


CSC 471 / 371 Mobile Application Development for iOS




Prof. Xiaoping Jia
School of Computing, CDM
DePaul University
xjia@cdm.depaul.edu
[@DePaulSWEng](https://twitter.com/DePaulSWEng)

2D Drawing

Outline

- 2-D drawing
- Core Graphics & Quartz 2D



DEPAUL UNIVERSITY

CG Types

- CG (Core Graphics) types begin with prefix CG
 - Used in graphics and animation API
 - CG types are structs. Hence, value types
- **CGFloat**
 - A floating-point type (32/64-bit), *always used for graphics.*
- **CGPoint**
 - location in 2D space: { x, y }
- **CGSize**
 - dimension: { width, height }
- **CGRect**
 - location and dimension: { origin, size }

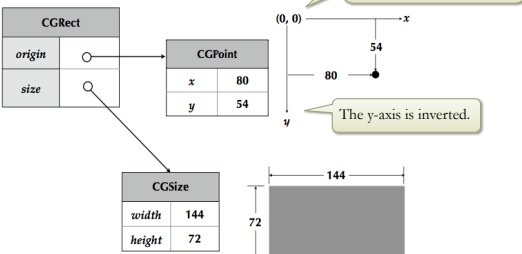
A CGPoint and a CGSize

DEPAUL UNIVERSITY

CG Types and Geometry

The origin of the 2-D coordinate system is always at the upper-left corner.

The y-axis is inverted.



CGRect

origin	→	CGPoint
size	→	CGSize

CGPoint

x	80
y	54

CGSize

width	144
height	72

DEPAUL UNIVERSITY

Geometry and CG Types

- **CGPoint**

```
struct CGPoint {
    var x: CGFloat
    var y: CGFloat
    init(x: CGFloat, y: CGFloat)
}
```
- Create CGPoint


```
var p1 = CGPoint(x: 10, y: 200)
var p2 = CGPointMake(10, 200);
p1.x = 300.0
p1.y = 30.0
```

DEPAUL UNIVERSITY

Geometry and CG Types

- CGSize


```
struct CGSize {
    var width: CGFloat
    var height: CGFloat
    init(width: CGFloat, height: CGFloat)
}
```
- Create CGSize


```
var s1 = CGSize(width: 10, height: 20)
var s2 = CGSizeMake(10, 20)
s1.width = 100.0
s1.height = 72.0
```

DEPAUL UNIVERSITY

7

Geometry and CG Types

- CGRect


```
struct CGRect {
    var origin: CGPoint
    var size: CGSize
    init(origin: CGPoint, size: CGSize)
    init(x: CGFloat, y: CGFloat,
         width: CGFloat, height: CGFloat)
}
```
- Create CGRect


```
var r1 = CGRect(x: 10, y: 10, width: 200, height: 40)
var r2 = CGRectMake(10, 10, 200, 40)
r1.origin.x = 0.0
r1.size.width = 50.0
```

DEPAUL UNIVERSITY

8

Core Graphics & Quartz 2D

Custom View Classes

- Define custom *view* classes
 - Subclass `UIView`
- Override the `drawRect:` method in `UIView` to draw a custom view


```
func drawRect(rect: CGRect)
```

 - The `rect` argument is the area to draw
 - If not overridden, the `backgroundColor` is used to fill the view

DEPAUL UNIVERSITY

10

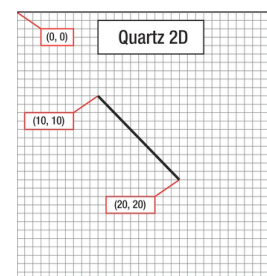
When Is It OK to Call drawRect:?

- The method `drawRect` is invoked by the system
- Don't call it directly!
- Be lazy
 - Being lazy is good for performance
 - System decides when and how often to call the `drawRect` method
- When a view needs to be redrawn, call the method `setNeedsDisplay()`

DEPAUL UNIVERSITY

11

Quartz 2D Coordinate System



- The coordinate system for drawing views in iOS is flipped
 - Different from the coordinate system used by OpenGL
- Consistent with the coordinate system used by UIKit

DEPAUL UNIVERSITY

12

Core Graphics and Quartz 2D

- UIKit offers very basic drawing functions


```
func UIRectFill(rect: CGRect)
func UIRectFrame(rect: CGRect)
```
- Core Graphics
 - A C-based drawing API, i.e., all functions.
 - Bridged to Swift
- CG and Quartz 2D drawing engine define simple but powerful graphics primitives
 - Graphics context ■ Transformations ■ Paths
 - Colors ■ Fonts ■ Painting operations

DEPAUL UNIVERSITY

13

Graphics Context

- A graphics context is an object that encapsulates information about drawing onto an output device
 - display (view)
 - bitmap image (buffer)
 - PDF file
 - Printer, etc.
- A graphics context supports
 - drawing shapes (paths), images, etc.
 - graphics attributes (states), colors, patterns, etc.

DEPAUL UNIVERSITY

14

Graphics Context

- A graphics context is setup automatically before invoking `drawRect`
 - Defines current path, line width, transform, etc.
 - Access the graphics context within `drawRect`: by calling


```
func UIGraphicsGetCurrentContext() -> CGContext!
```
 - Use CG calls to change settings
- A `CGContext` object is only valid for the current call to `drawRect`
 - **Do not cache a `CGContext` object**

DEPAUL UNIVERSITY

15

CG Wrappers in UIKit

- Some CG functionalities are wrapped by UIKit
- **UIColor**
 - Convenience for common colors
 - Easily set the fill and/or stroke colors when drawing


```
let redColor = UIColor.redColor()
redColor.set() // drawing will be done in red
```
- **UIFont**
 - Access system font. Get font by name

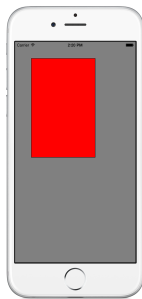

```
let font = UIFont.systemFontOfSize(17)
myLabel.font = font
```

DEPAUL UNIVERSITY

16

A Simple Drawing App

- A single view app with a custom view class
- Use the simple drawing functions in UIKit
 - Draw background
 - Draw a rectangle



DEPAUL UNIVERSITY

17

Create a Custom View App

- Start with a single view app
- New | File | iOS source | Cocoa Touch Class
 - Class name: `MyView`
 - Subclass of: `UIView`
 - Language: Swift
 - Click “Next”, select folder to create the new file
- In the main storyboard
 - Select the *View* of the initial scene
 - Use the *Identify Inspector* to change the class to `MyView`

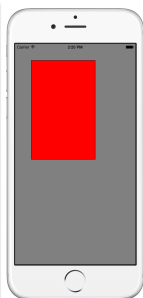
DEPAUL UNIVERSITY

18

The Custom View Class for Simple Drawing

```
import UIKit
class MyView: UIView {
    override func drawRect(rect: CGRect) {
        UIColor.grayColor().set()
        UIRectFill(bounds);

        let rect = CGRect(x: 50, y: 50,
                          width: 200, height: 300)
        UIColor.redColor().set()
        UIRectFill(rect)
        UIColor.blackColor().set()
        UIRectFrame(rect)
    }
}
```



DEPAUL UNIVERSITY

19

Drawing More Complex Shapes

- Common steps for Quartz drawing:
 - Get the current graphics context
 - Define a path
 - Set a color
 - Stroke or fill path
 - Repeat, if necessary

DEPAUL UNIVERSITY

20

Graphics States

- Quartz modifies the results of drawing operations according to the parameters in the **current graphics state**.
- The parameters include
 - Transformation matrix
 - Clipping area
 - Color and transparency
 - Line attributes: width, join, cap, dash
 - Font for text
 - etc.

DEPAUL UNIVERSITY

21

Path Based Drawing

- A **path** defines one or more shapes
- A path can consist of straight lines, curves, or both.
- It can be open or closed.



DEPAUL UNIVERSITY

22

Building Blocks

Open shapes

- Points
- Lines
- Arcs
- Curves
 - cubic Bézier curve
 - quadratic Bézier curve

Closed shapes

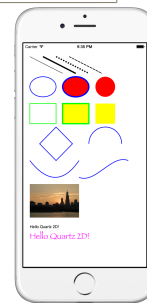
- Ellipses
- Rectangles

DEPAUL UNIVERSITY

23

Quartz Demo

- Solid and dashed lines
- Stroke and fill ellipses and circles
- Stroke and fill rectangles and squares
- Paths
- Arcs
- Quadratic and cubic Bézier curves
- Images
- Text



DEPAUL UNIVERSITY

24

Quartz Demo – Draw Solid Lines

```
override func drawRect(rect: CGRect) {
    let context = UIGraphicsGetCurrentContext()

    CGContextMoveToPoint(context, 20, 40)
    CGContextAddLineToPoint(context, 120, 90)
    CGContextStrokePath(context)

    CGContextSetLineWidth(context, 4)
    CGContextMoveToPoint(context, 60, 40)
    CGContextAddLineToPoint(context, 160, 90)
    CGContextStrokePath(context)

    ...
}
```

 DePAUL UNIVERSITY

25

Quartz Demo – Draw Dashed Lines

```
let shortDash : [CGFloat] = [ 4, 4 ]
CGContextSetLineDash(context, 0, shortDash, 2)
CGContextMoveToPoint(context, 100, 40)
CGContextAddLineToPoint(context, 200, 90)
CGContextStrokePath(context)

CGContextSetLineWidth(context, 2)
let longDash : [CGFloat] = [ 8, 2 ]
CGContextSetLineDash(context, 0, longDash, 2)
CGContextMoveToPoint(context, 140, 40)
CGContextAddLineToPoint(context, 240, 90)
CGContextStrokePath(context)
```

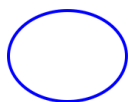
 DePAUL UNIVERSITY

26

Quartz Demo – Draw Ovals and Circles

```
CGContextSetStrokeColorWithColor(context,
    UIColor.blueColor().CGColor)
CGContextSetFillColorWithColor(context,
    UIColor.redColor().CGColor)
CGContextSetLineDash(context, 0, nil, 0)

let rect1 = CGRect(x: 20, y: 100, width: 80, height: 60)
CGContextStrokeEllipseInRect(context, rect1)
```

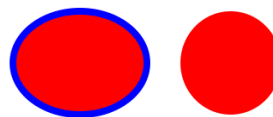

 DePAUL UNIVERSITY

27

Quartz Demo – Draw Ovals and Circles

```
CGContextSetLineWidth(context, 4)
let rect2 = CGRect(x: 120, y: 100, width: 80, height: 60)
CGContextFillEllipseInRect(context, rect2)
CGContextStrokeEllipseInRect(context, rect2)

let rect3 = CGRect(x: 220, y: 100, width: 60, height: 60)
CGContextFillEllipseInRect(context, rect3)
```


 DePAUL UNIVERSITY

28

Quartz Demo – Draw Rectangles and Squares

```
CGContextSetStrokeColorWithColor(context,
    UIColor.greenColor().CGColor)
CGContextSetFillColorWithColor(context,
    UIColor.yellowColor().CGColor)
CGContextSetLineWidth(context, 2)

let rect4 = CGRect(x: 20, y: 180, width: 80, height: 60)
CGContextStrokeRect(context, rect4)
```

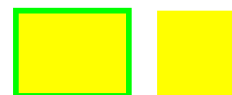

 DePAUL UNIVERSITY

29

Quartz Demo – Draw Rectangles and Squares

```
CGContextSetLineWidth(context, 4)
let rect5 = CGRect(x: 120, y: 180, width: 80, height: 60)
CGContextFillRect(context, rect5)
CGContextStrokeRect(context, rect5)

let rect6 = CGRect(x: 220, y: 180, width: 60, height: 60)
CGContextFillRect(context, rect6)
```


 DePAUL UNIVERSITY

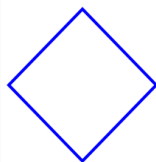
30

Quartz Demo – Draw Paths

- Draw a diamond by tracing the edges

```
CGContextSetStrokeColorWithColor(context,
UIColor.blueColor().CGColor)
CGContextSetLineWidth(context, 2)

CGContextMoveToPoint(context, 100, 250)
CGContextAddLineToPoint(context, 150, 300)
CGContextAddLineToPoint(context, 100, 350)
CGContextAddLineToPoint(context, 50, 300)
CGContextClosePath(context)
CGContextStrokePath(context)
```



Quartz Demo – Draw Arcs

```
CGContextAddArc(context,
250, 300, // the center
50, // the radius
0, // the start angle
CGFloat(M_PI), // the end angle
1) // counter-clockwise
CGContextStrokePath(context)
```



Quartz Demo – Draw Bézier Curves

```
CGContextMoveToPoint(context, 20, 350)
CGContextAddQuadCurveToPoint(context,
100, 450, // the control point
170, 350) // the end point
CGContextStrokePath(context)
```

A quadratic Bézier curve.

```
CGContextMoveToPoint(context, 170, 400)
CGContextAddCurveToPoint(context,
220, 420, // the 1st control point
270, 330, // the 2nd control point
320, 350) // the end point
CGContextStrokePath(context)
```

A cubic Bézier curve.

Quartz Demo – Draw Images

```
let image = UIImage(named: "Chicago")
let rect = CGRect(x: 20, y: 420, width: 150, height: 100)
image?.drawInRect(rect)
```



```
image?.drawAtPoint(CGRect(x: 20, y: 420))
```

Draw the image at the full size.

Quartz Demo – Draw Text

```
let text: NSString = "Hello Quartz 2D!"
text.drawAtPoint(CGPoint(x: 20, y: 540),
withAttributes: nil)
```

The top-left corner of the text

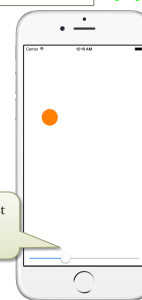
```
let textAttr = [
NSForegroundColorAttributeName : Hello Quartz 2D!
UIColor.magentaColor(),
NSFontAttributeName :
UIFont(name: "Papyrus", size: 24)!
]
text.drawAtPoint(CGPoint(x: 20, y: 560),
withAttributes: textAttr)
```

Hello Quartz 2D!

The Bouncing Ball App

- Simple animation
 - Using a timer and
 - Quartz drawing
- A custom view class: *Canvas View*
- The top view container contains
 - A *Canvas View*, and
 - A *Slider*
 - To control the velocity
 - Range: 1 – 10

A slider to adjust the velocity of the ball



The Canvas View Class – Drawing

```
class CanvasView: UIView {
    var x: CGFloat = 0, y: CGFloat = 0, r: CGFloat = 25
    var velocity: CGFloat = 1
    var dx: CGFloat = 1, dy: CGFloat = 1
    override func drawRect(rect: CGRect) {
        let context = UIGraphicsGetCurrentContext()
        CGContextSetFillColorWithColor(context,
            UIColor.orangeColor().CGColor)
        let rect = CGRect(x: x - r, y: y - r,
            width: 2 * r, height: 2 * r)
        CGContextFillEllipseInRect(context, rect)
    }
    ...
}
```

Draw the ball at the current position

The Canvas View Class – Start Animation

- Use a time that fires at a specified interval
 - A method is called when the timer fires
 - A timer can fire once only or fire repeatedly

```
func start() {
    NSTimer.scheduledTimerWithTimeInterval(1.0/60.0,
        target: self, selector: "update:",
        userInfo: nil, repeats: true)
    x = bounds.width / 2
    y = bounds.height / 2
}
```

A repeating timer.

Call this method each time the timer fires. You must have a colon on the end of the selector!

The Canvas View Class – Updating

- Update the ball position, then request the canvas be redrawn

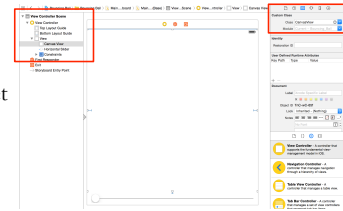
```
func update(timer: NSTimer) {
    x += dx * velocity
    y += dy * velocity
    if x < r || x > bounds.width - r {
        dx = -dx
    }
    if y < r || y > bounds.height - r {
        dy = -dy
    }
    self.setNeedsDisplay()
}
```

Invoked when each time the timer fires

Request the canvas be redrawn. Never call drawRect directly

The Storyboard Scene

- Insert a *View* and a *Horizontal Slider* to the top container
- Select the *View*
 - Change class to: *Canvas View*
- Connect an outlet to the *canvas view*



The View Controller Class

```
class ViewController: UIViewController {
    @IBOutlet weak var canvas: CanvasView!

    override func viewWillAppear(animated: Bool) {
        canvas.start()
    }

    @IBAction func velocityChanged(sender: UISlider) {
        canvas.velocity = CGFloat(sender.value)
    }
    ...
}
```

Kick off the animation


Sample Code

- Drawing Demo.zip
- Quartz Demo.zip
- Bouncing Ball.zip

Next ...

- Touch events
- Gestures & gesture recognizers

❖ iOS is a trademark of Apple Inc.

 DEPAUL UNIVERSITY

42

