class DrawingViewController: UIViewController {

override func viewDidLoad() {

super.viewDidLoad()

// Create a custom CALayer

let customLayer = CALayer()

customLayer.frame = CGRect(x: 50, y: 50, width: 100, height: 100)

// Custom drawing using CALayer's delegate

customLayer.delegate = self

customLayer.setNeedsDisplay() // Trigger drawing

// Add the CALayer to the view's layer

view.layer.addSublayer(customLayer)

}

}

extension DrawingViewController: CALayerDelegate {

// Implement the drawing logic here

func draw(\_ layer: CALayer, in ctx: CGContext) {

ctx.setFillColor(UIColor.red.cgColor)

ctx.fill(layer.bounds)

}

}

class AnimationViewController: UIViewController {

override func viewDidLoad() {

super.viewDidLoad()

// Create a CALayer

let animatedLayer = CALayer()

animatedLayer.frame = CGRect(x: 50, y: 50, width: 100, height: 100)

animatedLayer.backgroundColor = UIColor.yellow.cgColor

// Add the CALayer to the view's layer

view.layer.addSublayer(animatedLayer)

// Create a basic animation for opacity

let animation = CABasicAnimation(keyPath: "opacity")

animation.fromValue = 1.0

animation.toValue = 0.2

animation.duration = 2.0

// Add the animation to the layer

animatedLayer.add(animation, forKey: "opacityAnimation")

// Update the layer's opacity

animatedLayer.opacity = 0.2

}

}

let animation = CABasicAnimation(keyPath: "position")

animation.fromValue = CGPoint(x: 0, y: 0)

animation.toValue = CGPoint(x: 200, y: 200)

animation.duration = 2.0

myLayer.add(animation, forKey: "positionAnimation")

let keyframeAnimation = CAKeyframeAnimation(keyPath: "position")

keyframeAnimation.values = [CGPoint(x: 50, y: 50),

CGPoint(x: 150, y: 150),

CGPoint(x: 250, y: 50)]

keyframeAnimation.keyTimes = [0, 0.5, 1.0]

keyframeAnimation.duration = 1.5

let animator = UIViewPropertyAnimator(duration: 1.0, curve: .easeInOut) {

myView.center = CGPoint(x: 200, y: 200)

}

animator.startAnimation()

let animator = UIViewPropertyAnimator(duration: 1.0, curve: .easeInOut) {

myView.alpha = 0.0

}

animator.fractionComplete = 0.5 // Adjust animation progress manually

animator.addCompletion { \_ in

myView.removeFromSuperview()

}

animator.startAnimation()

let animator = UIViewPropertyAnimator(duration: 1.0, curve: .easeInOut) {

// Animation code

}

// Pause animation

animator.pauseAnimation()

// Resume animation

animator.continueAnimation(withTimingParameters: nil, durationFactor: 0)

// Reverse animation

animator.isReversed = true

let blurFilter = CIFilter(name: "CIGaussianBlur")

blurFilter?.setValue(10, forKey: "inputRadius")

myLayer.filters = [blurFilter]

func makeEmitterCell(with image: UIImage) -> CAEmitterCell {

let cell = CAEmitterCell()

cell.contents = image.cgImage

cell.birthRate = 5

cell.lifetime = 10

cell.velocity = 50

return cell

}

let emitterLayer = CAEmitterLayer()

emitterLayer.emitterPosition = CGPoint(x: view.bounds.width / 2, y: -50)

emitterLayer.emitterSize = CGSize(width: view.bounds.width, height: 0)

emitterLayer.emitterShape = .line

let snowflake = makeEmitterCell(with: UIImage(named: "snowflake")!)

emitterLayer.emitterCells = [snowflake]

view.layer.addSublayer(emitterLayer)

let path = UIBezierPath()

path.move(to: CGPoint(x: 100, y: 100))

path.addLine(to: CGPoint(x: 200, y: 200))

path.addLine(to: CGPoint(x: 200, y: 100))

path.close()

let context = UIGraphicsGetCurrentContext()

context?.addRect(CGRect(x: 100, y: 100, width: 100, height: 100))

context?.setFillColor(UIColor.red.cgColor)

context?.fillPath()

let text = "Hello, Core Graphics!"

let attributes = [NSAttributedString.Key.font: UIFont.systemFont(ofSize: 20)]

text.draw(at: CGPoint(x: 100, y: 100), withAttributes: attributes)

class CustomView: UIView {

override func draw(\_ rect: CGRect) {

guard let context = UIGraphicsGetCurrentContext() else { return }

// Define the circle

let center = CGPoint(x: bounds.midX, y: bounds.midY)

let radius = min(bounds.width, bounds.height) / 2

let startAngle = CGFloat(0)

let endAngle = CGFloat.pi \* 2

// Create a path

let path = CGMutablePath()

path.addArc(center: center, radius: radius,

startAngle: startAngle, endAngle: endAngle, clockwise: false)

// Define the gradient

let colors = [UIColor.red.cgColor, UIColor.blue.cgColor] as CFArray

let gradient = CGGradient(colorsSpace: CGColorSpaceCreateDeviceRGB(),

colors: colors, locations: [0, 1])!

// Draw the path

context.addPath(path)

context.clip()

context.drawLinearGradient(gradient,

start: CGPoint(x: bounds.minX, y: bounds.minY),

end: CGPoint(x: bounds.maxX, y: bounds.maxY),

options: [])

// Stroke the path

context.addPath(path)

context.setStrokeColor(UIColor.black.cgColor)

context.strokePath()

}

}