

Graphics and Animation

Mobile Application Development in iOS

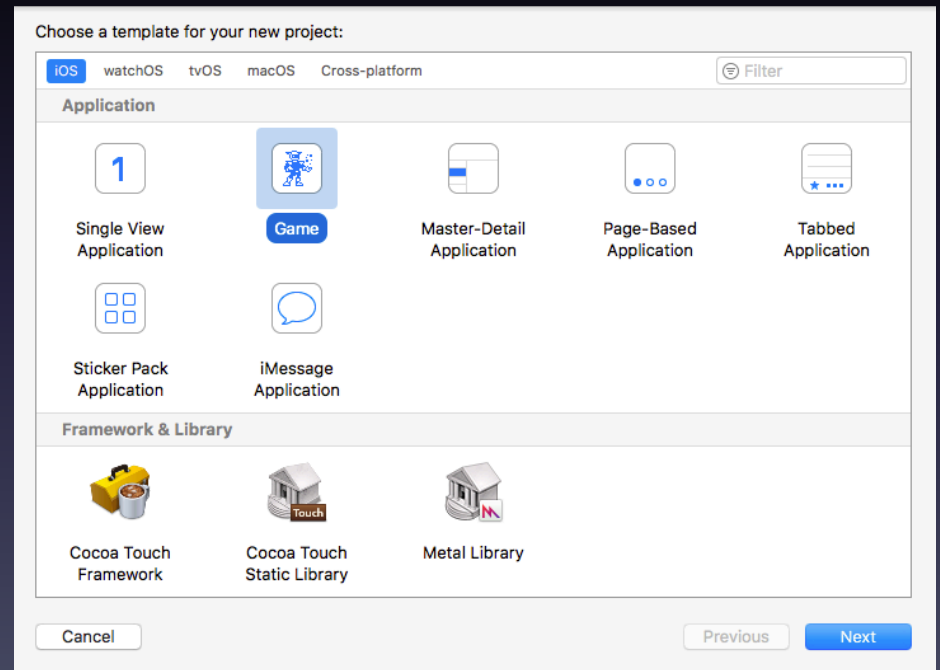
School of EECS

Washington State University

Instructor: Larry Holder

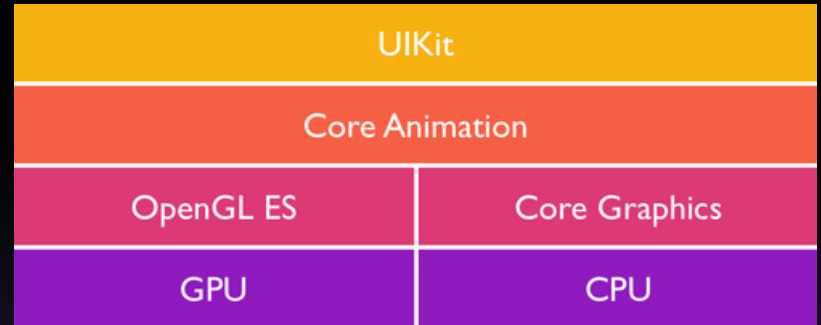
Outline

- iOS frameworks for graphics and animation
- Core Graphics
- SpriteKit
- SceneKit



iOS Frameworks (old)

- UIKit graphics
 - Animate elements of view
- Core Graphics and Core Animation
 - 2D graphics and animation engine
 - Part of UIView
- OpenGL ES and GLKit
 - 2D and 3D rendering for GPUs on Embedded Systems (ES)

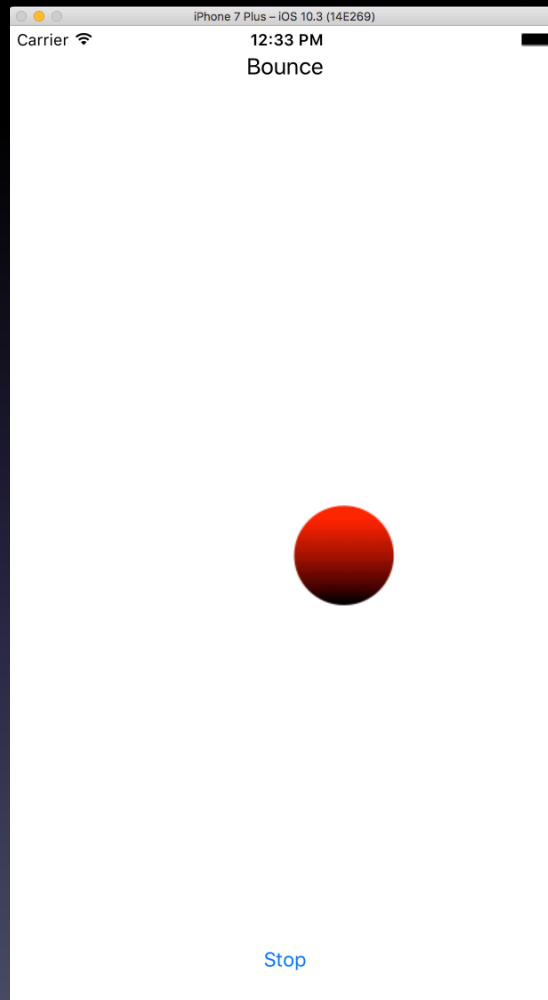


iOS Frameworks (new)

- SpriteKit
 - 2D game engine
 - Most components accessible via Storyboard
- SceneKit
 - 3D game engine
 - Most components accessible via Storyboard
- Metal
 - More direct access to GPU for graphics and computation



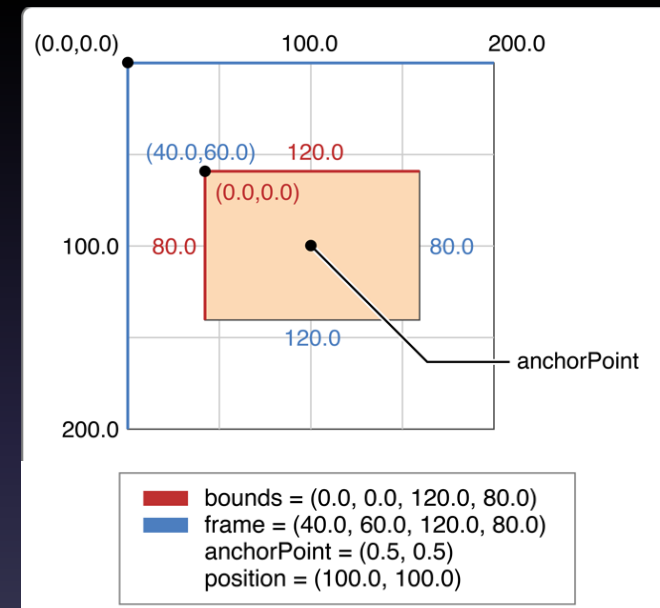
Bounce



Core Graphics

Core Graphics Approach

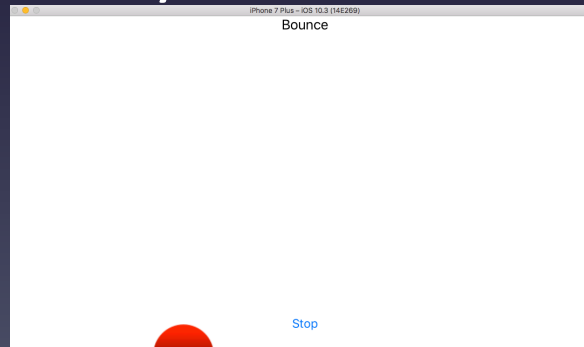
- Coordinate system (upper-left origin)
- Points vs. pixels



- Frame vs. bounds
 - CGRect = {origin.x, origin.y, size.width, size.height}
 - CGRect self.frame, self.bounds

Core Graphics Approach

- Add a UIView as a subView of the main view
- Implement gameUpdate() method
 - Modify subView's position, etc.
- Use Timer to call gameUpdate() method repeatedly
- Watch out for auto layout and orientation changes



Core Graphics Approach

```
class ViewController: UIViewController {

    let frameRate = 30.0 // updates per seconds
    let ballSpeed = 200.0 // points per second
    var ballDirection = CGPoint(x: 1.0, y: -1.0)
    var ballImageView: UIImageView!
    var gameTimer: Timer!

    func initGame() {
        let ballImage = UIImage(named: "redball.png")!
        ballImageView = UIImageView()
        ballImageView.image = ballImage
        ballImageView.frame = CGRect(x: 0, y: 0, width:
            ballImage.size.width, height: ballImage.size.height)
        self.view.addSubview(ballImageView)
    }
}
```

Core Graphics Approach

```
func startGame () {  
    self.gameTimer = Timer.scheduledTimer(withTimeInterval:  
        (1.0 / frameRate), repeats: true, block: updateGame)  
}  
  
func pauseGame () {  
    self.gameTimer.invalidate()  
}
```

Core Graphics Approach

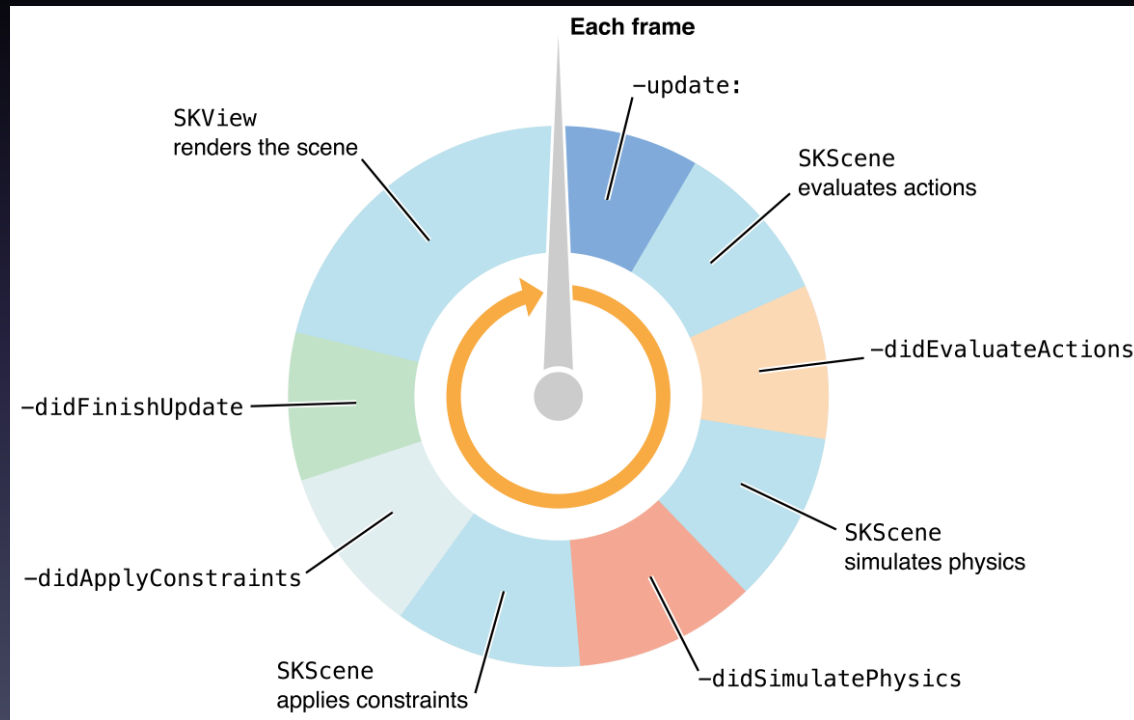
```
func updateGame (timer: Timer) {
    let x = self.ballImageView.frame.origin.x
    let y = self.ballImageView.frame.origin.y
    let width = self.ballImageView.frame.width
    let height = self.ballImageView.frame.height
    // if ball hits wall, then change direction
    if (x < 0) { // Hit left wall
        self.ballDirection.x = -self.ballDirection.x
    }
    if ((x + width) > self.view.frame.width) { // Hit right wall
        self.ballDirection.x = -self.ballDirection.x
    }
    // Handle top and bottom walls...
    // Update ball location
    let xOffset = CGFloat(self.ballSpeed / self.frameRate) * self.ballDirection.x
    let yOffset = CGFloat(self.ballSpeed / self.frameRate) * self.ballDirection.y
    self.ballImageView.frame.origin.x = x + xOffset
    self.ballImageView.frame.origin.y = y + yOffset
}
```

SpriteKit



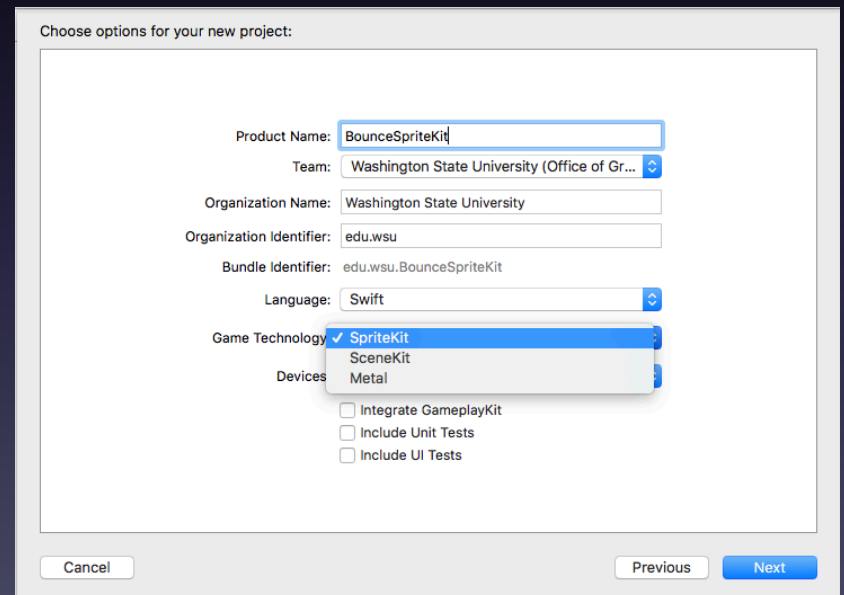
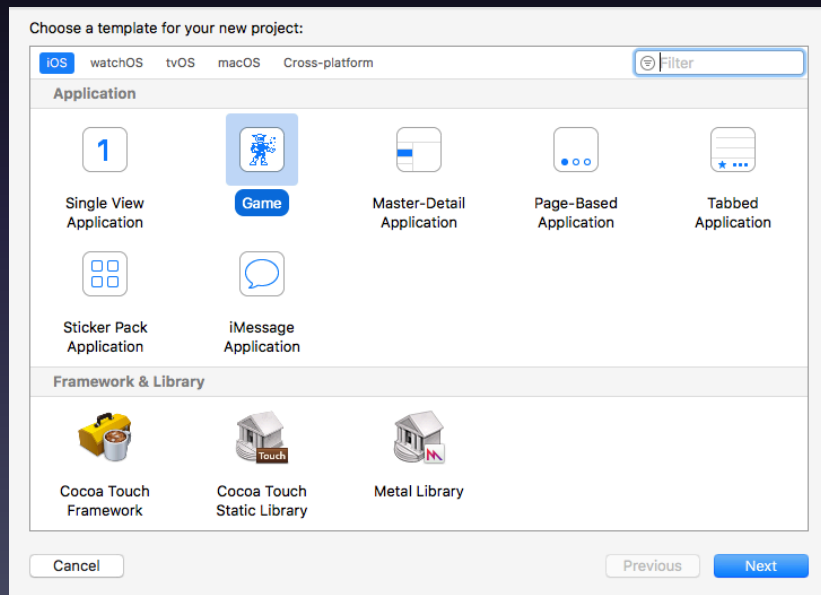
SpriteKit Approach

- Update/render loop



SpriteKit Approach

- Create new Game project
 - Game Technology: SpriteKit



SpriteKit Organization

- Scene(s) of type SKScene
 - Edit in Sprite Editor (.sks file)
- Main view of type SKView
- Present SKScene in SKView

GameScene.swift

```
import SpriteKit
import GameplayKit

class GameScene: SKScene {
    // . . .
```

GameViewController.swift

```
override func viewDidLoad() {
    super.viewDidLoad()
    if let view = self.view as! SKView? {
        // Load the SKScene from 'GameScene.sks'
        if let scene = SKScene(fileNamed: "GameScene") {
            // Set the scale mode to scale to fit the window
            scene.scaleMode = .aspectFill
            // Present the scene
            view.presentScene(scene)
        }
    }
}
```

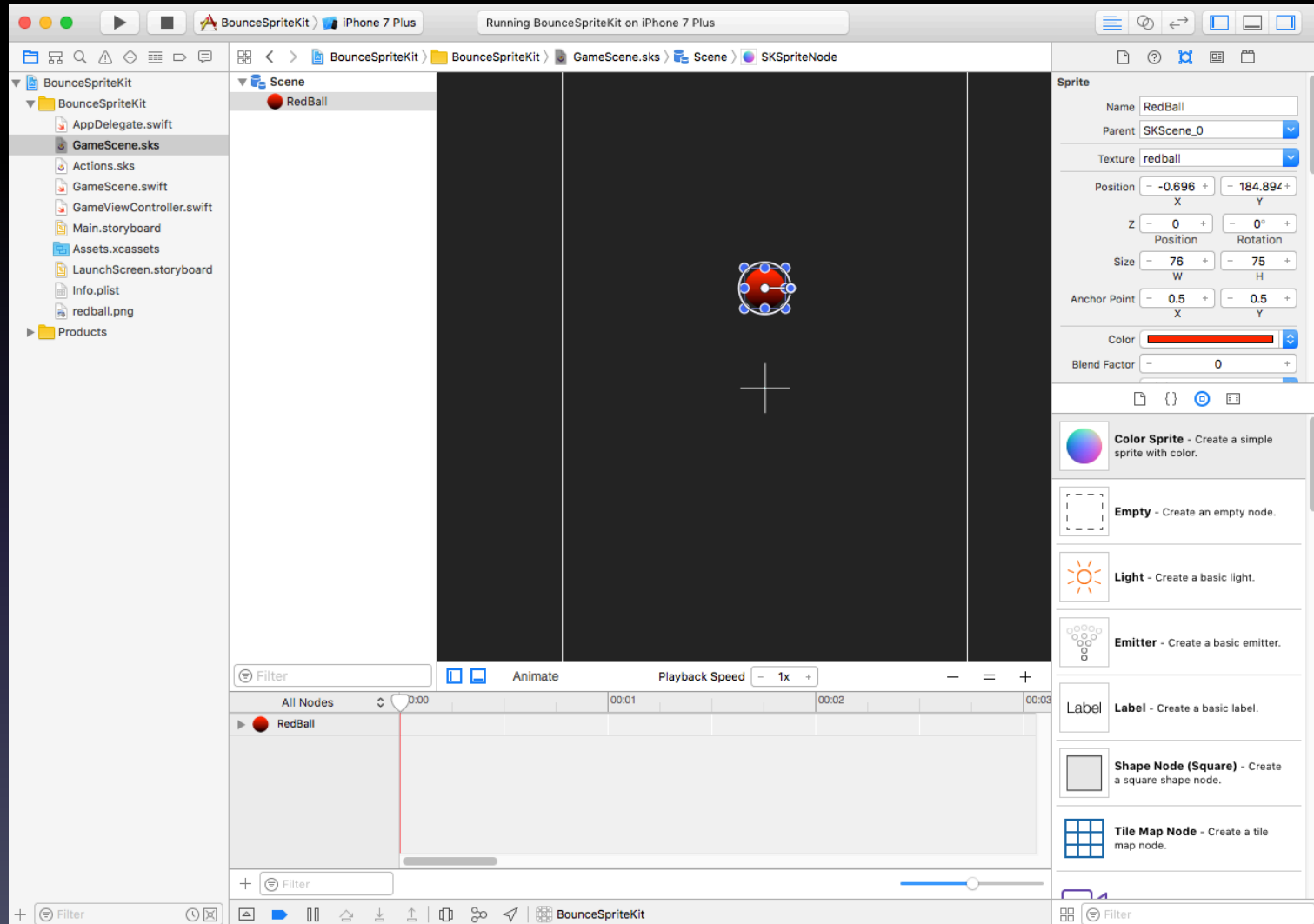
Sprites



redball.png

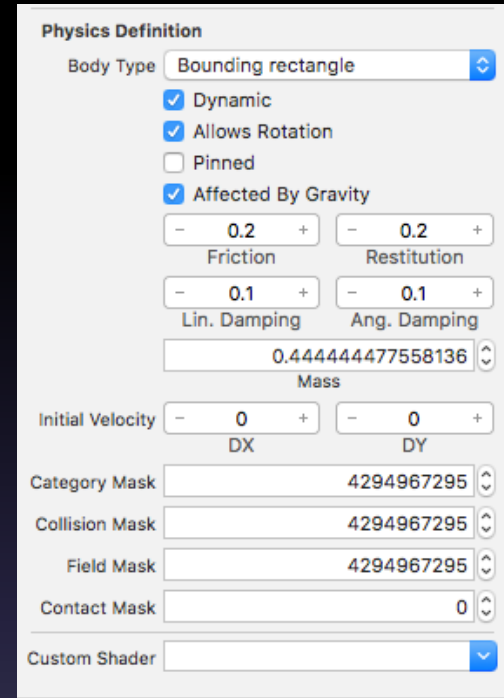
- Sprite is a rectangle with a texture (image)
- **SKSpriteNode** is a sprite with many properties
 - **SKAction** for actions to execute (e.g., fade in/out)
 - **SKPhysicsBody** for physical effects (e.g., gravity)
- Other types of **SKNode**'s (e.g., **SKLabelNode**)
- **SKScene** is a collection of **SKNode**'s

SpriteKit Scene Editor



SpriteKit Physics

- Body Type
- Dynamic
- Pinned (fixed to parent)
- Allows Rotation, Ang. Damping
- Affected By Gravity, Linear Damping, Mass
- Friction, Restitution



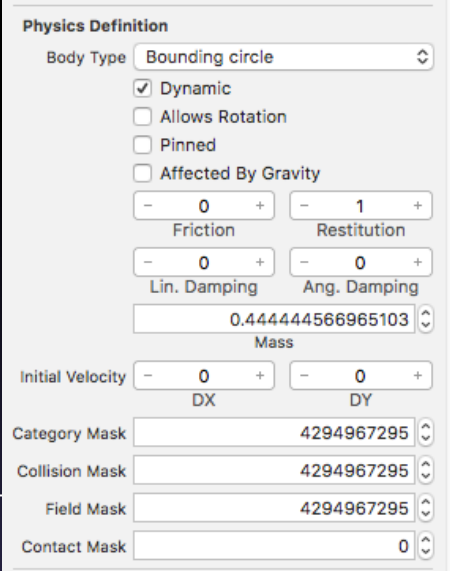
The screenshot shows the 'Physics Definition' panel in Xcode. It contains the following settings:

- Body Type:** Bounding rectangle
- ☒ **Dynamic**
- ☒ **Allows Rotation**
- ☐ **Pinned**
- ☒ **Affected By Gravity**
- Friction:** 0.2
- Restitution:** 0.2
- Lin. Damping:** 0.1
- Ang. Damping:** 0.1
- Mass:** 0.444444477558136
- Initial Velocity:** DX: 0, DY: 0
- Category Mask:** 4294967295
- Collision Mask:** 4294967295
- Field Mask:** 4294967295
- Contact Mask:** 0
- ☒ **Custom Shader**

SpriteKit Physics

- Bouncing off edge of screen

Ball:



The screenshot shows the 'Physics Definition' panel in Xcode. The 'Body Type' is set to 'Bounding circle'. The 'Dynamic' checkbox is checked. 'Allows Rotation' and 'Pinned' are unchecked. 'Affected By Gravity' is unchecked. The 'Friction' slider is at 0, and the 'Restitution' slider is at 1. 'Lin. Damping' is at 0, and 'Ang. Damping' is at 0. The 'Mass' field is set to 0.444444566965103. The 'Initial Velocity' sliders for DX and DY are both at 0. The 'Category Mask', 'Collision Mask', and 'Field Mask' are all set to 4294967295. The 'Contact Mask' is set to 0.

```
override func didMove(to view: SKView) {  
    . . .  
    // Set screen edge to bounce with no friction  
    let screenPhysicsBody = SKPhysicsBody(edgeLoopFrom: self.frame)  
    screenPhysicsBody.friction = 0.0  
    self.physicsBody = screenPhysicsBody  
}  
  
redBallNode.physicsBody?.applyImpulse(CGVector(dx: 200.0, dy: 200.0))
```

SpriteKit: Adding Nodes Programmatically

```
// Add green ball programmatically
greenBallNode = SKSpriteNode(imageNamed: "greenball.png")
greenBallNode.physicsBody = SKPhysicsBody(circleOfRadius: 50.0)
greenBallNode.physicsBody?.affectedByGravity = false
greenBallNode.physicsBody?.friction = 0.0
greenBallNode.physicsBody?.restitution = 1.0
greenBallNode.physicsBody?.linearDamping = 0.0
self.addChild(greenBallNode)
```

SpriteKit Physics: Collisions

- Mask is a bit string (4294967295 = all 1s)
- Category
 - Mask that is a unique power of 2 for each object type
 - E.g., ball: 0001, brick: 0010, wall: 0100
- Category Mask (`SKPhysicsBody.categoryBitMask`)
 - Categories this body belongs to
- Collision Mask (`SKPhysicsBody.collisionBitMask`)
 - Categories this body collides with
- Field Mask (`SKPhysicsBody.fieldBitMask`)
 - Fields this body is affected by
- Contact Mask (`SKPhysicsBody.contactTestBitMask`)
 - Categories generating Contact delegate call, if contact this body

Physics Definition

Body Type: Bounding circle

☒ Dynamic

☐ Allows Rotation

☐ Pinned

☐ Affected By Gravity

Friction: 0 Restitution: 1

Lin. Damping: 0 Ang. Damping: 0.444444566965103

Mass: 0.444444566965103

Initial Velocity: DX: 0 DY: 0

Category Mask: 4294967295

Collision Mask: 4294967295

Field Mask: 4294967295

Contact Mask: 0

```
Body 1 Category Mask: 0010
Body 2 Collision Mask: 0011
Bitwise And:          0010 > 0
                        Collision!
```

SpriteKit Physics: Contacts

- Delegate
 - `SKPhysicsContactDelegate`
 - `SKScene.physicsWorld.contactDelegate = self`
- Delegate methods
 - `didBegin(_ contact: SKPhysicsContact)`
 - `didEnd(_ contact: SKPhysicsContact)`
 - `contact.bodyA.node`
 - `contact.bodyB.node`

SpriteKit Physics: Contacts

```
func didBegin(_ contact: SKPhysicsContact) {  
    let bodyNameA = String(describing: contact.bodyA.node?.name)  
    let bodyNameB = String(describing: contact.bodyB.node?.name)  
    print("Contact: \(bodyNameA), \(bodyNameB)")  
}
```

SpriteKit Touches

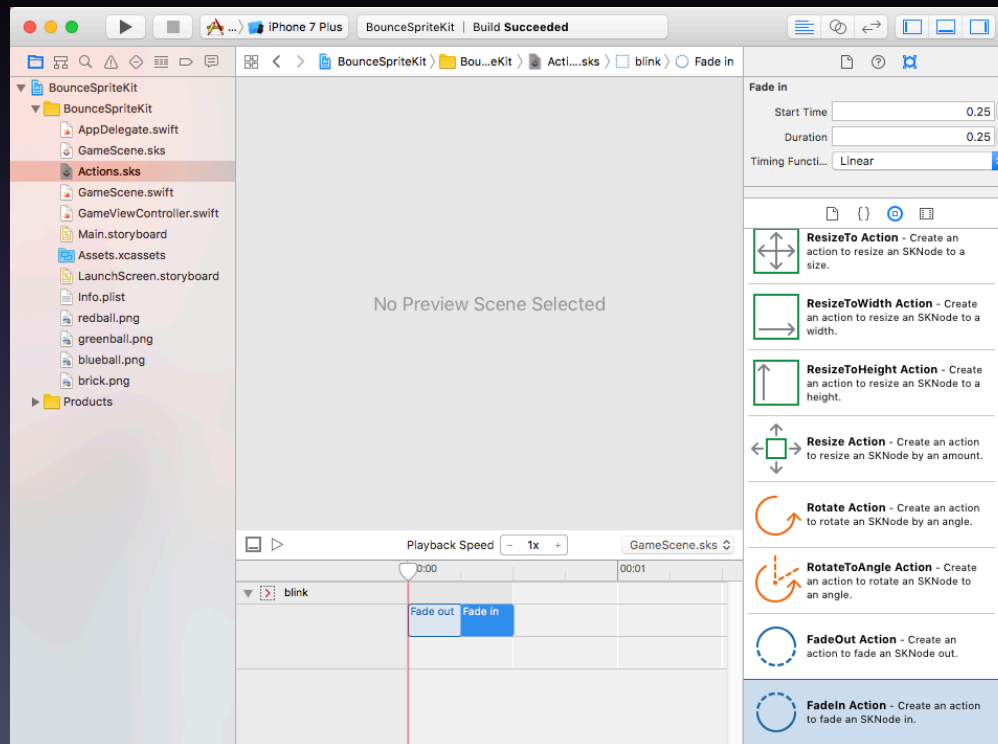
- Same as for UIView
 - `func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?)`
 - `func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?)`
 - `func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?)`

SpriteKit Touches

```
override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {  
    for touch in touches {  
        let point = touch.location(in: self)  
        let nodeArray = nodes(at: point)  
        for node in nodeArray {  
            if node.name == "StartStop" {  
                if (self.isPaused) {  
                    self.startGame()  
                } else {  
                    self.pauseGame()  
                }  
            }  
        }  
    }  
}
```

SpriteKit Actions: Option 1

- Create **SKAction** in SpriteKit Action Editor
 - Execute **SKNode.run(SKAction.init(named: “blink”))**



SpriteKit Actions: Option 1

```
// Only called when two balls contact
func didBegin(_ contact: SKPhysicsContact) {
    let nodeA = contact.bodyA.node!
    let nodeB = contact.bodyB.node!
    let blinkAction = SKAction.init(named: "blink")!
    nodeA.run(blinkAction)
    nodeB.run(blinkAction)
}
```

SpriteKit Actions: Option 2

- Create `SKAction` programmatically
- Execute `SKNode.run(SKAction)`

```
// Only called when two balls contact
func didBegin(_ contact: SKPhysicsContact) {
    let nodeA = contact.bodyA.node!
    let nodeB = contact.bodyB.node!
    let action1 = SKAction.fadeOut(withDuration: 0.25)
    let action2 = SKAction.fadeIn(withDuration: 0.25)
    let blinkAction = SKAction.sequence([action1, action2])
    nodeA.run(blinkAction)
    nodeB.run(blinkAction)
}
```

SpriteKit Audio

- Sound effects
 - `SKAction.playSoundFileNamed`
- Background music
 - `AVAudioPlayer`
- `SKAudioNode` (work in progress...)
 - Positional
 - Effects, e.g., reverb

SpriteKit Audio: Sound Effects

- Create `SKAction.playSoundFileNamed`
- Execute `SKScene.run(SKAction)`

```
let bounceSoundAction = SKAction.playSoundFileNamed("bounce.mp3",
    waitForCompletion: false)

func didBegin(_ contact: SKPhysicsContact) {
    let nodeA = contact.bodyA.node!
    let nodeB = contact.bodyB.node!
    ...
    run(bounceSoundAction)
}
```

Background Music

- Import `AVFoundation`
- Create `AVAudioPlayer` from URL to music file
 - `AVAudioPlayer(contentsOf: URL)`
- Set `volume`, `numberOfLoops` (-1 = loop continuously), ...
- Methods: `play`, `pause`, `stop`, ...

Background Music

```
import AVFoundation

var audioPlayer: AVAudioPlayer!

let musicURL = Bundle.main.url(forResource: "WSU-Fight-Song.mp3",
    withExtension: nil)
do {
    audioPlayer = try AVAudioPlayer(contentsOf: musicURL!)
} catch {
    print("error accessing music")
}
audioPlayer.volume = 0.25
audioPlayer.numberOfLoops = -1 // loop forever

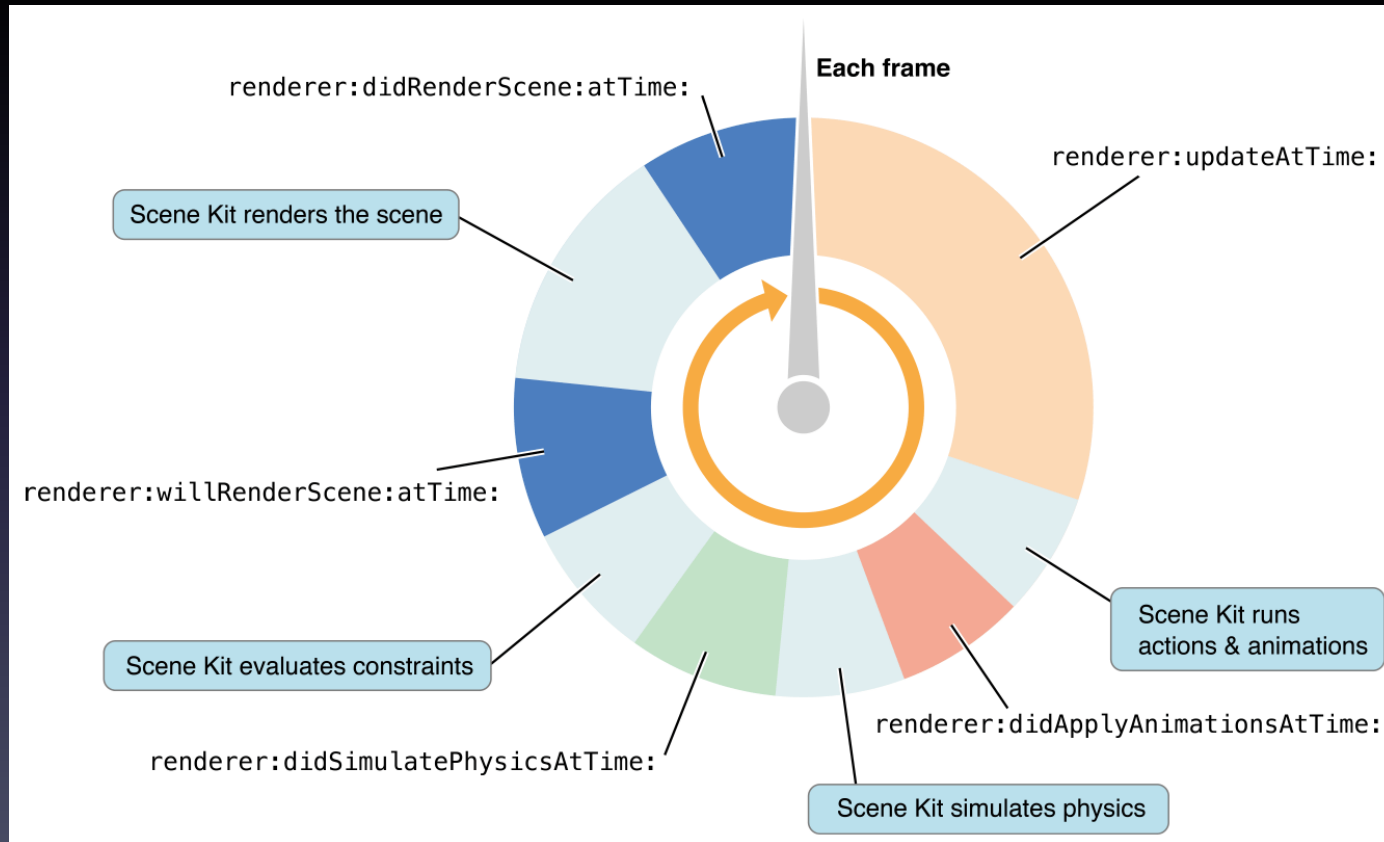
audioPlayer.play() // In startGame()
audioPlayer.pause() // In pauseGame()
```


SceneKit



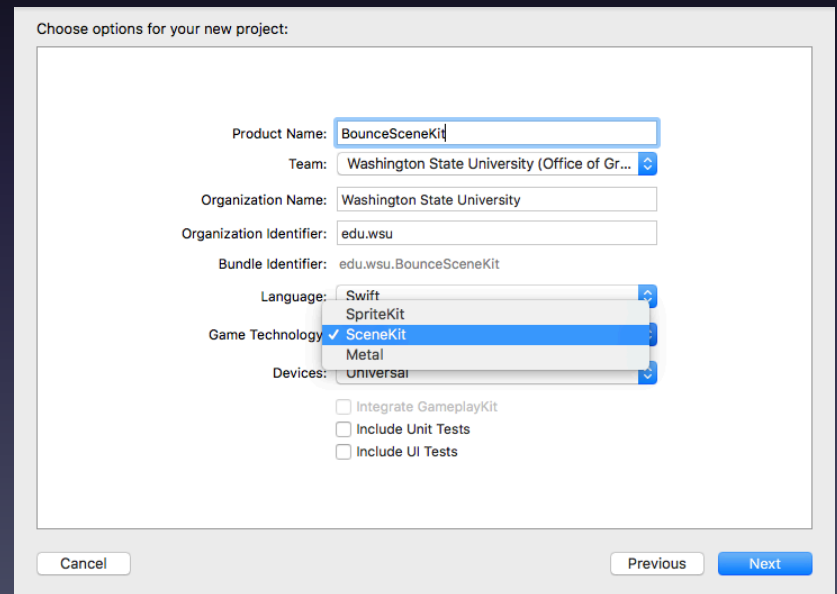
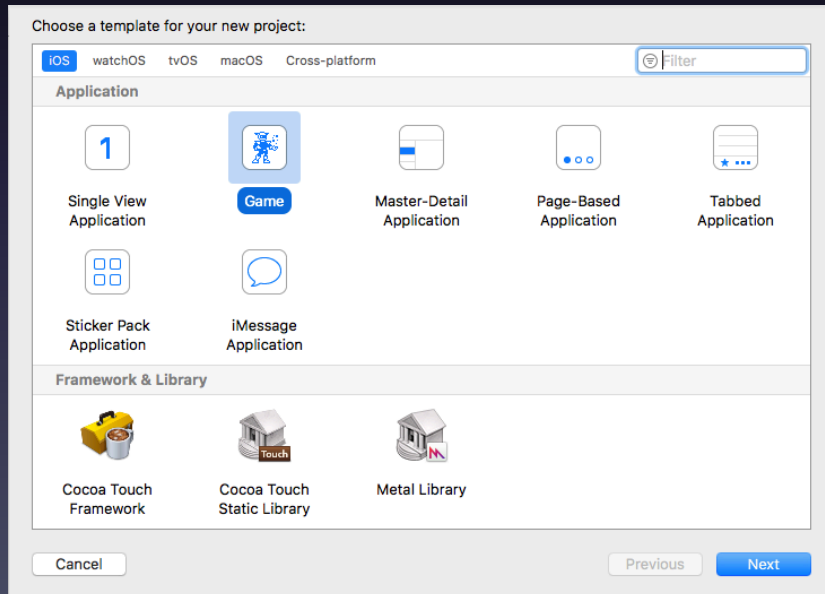
SceneKit Approach

- Update/render loop



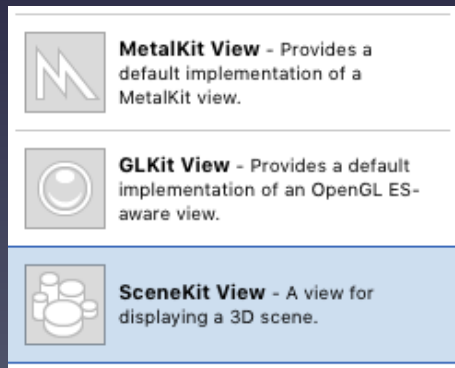
SceneKit Approach

- Create new Game project
 - Game Technology: SceneKit



SceneKit Organization

- Scene(s) of type SCNScene
 - Edit in SceneKit Editor (.scn file)
 - Or, build programmatically
- Main view of type SCNView
- Also available in StoryBoard



GameViewController.swift

```
import UIKit
import SceneKit

class GameViewController: UIViewController {

    var scene: SCNScene!

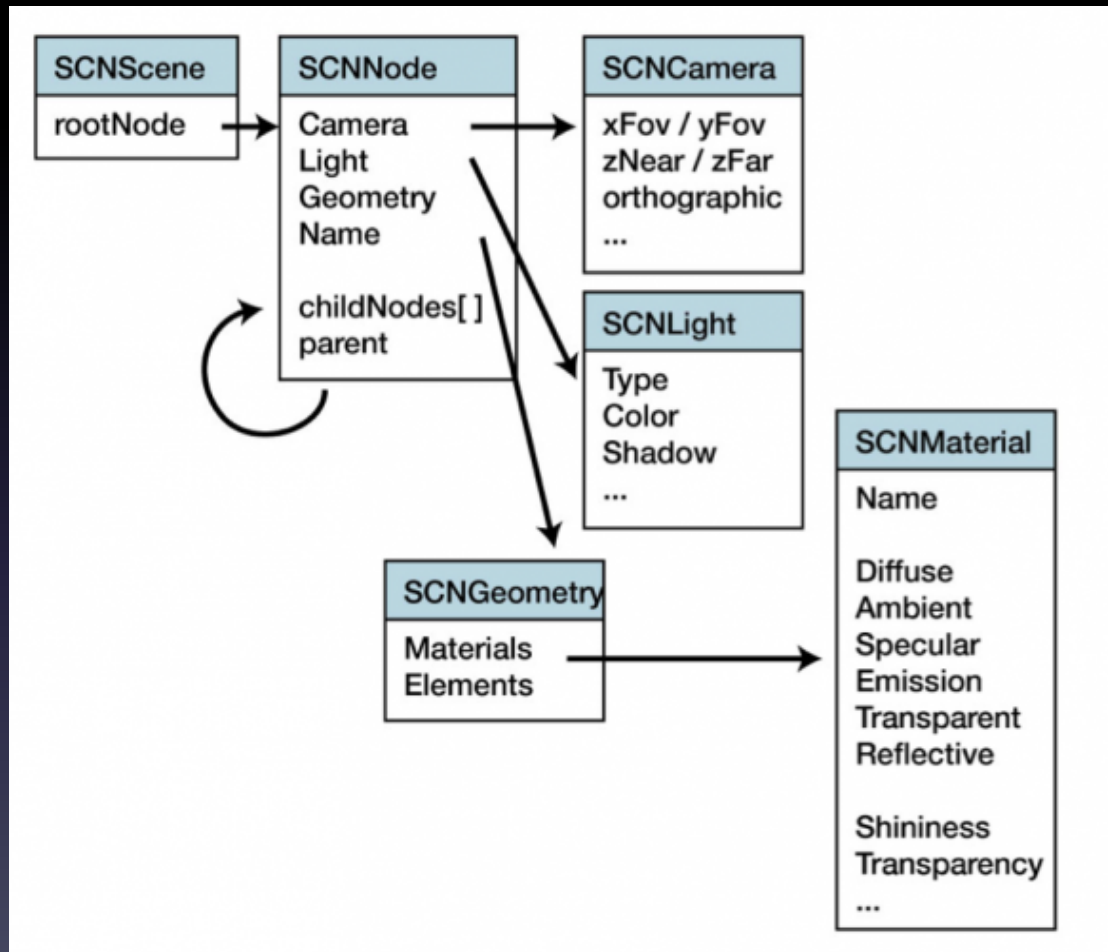
    override func viewDidLoad() {
        super.viewDidLoad()

        // create a new scene
        scene = SCNScene()

        // retrieve the SCNView
        let scnView = self.view as! SCNView

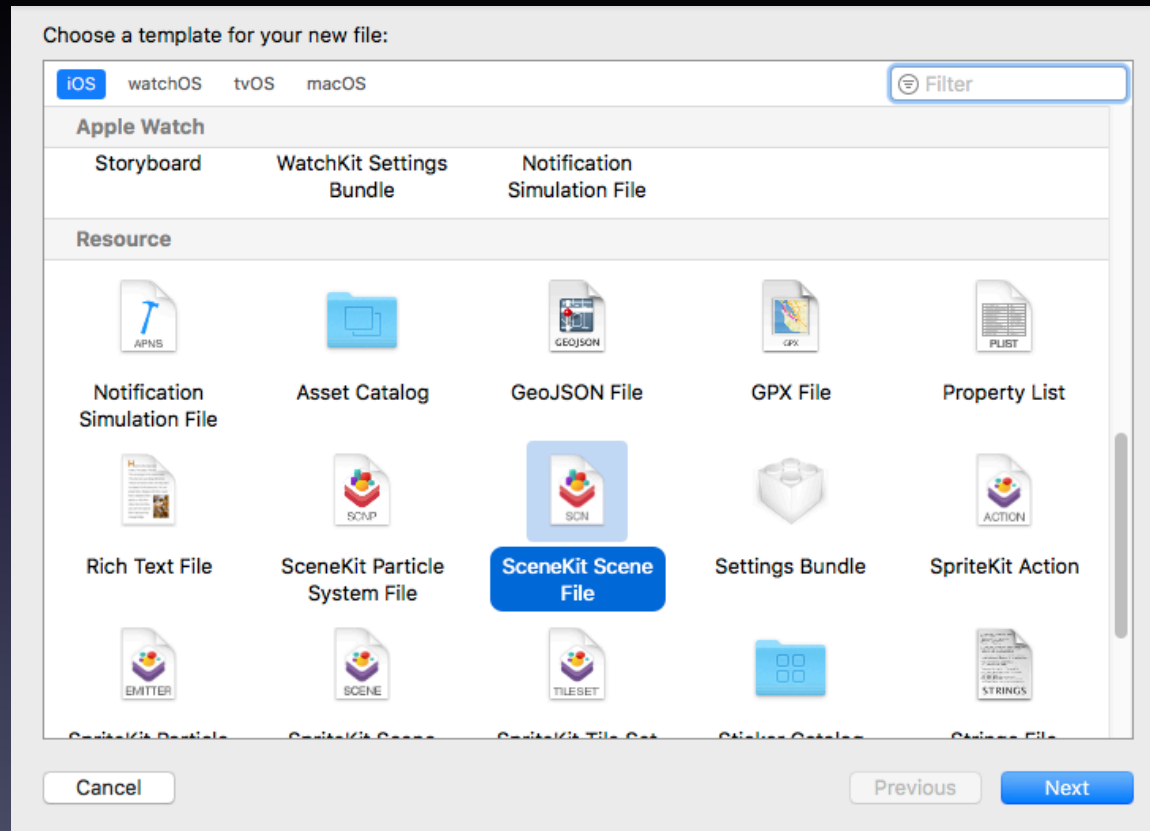
        // set the scene to the view
        scnView.scene = scene
    }
}
```

Scene Graph Layout



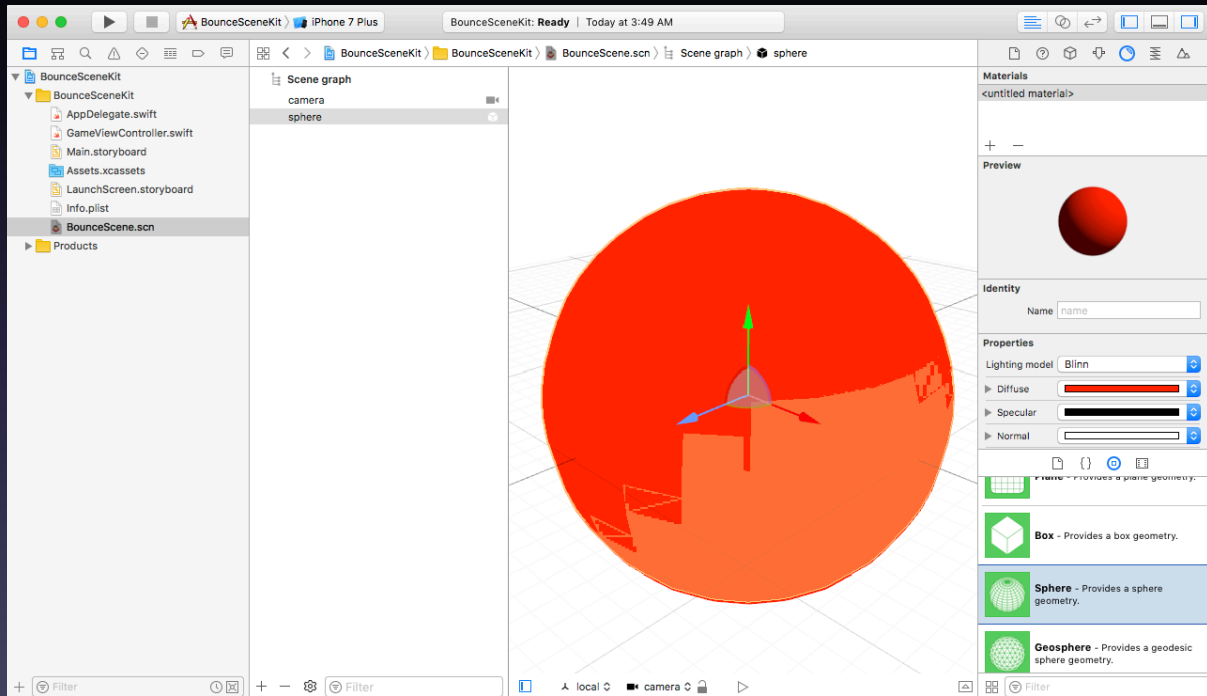
SceneKit Scene Editor













- First, add scene file (.scn)



SceneKit Scene Editor

- Add SCNNode's to scene



	Empty Node - An empty SCNNode.
	Omni light - Illuminates the scene from a point in every direction.
	Directional light - Illuminates the scene in a specific direction.
	Spot light - Illuminates the scene from a point and spreads out as a cone.
	Ambient light - Illuminates the scene equally in every directions.
	IES Light - Illuminates the scene with an IES profile.
	Light Probe - Sample the lighting at a specific location.
	Camera - Provides a camera that can be used as a point of view to render the scene.
	Plane - Provides a plane geometry.
	Box - Provides a box geometry.
	Sphere - Provides a sphere geometry.
	Geosphere - Provides a geodesic sphere geometry.

SceneKit Scene Editor

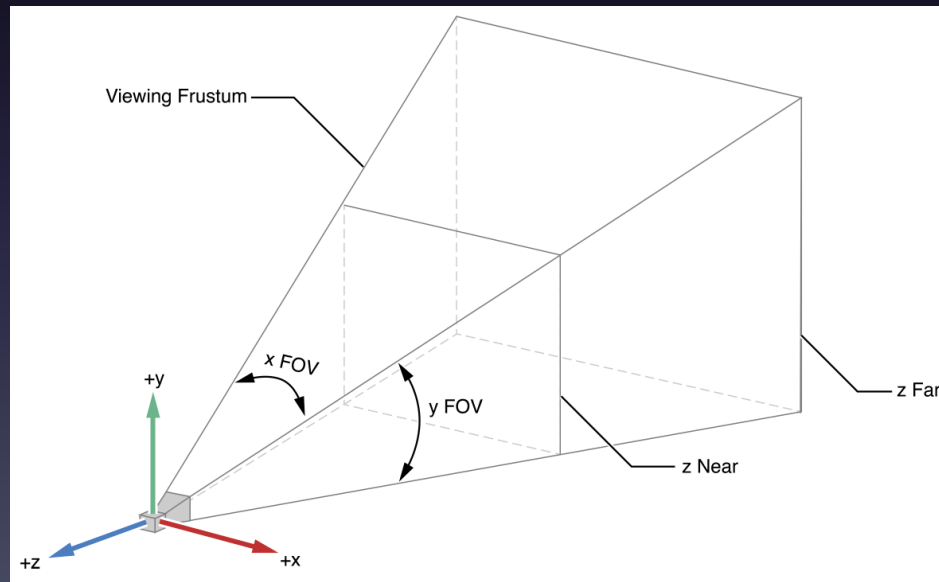
- Load scene file into SCNView
 - `let scene = SCNScene(named: "BounceScene.scn")!`
- Good for creating specific elements of game
 - Collections of nodes
 - Fields
 - Actions
- Cumbersome for creating entire game

Elements of a Scene: SCNNode

- Camera
- Light: Ambient, Directional, Omni, Spot
- Geometry: Plane, Box, Sphere, Text, ...
- Physics
 - Fields: Drag, Gravity, Electric, Magnetic, ...
- Actions: Move, Scale, Rotate, Fade
- Materials

SceneKit Camera

- Default camera unless you add one
- Can allow user control



SceneKit Camera

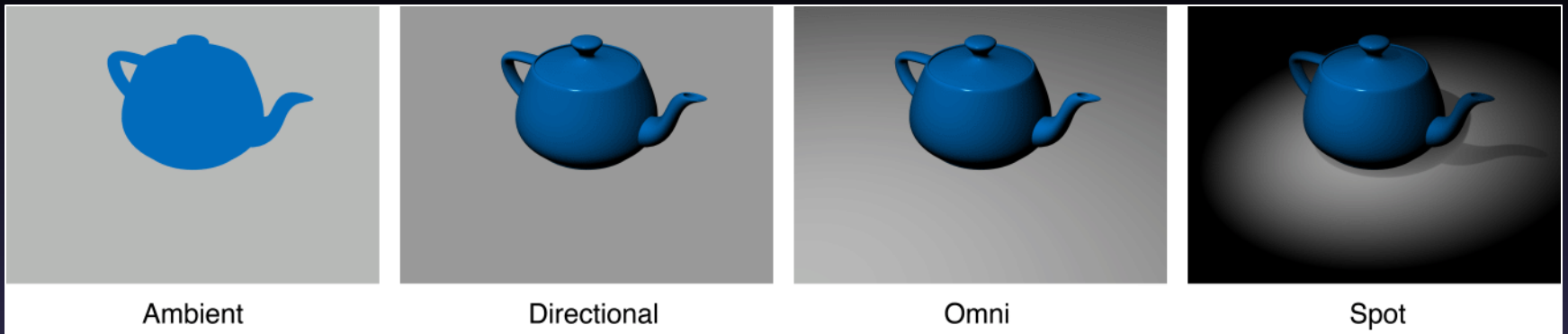
```
// create and add a camera to the scene
let cameraNode = SCNNode()
cameraNode.camera = SCNCamera()
self.scene.rootNode.addChildNode(cameraNode)

// place the camera
cameraNode.position = SCNVector3(x: 0, y: 0, z: 30)

// allow the user to manipulate the camera
scrollView.allowsCameraControl = true
```

SceneKit Light

- Default ambient light unless you add more



SceneKit Light

```
// create and add point light source to the scene
let lightNode = SCNNode()
lightNode.light = SCNLight()
lightNode.light!.type = .omni
lightNode.position = SCNVector3(x: 0, y: 10, z: 10)
self.scene.rootNode.addChildNode(lightNode)

// create and add ambient light to the scene
let ambientLightNode = SCNNode()
ambientLightNode.light = SCNLight()
ambientLightNode.light!.type = .ambient
ambientLightNode.light!.color = UIColor.darkGray
self.scene.rootNode.addChildNode(ambientLightNode)
```

SceneKit Geometry



Plane - Provides a plane geometry.



Box - Provides a box geometry.



Sphere - Provides a sphere geometry.



Geosphere - Provides a geodesic sphere geometry.



Pyramid - Provides a pyramid geometry.



Cylinder - Provides a cylinder geometry.



Cone - Provides a cone geometry.



Tube - Provides a tube geometry.



Capsule - Provides a capsule geometry.



Torus - Provides a torus geometry.



Floor - Provides an infinite plane with reflection support.



3D Text - Provides 3D Text with extrusion and chamfer support.

SceneKit SCNNode

- Create SCNGeometry
 - Set geometry properties
- Create SCNNode from geometry
 - Set node properties
- Add node as child of scene's root node

SceneKit Geometry: SCNSphere

```
// Red ball
let redBallGeometry = SCNSphere(radius: 1.0)
let redBallPhysicsShape = SCNPhysicsShape(geometry:
    redBallGeometry, options: [:])
redBallGeometry.firstMaterial!.diffuse.contents = UIColor.red
redBallNode = SCNNode(geometry: redBallGeometry)
redBallNode.name = "RedBall"
redBallNode.physicsBody = SCNPhysicsBody(type: .dynamic,
    shape: redBallPhysicsShape)
redBallNode.physicsBody!.isAffectedByGravity = false
redBallNode.physicsBody!.friction = 0.0
redBallNode.physicsBody!.restitution = 1.0
redBallNode.physicsBody!.damping = 0.0
redBallNode.physicsBody!.angularDamping = 0.0
self.scene.rootNode.addChildNode(redBallNode)
```


SceneKit Geometry: SCNText

```
// Bounce text
let bounceTextGeometry = SCNText(string: "Bounce",
    extrusionDepth: 0.5)
bounceTextGeometry.firstMaterial!.diffuse.contents =
    UIColor.lightGray
let bounceTextNode = SCNNode(geometry: bounceTextGeometry)
// Primitive positioning and scaling; could do better
bounceTextNode.position = SCNVector3(-2.0, 10.0, 0.0)
bounceTextNode.scale = SCNVector3(0.1, 0.1, 0.1)
self.scene.rootNode.addChildNode(bounceTextNode)
```

SceneKit Geometry: SCNText

```
// Start/Stop text
let startStopTextGeometry = SCNText(string: "Start",
    extrusionDepth: 0.5)
startStopTextGeometry.firstMaterial!.diffuse.contents =
    UIColor.lightGray
startStopTextNode = SCNNode(geometry: startStopTextGeometry)
startStopTextNode.position = SCNVector3(-2.0, -10.0, 0.0)
startStopTextNode.scale = SCNVector3(0.1, 0.1, 0.1)
startStopTextNode.name = "StartStop"
self.scene.rootNode.addChildNode(startStopTextNode)

// Change start/stop text
let textGeom = startStopTextNode.geometry as! SCNText
textGeom.string = "Stop"
```

SceneKit Geometry: SCNPlane

```
// Top wall
let wallGeometry = SCNPlane(width: 20.0, height: 20.0)
let wallPhysicsShape = SCNPhysicsShape(geometry: wallGeometry,
    options: [:])
let wallNode = SCNNode(geometry: wallGeometry)
wallNode.opacity = 0.0 // invisible
wallNode.physicsBody = SCNPhysicsBody(type: .static,
    shape: wallPhysicsShape)
wallNode.physicsBody!.friction = 0.0
wallNode.physicsBody!.restitution = 1.0
wallNode.physicsBody!.rollingFriction = 0.0
wallNode.position = SCNVector3(0.0, 10.0, 0.0)
wallNode.rotation = SCNVector4(1.0, 0.0, 0.0, -Double.pi / 2.0)
```

```
// Show walls
wallGeometry.firstMaterial!.isDoubleSided = true
wallGeometry.firstMaterial!.diffuse.contents = UIColor.blue
wallNode.opacity = 0.2
```

SceneKit Physics: Contacts

- Delegate
 - `SCNPhysicsContactDelegate`
 - `SCNScene.physicsWorld.contactDelegate = self`
- Delegate methods
 - `func physicsWorld(_ world: SCNPhysicsWorld, didBegin contact: SCNPhysicsContact)`
 - `func physicsWorld(_ world: SCNPhysicsWorld, didEnd contact: SCNPhysicsContact)`
 - `contact.nodeA`
 - `contact.nodeB`
- Category, collision, contact bit masks same as for SpriteKit

SceneKit Physics: Contacts

```
func physicsWorld(_ world: SCNPhysicsWorld, didBegin contact:
    SCNPhysicsContact) {
    let nodeA = contact.nodeA
    let nodeB = contact.nodeB
    let nameA = nodeA.name!
    let nameB = nodeB.name!
    print("contact between \(nameA) and \(nameB)")
}
```

SceneKit Interaction

- Option 1: `touchesEnded` with `hitTest`

```
override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {  
    let scnView = self.view as! SCNView  
    for touch in touches {  
        let point = touch.location(in: scnView)  
        let hitResults = scnView.hitTest(point, options: nil)  
        for hitResult in hitResults {  
            print("touched node \(hitResult.node.name)")  
        }  
    }  
}
```

SceneKit Interaction

- Option 2: `UITapGestureRecognizer` with `hitTest`

```
// In viewDidLoad...
let tapRecognizer = UITapGestureRecognizer()
tapRecognizer.numberOfTapsRequired = 1
tapRecognizer.numberOfTouchesRequired = 1
tapRecognizer.addTarget(self, action: #selector(sceneTapped))
scnView.gestureRecognizers = [tapRecognizer]

func sceneTapped(recognizer: UITapGestureRecognizer) {
    let scnView = self.view as! SCNView
    let location = recognizer.location(in: scnView)
    let hitResults = scnView.hitTest(location, options: nil)
    for hitResult in hitResults {
        print("tapped node \(hitResult.node.name)")
    }
}
```

SceneKit Update

- Add `SCNSceneRendererDelegate` to `scnView`
- `scnView.delegate = self`
- Implement `updateAtTime` delegate method

```
func renderer(_ renderer: SCNSceneRenderer, updateAtTime time: TimeInterval) {  
    print("rendered called at \(time)")  
}
```


SceneKit Actions

- Create **SCNAction** programmatically
- Execute **SCNNode.runAction(SCNAction)**

```
func physicsWorld(_ world: SCNPhysicsWorld,
                  didBegin contact: SCNPhysicsContact) {
    let nodeA = contact.nodeA
    let nodeB = contact.nodeB
    let nameA = nodeA.name!
    let nameB = nodeB.name!
    if ((nameA == "Box" ) || (nameB == "Box")) {
        print("You hit the box!")
        let action1 = SCNAction.fadeOut(duration: 0.25)
        let action2 = SCNAction.fadeIn(duration: 0.25)
        let blinkAction = SCNAction.sequence([action1,action2])
        self.redBallNode.runAction(blinkAction)
    }
}
```



Move Action - Create an action to move a node by an offset.



MoveTo Action - Create an action to move a node to a location.



Scale Action - Create an action to scale a node by a factor.



ScaleTo Action - Create an action to scale a node to a factor.



Rotate Action - Create an action to rotate a node by an angle.



RotateTo Action - Create an action to rotate a node to an angle.



RotateTo Action (shortest) - Create an action to rotate a node to an angle using the shortest unit...



RotateBy Axis Angle Action - Create an action to rotate a node by an angle around an axis.



RotateTo Axis Angle Action - Create an action to rotate a node to an angle around an axis.



FadeOut Action - Create an action to fade a node out.



FadeIn Action - Create an action to fade a node in.



FadeOpacityTo Action - Create an action to fade a node to an opacity value.

SceneKit Audio

- Sound effects
 - Create `SCNAudioSource`
 - Create `SCNAction.playAudio`
 - Run action on some `SCNNode`
- Background music
 - Same as SpriteKit, i.e., `AVAudioPlayer`
- `SCNAudioPlayer` (work in progress...)
 - Positional
 - Effects, e.g., reverb

SceneKit Audio: Sound Effects

```
var bounceSoundAction: SCNAction!  
  
let audioSource = SCNAudioSource(named: "bounce.mp3")  
bounceSoundAction = SCNAction.playAudio(audioSource!,  
    waitForCompletion: false)  
  
func physicsWorld(_ world: SCNPhysicsWorld, didBegin contact:  
    SCNPhysicsContact) {  
    let nodeA = contact.bodyA.node!  
    let nodeB = contact.bodyB.node!  
    ...  
    nodeA.runAction(bounceSoundAction)  
}
```

Resources

- Core Graphics
 - developer.apple.com/reference/coregraphics
- Sprite Kit
 - developer.apple.com/spritekit/
- Scene Kit
 - developer.apple.com/scenekit/
- Gameplay Kit
 - developer.apple.com/reference/gameplaykit
- AVFoundation
 - developer.apple.com/av-foundation/