



Graficos con Quartz

Yolanda Martínez Treviño

Quartz

- + El framework de Core Graphics es un API basado en C y en el engine de dibujo Quartz.
- + Quartz es un engine para hacer gráficos en 2D
- + La guía de programación de Quartz se encuentra en:

https://developer.apple.com/library/tvos/documentation/GraphicsImaging/Conceptual/drawingwithquartz2d/Introduction/Introduction.html#/apple_ref/doc/uid/TP30001066

Quartz

- + Quartz usa el modelo del pintor, esto significa que cada vez que se agrega algo a la vista, esto se coloca encima de lo que ya estaba. Lo que dibujes en la pantalla se mostrará en el orden en el que lo hagas.
- + Cuando se usa Quartz se agrega el código de dibujo a la vista (**UIView**) en la que se quiere dibujar.

La clase UIView

- + El código de dibujo se coloca en el método **draw**.
- + El método **draw** se llama cada vez que una vista se muestra por primera vez o se necesita redibujar.
- + Al crear la clase hija de **UIView**, se debe implementar el método **draw** que tiene el siguiente encabezado:

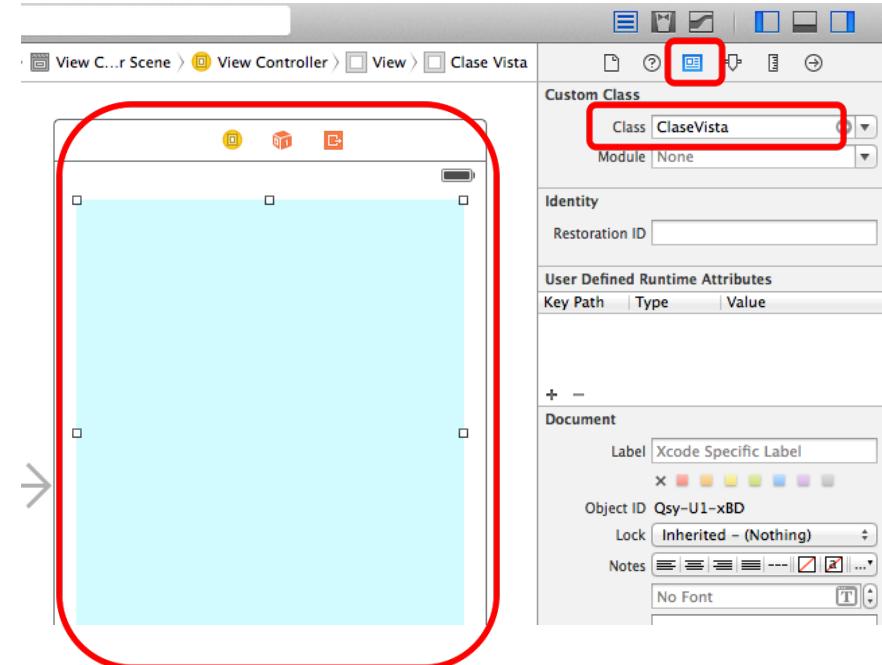
```
override func draw(_ rect: CGRect)
```

El método draw

- + El método **draw** es responsable de dibujar el contenido en el **contexto gráfico** de la vista. Esto crea una representación visual que se mostrará en la vista.
- + No es recomendable implementar el método **draw** y dejarlo vacío, si no se requiere, no se debe implementar.
- + Cuando el contenido de la vista cambia, es responsabilidad del programador notificar que la vista necesita ser redibujada llamando al método **setNeedsDisplay**.
- + El método **setNeedsDisplay** se ejecutará en el siguiente ciclo de dibujo.

Usando la subclase de UIView

- + Crea una subclase de la clase **UIView**.
- + En el storyboard agrega un objeto de tipo **UIView** y en el identity Inspector cambia su clase a la clase que creaste.



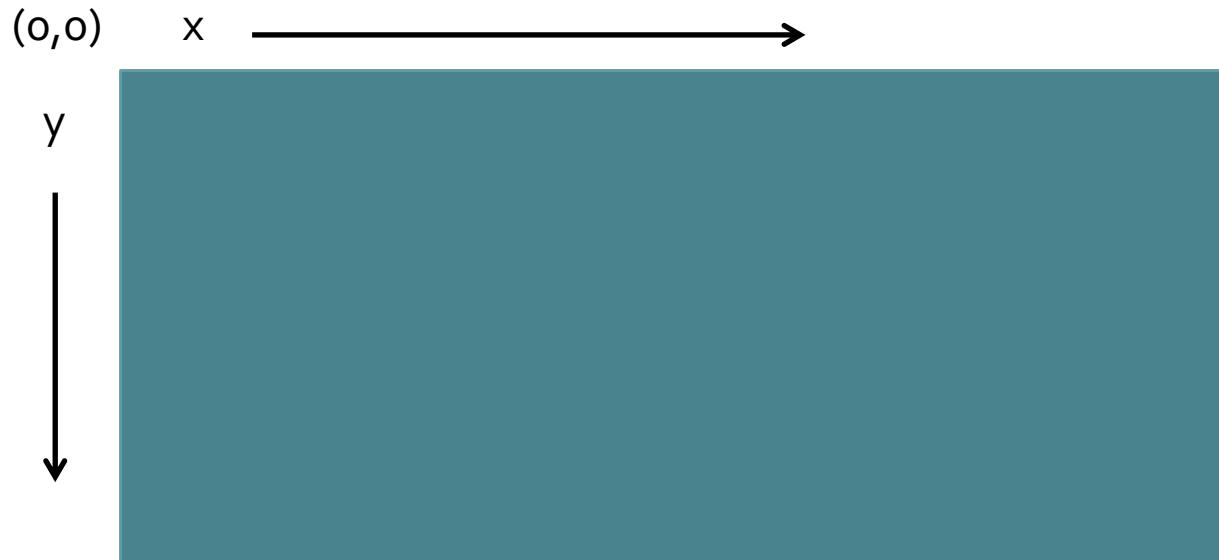
El contexto

- + Al usar Quartz se dibuja en un contexto gráfico, usualmente llamado solamente el contexto.
- + Cada vista tiene un contexto asociado. Cuando quieres dibujar en una vista, tienes que obtener su contexto.
- + Para obtener el contexto usas:

```
let contexto = UIGraphicsGetCurrentContext()!
```

El sistema de coordenadas

- + El contexto que obtenemos de un **UIView** usa el siguiente sistema de coordenadas:



Colores

- + UIKit nos da una clase UIColor, pero no se puede usar directamente en Core Graphics.
- + UIColor es un “wrapper” sobre la clase CGColor entonces se puede obtener el color usando la propiedad cgColor de la clase UIColor.

Por ejemplo:

```
let color = UIColor.yellow.cgColor
```

Colores

- + Los colores se representan con 4 valores de tipo **CGFloat** que guardan valores entre 0 y 1
 - + Red
 - + Green
 - + Blue
 - + Alfa – nivel de transparencia donde 1 es 100% opaco
- + Puedes crear el color que necesites usando:

```
let color = UIColor(red: 120.0/255.0, green: 190.0/255.0, blue: 255/255.0, alpha: 0.5)
```

Colores

- + Propiedades para obtener colores predefinidos:

```
let color = UIColor.  
    UIColor black  
    UIColor blue  
    UIColor brown  
    UIColor clear  
with: NSZone?) -> Any copy(self: UIColor)  
    UIColor cyan  
    UIColor darkGray  
    UIColor darkText
```

Color con transparencia

- + Si tienes un color sólido y lo quieres obtener con transparencia puedes usar:

```
let colorConTransparencia =  
    UIColor.magenta.cgColor.copy(alpha: 0.5)
```

Antes de dibujar

- + Para dibujar figuras necesitas definir
- + El grosor de la línea

`contexto.setLineWidth(3.0)`

- + El color de la línea:

`contexto.setStrokeColor(unColor.cgColor)`

El color de relleno

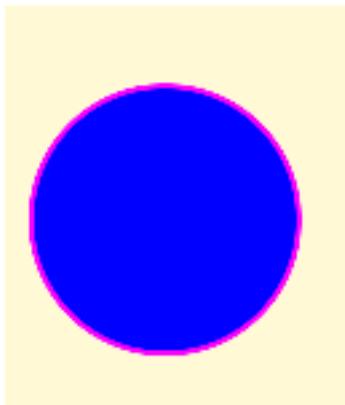
`contexto.setFillColor(otroColor.cgColor)`

Estos valores permanecen “activos” mientras no se vuelva a utilizar el método para cambiar dicho valor.

Dibujar algunas figuras

Elipse

```
+ let theRect : CGRect =  
    CGRect(x: 10, y: 30, width: 100, height: 100)  
contexto.fillEllipse(in: theRect)  
contexto.strokeEllipse(in: theRect)
```



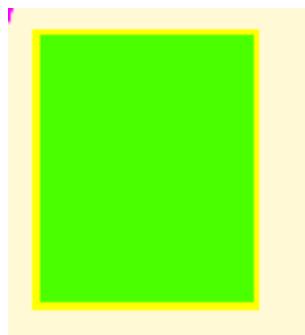
Usa Stroke para dibujar la orilla

Usa Fill para dibujar el relleno

Dibujar algunas figuras

Rectángulo

```
let aRect : CGRect = CGRect(x:120, y:100, width:80,  
height:100)  
contexto.stroke(aRect)  
contexto.fill(aRect)
```



Dibujar rutas (Paths)

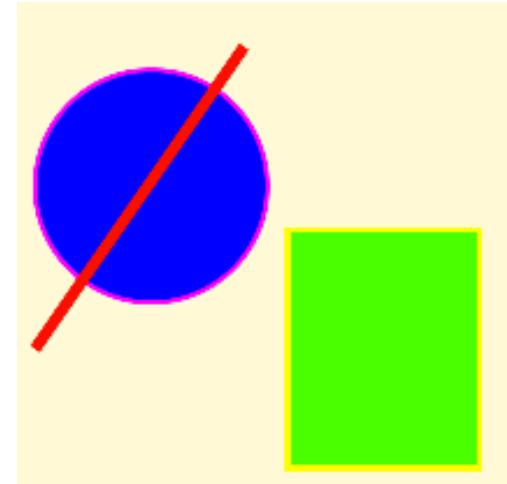
- + Para dibujar una ruta primero debes crearla
- + Después pides a quartz que la muestre, puedes
 - + dibujar el path (stroke path)
 - + llenar el path (fill path)

Métodos para dibujar rutas

- + Para dibujar una línea

```
// primero crear la ruta  
contexto.move(to: CGPoint(x: 10.0, y: 150.0))  
contexto.addLine(to: CGPoint(x: 100.0, y:20.0))
```

```
// luego dibujarla  
contexto.strokePath()
```



Polígonos

// primero dibujar la ruta

```
contexto.move(to: CGPoint(x: ___, y: ____))
```

```
contexto.addLine(to: CGPoint(x: ___, y:____))
```

```
contexto.addLine(to: CGPoint(x: ___, y: ____))
```

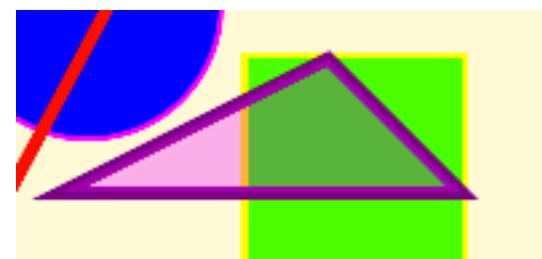
...

// luego llenar o dibujar la ruta

```
contexto.closePath()
```

```
contexto.fillPath()
```

```
contexto.strokePath()
```



Una vez que dibujas una ruta (relleno u orilla) ésta se elimina del contexto.

Capas (layers)

- + Puedes obtener una capa (layer) del contexto usando:

```
let capa = CGLayer(contexto, size: self.frame.size,  
auxiliaryInfo: nil)
```

Self se refiere a la vista (estamos en la clase UIView).

- + Para dibujar requieres el contexto del layer:

```
let contextoCapa = capa!.context!
```

Dibuja en la capa de la misma forma que se hace en el contexto de la vista. La diferencia es que no se dibuja inmediatamente, sino hasta que la mandas dibujar.

Capas

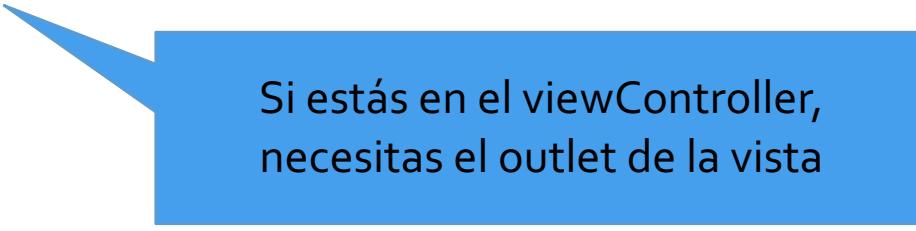
- + Una vez que tengas la capa completa muestralá en la vista usando:

```
contexto.draw(capa!, in: self.frame)
```

Redibujar una vista

- + Si modificas el contenido de una vista puedes indicar que se haga el repintado utilizando:

```
vista.setNeedsDisplay()
```



Si estás en el viewController,
necesitas el outlet de la vista

Gradiente

- + Es un efecto de que pasa de un color a otro
- + Hay diferentes tipos, el más sencillo es un gradiente lineal como este



Quartz 2D Programming Guide

Table of Contents

- Introduction
- Overview of Quartz 2D
- Graphics Contexts
- Paths
- Color and Color Spaces
- Transforms
- Patterns
- Shadows
- Gradients
- Transparency Layers

Overview of Quartz 2D

Quartz 2D is a two-dimensional drawing engine accessible in environments outside of the kernel. You can use the Quartz 2D API to features such as path-based drawing, painting with transparency, color management, anti-aliased rendering, PDF document generation, and more. Quartz 2D leverages the power of the graphics hardware.

In Mac OS X, Quartz 2D can work with all other graphics and imaging frameworks in Mac OS X, including QuickTime. It's possible to create an image in Quartz and then save it as a QuickTime movie.

Crear un gradiente

Con estos pasos creas un gradiente

```
let contexto = UIGraphicsGetCurrentContext()!

//crea el arreglo de colores para el gradiente
let startColor = UIColor.blue
let endColor = UIColor.green
let colores = [startColor.cgColor, endColor.cgColor]

// establecer el espacio de color
let espacioDeColor = CGColorSpaceCreateDeviceRGB()

// establece los lugares en los que cambia el color
let localizacionColores :[CGFloat] = [0.0, 1.0]

// crea el gradiente
let gradiente = CGGradient(colorsSpace: espacioDeColor,
                            colors: colores as CFArray, locations: localizacionColores)
```

Continua...

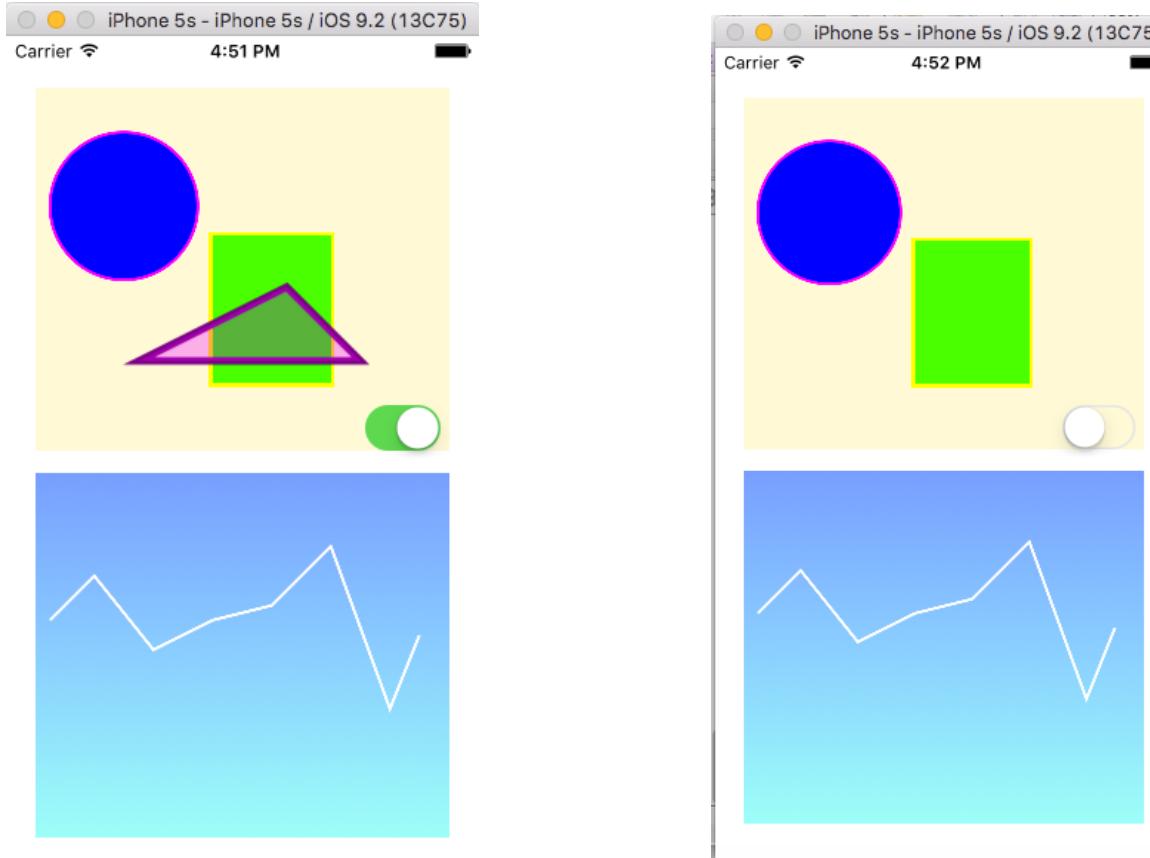
Crear un gradiente

+ Continuación...

```
//dibuja el gradiente
let puntoInicio = CGPoint.zero
let puntoFin = CGPoint(x:0, y:self.bounds.height)
contexto.drawLinearGradient(gradiente!,
    start: puntoInicio,
    end: puntoFin, options:
CGGradientDrawingOptions.drawsAfterEndLocation)
```

Ejercicio

ve la descripción en la siguiente filmina



Ejercicio

- + Haz 2 objetos de tipo UIView
 - + En el de arriba agrega un rectangulo y un elipse, que tengan diferente color de orilla y de relleno.
 - + Agrega una capa y en ella coloca un polígono con un color de orilla y otro de relleno, el color de relleno debe tener transparencia y la figura debe quedar arriba de alguna de las otras para que se note la transparencia.
 - + Agrega un switch, cuando esté **off**, no muestres la capa, cuando esté **on** muéstralala.
 - + En el view de abajo agrega un gradiente y agrega una ruta (path) sin cerrar que contenga al menos 4 puntos.

Rúbrica para la evaluación del ejercicio

Puntos	Descripción
10	Rectangulo con color diferente de orilla y de relleno
10	Circulo con color diferente de orilla y de relleno
15	Triangulo con color transparente de relleno y arriba de otra figura
15	El triangulo tiene un color diferente de relleno y otro de orilla
25	Switch que al poner en on se ve el triangulo y al poner en off no se ve el triangulo
10	Un segundo view con un gradiente de color de fondo
15	En el segundo view hay una ruta sin cerrar que tiene al menos 4 puntos