CSC 471 / 371 Mobile Application Development for iOS

Prof. Xiaoping Jia
School of Computing, CDM
DePaul University
xjia@cdm.depaul.edu
@DePaulSWEng

1

2D Drawing

2

Outline

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

3

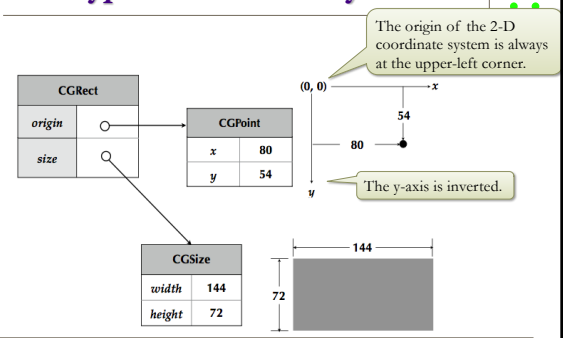
Core Graphics Types

- 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`
  - A point or position in 2D space: { x , y }
- `CGSize`
  - The dimensions: { width , height }
- `CGRect`
  - The location and dimension of a rectangle: { origin , size }

A `CGPoint` and a `CGSize`

4

CG Types and Geometry



5

Geometry and CG Types

- CGPoint

```
struct CGPoint {
    var x: CGFloat
    var y: CGFloat
    init(x: CGFloat, y: CGFloat)
    init(x: Int, y: Int)
    init(x: Double, y: Double)
}
```

- Create CGPoint

```
var p1 = CGPoint(x: 10, y: 200)
var p2 = CGPoint(x: 10.5, y: 200.3);
p1.x = 300.0
p1.y = 30.0
```


6

Geometry and CG Types

- CGSize

```
struct CGSize {
    var width: CGFloat
    var height: CGFloat
    init(width: CGFloat, height: CGFloat)
    init(width: Int, height: Int)
    init(width: Double, height: Double)
}
```

- Create CGSize

```
var s1 = CGSize(width: 10, height: 20)
var s2 = CGSize(width: 10.5, height: 20.8)
s1.width = 100.0
s1.height = 72.0
```

DePaul University 7

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)
}
```

Also Int and Double.

- Create CGRect

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

DePaul University 8

8

Core Graphics & Quartz 2D

9

Custom View Classes

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

```
func draw(_ rect: CGRect)
```

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

DePaul University 10

10

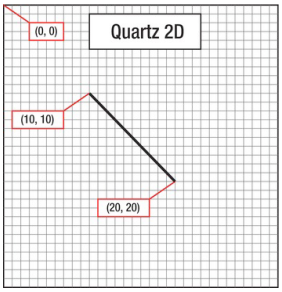
When Is Okay to Call draw?

- The method `draw` 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 `draw` method
- When a view needs to be redrawn, call the method `setNeedsDisplay()`

DePaul University 11

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

12

Core Graphics and Quartz 2D

- UIKit offers some very basic drawing functions

```
func UIRectFill(_ rect: CGRect)
func UIRectFrame(_ rect: CGRect)
```

- Core Graphics
  - A full-scale 2D drawing/graphics library
- Core Graphics and Quartz 2D drawing engine define some simple but powerful graphics primitives
  - Graphics context ■ Transformations ■ Paths
  - Colors ■ Fonts ■ Painting operations

DePaul University 13

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

14

Graphics Context

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

```
func UIGraphicsGetCurrentContext() -> CGContext?
```

- A *CGContext* object is only valid for the current call of `draw`
  - **Do not cache a** *CGContext* object

DePaul University 15

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.red
redColor.set() // drawing will be done in red
```

- `UIFont`
  - Access system fonts. Get font by name

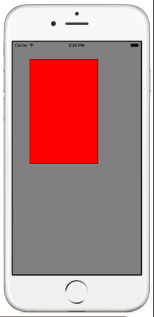
```
let font = UIFont.systemFont(ofSize: 17)
myLabel.font = font
```

DePaul University 16

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

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 *Identity Inspector* to change the class to *MyView*

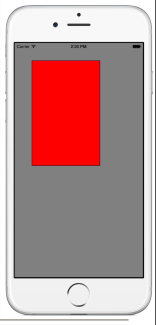
DePaul University 18

18

The Custom View Class for Simple Drawing

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

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



DePaul University 19

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

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

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

22

Building Blocks

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

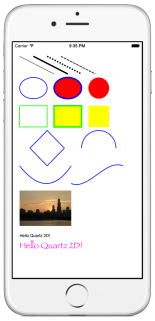
Closed shapes
- Ellipses
- Rectangles

DePaul University 23

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

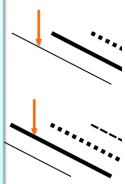
24

Quartz Demo – Draw Solid Lines

```
override func draw(_ rect: CGRect) {
  if let context = UIGraphicsGetCurrentContext() {

    context.move(to: CGPoint(x: 20, y: 40))
    context.addLine(to: CGPoint(x: 120, y: 90))
    context.strokePath()

    context.setLineWidth(4)
    context.move(to: CGPoint(x: 60, y: 40))
    context.addLine(to: CGPoint(x: 160, y: 90))
    context.strokePath()
    …
  }
}
```



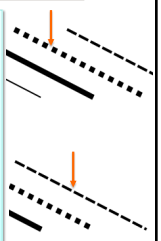
DePaul University 25

25

Quartz Demo – Draw Dashed Lines

```
let shortDash : [CGFloat] = [ 4, 4 ]
context.setLineDash(phase: 0,
                    lengths: shortDash)
context.move(to: CGPoint(x: 100, y: 40))
context.addLine(to: CGPoint(x: 200, y: 90))
context.strokePath()

context.setLineWidth(2)
let longDash : [CGFloat] = [ 8, 2 ]
context.setLineDash(phase:0,
                    lengths: longDash)
context.move(to: CGPoint(x: 140, y: 40))
context.addLine(to: CGPoint(x: 240, y: 90))
context.strokePath()
```



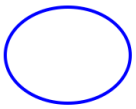
DePaul University 26

26

Quartz Demo – Draw Ovals and Circles

```
context.setStrokeColor(UIColor.blue.cgColor)
context.setFillColor(UIColor.red.cgColor)
context.setLineDash(phase:0, lengths: [])

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



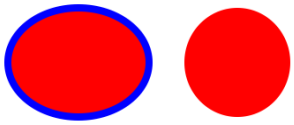
DePaul University 27

27

Quartz Demo – Draw Ovals and Circles

```
context.setLineWidth(4)
let rect2 = CGRect(x: 120, y: 100, width: 80, height: 60)
context.fillEllipse(in: rect2)
context.strokeEllipse(in: rect2)

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



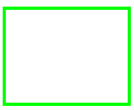
DePaul University 28

28

Quartz Demo – Draw Rectangles and Squares

```
context.setStrokeColor(UIColor.green.cgColor)
context.setFillColor(UIColor.yellow.cgColor)
context.setLineWidth(2)

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



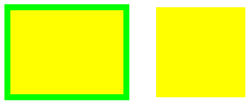
DePaul University 29

29

Quartz Demo – Draw Rectangles and Squares

```
context.setLineWidth(4)
let rect5 = CGRect(x: 120, y: 180, width: 80, height: 60)
context.fill(rect5)
context.stroke(rect5)

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



DePaul University 30

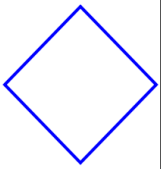
30

Quartz Demo – Draw Paths

- Draw a diamond by tracing the edges

```
context.setStrokeColor(UIColor.blue.cgColor)
context.setLineWidth(2)

context.move(to: CGPoint(x: 100, y: 250))
context.addLine(to: CGPoint(x: 150, y: 300))
context.addLine(to: CGPoint(x: 100, y: 350))
context.addLine(to: CGPoint(x: 50, y: 300))
context.closePath()
context.strokePath()
```



DePaul University 31

31

Quartz Demo – Draw Arcs

```
context.addArc(
    center: CGPoint(x: 250, y: 300),
    radius: 50,
    startAngle: 0,
    endAngle: CGFloat(M_PI),
    clockwise: true)
context.strokePath()
```

DePaul University 32

32

Quartz Demo – Draw Bézier Curves

```
context.move(to: CGPoint(x: 20, y: 350))
context.addQuadCurve(to: CGPoint(x: 170, y: 350),
    control: CGPoint(x: 100, y: 450))
context.strokePath()

context.move(to: CGPoint(x: 170, y: 400))
context.addCurve(to: CGPoint(x: 320, y: 350),
    control1: CGPoint(x: 220, y: 420),
    control2: CGPoint(x: 270, y: 330))
context.strokePath()
```

A quadratic Bézier curve.

A cubic Bézier curve.

DePaul University 33

33

Quartz Demo – Draw Images

```
if let image = UIImage(named: "Chicago") {
    let rect = CGRect(x: 20, y: 420, width: 150, height: 100)
    image.draw(in: rect)
}
```

```
image.draw(at: CGPoint(x: 20, y: 420))
```

Draw the image at the full size.

DePaul University 34

34

Quartz Demo – Draw Text

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

let textAttr : [NSAttributedString.Key : Any] = [
    .foregroundColor : UIColor.magenta,
    .font : UIFont(name: "Papyrus", size: 24)!
]
text.draw(at: CGPoint(x: 20, y: 560),
          withAttributes: textAttr)
```

The top-left corner of the text

Hello Quartz 2D!

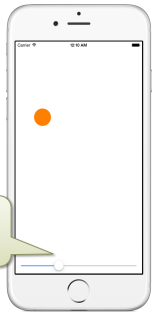
Hello Quartz 2D!

DePaul University 35

35

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

DePaul University 36

36

The Canvas View Class – Draw the Ball at the Current Position

```
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 draw(_ rect: CGRect) {
        if let context = UIGraphicsGetCurrentContext() {
            context.setFillColor(UIColor.orange.cgColor)
            let rect = CGRect(x: x - r, y: y - r,
                              width: 2 * r, height: 2 * r)
            context.fillEllipse(in: rect)
        }
    }
    …
}
```


37

The Canvas View Class – Initialize and Update the Ball Position

```
func start() {
    x = bounds.width / 2
    y = bounds.height / 2
}

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

Set the initial ball position

Update the ball position

But, the ball is not moving!

38

The Canvas View Class – Draw the Ball at the Current Position

```
class CanvasView: UIView {
    …
    var done = false
    override func draw(_ rect: CGRect) {
        if let context = UIGraphicsGetCurrentContext() {
            context.setFillColor(UIColor.orange.cgColor)
            let rect = CGRect(x: x - r, y: y - r,
                              width: 2 * r, height: 2 * r)
            context.fillEllipse(in: rect)
        }
        if !done {
            DispatchQueue.main.asyncAfter(deadline:
                                .now() + .milliseconds(15)) {
                self.update()
            }
        }
    }
}
```

Call the update method after 15 milliseconds delay

39

The Canvas View Class – Update the Ball Position and Redraw

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

```
func update() {
    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()
}
```

Request the canvas be redrawn. Never call `draw` directly

40

The Canvas View Class – Start and Stop the Animation

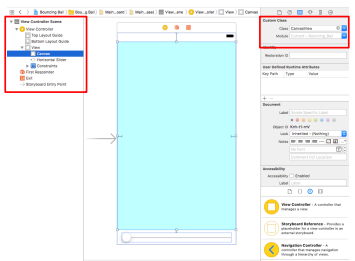
```
func start() {
    x = bounds.width / 2
    y = bounds.height / 2
    done = false
    self.setNeedsDisplay()
}

func stop() {
    done = true
}
```


41

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*



42

The View Controller Class

```
class ViewController: UIViewController {

    @IBOutlet weak var canvas: CanvasView!

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

    override func viewWillDisappear(_ animated: Bool) {
        canvas.stop()
    }

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

Kick off the animation

DePaul University 43

43

Bouncing Ball App – Using a Timer

- An alternative method for animation
- The `Timer` class
  - A timer waits until a specified interval has elapsed and then fires
    - A timer can fire once only or fire repeatedly
  - When a timer fires, it invokes a block of code provided as a closure

DePaul University 44

44

The Canvas View Class – Draw the Ball at the Current Position

```
class CanvasView : UIView {
    var x: CGFloat = 0, y: CGFloat = 0, r: CGFloat = 25
    var velocity: CGFloat = 1
    var dx: CGFloat = 1, dy: CGFloat = 1
    var timer : Timer?
    override func draw(_ rect: CGRect) {
        if let context = UIGraphicsGetCurrentContext() {
            context.setFillColor(UIColor.orange.cgColor)
            let rect = CGRect(x: x – r, y: y – r,
                              width: 2 * r, height: 2 * r)
            context.fillEllipse(in: rect)
        }
    }
    …
}
```

Draw the ball at the current position

DePaul University 45

45

The Canvas View Class – Start Animation

```
func start() {
    timer = Timer.scheduledTimer(timeInterval: 1.0/60.0,
                                 repeats: true) { _ in
        self.update()
    }
    x = bounds.width / 2
    y = bounds.height / 2
}

func stop() {
    timer?.invalidate()
}
```

Call this closure each time the timer fires.

A repeating timer.

DePaul University 46

46

The Canvas View Class – Update the Ball Position

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

```
func update() {
    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 `draw` directly

DePaul University 47

47

The View Controller Class

```
class ViewController: UIViewController {

    @IBOutlet weak var canvas: CanvasView!

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

    override func viewWillDisappear(_ animated: Bool) {
        canvas.stop()
    }

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

Kick off the animation

DePaul University 48

48

Sample Code

- Drawing Demo – SB.zip
- Quartz Demo – SB.zip
- Bouncing Ball – SB.zip
- Bouncing Ball – Timer – SB.zip

49

Next …

- Touch events
- Gestures & gesture recognizers

✧ iOS is a trademark of Apple Inc.

50