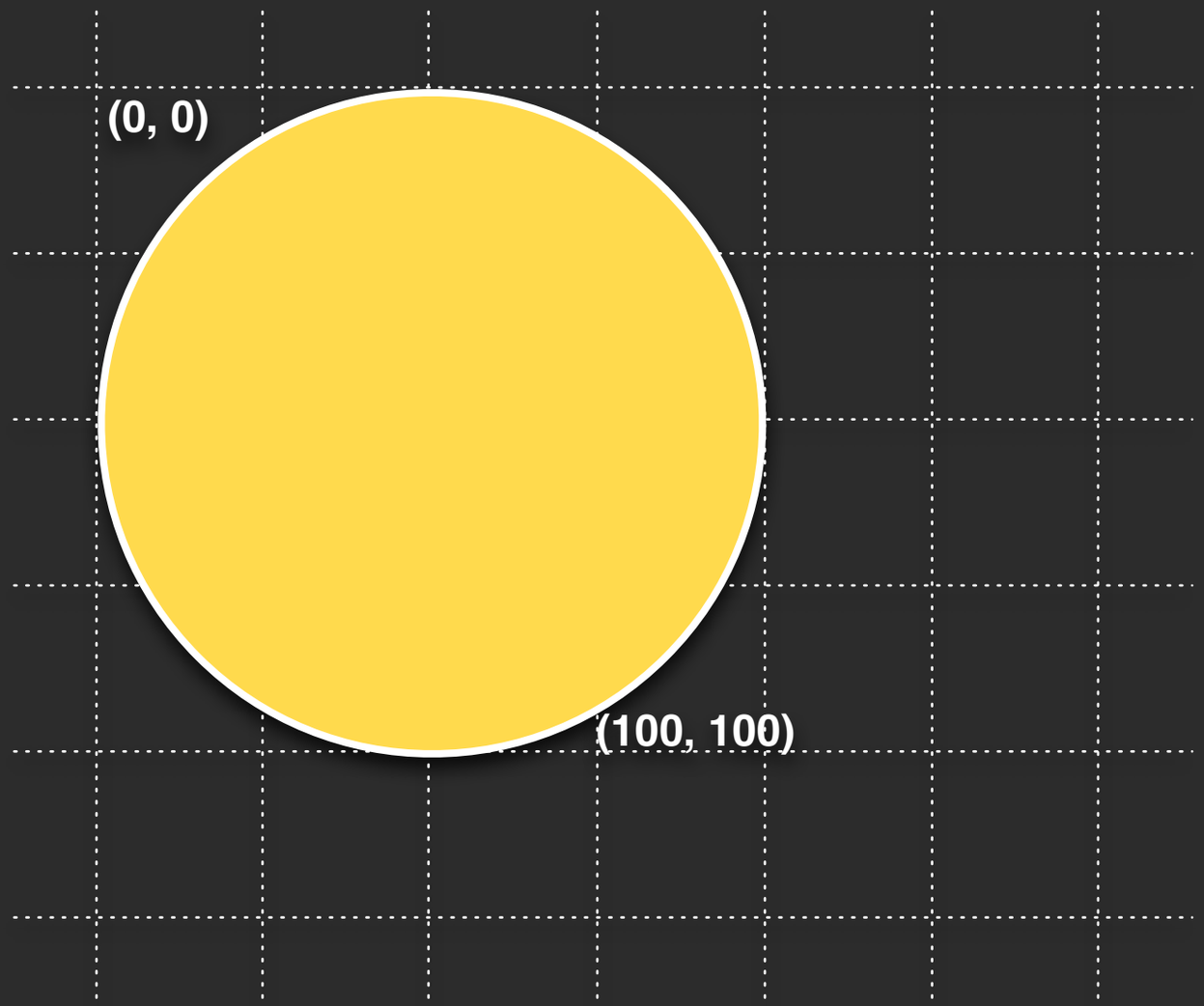
Defining Paths




```
CGContextSetFillColorWithColor(context, yellow);
CGContextDrawPath(context, kCGPathFill);
```

CGContext Drawing

Defining Paths




```
CGContextDrawPath(context, kCGPathFillStroke);
```

Defining Colors

CGColorRef

- Colors must be defined before drawing
- Fill Colors
  - `CGContextSetFillColor`
  - `CGContextSetFillColorWithColor`
  - `CGContextSetRGBFillColor`
- Stroke Colors
  - `CGContextSetStrokeColor`
  - `CGContextSetStrokeColorWithColor`
  - `CGContextSetRGBStrokeColor`

Defining Colors

CGColor vs UIColor

- `CGColor` is the native Quartz color type
- `UIColor` is the native UIKit color type
- Use `UIColor` in almost all cases
  - Easier to create
    - Autoreleased convenience initializers
    - Named color initializers
  - Easy to convert to and from `CGColor`

```
CGColorRef redColor = [UIColor redColor].CGColor;
CGContextSetFillColorWithColor(context, redColor);
```

Drawing Lines

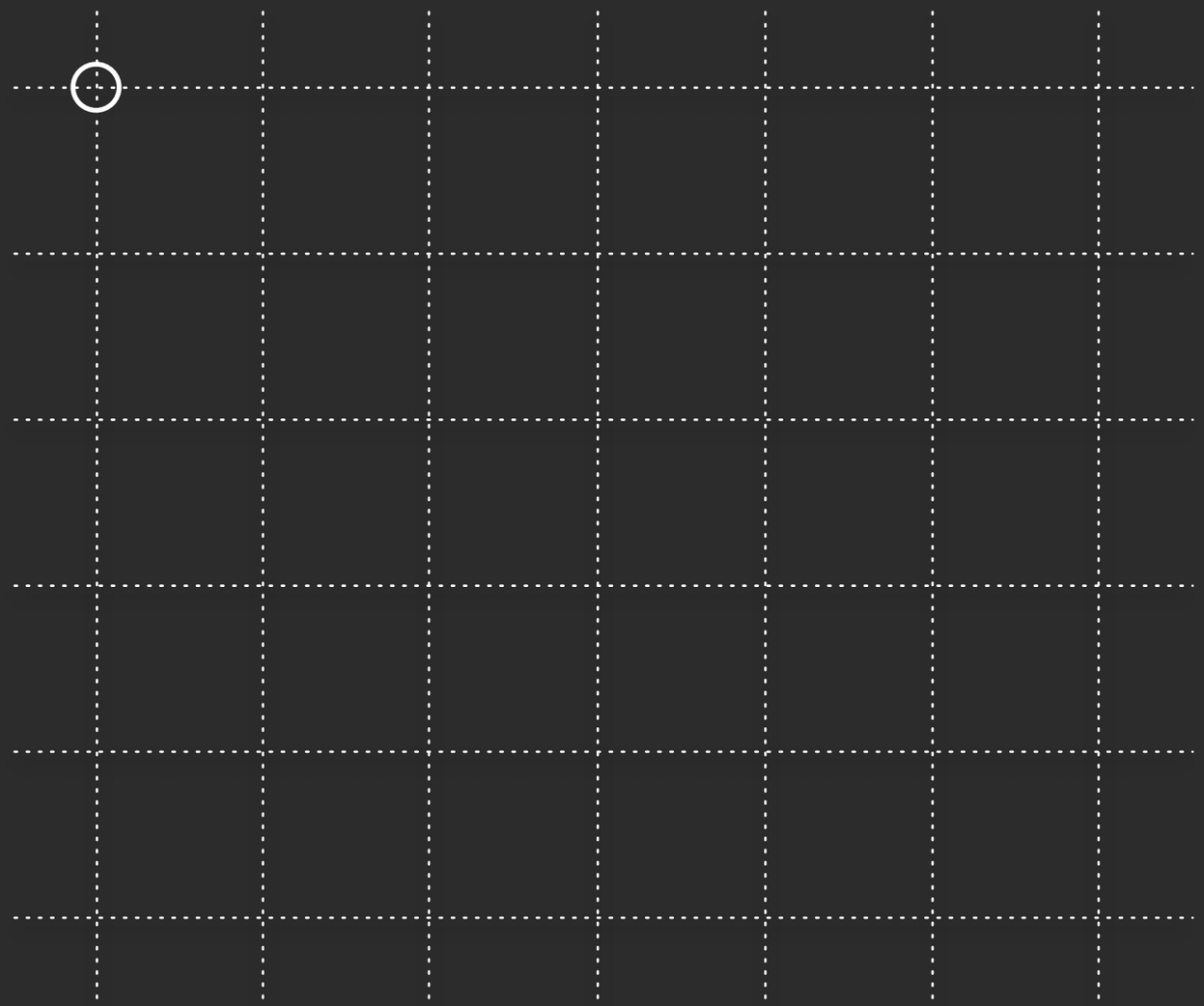
Defining Paths

- Lines are the most essential primitive
- Lines are always defined by their endpoints
- Basic Recipe:

1. Begin a new path
2. Move to initial starting point
3. Add lines defining shape
4. Optionally close the path
5. Draw Path

CGContext Drawing

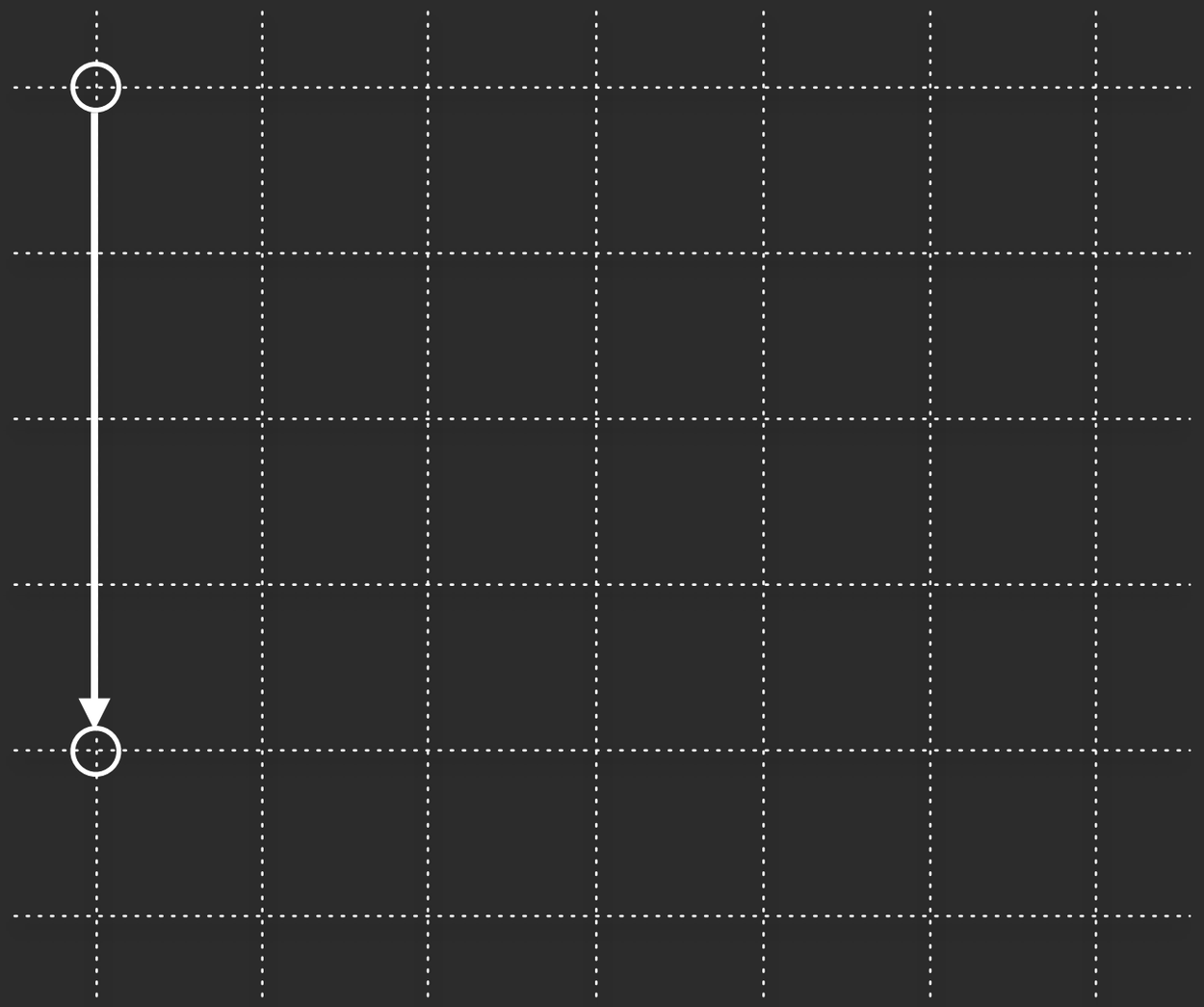
Defining Paths



```
CGContextBeginPath(context);
CGContextMoveToPoint(context, 0.0f, 0.0f);
```

CGContext Drawing

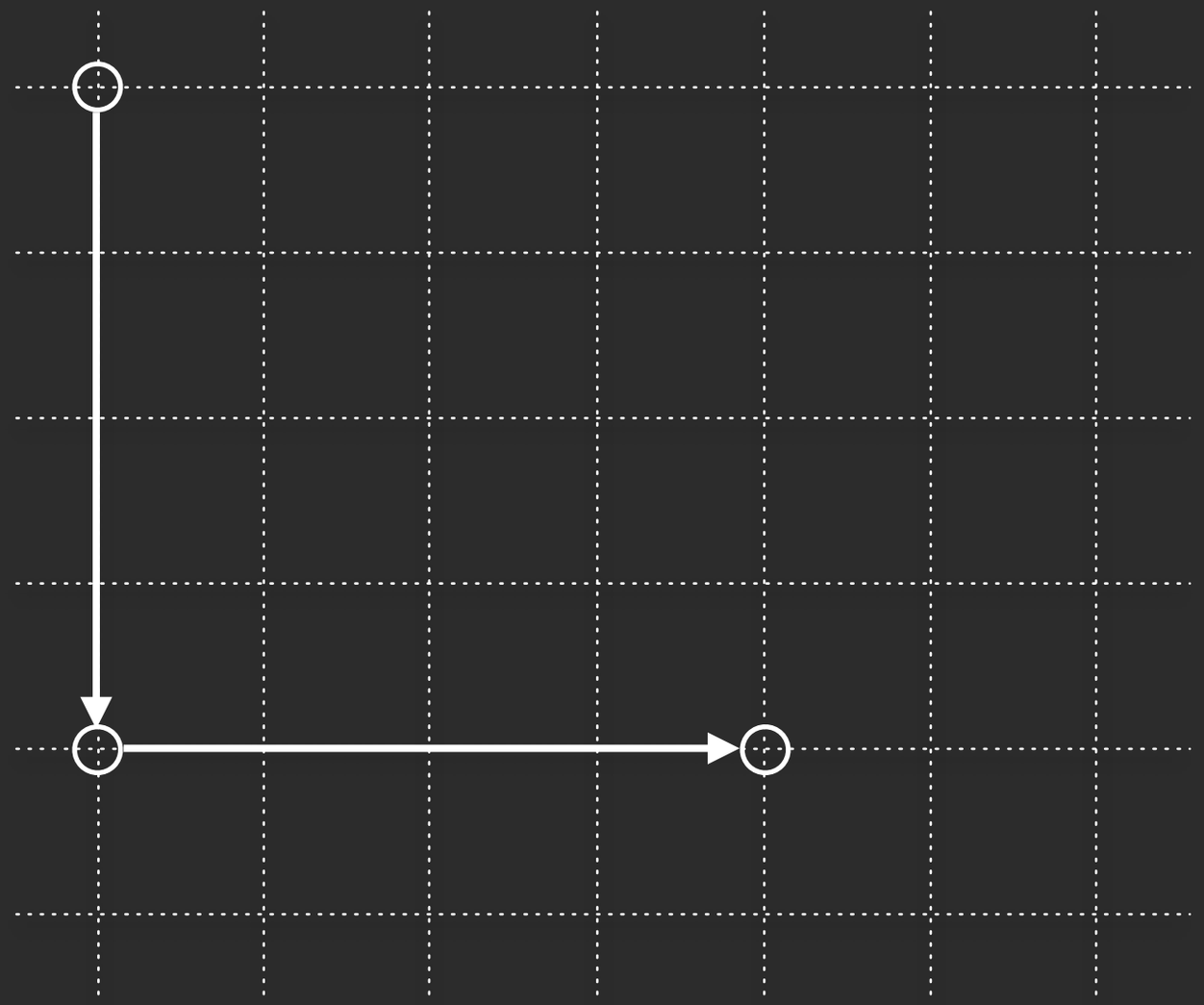
Defining Paths



```
CGContextAddLineToPoint(context, 0.0f, 4.0f);
```

CGContext Drawing

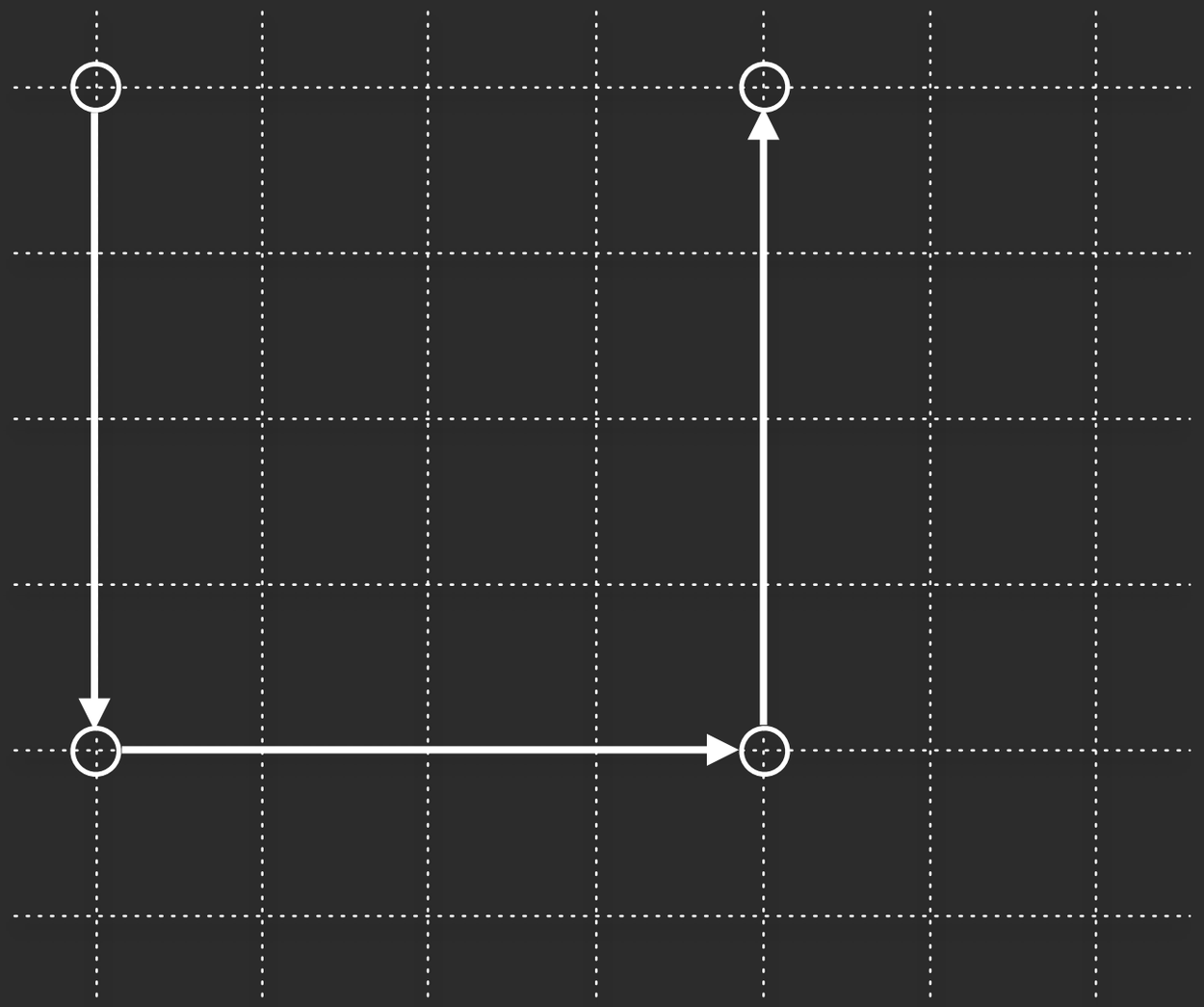
Defining Paths



```
CGContextAddLineToPoint(context, 4.0f, 4.0f);
```

CGContext Drawing

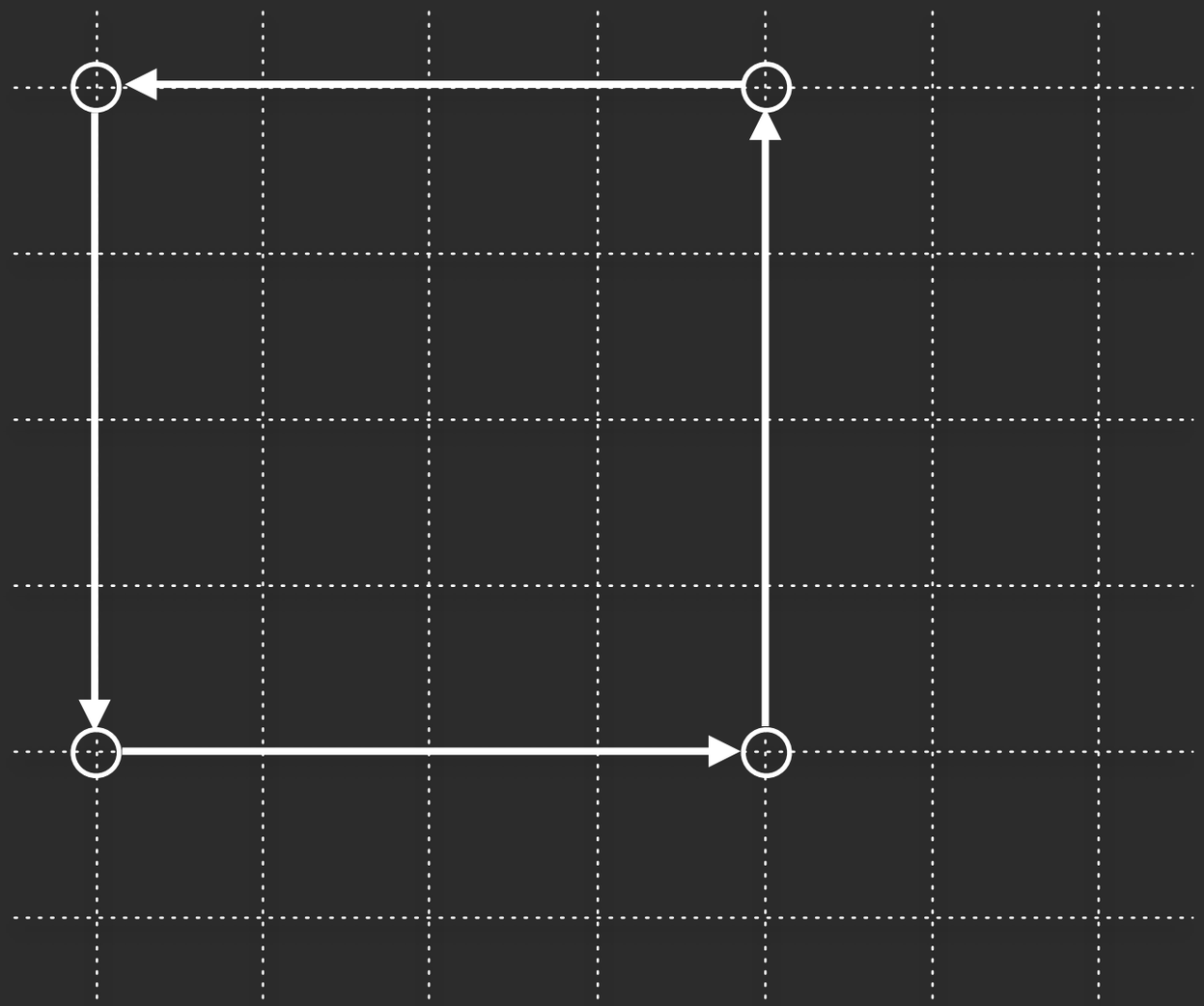
Defining Paths



```
CGContextAddLineToPoint(context, 4.0f, 0.0f);
```

CGContext Drawing

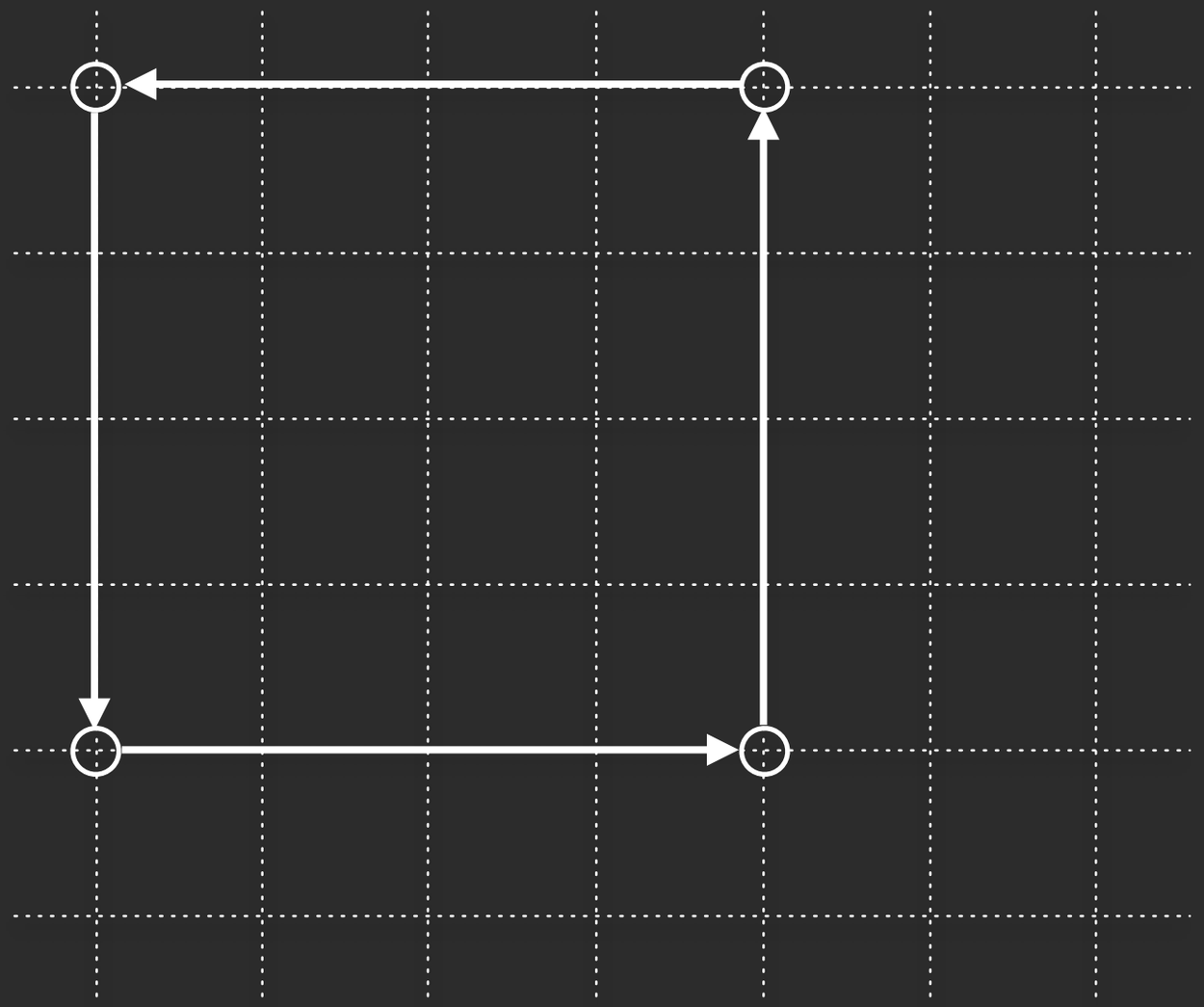
Defining Paths



```
CGContextClosePath(context);
```

CGContext Drawing

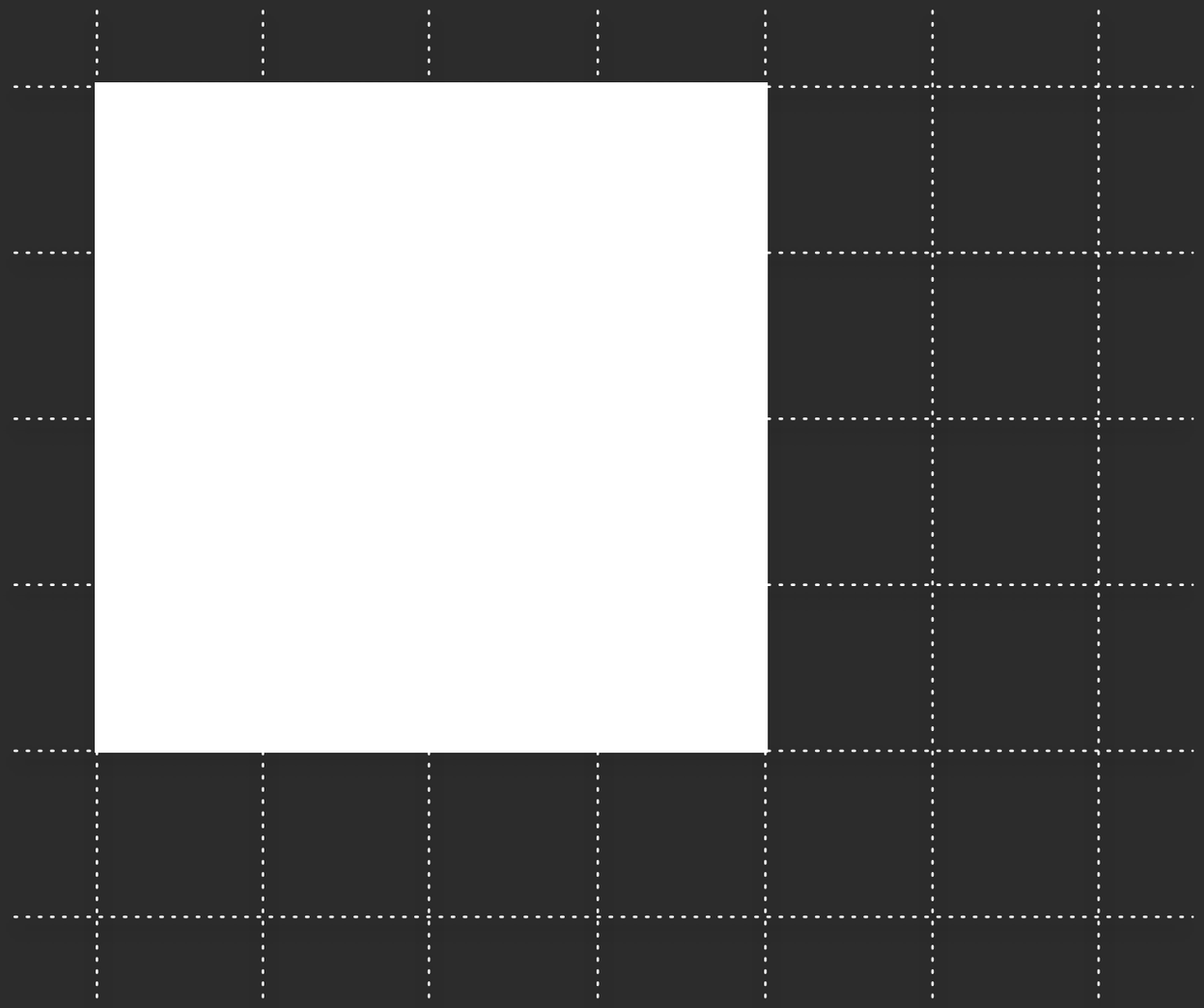
Defining Paths



```
CGColorRef color = [UIColor whiteColor].CGColor;
CGContextSetFillColorWithColor(context, color);
```

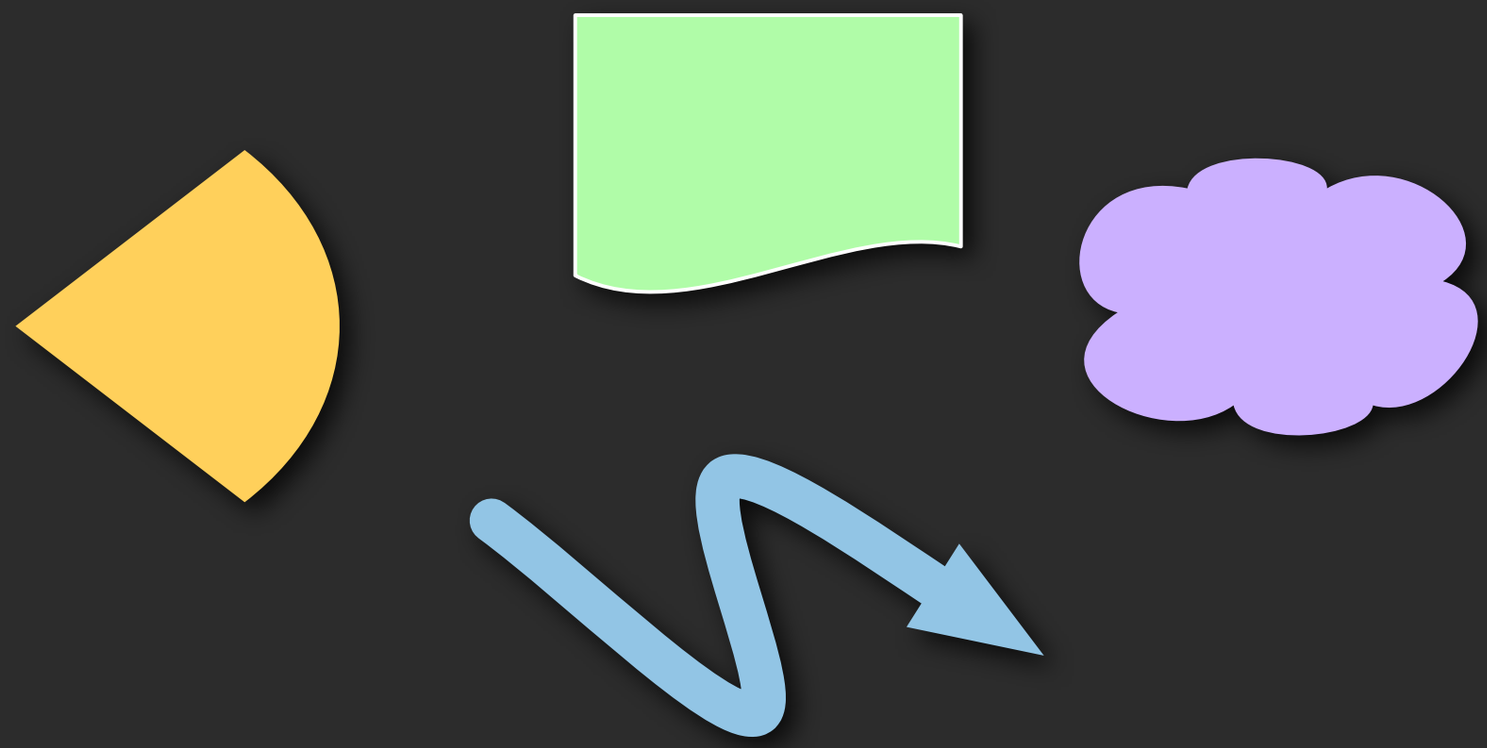
CGContext Drawing

Defining Paths



```
CGContextDrawPath(context, kCGPathFill);
```

Advanced Paths



Drawing Arcs

Defining Paths

- Arcs define circle segments
- Arcs are defined with the following:
  - Center point x, y coordinates
  - Radius
  - Start and stop angles (in radians)
  - Direction (clockwise or counter-clockwise)
- Created using:
  - `CGContextAddArc`
  - `CGContextAddArcToPoint`

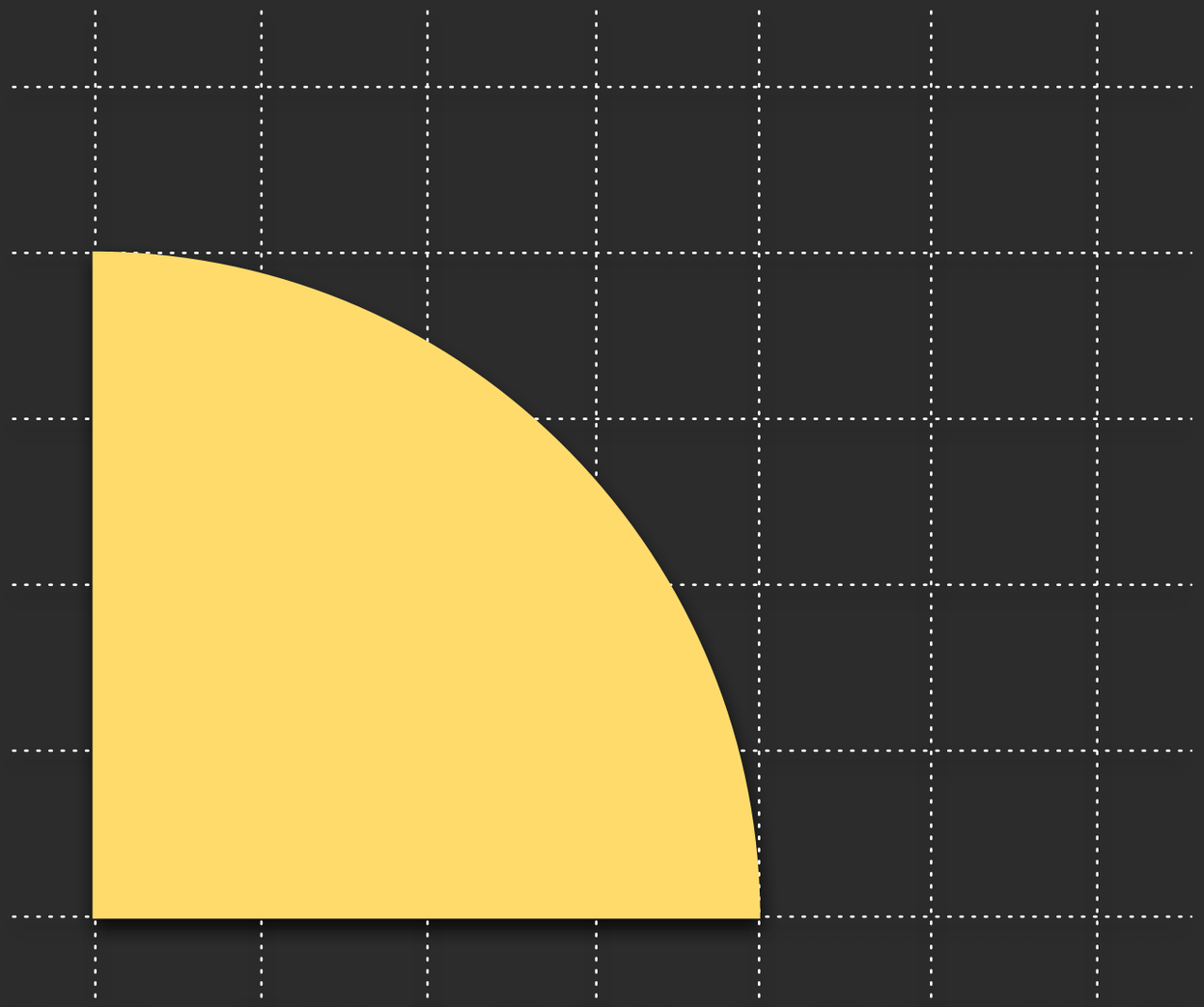
UIBezierPath

More iOS-friendly solution

- Arc functions are not aware of flipped coordinate system
  - Need to think upside down
  - Transform coordinate system before using
- `UIBezierPath` provides a lightweight wrapper over Quartz path functionality
  - Access to underlying path with `CGPath` property
- Easier to use `UIBezierPath` on iOS

Drawing Arcs

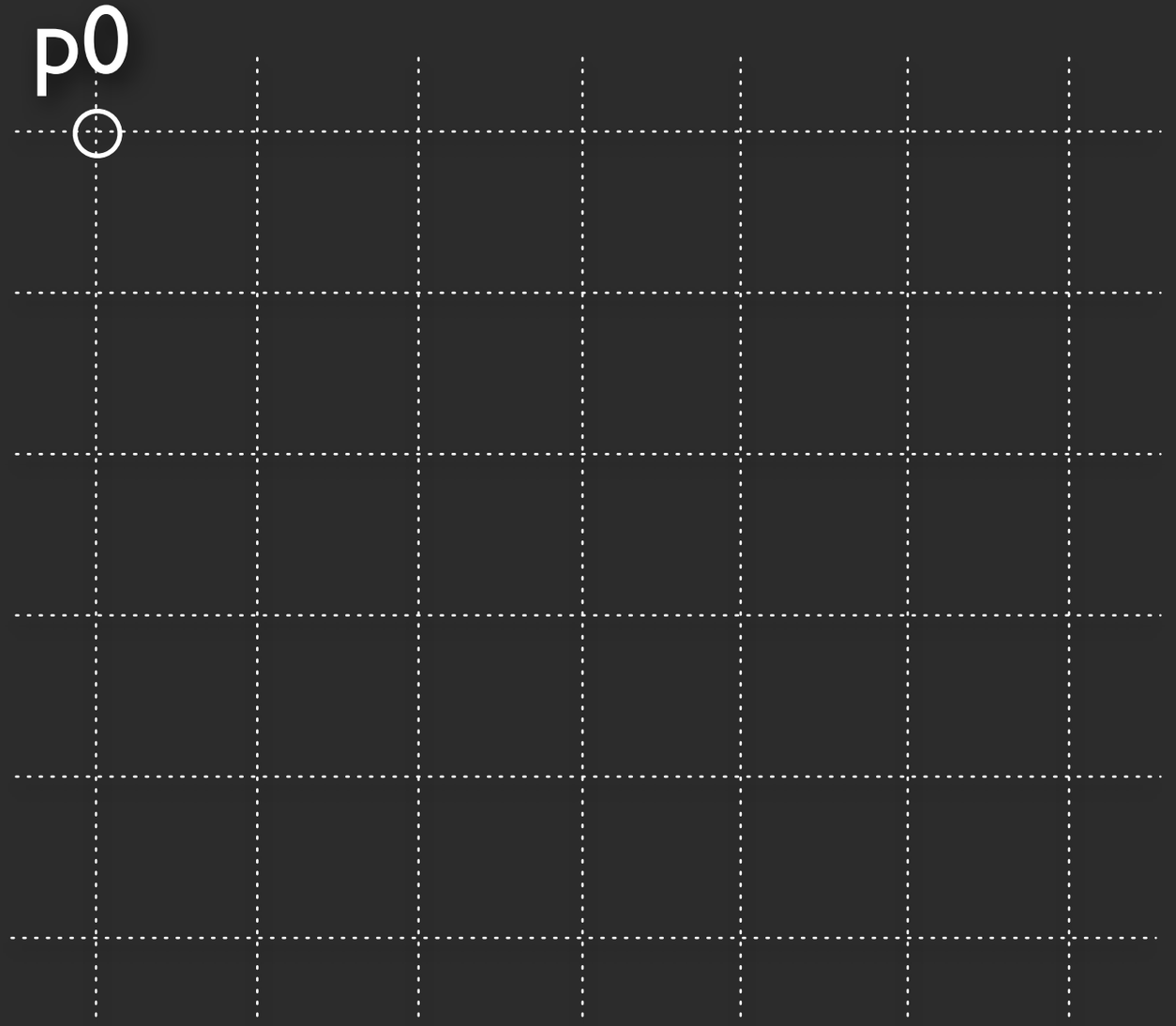
Defining Paths



```
UIBezierPath *path = [UIBezierPath bezierPathWithArcCenter:centerPoint
                                                    radius:radius
                                                startAngle:0
                                                  endAngle:DEGREES(90)
                                                 clockwise:NO];
CGContextAddPath(context, path.CGPath);
```

Quadratic Bézier Curves

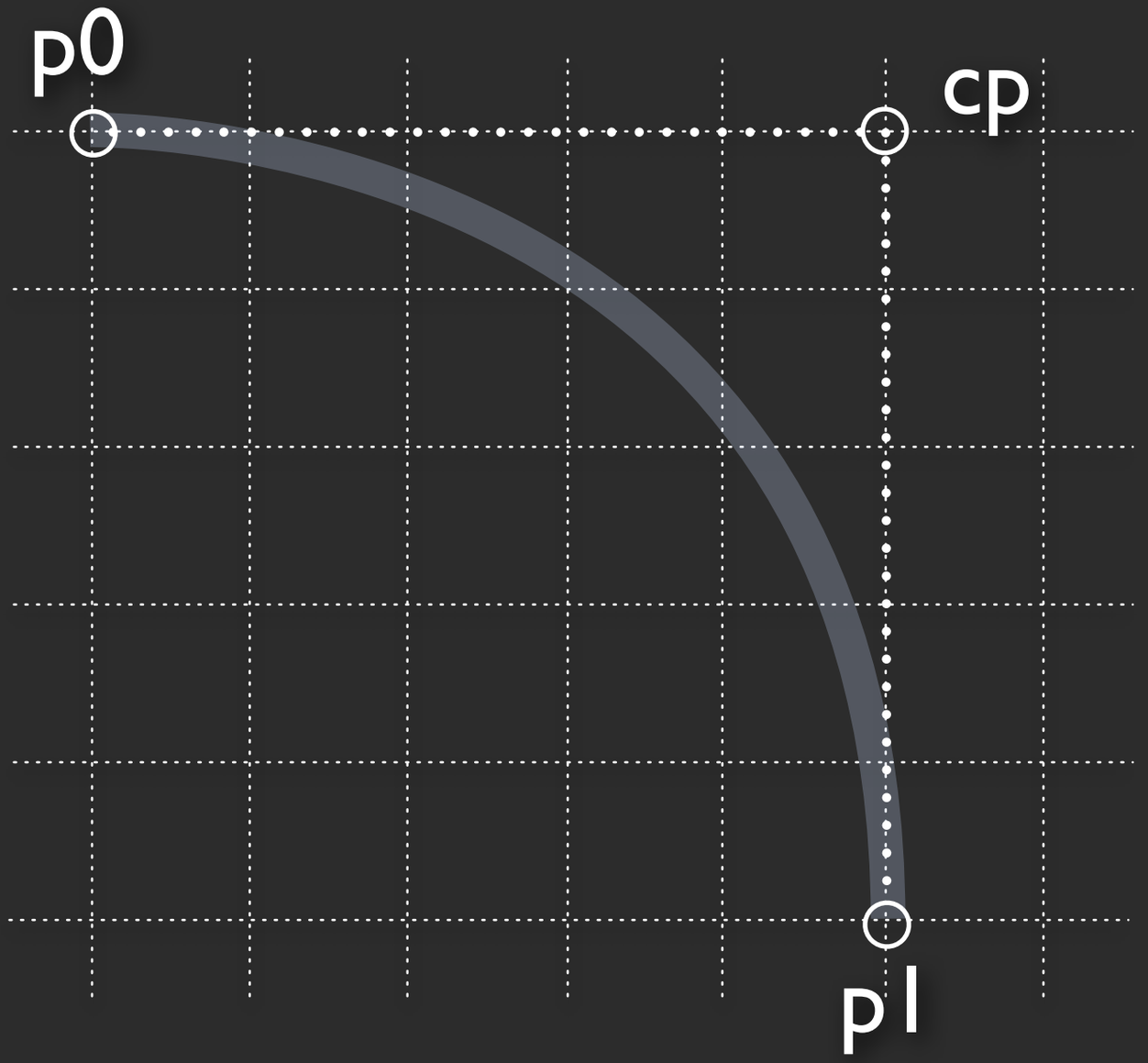
Single Control Point




```
CGContextBeginPath(context);
CGContextMoveToPoint(context, startPoint.x, startPoint.y);
```

Quadratic Bézier Curves

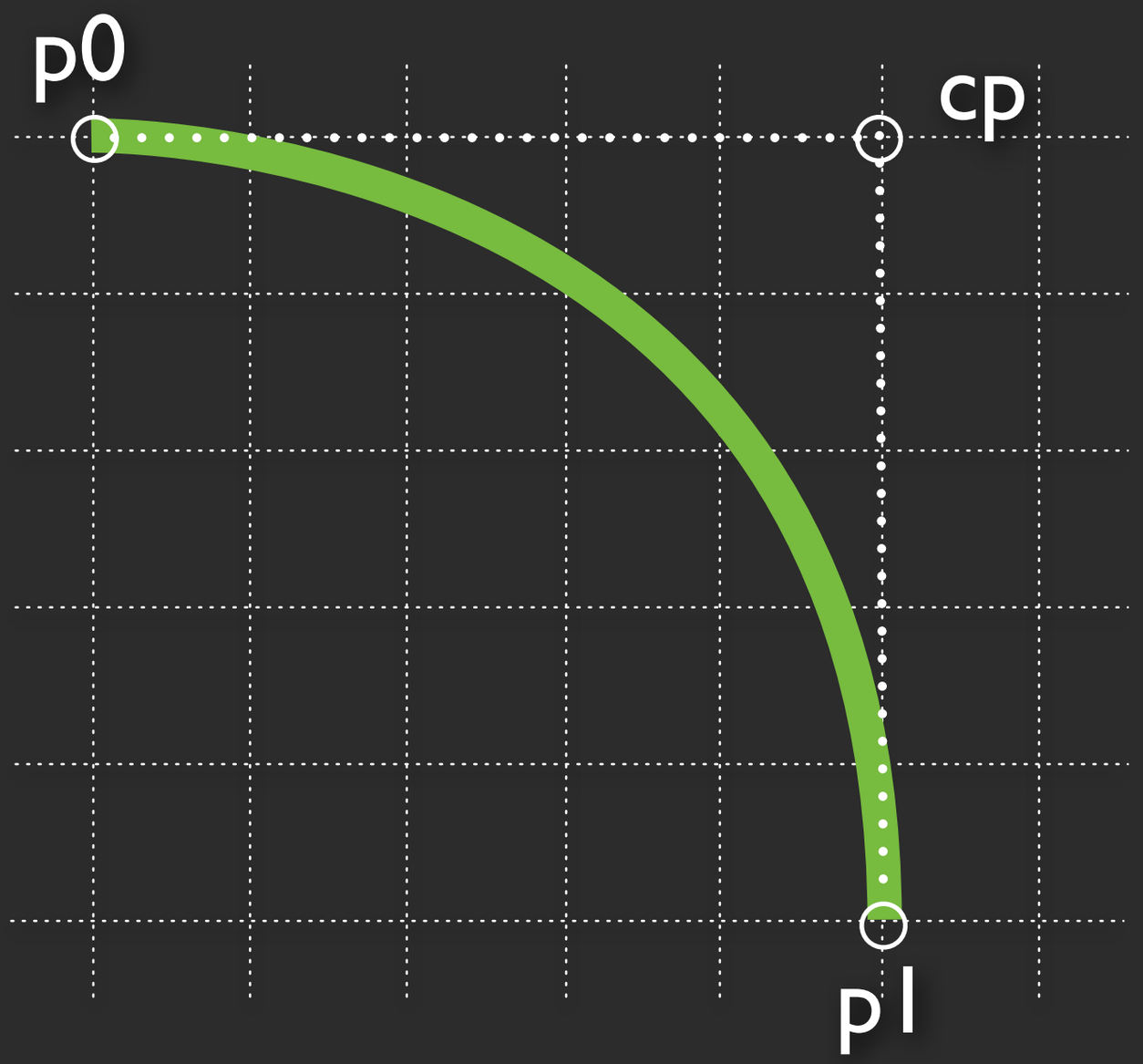
Single Control Point




```
CGContextAddQuadCurveToPoint(context, cp.x, cp.y, endPoint.x, endPoint.y);
```

Quadratic Bézier Curves

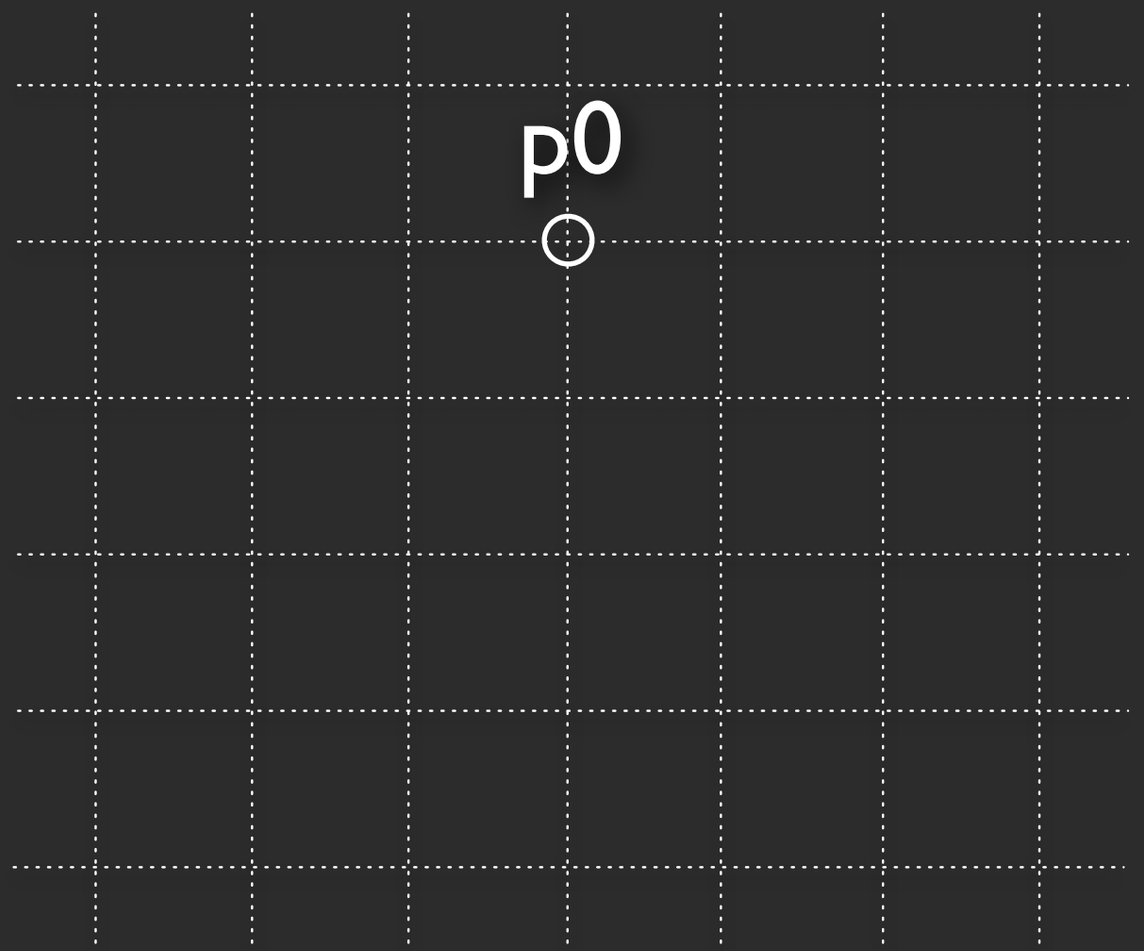
Single Control Point




```
CGColorRef greenColor = [UIColor greenColor].CGColor;
CGContextSetStrokeColorWithColor(context, greenColor);
CGContextStrokePath(context);
```

Cubic Bézier Curves

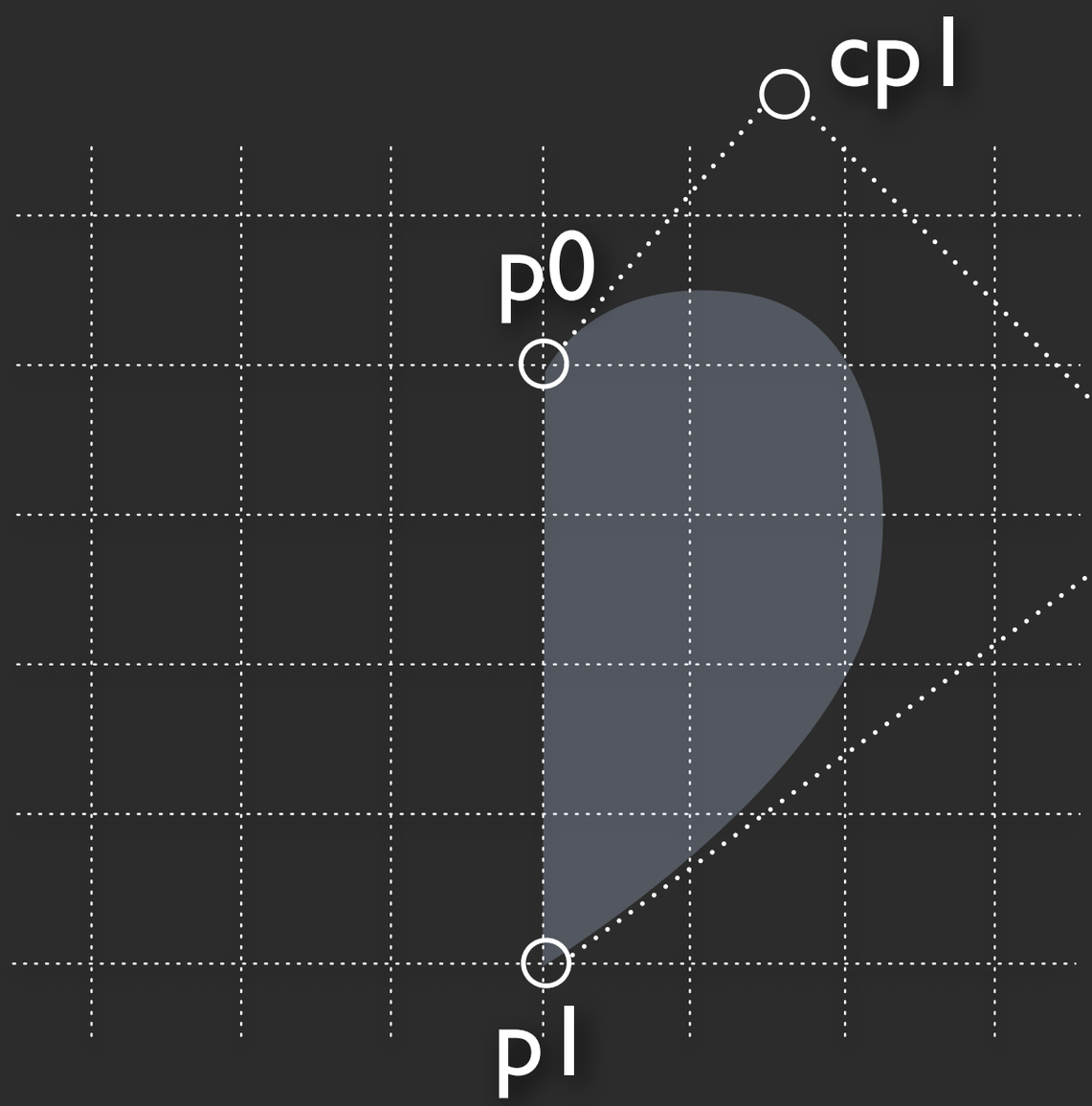
Two Control Points




```
CGContextBeginPath(context);
CGContextMoveToPoint(context, startPoint.x, startPoint.y);
```

Cubic Bézier Curves

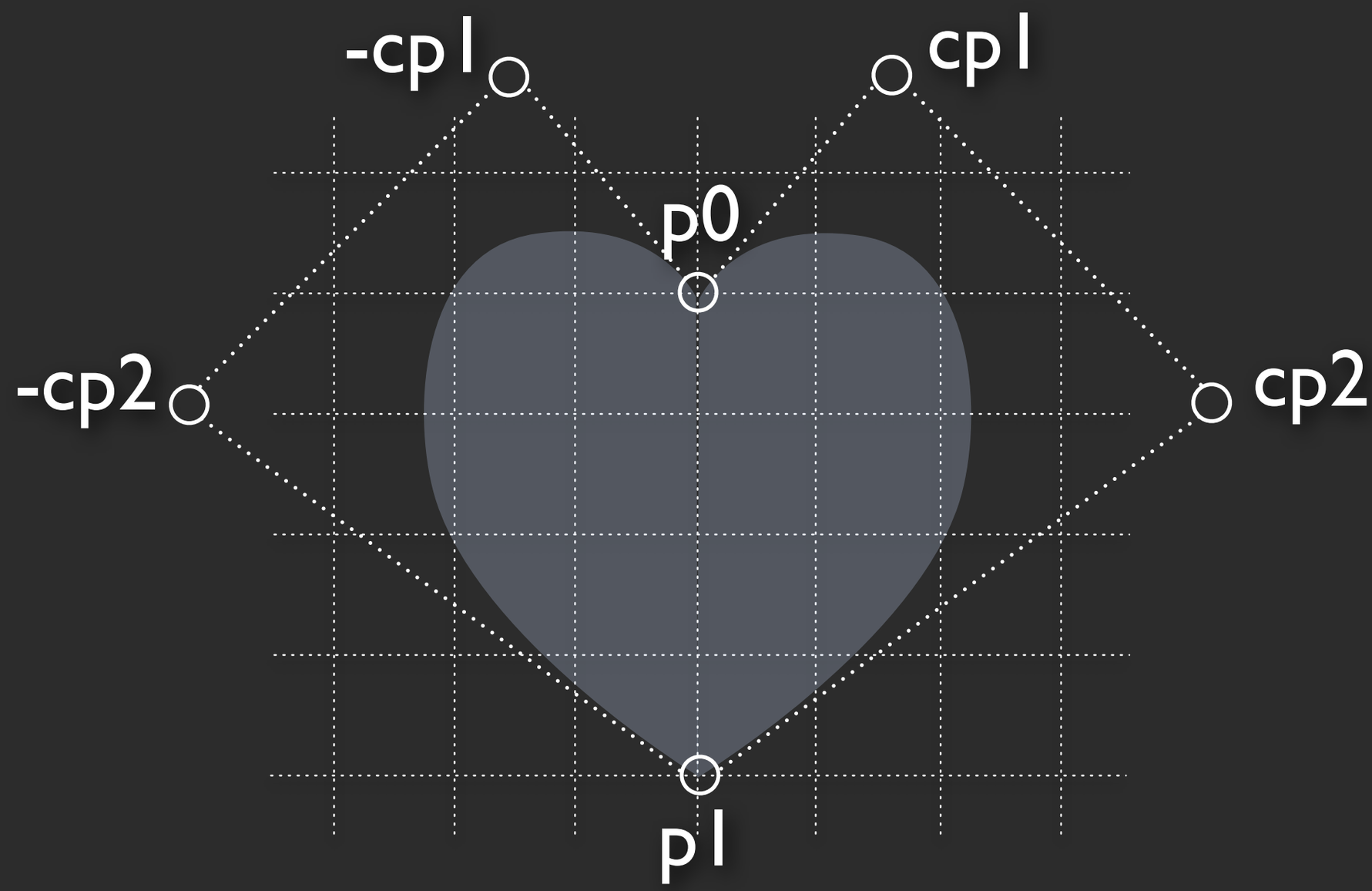
Two Control Points




```
CGContextAddCurveToPoint(context,
                         cp1.x, cp1.y,
                         cp2.x, cp2.y,
                         endPoint.x, endPoint.y);
```

Cubic Bézier Curves

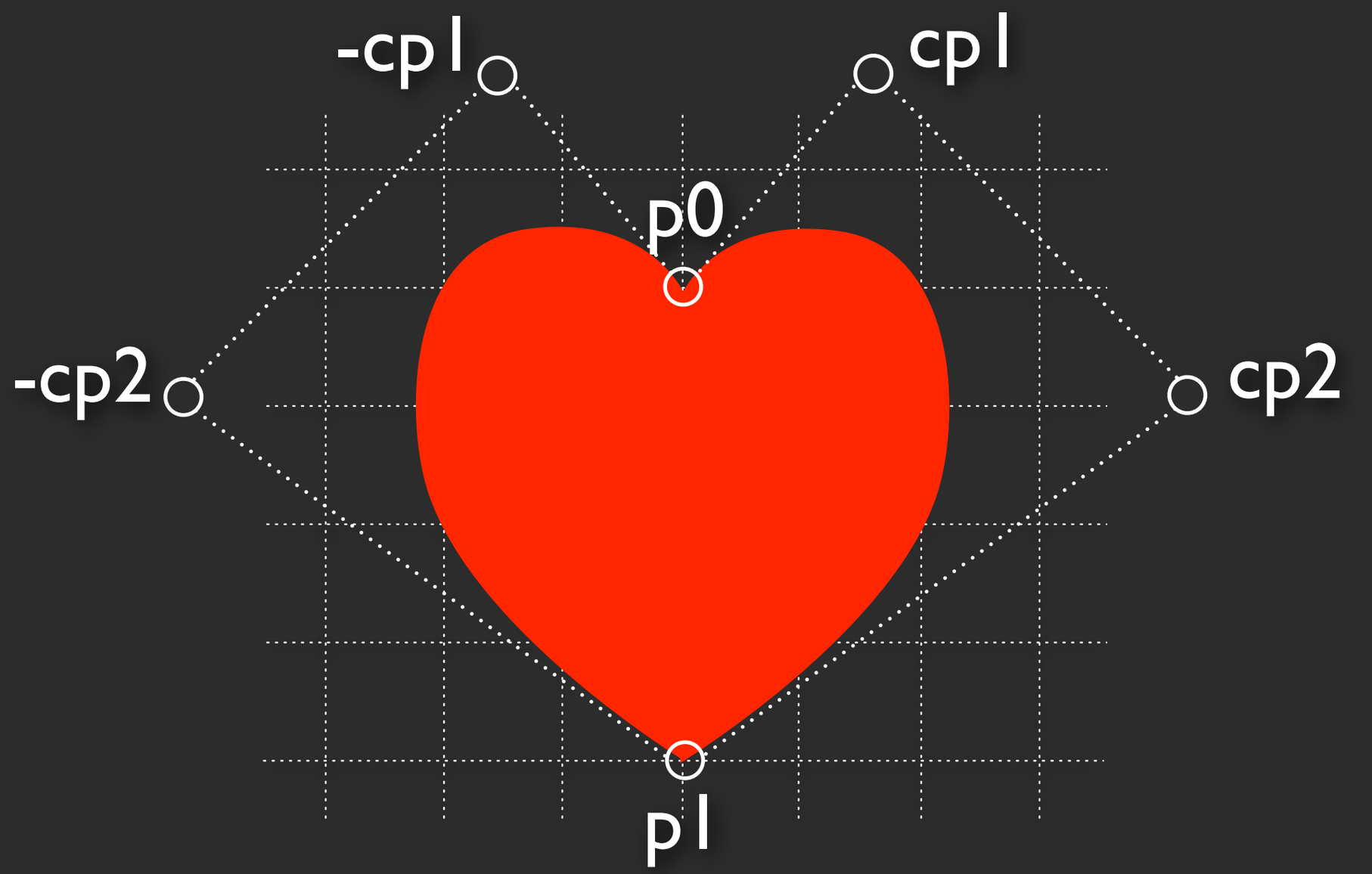
Two Control Points




```
CGContextAddCurveToPoint(context,
                         -cp2.x, cp2.y,
                         -cp1.x, cp1.y,
                         endPoint.x, endPoint.y);
```

Cubic Bézier Curves

Two Control Points

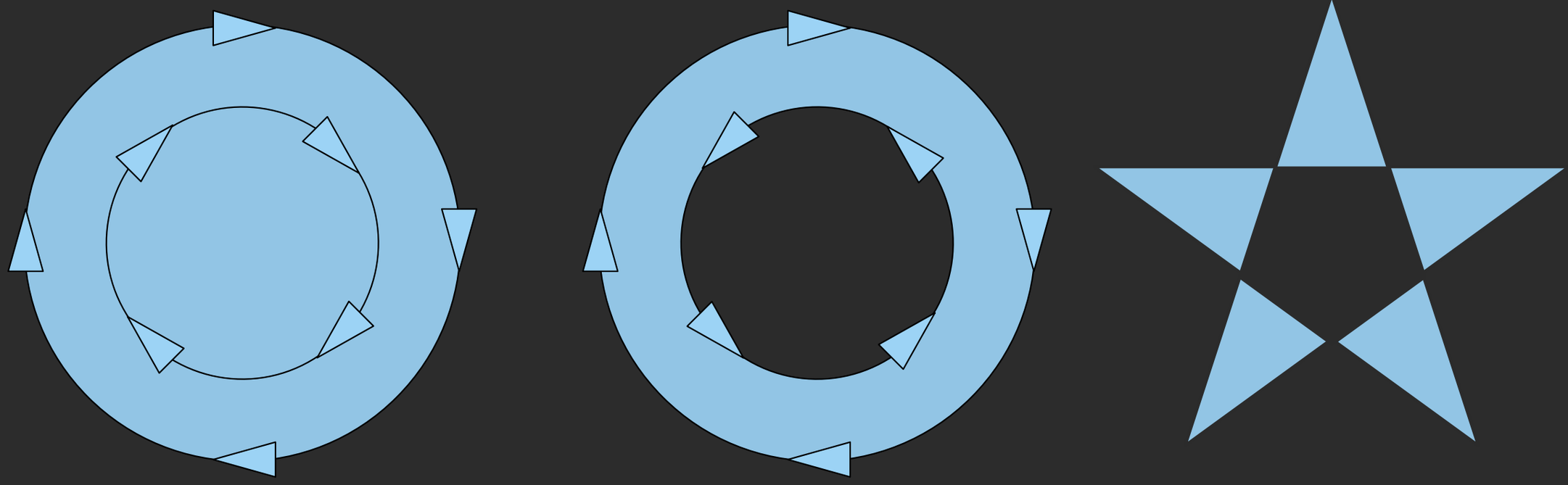



```
CGColorRef fillColor = [UIColor redColor].CGColor;
CGContextSetFillColorWithColor(context, fillColor);
CGContextClosePath(context);
CGContextDrawPath(context, kCGPathFillStroke);
```

Fill Rules

To fill, or not to fill

- Quartz uses fill rules to determine what's inside and outside of a path.
- Uses one of the following:
  - Non-Zero Winding Rule (Default)
  - Even Odd Rule



Reusable Paths

CGPathRef and UIBezierPath

- Paths are flushed from context when drawn
- `CGPath` or `UIBezierPath` should be used when you need a reusable path

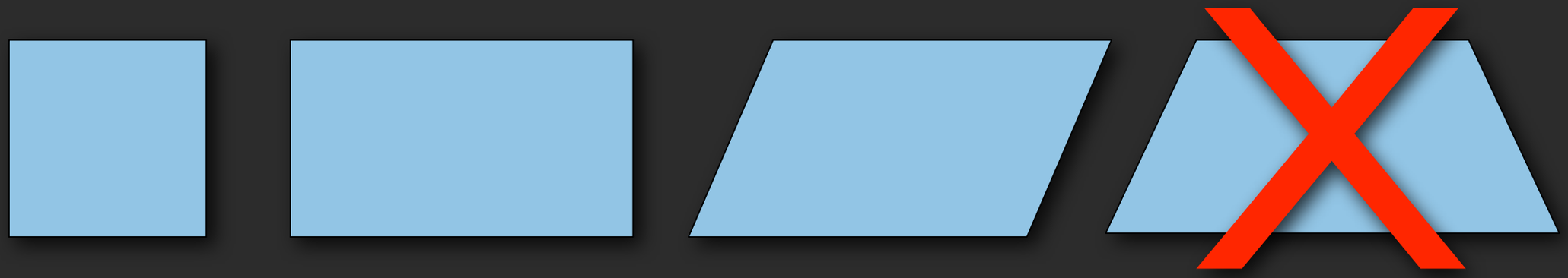
```
CGMutablePathRef path = CGPathCreateMutable();
CGPathMoveToPoint(path, NULL, 0, 55);
CGPathAddQuadCurveToPoint(path, NULL, -50, 155, 0, 155);
CGPathAddQuadCurveToPoint(path, NULL, 50, 155, 0, 55);

CGContextAddPath(context, path);
CGContextDrawPath(context, kCGPathFillStroke);

CGPathRelease(path);
```

Transformations

Affine Transformations

Transforming the Coordinate System

- Affine transformations are essential to all graphics programming regardless of platform
- Allow you to manipulate the coordinate system to translate, scale, skew, and rotate
- Transform preserves collinearity of lines



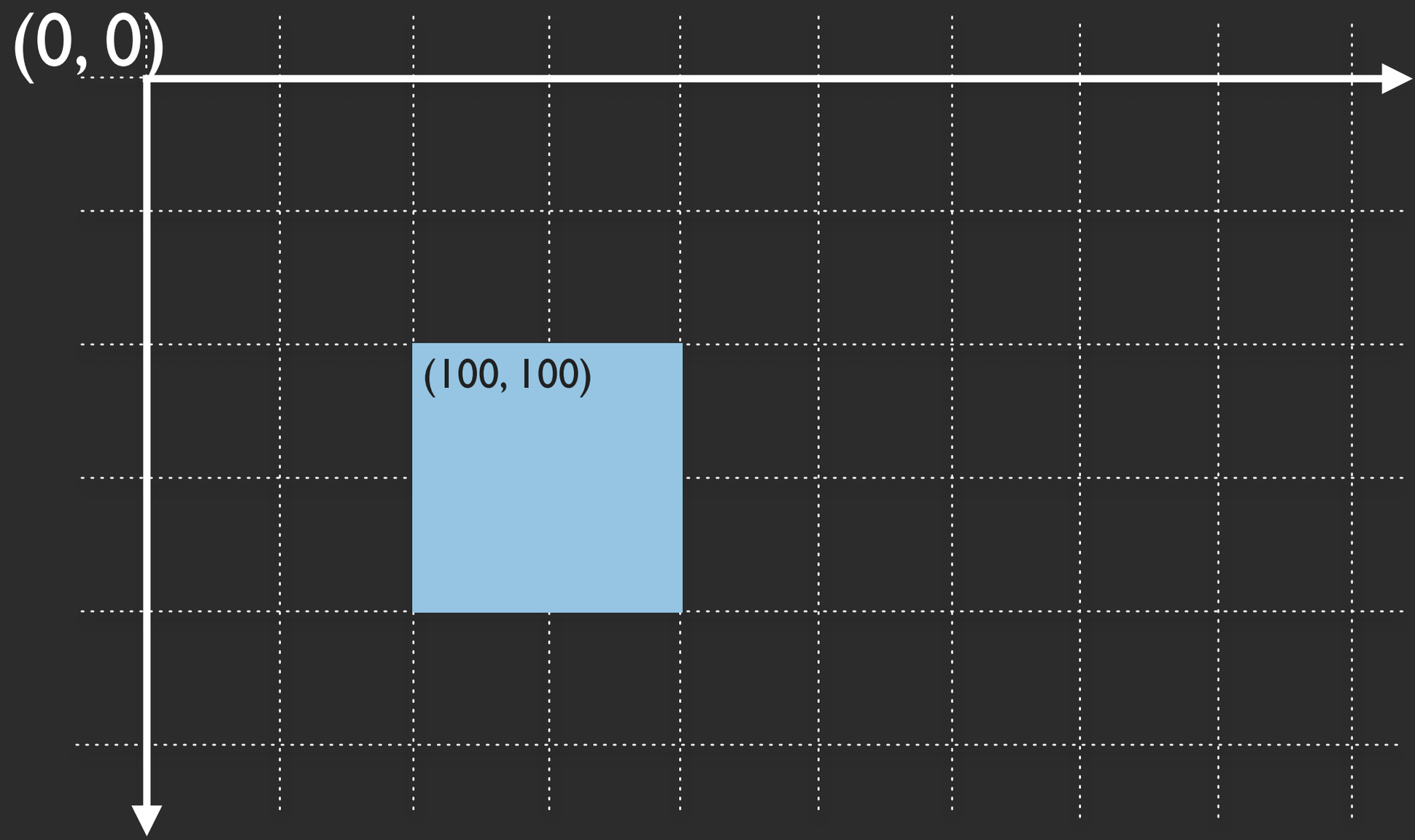
Current Transformation Matrix

Modifying the CTM

- CTM can be modified with the following:
  - `CGContextTranslateCTM`
  - `CGContextScaleCTM`
  - `CGContextRotateCTM`
- Additionally can create a `CGAffineTransform`

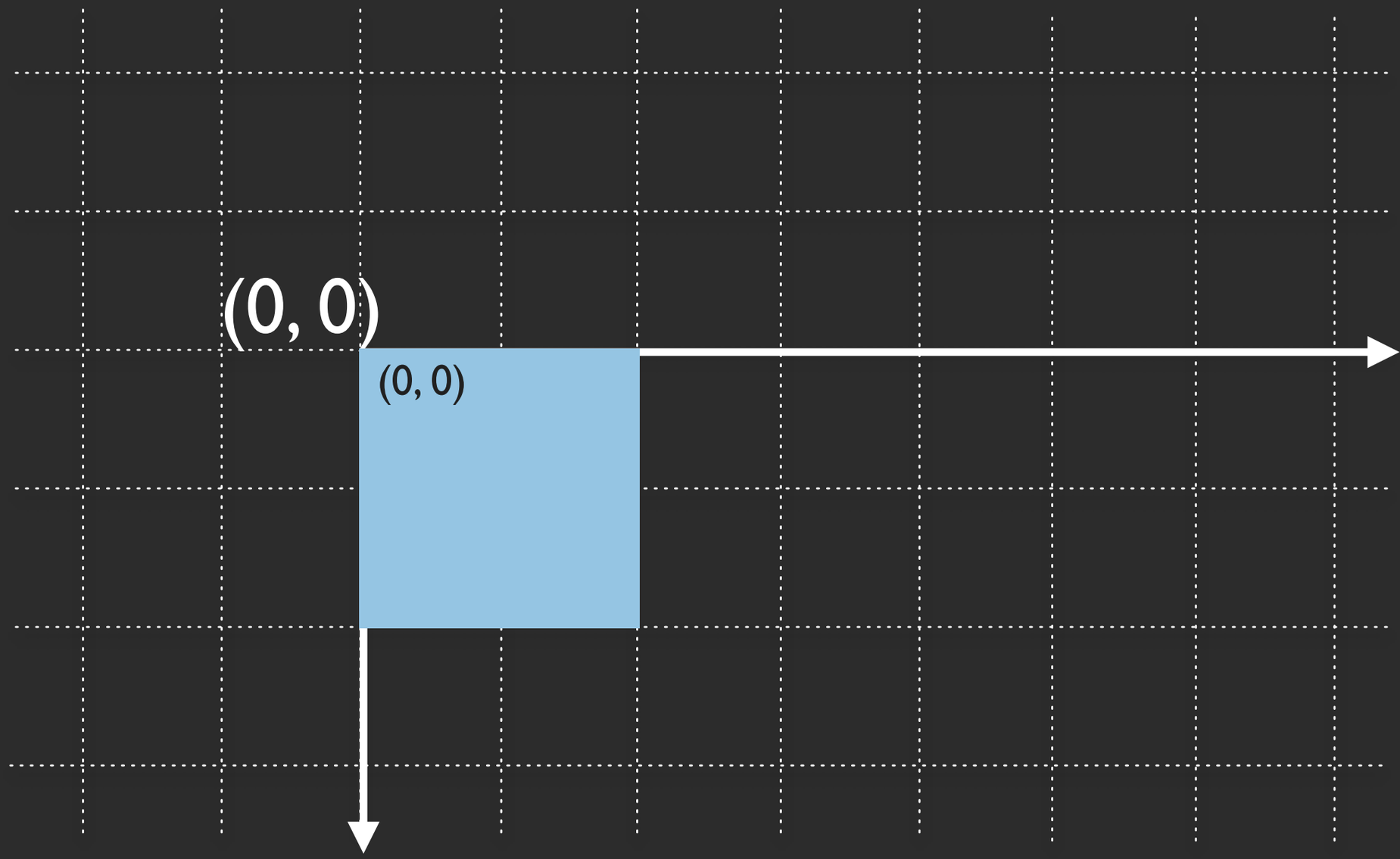
CGContextTranslateCTM

Moving the Origin Point



CGContextTranslateCTM

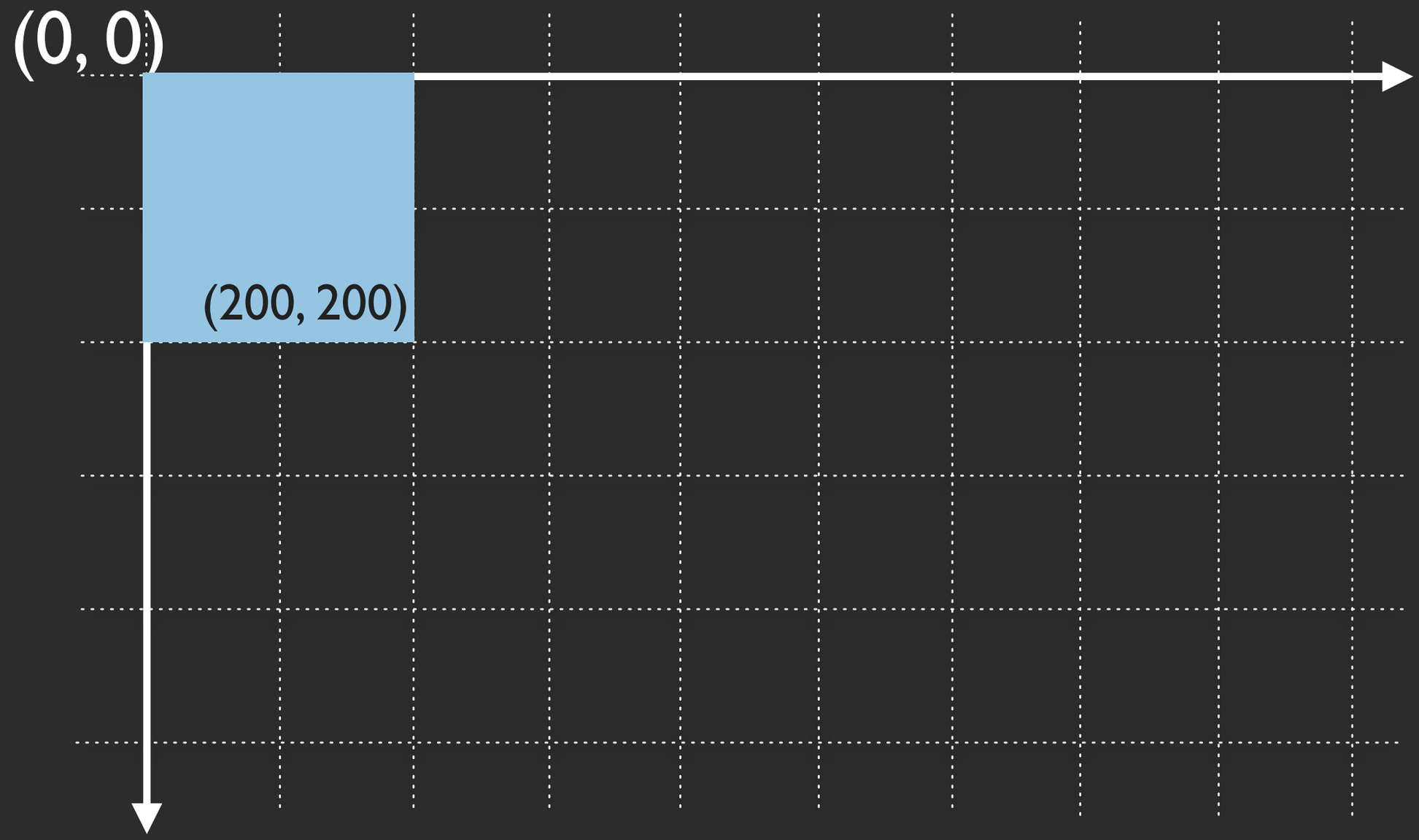
Moving the Origin Point




```
CGContextTranslateCTM(context, 100.0f, 100.0f);
```

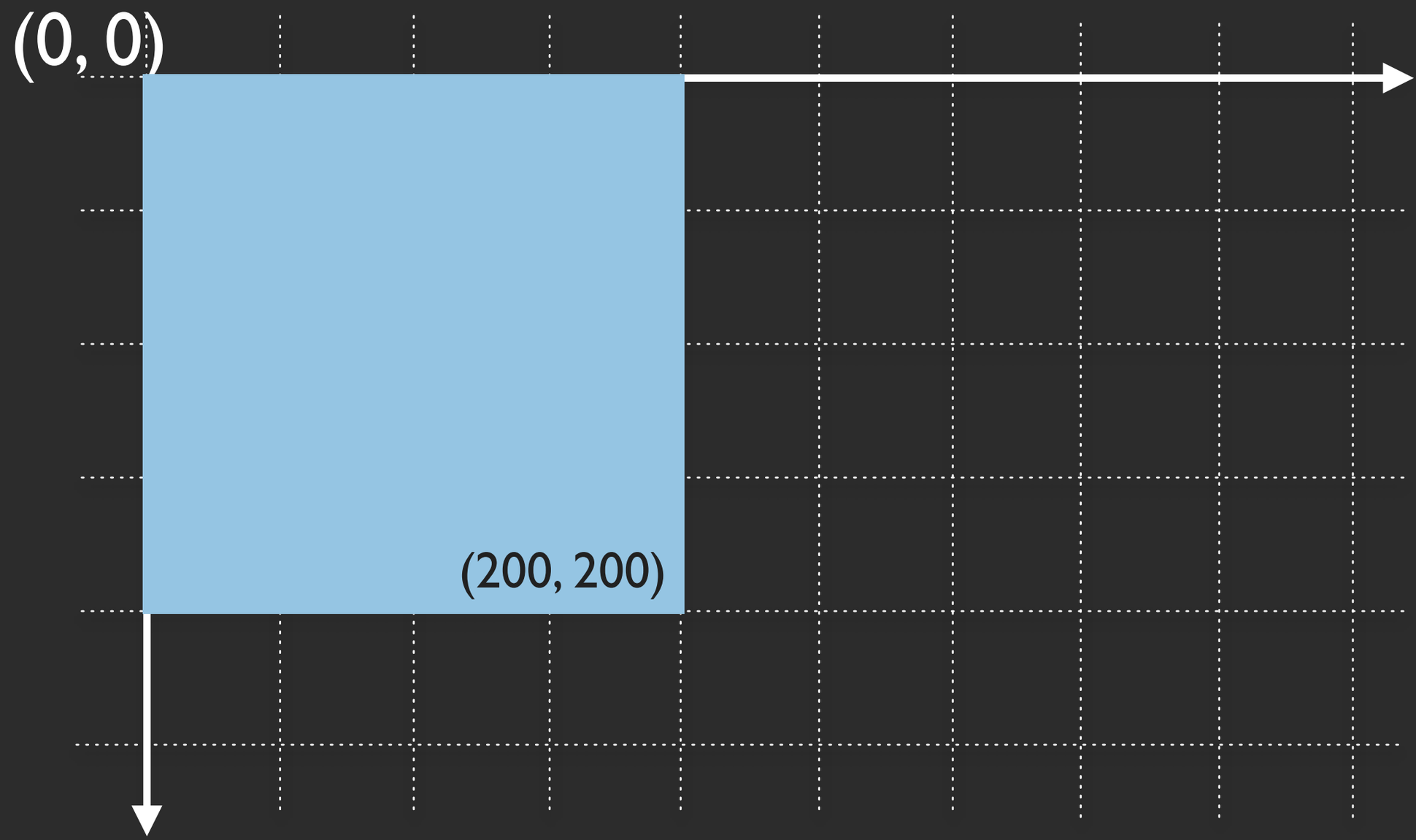
CGContextScaleCTM

Scaling the Coordinates



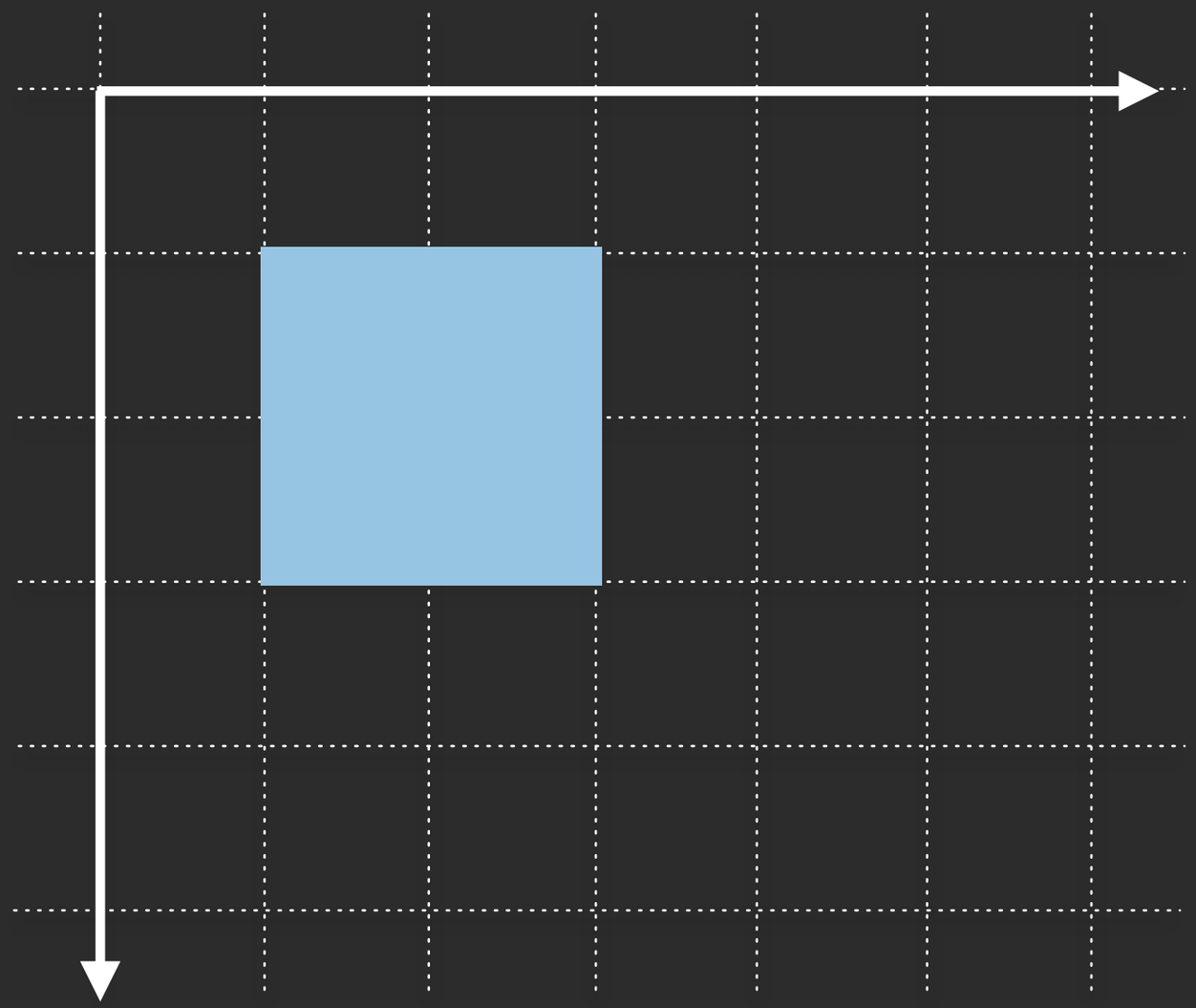
CGContextScaleCTM

Scaling the Unit Size

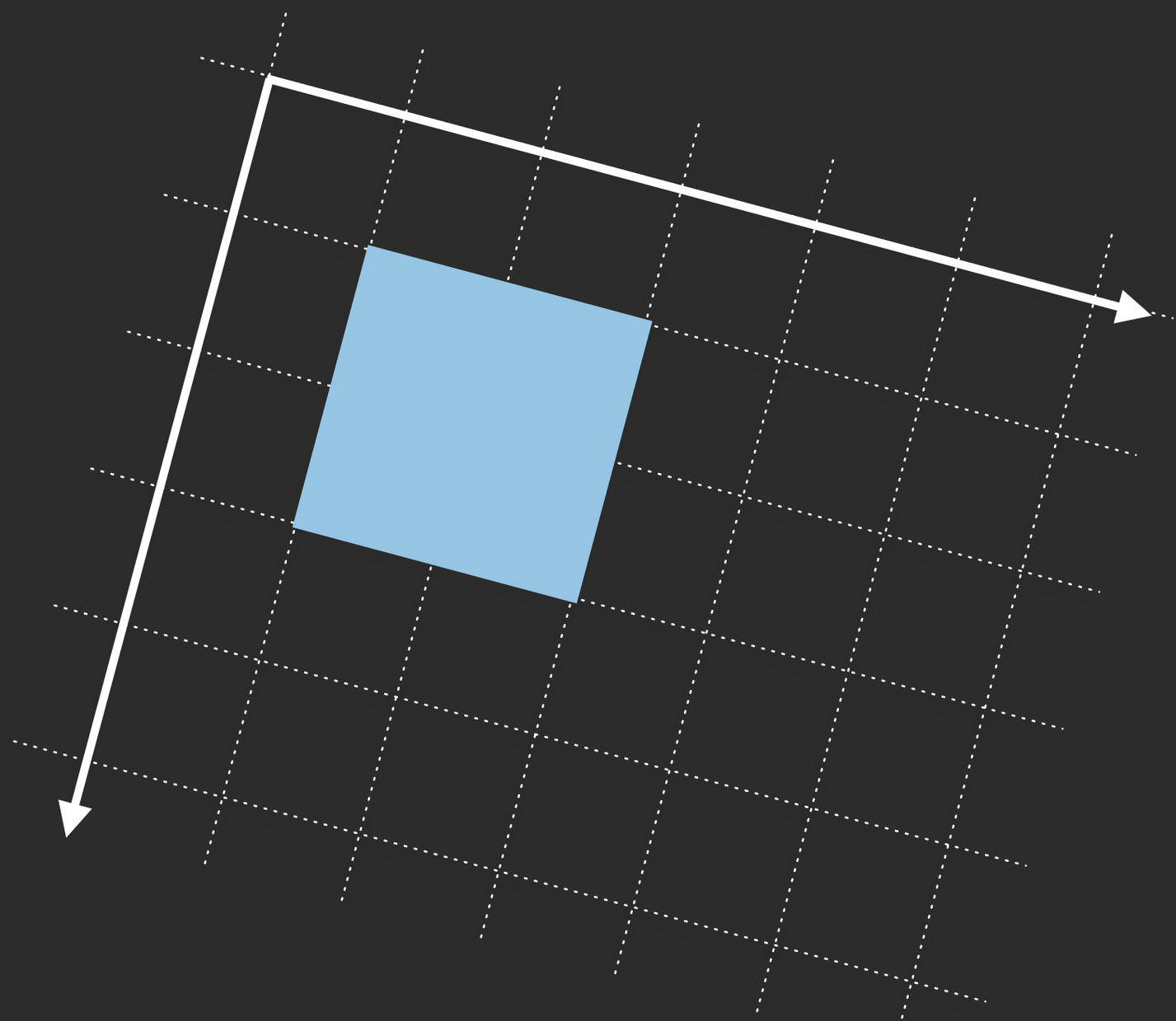



```
CGContextScaleCTM(context, 2.0f, 2.0f);
```

CGContextRotateCTM



CGContextRotateCTM



```
CGContextRotateCTM(context, DEGREES(15));
```

Gradients

Gradients

Drawing Gradients

- A gradient is a fill that varies from one color to another
- Quartz supports two different gradient types
  - Linear varies between two endpoints
  - Radial varies between two radial endpoints

Linear

Radial

Creating a CGGradient

CGGradientRef

- Gradients can be created using either:
  - `CGGradientCreateWithColors`
  - `CGGradientCreateWithColorComponents`
- Both functions require the following:
  - `CGColorSpace`
  - Colors (components or array of `CGColorRef`)
  - `CGFloat[]` defining gradient color stops

CGGradientRef

Building the gradient code

```
// Define Colors
CGColorRef startColor = [UIColor greenColor].CGColor;
CGColorRef endColor = [UIColor yellowColor].CGColor;
NSMutableArray *colors = [NSMutableArray array];
[colors addObject:(id)startColor];
[colors addObject:(id)endColor];

// Define Color Locations
CGFloat locations[] = {0.0, 1.0};

// Create Gradient
CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceRGB();
CGGradientRef gradient =
     CGGradientCreateWithColors(colorSpace, (CFArrayRef)colors, locations);

// Define start and end points and draw gradient
CGPoint startPoint = CGPointMake(CGRectGetMidX(rect), CGRectGetMinY(rect));
CGPoint endPoint = CGPointMake(CGRectGetMidX(rect), CGRectGetMaxY(rect));

CGContextDrawLinearGradient(context, gradient, startPoint, endPoint, 0);
```

Gradient Examples

Images

Working with Images

CGImageRef

- Quartz has broad support for images
- Represented as `CGImageRef`
- Can be drawn using:
  - `CGContextDrawImage`
  - `CGContextDrawTiledImage`
- `CGBitmapContext` allows for dynamically building image data

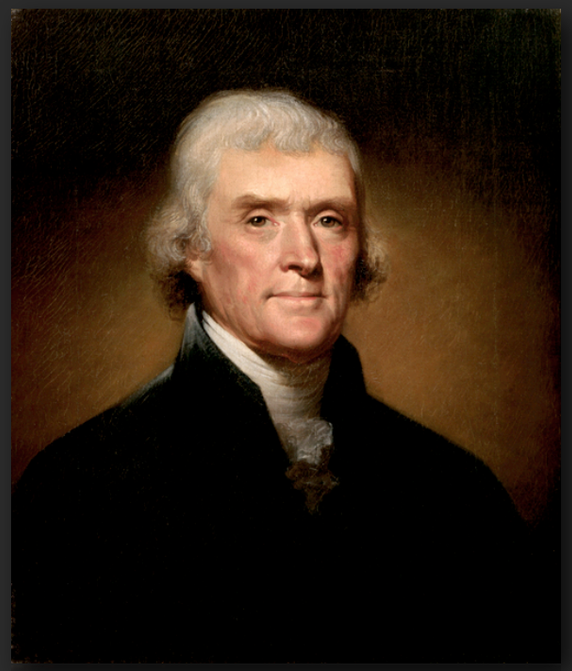
CGImage vs UIImage

Which should I use?

- Don’t draw in Quartz. Use UIImageView.
- Choose UIImage over CGImage
  - Easier to load and cache image content
  - Provides methods for drawing
    - Is aware of flipped coordinate system
    - Easy to convert to and from CGImage
- Use CGImage for more advanced cases
  - Cropping, scaling, masking
  - Dynamic image creation

Using Image Masks

Hiding behind a mask



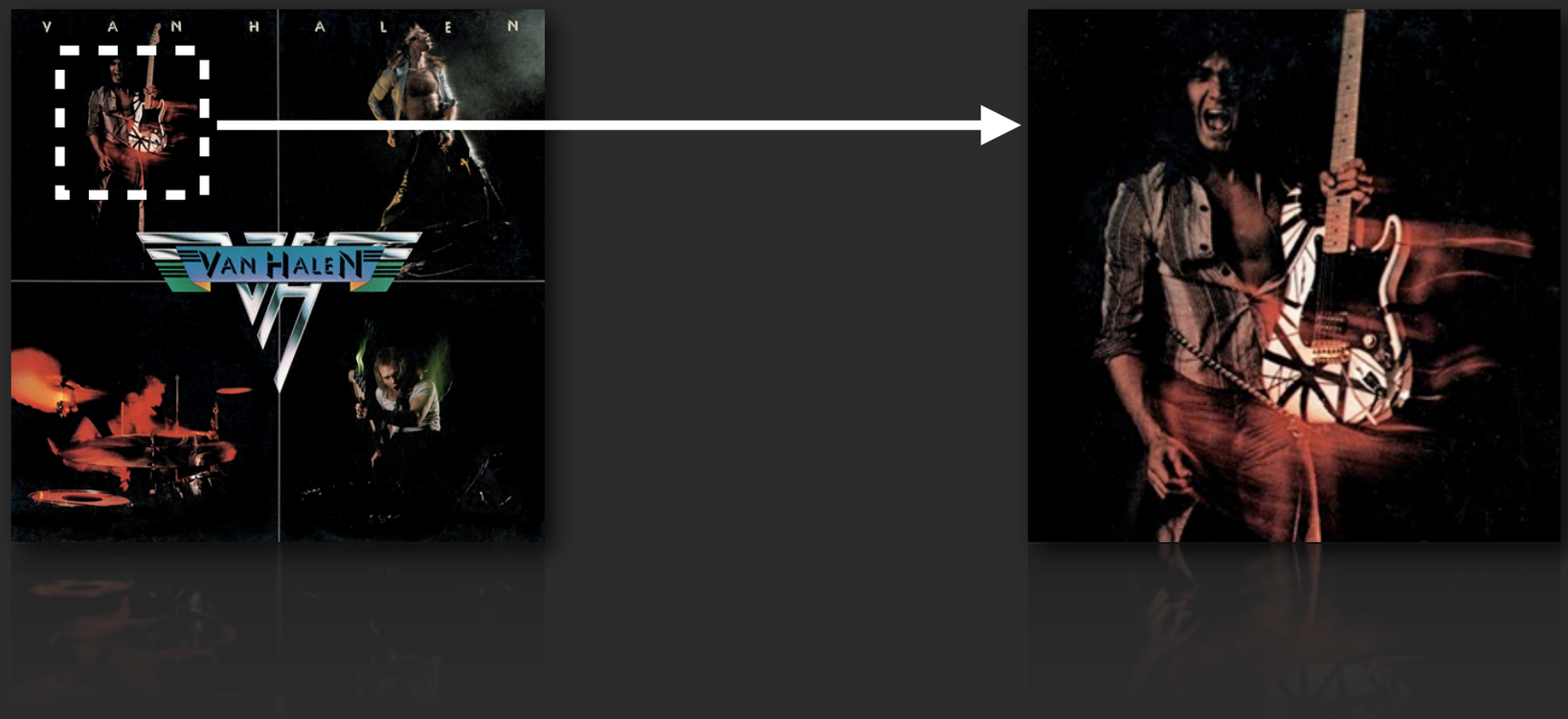
Using Image Masks

Hiding behind a mask

```
UIImage *maskImage = [UIImage imageNamed:@"round_mask"];
UIImage *jeffersonImage = [UIImage imageNamed:@"jefferson"];

CGImageRef maskRef = maskImage.CGImage;
CGImageRef imageMask = CGImageMaskCreate(CGImageGetWidth(maskRef),
                                         CGImageGetHeight(maskRef),
                                         CGImageGetBitsPerComponent(maskRef),
                                         CGImageGetBitsPerPixel(maskRef),
                                         CGImageGetBytesPerRow(maskRef),
                                         CGImageGetDataProvider(maskRef), NULL, false);

CGImageRef masked = CGImageCreateWithMask(jeffersonImage, imageMask);
UIImage *maskedImage = [UIImage imageWithCGImage:masked];
```

Cropping Images

CGImageCreateWithImageInRect




```
UIImage *albumImage = [UIImage imageNamed:@"vh1"];
CGRect cropRect = CGRectMake(20.0f, 20.0f, 200.0f, 200.0f);
CGImageRef image = CGImageCreateWithImageInRect(albumImage.CGImage, cropRect);

CGRect imageRect = CGRectMake(0.0f, 0.0f, 200.0f, 200.0f);
// Draw image in current context
[[UIImage imageWithCGImage:image] drawInRect:imageRect];

CGImageRelease(image);
```

Summary

- Quartz is a powerful, comprehensive drawing engine and API for Apple platforms
- Fairly steep learning curve, but consistent interface helps considerably
- Understanding is essential to building beautiful apps on Mac and iOS

Resources

Quartz 2D Programming Guide

http://developer.apple.com/

Quartz Demos

https://github.com/tapharmonic/QuartzDemos

Programming with Quartz

David Gelphman

Bunny Laden