Shadows UIKit

Move iT!

Versión del Documento: 1.0

Autor: Rodolfo Gonzalez Hernandez

Cargo: Developer iOS

Colaboradores: Felipe Carrasco Galindo
Mauricio Caro

Fecha de Emisión: 10 de Enero de 2024

Shadows UIKit

Version	Descripción del cambio
V 1.0	Version inicial de la documentación

Objetivo

El objetivo principal de establecer y documentar la implementación de sombras a través de la creacion de capas con UIKit, para hacer un diseño y estilo personalizado, junto con Swift y UIKit dentro del desarrollo y diseño de aplicaciones iOS. Esta documentación se centra en fomentar la eficiencia, la entrega de software de alta calidad, definir y aplicar TECNICAS de codigo Swift e interfaz con UIKit. A través de la implementación de sombras, capas y componentes gráficos, buscamos alcanzar los siguientes propósitos clave:

1.

Promover la Colaboración Efectiva:

- Facilitar un entorno de desarrollo en el cual los miembros del equipo puedan colaborar de manera armoniosa, permitiendo una integración continua de cambios y un desarrollo de UI fluido y prolijo entre los desarrolladores y su codigo Swift.

2.

Consistencia de estilo:

- Asegurarse que el codigo siga un estilo de diseño y UI consistente.

3.

Prevenir malas practicas de diseño:

- Evita la adopción de malas practicas de diseño, al estar definida y parametrizada la configuración de sombras.

4.

Mantenimiento consistente del Diseño

- Contribuye a un mantenimiento mas consistente del diseño a lo largo del tiempo.

5.

Aplicar efecto Neomorfismo a través de Shadows

- Ya comprendido el concepto de sombras en UIView de UIKit es que aplicaremos un estilo neomorfismo a nuestros componentes UIVIew, a través de sombras capas y componentes graficos.
- Que en el diseño se logre apreciar una distinción de profundidades negativas y profundidades positivas en los componentes graficos, gracias a las sombras creadas

Flujo trabajo Shadows UIKit

El flujo de trabajo propuesto para la implementación de sombras en el diseño y desarrollo de aplicaciones iOS establece un conjunto de prácticas y procedimientos estructurados que guían el diseño del software, desde la concepción de nuevos estilos, hasta su despliegue , siempre enfocado en estrictas reglas y buenas practicas, con el fin de conseguir un diseño y codigo Swift prolijo y desarrollo profesional. Durante todo el transcurso del desarrollo es que se busca normar o unificar criterios en cuanto a estilo y diseño de la App. iOS. Desempeñando un papel primordial en el proceso de desarrollo hasta el producto final.

A la hora de aplicar sombras a nuestro proyecto podemos optar tanto como sombras interiores, como sombras exteriores.

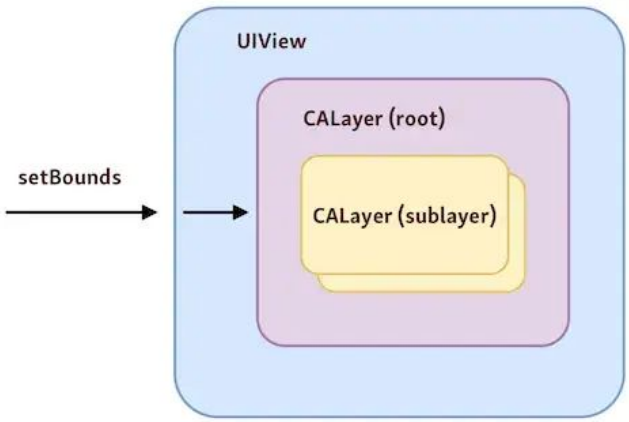
Documentación extra

Creating inner shadows in UIView to replicate Neumorphic Style

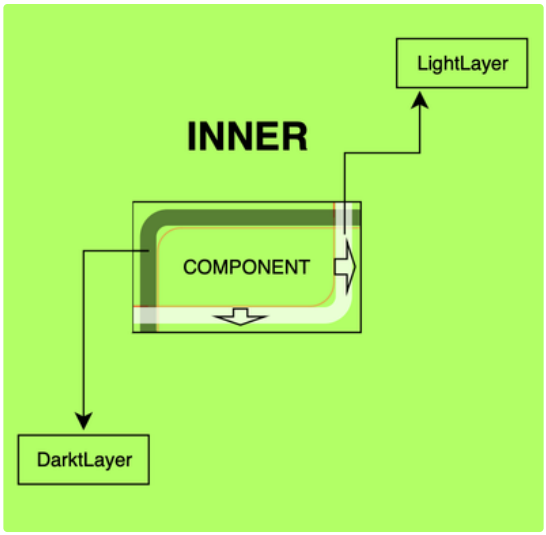
GitHub - noblakit01/SwiftyShadow: Helper Shadow Path for UIView in Swift iOS

GitHub - Rodolfo-Swift-dev/INNER-OVERshadow: implementación de sombras a través de la creacion de capas con UIKit, para hacer un diseño y estilo personalizado, junto con Swift y UIKit dentro del desarrollo y diseño de aplicaciones iOS

Vista de ORDEN de capas añadidas a super capa de una UIView



WorkFlow Inner Shadow



1. **Creacion de DarkShadowLayer**

Creamos una capa a la cual le damos valor a su propiedad frame y cornerRadius las cuales corresponderán a valores captados de la Supercapa, a la cual mas tarde agregaremos esta capa recién creada.

```
let topShadow = CALayer()
topShadow.frame = self.layer.bounds
topShadow.cornerRadius = self.layer.cornerRadius
```

2. **Creacion de UIBezierPath**

Este código se utiliza para definir una forma compleja mediante 2 rutas. Una ruta me indica el contorno de la super capa de la UIView. La otra ruta me indica el contorno o figura producto del `insetBy(dx: , dy:)`, el cual sera directamente las sombras para nuestro aplicativo.

UiBezierPath se utilice en el contexto de gráficos o interfaz de usuario en una aplicación iOS.

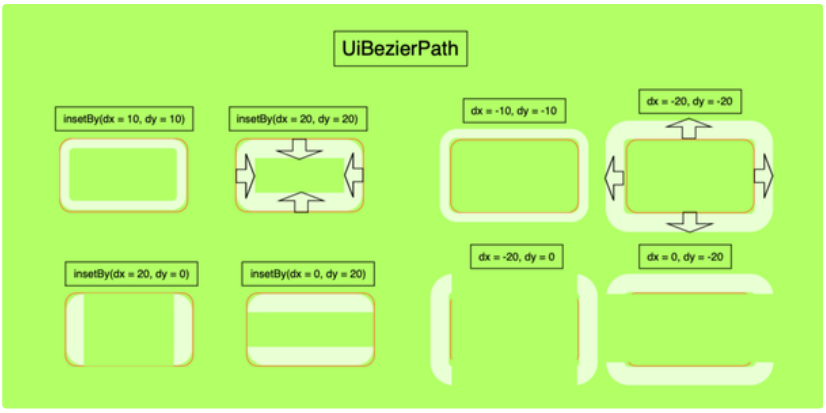
```
let scaleFactorX: CGFloat = self.bounds.width * -0.2
let scaleFactorY: CGFloat = self.bounds.height * -0.2
let path = UIBezierPath(roundedRect: topShadow.bounds.insetBy(dx: scaleFactorX, dy: scaleFactorY), cornerRadius:
let cutout = UIBezierPath(roundedRect: topShadow.bounds, cornerRadius: radius).reversing()
path.append(cutout)
topShadow.shadowPath = path.cgPath
```

Por ultimo añadimos nuestras rutas a nuestra capa(Layer) anteriormente creada en el paso 1.

Contexto UiBezierPath

Signo - hacia afuera de los bordes

Signo + hacia dentro de los bordes



3. **Configuración de DarkShadowLayer**

Configuramos la capa a la cual le damos valores a sus propiedades , a la cual mas tarde agregaremos esta capa recién creada.

Cabe señalar que la finalidad de este ejemplo es crear una capa iluminada que representa a una sombra de una de las esquinas del componente UIView

```
topShadow.shadowColor = UIColor.lightGray.cgColor
topShadow.shadowOffset = CGSize(width: 5, height: 5)
topShadow.shadowOpacity = darkShadowFactor
topShadow.shadowRadius = 5
topShadow.cornerRadius = radius
self.layer.addSublayer(topShadow)
```

Por ultimo añadimos nuestra capa(Layer) ya configurada a nuestra super capa de la UIView.

Contexto ShadowOffset

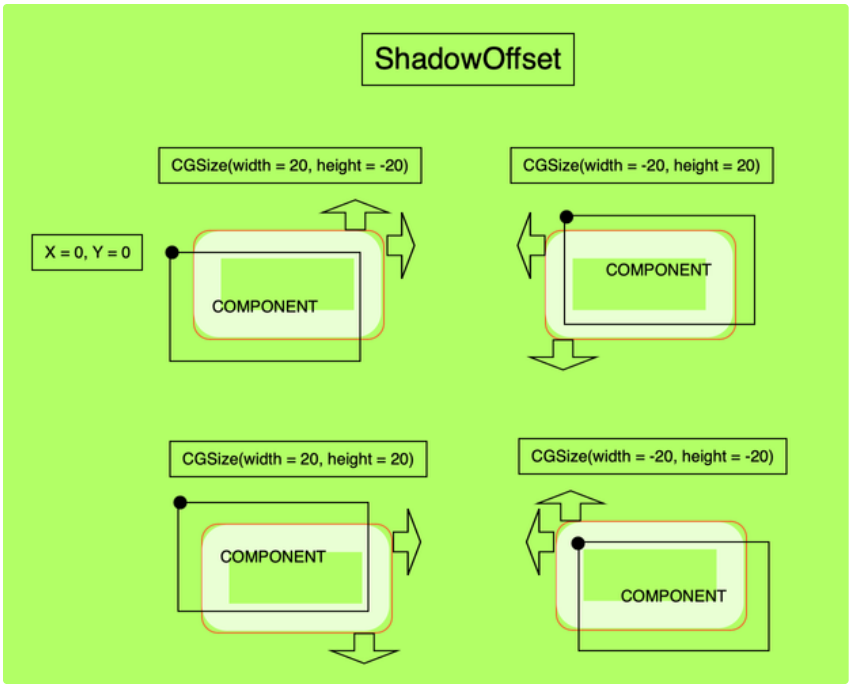
Cabe señalar que nuestro origen sera en relación a la super capa de la UIView, de la cual tomamos su frame en el paso 1.

Signo - eje X = hacia izquierda en el eje X en relación a su punto de origen.

Signo + eje X= hacia derecha en el eje X en relación a su punto de origen.

Signo - eje Y = hacia arriba en el eje Y en relación a su punto de origen.

Signo + eje Y= hacia abajo en el eje Y en relación a su punto de origen.



4. **Creacion de LightkShadowLayer**

Creamos una capa a la cual le damos valor a su propiedad frame y cornerRadius las cuales corresponderán a valores captados de la Supercapa, a la cual mas tarde agregaremos esta capa recién creada.

```
let bottomShadow = CALayer()
bottomShadow.frame = self.layer.bounds
bottomShadow.cornerRadius = self.layer.cornerRadius
```

5. **Creacion de UIBezierPath**

Este código se utiliza para definir una forma compleja mediante 2 rutas. Una ruta me indica el contorno de la super capa de la UIView. La otra ruta me indica el contorno o figura producto del `insetBy(dx: , dy:)`, el cual sera directamente las sombras para nuestro aplicativo.

Para este paso necesitamos acceder a los scaleFactorX e scaleFactorY creados en el paso 2.

UiBezierPath se utilice en el contexto de gráficos o interfaz de usuario en una aplicación iOS.

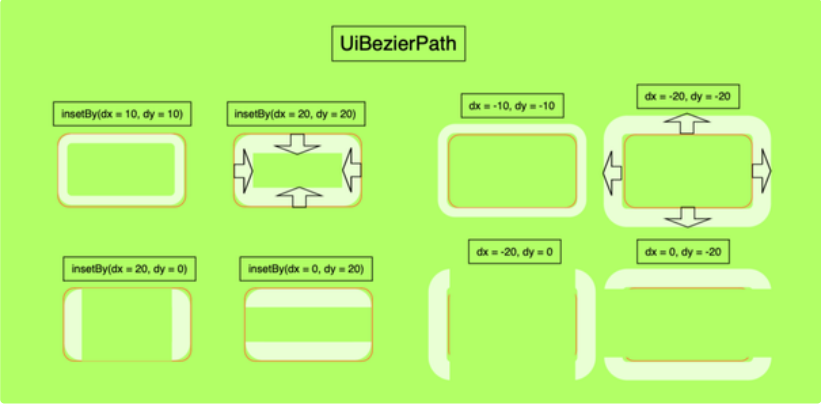
```
let path2 = UIBezierPath(roundedRect: bottomShadow.bounds.insetBy(dx: scaleFactorX, dy: scaleFactorY), cornerRadi
let cutout2 = UIBezierPath(roundedRect: bottomShadow.bounds, cornerRadius: radius).reversing()
path2.append(cutout2)
bottomShadow.shadowPath = path2.cgPath
```

Por ultimo añadimos nuestras rutas a nuestra capa(Layer) anteriormente creada en el paso 4.

Contexto UiBezierPath

Signo - hacia afuera de los bordes

Signo + hacia dentro de los bordes



6. **Configuración de LightShadowLayer**

Configuramos la capa a la cual le damos valores a sus propiedades , a la cual mas tarde agregaremos esta capa recién creada.

Cabe señalar que la finalidad de este ejemplo es crear una capa iluminada que representa a una sombra de una de las esquinas del componente UIView

```
bottomShadow.shadowColor = UIColor.white.cgColor
bottomShadow.shadowOffset = CGSize(width: -5, height: -5)
bottomShadow.shadowOpacity = lightShadowFactor
bottomShadow.shadowRadius = 5
bottomShadow.cornerRadius = radius
self.layer.addSublayer(bottomShadow)
```

Por ultimo añadimos nuestra capa(Layer) ya configurada a nuestra super capa de la UIView.

Contexto ShadowOffset

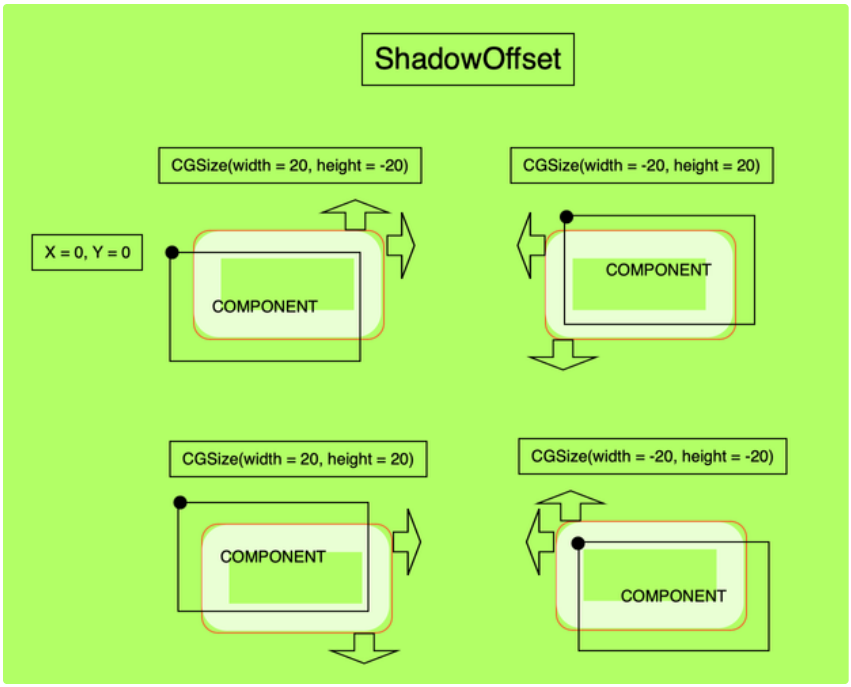
Cabe señalar que nuestro origen sera en relación a la super capa de la UIView, de la cual tomamos su frame en el paso 4.

Signo - eje X = hacia izquierda en el eje X en relación a su punto de origen.

Signo + eje X= hacia derecha en el eje X en relación a su punto de origen.

Signo - eje Y = hacia arriba en el eje Y en relación a su punto de origen.

Signo + eje Y= hacia abajo en el eje Y en relación a su punto de origen.

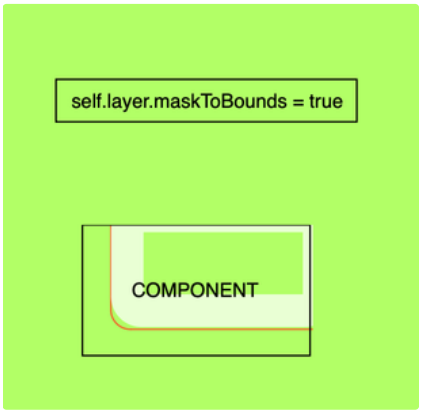


7. **Eliminar vision fuera de los limites**

Para lograr un efecto de sombra interior con efecto de profundidad negativa tenemos que configurar la propiedad maskToBounds y asignarle el valor de true para eliminar la vision de cualquier subcapa fuera de los limites de la UIView.

Cabe señalar que la finalidad de este ejemplo es eliminar las sombras visibles creadas en las layer, que sobresalen de los limites del componente UIView que contiene las capas con sombras.

```
self.layer.masksToBounds = true
```



8. **Mejoras en renderizado de componentes graficos**

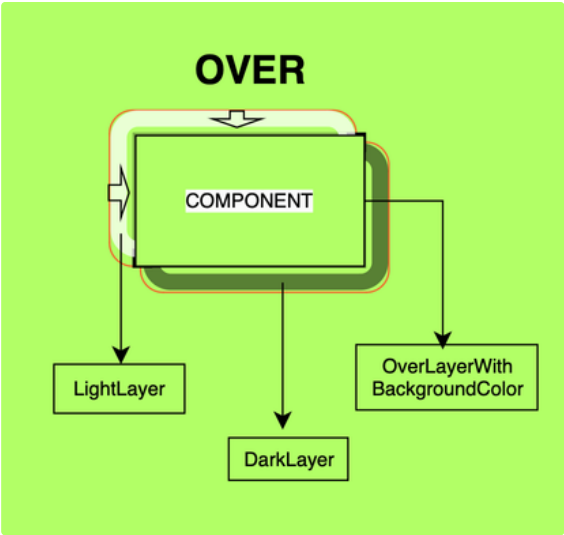
Directamente relacionado con las cache de creacion de componentes graficos es que adoptamos la siguiente configuración en nuestro codigo.

```
self.layer.shouldRasterize = true
self.layer.rasterizationScale = UIScreen.main.scale
```

9. **Recomendaciones**

Tener en consideración el no abusar de la opacidad de Subcapas y Supercapa. ya que a medida que se aplica opacidad a las subcapas el efecto va incrementándose.

WorkFlow Over Shadow



1. **Creacion de DarkShadowLayer**

Creamos una capa a la cual le damos valor a su propiedad frame y cornerRadius las cuales corresponderán a valores captados de la Supercapa, a la cual mas tarde agregaremos esta capa recién creada.

```
let topShadow = CALayer()
topShadow.frame = self.layer.bounds
topShadow.cornerRadius = self.layer.cornerRadius
```

2. **Creacion de UIBezierPath**

Este código se utiliza para definir una forma compleja mediante 2 rutas. Una ruta me indica el contorno de la super capa de la UIView. La otra ruta me indica el contorno o figura producto del `insetBy(dx: , dy:)`, el cual sera directamente las sombras para nuestro aplicativo.

UiBezierPath se utilice en el contexto de gráficos o interfaz de usuario en una aplicación iOS.

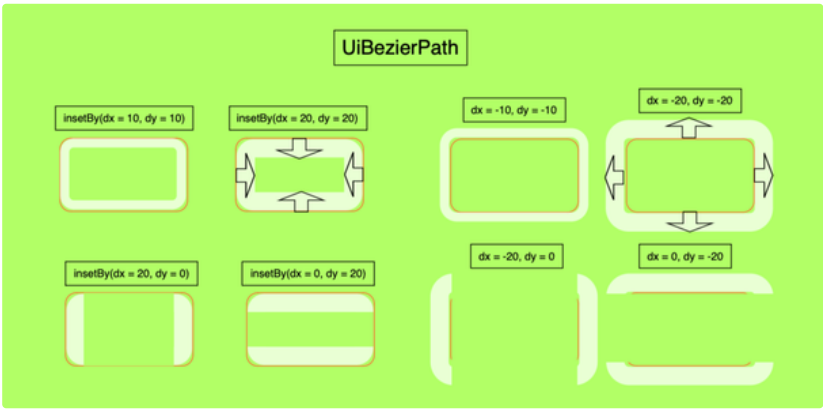
```
let scaleFactorX: CGFloat = self.bounds.width * 0.2
let scaleFactorY: CGFloat = self.bounds.height * 0.2
let path = UIBezierPath(roundedRect: topShadow.bounds.insetBy(dx: scaleFactorX, dy: scaleFactorY), cornerRadius:
let cutout = UIBezierPath(roundedRect: topShadow.bounds, cornerRadius: radius).reversing()
path.append(cutout)
topShadow.shadowPath = path.cgPath

```

Por ultimo añadimos nuestras rutas a nuestra capa(Layer) anteriormente creada en el paso 1.

Contexto UiBezierPath

Signo - hacia afuera de los bordes

Signo + hacia dentro de los bordes



3. **Configuración de DarkShadowLayer**

Configuramos la capa a la cual le damos valores a sus propiedades , a la cual mas tarde agregaremos esta capa recién creada.

Cabe señalar que la finalidad de este ejemplo es crear una capa iluminada que representa a una sombra de una de las esquinas del componente UIView

```
topShadow.shadowColor = UIColor.lightGray.cgColor
topShadow.shadowOffset = CGSize(width: 5, height: 5)
topShadow.shadowOpacity = darkShadowFactor
topShadow.shadowRadius = 5
topShadow.cornerRadius = radius
self.layer.addSublayer(topShadow)
```

Por ultimo añadimos nuestra capa(Layer) ya configurada a nuestra super capa de la UIView.

Contexto ShadowOffset

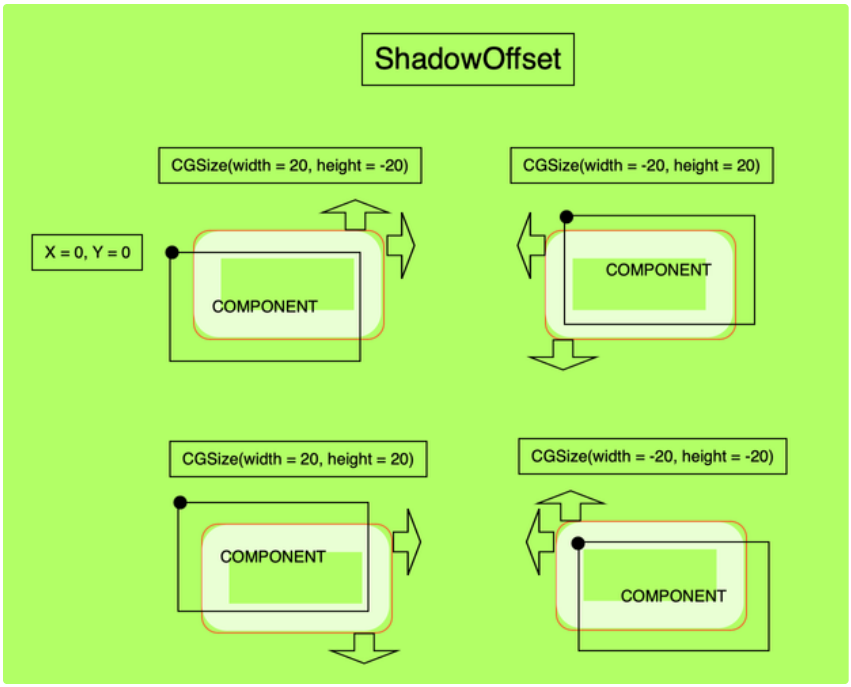
Cabe señalar que nuestro origen sera en relación a la super capa de la UIView, de la cual tomamos su frame en el paso 1.

Signo - eje X = hacia izquierda en el eje X en relación a su punto de origen.

Signo + eje X= hacia derecha en el eje X en relación a su punto de origen.

Signo - eje Y = hacia arriba en el eje Y en relación a su punto de origen.

Signo + eje Y= hacia abajo en el eje Y en relación a su punto de origen.



4. **Creacion de LightkShadowLayer**

Creamos una capa a la cual le damos valor a su propiedad frame y cornerRadius las cuales corresponderán a valores captados de la Supercapa, a la cual mas tarde agregaremos esta capa recién creada.

```
let bottomShadow = CALayer()
bottomShadow.frame = self.layer.bounds
bottomShadow.cornerRadius = self.layer.cornerRadius
```

5. **Creacion de UIBezierPath**

Este código se utiliza para definir una forma compleja mediante 2 rutas. Una ruta me indica el contorno de la super capa de la UIView. La otra ruta me indica el contorno o figura producto del `insetBy(dx: , dy:)`, el cual sera directamente las sombras para nuestro aplicativo.

Para este paso necesitamos acceder a los scaleFactorX e scaleFactorY creados en el paso 2.

UiBezierPath se utilice en el contexto de gráficos o interfaz de usuario en una aplicación iOS.

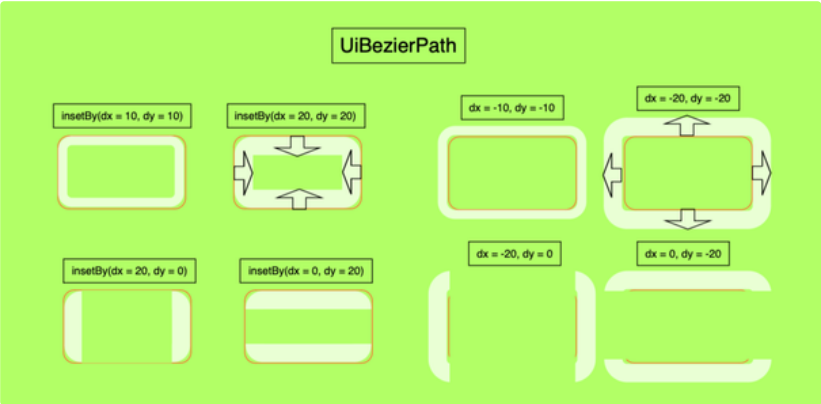
```
let path2 = UIBezierPath(roundedRect: bottomShadow.bounds.insetBy(dx: scaleFactorX, dy: scaleFactorY), cornerRadi
let cutout2 = UIBezierPath(roundedRect: bottomShadow.bounds, cornerRadius: radius).reversing()
path2.append(cutout2)
bottomShadow.shadowPath = path2.cgPath

```

Por ultimo añadimos nuestras rutas a nuestra capa(Layer) anteriormente creada en el paso 4.

Contexto UiBezierPath

Signo - hacia afuera de los bordes

Signo + hacia dentro de los bordes



6. **Configuración de LightShadowLayer**

Configuramos la capa a la cual le damos valores a sus propiedades , a la cual mas tarde agregaremos esta capa recién creada.

Cabe señalar que la finalidad de este ejemplo es crear una capa iluminada que representa a una sombra de una de las esquinas del componente UIView

```
bottomShadow.shadowColor = UIColor.white.cgColor
bottomShadow.shadowOffset = CGSize(width: -5, height: -5)
bottomShadow.shadowOpacity = lightShadowFactor
bottomShadow.shadowRadius = 5
bottomShadow.cornerRadius = radius
self.layer.addSublayer(bottomShadow)
```

Por ultimo añadimos nuestra capa(Layer) ya configurada a nuestra super capa de la UIView.

Contexto ShadowOffset

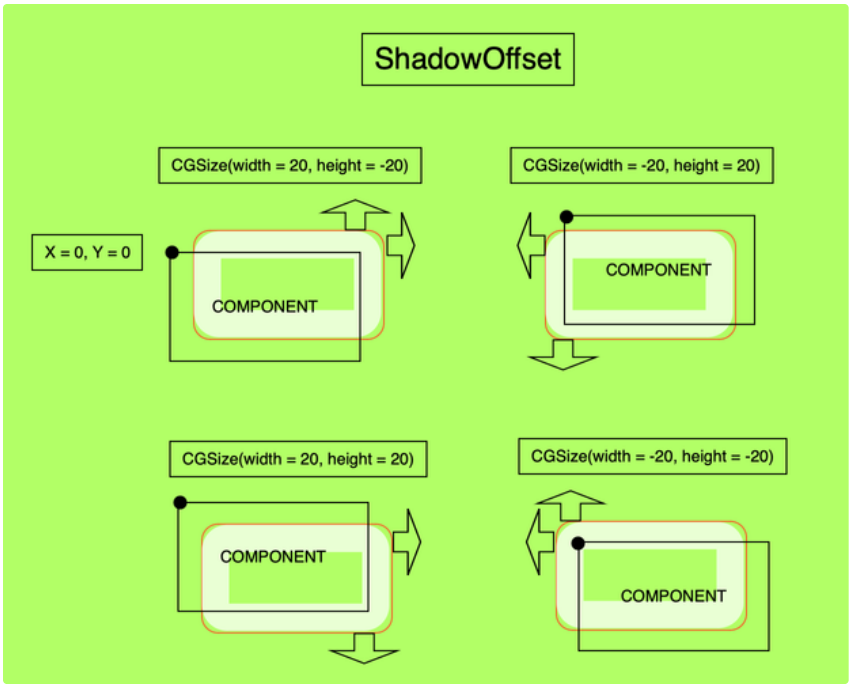
Cabe señalar que nuestro origen sera en relación a la super capa de la UIView, de la cual tomamos su frame en el paso 4.

Signo - eje X = hacia izquierda en el eje X en relación a su punto de origen.

Signo + eje X= hacia derecha en el eje X en relación a su punto de origen.

Signo - eje Y = hacia arriba en el eje Y en relación a su punto de origen.

Signo + eje Y= hacia abajo en el eje Y en relación a su punto de origen.



7. **Creacion de ColorBackgroundLayer**

Creamos una capa a la cual le damos valor a su propiedad frame y cornerRadius las cuales corresponderán a valores captados de la Supercapa, a la cual mas tarde agregaremos esta capa recién creada. Ademas le damos valor a su propiedad backgroundColor para que sea de color, en este caso el del background fuera de la view tratando de aplicar neomorfismo.

Cabe señalar que la finalidad de esta capa recién creada es tapar el contenido de sombras que están dentro del marco de la UIView, dejando visible solo las sombras que sobresalen desde los bordes de la UIView.

```
let contentLayer = CALayer()
contentLayer.frame = self.layer.bounds
contentLayer.cornerRadius = self.layer.cornerRadius
contentLayer.backgroundColor = UIColor(red: 230 / 255, green: 240 / 255, blue: 255 / 255, alpha: 1).cgColor
self.layer.addSublayer(contentLayer)

```

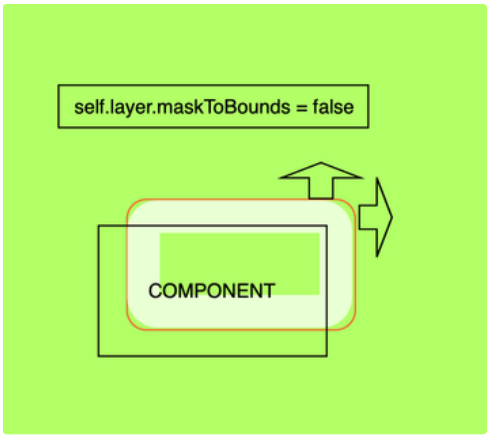
Por ultimo añadimos nuestra capa(Layer) ya configurada a nuestra super capa de la UIView.

8. **Eliminar vision fuera de los limites**

Para lograr un efecto de sombra exterior con efecto de profundidad positiva tenemos que configurar la propiedad maskToBounds y asignarle el valor de false para dejar ver la vision de cualquier subcapa fuera de los limites de la UIView.

Cabe señalar que la finalidad de este ejemplo es dejar a la vista las sombras visibles exteriores creadas en las layer, que sobresalen de los limites del componente UIView que contiene las capas con sombras.

```
self.layer.masksToBounds = false
```



8. **Mejoras en renderizado de componentes graficos**

Directamente relacionado con las cache de creacion de componentes graficos es que adoptamos la siguiente configuración en nuestro codigo.

```
self.layer.shouldRasterize = true
self.layer.rasterizationScale = UIScreen.main.scale
```

9. **Recomendaciones**

Tener en consideración el no abusar de la opacidad de Subcapas y Supercapa. ya que a medida que se aplica opacidad a las subcapas el efecto va incrementándose.