Going Beyond 2D with SpriteKit

Session 609

Ross Dexter, Game Technologies Engineer
SpriteKit

What is SpriteKit?

2D graphics framework for games

Flexible, fast, easy to use

Supported on iOS, macOS, tvOS, and watchOS

Xcode integrated live editor
Apple Graphics Frameworks

- SceneKit
- SpriteKit
- ARKit

Metal
Going Beyond 2D with SpriteKit

Rendering content in ARKit with SpriteKit
Using SpriteKit and SceneKit in ARKit
Introduction to SKRenderer
Working with ARKit
What is Augmented Reality?
Working with ARKit

What is ARKit?

ARKit does all the hard work for you
Tracks your device’s pose
Provide it content you want to appear in AR
Updates the relative position of your content
See Introducing ARKit for more info
Working with ARKit
Content anchors

Anchors are what makes AR work
3D points tied to real-world features
ARKit uses device hardware to detect them
Easy to create
Working with ARKit
How ARKit works with SpriteKit

ARKit directly interacts with SpriteKit
Asks for nodes it will associate with anchors
Nodes are updated as the device moves
Sprites always face the camera
Working with ARKit

How sprites work in 3D
Working with ARKit

How sprites work in 3D
Working with ARKit
How sprites work in 3D

World

Camera View
Working with ARKit
How sprites work in 3D
Working with ARKit
How sprites work in 3D

World

Camera View
Working with ARKit

How sprites work in 3D
Working with ARKit

How sprites work in 3D

World

Camera View
Working with ARKit

How sprites work in 3D

World

Camera View
Working with ARKit

ARKit Objects

- ARSession
- ARAnchor
- ARSKView
- ARSKViewDelegate
Working with ARKit

ARKit Objects

- ARSession
- ARAnchor
- ARSKView
- ARSKViewDelegate
ARSession
The heart of ARKit

Handles device tracking

Orchestrates interactions with SpriteKit

Methods for adding and removing anchors

Call run method to start
• Requires an ARSessionConfiguration
• Use ARWorldTrackingSessionConfiguration
ARAnchor
Defines real-world feature points

Contains transform data, unique identifier

ARAnchors are mapped to SKNodes
ARSKView
Bridge between SpriteKit and ARKit

Derived from SKView

Creates and contains ARSession

Methods for getting related anchors/nodes

hitTest method for creating anchors
ARSKViewDelegate
Callbacks for anchor updates

Protocol derived from SKViewDelegate
All methods are optional
Working with ARKit

Creating a project

New AR app template in Xcode 9
Working with ARKit
Creating a project

New AR app template in Xcode 9
Select SpriteKit as the content technology
Working with ARKit
Creating a project

New AR app template in Xcode 9
Select SpriteKit as the content technology
Ready to enter augmented reality!
Working with ARKit

Scene.sks

The SpriteKit scene

Add non-AR content here

• Good for HUD, help text, etc.
• $Z \geq 0$ will draw over AR content

AR content is managed by ARKit
• AR content $z$ positions are $< 0$
Working with ARKit

Scene.swift

The SpriteKit scene’s corresponding source file

Put code to manage your scene in here

• Gameplay, logic, etc.
Working with ARKit
ViewController.swift

App’s view controller
Conforms to ARSKViewDelegate
ARSKView sceneView property
Primary means of ARKit interaction
Calls run on ARSession
Implement delegate methods here
Working with ARKit

Anchor events
Working with ARKit

Anchor events

Anchor added
Working with ARKit

Anchor events

Anchor added ➔ Create AR content
Working with ARKit

Anchor events

- Anchor added
- Anchor updated
- Create AR content
Working with ARKit

Anchor events

- Anchor added
- Anchor updated

- Create AR content
- React to update
Working with ARKit

Anchor events

Anchor added → Create AR content

Anchor updated → React to update

Anchor removed
Working with ARKit

Anchor events

- Anchor added
- Anchor updated
- Anchor removed

Create AR content
React to update
Perform cleanup
Working with ARKit

Anchor events

- Anchor added
- Anchor updated
- Anchor removed

view(view:nodeFor anchor:)
view(view:didAdd node: for anchor:)
view(view:willUpdate node: for anchor:)
view(view:didUpdate node: for anchor:)
view(view:didRemove node: for anchor:)
ARSKViewDelegate Methods
Anchor added

Create a custom node for an anchor
If not implemented, SKNode created for you
Node will be modified by ARKit
Automatically added to the scene graph
ARSKViewDelegate Methods

Anchor added

Called after an SKNode is mapped to an anchor

If `view(view:nodeFor anchor:)` implemented
- Node is the result of that method
- Otherwise a default empty node

Add content as children of node

Children won’t be modified by ARKit
ARSKViewDelegate Methods

Anchor updated

Called when node is updated with anchor’s data
- willUpdate called before update
- didUpdate called after update

Occurs when device moves & view changes

Node’s transform may change

view(view:nodeFor anchor:)
view(view:didAdd node: for anchor:)
view(view:willUpdate node: for anchor:)
view(view:didUpdate node: for anchor:)
view(view:didRemove node: for anchor:')
ARSKViewDelegate Methods

Anchor removed

Called when node removed from scene graph

Occurs on anchor removed from ARSession

view(view:willUpdate node: for anchor:)
view(view:didUpdate node: for anchor:)
view(view:nodeFor anchor:)
view(view:didAdd node: for anchor:)
view(view:willUpdate node: for anchor:)
view(view:didUpdate node: for anchor:)
view(view:didRemove node: for anchor:)
// Creating anchors - Scene.swift

override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
    guard let sceneView = self.view as? ARSKView else {
        return
    }

    if let touchLocation = touches.first?.location(in: sceneView) {

        // Create anchor with ARSKView's hitTest method
        if let hit = sceneView.hitTest(touchLocation, types: .featurePoint).first {
            sceneView.session.add(anchor: ARAnchor(transform: hit.worldTransform))
        }
    }
}
// Creating anchors - Scene.swift

override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
    guard let sceneView = self.view as? ARSKView else {
        return
    }

    if let touchLocation = touches.first?.location(in: sceneView) {
        // Create anchor with ARSKView's hitTest method
        if let hit = sceneView.hitTest(touchLocation, types: .featurePoint).first {
            sceneView.session.add(anchor: ARAnchor(transform: hit.worldTransform))
        }
    }
}
// Creating anchors - Scene.swift

override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
    guard let sceneView = self.view as? ARSKView else {
        return
    }

    if let touchLocation = touches.first?.location(in: sceneView) {

        // Create anchor with ARSKView's hitTest method
        if let hit = sceneView.hitTest(touchLocation, types: .featurePoint).first {
            sceneView.session.add(Anchor: ARAnchor(transform: hit.worldTransform))
        }
    }
}
// Creating anchors - Scene.swift

override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
    guard let sceneView = self.view as? ARSKView else {
        return
    }

    if let touchLocation = touches.first?.location(in: sceneView) {
        // Create anchor with ARSKView's hitTest method
        if let hit = sceneView.hitTest(touchLocation, types: .featurePoint).first {
            sceneView.session.add(anchor: ARAnchor(transform: hit.worldTransform))
        }
    }
}
// Providing SpriteKit content - ViewController.swift

// Implementation of delegate method
func view(_ view: ARSKView, didAdd node: SKNode, for anchor: ARAnchor) {
    let labelNode = SKLabelNode(text: "👾")

    labelNode.horizontalAlignmentMode = .center
    labelNode.verticalAlignmentMode = .center

    node.addChild(labelNode)
}
// Providing SpriteKit content - ViewController.swift

// Implementation of delegate method
func view(_ view: ARSKView, didAdd node: SKNode, for anchor: ARAnchor) {
    let labelNode = SKLabelNode(text: "👾")

    labelNode.horizontalAlignmentMode = .center
    labelNode.verticalAlignmentMode = .center

    node.addChild(labelNode)
}
func view(_ view: ARSKView, didAdd node: SKNode, for anchor: ARAnchor) {
    let labelNode = SKLabelNode(text: "👾")
    labelNode.horizontalAlignmentMode = .center
    labelNode.verticalAlignmentMode = .center

    node.addChild(labelNode)
}
// Providing SpriteKit content - ViewController.swift

// Implementation of delegate method
func view(_ view: ARSKView, didAdd node: SKNode, for anchor: ARAnchor) {
    let labelNode = SKLabelNode(text: "👾")
    
    labelNode.horizontalAlignmentMode = .center
    labelNode.verticalAlignmentMode = .center

    node.addChild(labelNode)
}
Demo
SpriteKit and ARKit
Attributed Strings

SKLabelNode now supports attributed strings

Allows you to mix colors, fonts in the same label

Uses NSAttributedString

Set it on the new attributedText property
Attributed Strings

SKLabelNode now supports attributed strings

Allows you to mix colors, fonts in the same label

Uses NSAttributedString

Set it on the new attributedText property
SKTransformNode

SKNode already has z rotation

SKTransformNode adds x, y rotation

Rotations apply to all children

Uses orthographic projection

Getters and setters for
  • Euler angles, rotation matrices, quaternions
View Debugger Support

View debugging now supports SpriteKit content
Displays full scene graph
Inspect node properties
For more info, see Debugging with Xcode 9
Working with ARKit

Takeaways

Quick and easy to get started
ARKit does all the hard work for you
SpriteKit makes rendering AR content a snap
New SpriteKit features give greater flexibility
Working with ARKit

Open questions

What if we don’t want billboarded sprites?

What if we want perspective?

What if we want to mix 2D and 3D content in augmented reality?

What if we want to take SpriteKit further into 3D?
SceneKit, SpriteKit, and ARKit
SceneKit, SpriteKit, and ARKit

SceneKit and SpriteKit

SceneKit is SpriteKit’s 3D counterpart
Use SpriteKit scenes as material on objects
Allows for 3D transforms and perspective
Mix 2D and 3D content
SceneKit, SpriteKit, and ARKit

ARKit and SceneKit

SceneKit is also integrated with ARKit

Use SceneKit as the content technology
Working with ARKit

ARKit API and SceneKit

API is similar by design

ARSSession
ARAnchor
ARSKView
ARSKViewDelegate
Working with ARKit

ARKit API and SceneKit

API is similar by design

Template creates ARSCNView for you

ViewController is ARSCNViewDelegate
SceneKit, SpriteKit, and ARKit

SpriteKit content in SceneKit

Normal SpriteKit rendering
• Set scene on SKView
• Works with UIKit/AppKit to render and display
SceneKit, SpriteKit, and ARKit

SpriteKit content in SceneKit

Rendering SpriteKit content in SceneKit

• Set scene on SCNMaterialProperty
• The material works with SceneKit to render
  - Uses SceneKit’s Metal command queue
• SpriteKit texture mapped onto geometry
SceneKit, SpriteKit, and ARKit

SpriteKit content in SceneKit
SceneKit, SpriteKit, and ARKit

SpriteKit content in SceneKit
SceneKit, SpriteKit, and ARKit

SpriteKit content in SceneKit
SceneKit, SpriteKit, and ARKit

SpriteKit content in SceneKit
// Using SpriteKit in SceneKit

// Get SpriteKit scene
let spriteKitScene = SKScene(fileNamed: "SpriteKitScene")

// Create plane geometry
let plane = SCNPlane(width: 10.0, height: 10.0)

// Set SpriteKit scene on the plane's material
plane.firstMaterial?.diffuse.contents = spriteKitScene

// If material double-sided, SpriteKit scene will show up on both sides of the plane
plane.firstMaterial?.isDoubleSided = true

// Create a SceneKit node for the plane
let sceneKitNode = SCNNode(geometry: plane)

// Add the SceneKit node to the SceneKit scene
scene.rootNode.addChildNode(sceneKitNode)
// Using SpriteKit in SceneKit

// Get SpriteKit scene
let spriteKitScene = SKScene(fileNamed: "SpriteKitScene")

// Create plane geometry
let plane = SCNPlane(width: 10.0, height: 10.0)

// Set SpriteKit scene on the plane's material
plane.firstMaterial?.diffuse.contents = spriteKitScene

// If material double-sided, SpriteKit scene will show up on both sides of the plane
plane.firstMaterial?.isDoubleSided = true

// Create a SceneKit node for the plane
let sceneKitNode = SCNNode(geometry: plane)

// Add the SceneKit node to the SceneKit scene
scene.rootNode.addChildNode(sceneKitNode)
// Using SpriteKit in SceneKit

// Get SpriteKit scene
let spriteKitScene = SKScene(fileNamed: "SpriteKitScene")

// Create plane geometry
let plane = SCNPlane(width: 10.0, height: 10.0)

// Set SpriteKit scene on the plane's material
plane.firstMaterial?.diffuse.contents = spriteKitScene

// If material double-sided, SpriteKit scene will show up on both sides of the plane
plane.firstMaterial?.isDoubleSided = true

// Create a SceneKit node for the plane
let sceneKitNode = SCNNode(geometry: plane)

// Add the SceneKit node to the SceneKit scene
scene.rootNode.addChildNode(sceneKitNode)
// Using SpriteKit in SceneKit

// Get SpriteKit scene
let spriteKitScene = SKScene(fileNamed: "SpriteKitScene")

// Create plane geometry
let plane = SCNPlane(width: 10.0, height: 10.0)

// Set SpriteKit scene on the plane's material
plane.firstMaterial?.diffuse.contents = spriteKitScene

// If material double-sided, SpriteKit scene will show up on both sides of the plane
plane.firstMaterial?.isDoubleSided = true

// Create a SceneKit node for the plane
let sceneKitNode = SCNNode(geometry: plane)

// Add the SceneKit node to the SceneKit scene
scene.rootNode.addChildNode(sceneKitNode)
// Using SpriteKit in SceneKit

// Get SpriteKit scene
let spriteKitScene = SKScene(fileNamed: "SpriteKitScene")

// Create plane geometry
let plane = SCNPlane(width: 10.0, height: 10.0)

// Set SpriteKit scene on the plane's material
plane.firstMaterial?.diffuse.contents = spriteKitScene

// If material double-sided, SpriteKit scene will show up on both sides of the plane
plane.firstMaterial?.isDoubleSided = true

// Create a SceneKit node for the plane
let sceneKitNode = SCNNode(geometry: plane)

// Add the SceneKit node to the SceneKit scene
scene.rootNode.addChildNode(sceneKitNode)
// Using SpriteKit in SceneKit

// Get SpriteKit scene
let spriteKitScene = SKScene(fileNamed: "SpriteKitScene")

// Create plane geometry
let plane = SCNPlane(width: 10.0, height: 10.0)

// Set SpriteKit scene on the plane's material
plane.firstMaterial?.diffuse.contents = spriteKitScene

// If material double-sided, SpriteKit scene will show up on both sides of the plane
plane.firstMaterial?.isDoubleSided = true

// Create a SceneKit node for the plane
let sceneKitNode = SCNNode(geometry: plane)

// Add the SceneKit node to the SceneKit scene
scene.rootNode.addChildNode(sceneKitNode)
// Using SpriteKit in SceneKit

// Get SpriteKit scene
let spriteKitScene = SKScene(fileNamed: "SpriteKitScene")

// Create plane geometry
let plane = SCNPlane(width: 10.0, height: 10.0)

// Set SpriteKit scene on the plane's material
plane.firstMaterial?.diffuse.contents = spriteKitScene

// If material double-sided, SpriteKit scene will show up on both sides of the plane
plane.firstMaterial?.isDoubleSided = true

// Create a SceneKit node for the plane
let sceneKitNode = SCNNode(geometry: plane)

// Add the SceneKit node to the SceneKit scene
scene.rootNode.addChildNode(sceneKitNode)
Demo

ARKit with SceneKit and SpriteKit
SceneKit, SpriteKit, and ARKit

Takeaways

- Allows for full 3D transforms, perspective
- Can mix 2D and 3D content
- Works with ARKit, or in general 3D apps
Introduction to SKRenderer
How SpriteKit Works

Normal usage
How SpriteKit Works

SpriteKit with SceneKit
How SpriteKit Works

SKRenderer

UIKit/AppKit

SKView

SKScene

SceneKit

SCNMaterialProperty

SKScene

Metal

SKRenderer

SKScene
Introduction to SKRenderer

Control update and render

Use instead of SKView

Control update and render

Works directly with Metal

SceneKit uses to render SpriteKit content
Introduction to SKRenderer

Using SKRenderer

- Initialize
- Set Scene
- Update
- Render
Introduction to SKRenderer

Initialization

```swift
var device: MTLDevice! = nil
var renderer: SKRenderer! = nil
var skScene: SKScene! = nil
var commandQueue: MTLCommandQueue! = nil

...

device = MTLCreateSystemDefaultDevice()
commandQueue = device.makeCommandQueue()
renderer = SKRenderer(device: device)
```
if let scene = SKScene(fileNamed: "GameScene") {
    skScene = scene
    renderer.scene = skScene
}
let currentTime = CACurrentMediaTime()
renderer.update(atTime: currentTime)
// Render for command buffer
renderer.render(withViewport: viewport, commandBuffer: commandBuffer, renderPassDescriptor: renderPassDesc)

// Render for command encoder
renderer.render(withViewport: viewport, renderCommandEncoder: renderCommandEncoder, renderPassDescriptor: renderPassDesc, commandQueue: commandQueue)
// Render for command buffer
renderer.render(withViewport: viewport, commandBuffer: commandBuffer, renderPassDescriptor: renderPassDesc)

// Render for command encoder
renderer.render(withViewport: viewport, renderCommandEncoder: renderCommandEncoder, renderPassDescriptor: renderPassDesc, commandQueue: commandQueue)
Introduction to SKRenderer

Rendering

// Render for command buffer
renderer.render(withViewport: viewport, commandBuffer: commandBuffer, renderPassDescriptor: renderPassDesc)

// Render for command encoder
renderer.render(withViewport: viewport, renderCommandEncoder: renderCommandEncoder, renderPassDescriptor: renderPassDesc, commandQueue: commandQueue)
// Render for command buffer
renderer.render(withViewport: viewport, commandBuffer: commandBuffer, renderPassDescriptor: renderPassDesc)

// Render for command encoder
renderer.render(withViewport: viewport, renderCommandEncoder: renderCommandEncoder, renderPassDescriptor: renderPassDesc, commandQueue: commandQueue)
// Render for command buffer
renderer.render(withViewport: viewport, commandBuffer: commandBuffer, renderPassDescriptor: renderPassDesc)

// Render for command encoder
renderer.render(withViewport: viewport, renderCommandEncoder: renderCommandEncoder, renderPassDescriptor: renderPassDesc, commandQueue: commandQueue)
Demo
Drawing SpriteKit content in 3D with Metal
Introduction to SKRenderer

Takeaways

SKRenderer gives more control over SpriteKit
Update and render exactly when you want
Use with Metal however you see fit
Summary

SpriteKit is useful in both 2D and 3D

Plays nice with SceneKit, Metal, and ARKit

New features give you more control than ever
More Information

<table>
<thead>
<tr>
<th>Session</th>
<th>Location</th>
<th>Date/Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introducing Metal 2</td>
<td>Executive Ballroom</td>
<td>Tuesday 1:50PM</td>
</tr>
<tr>
<td>Introducing ARKit: Augmented Reality for iOS</td>
<td>Hall 3</td>
<td>Tuesday 5:10PM</td>
</tr>
<tr>
<td>SceneKit: What’s New</td>
<td>Grand Ballroom A</td>
<td>Wednesday 11:00AM</td>
</tr>
<tr>
<td>Debugging with Xcode 9</td>
<td>Hall 2</td>
<td>Wednesday 10:00AM</td>
</tr>
</tbody>
</table>
## Labs

<table>
<thead>
<tr>
<th>Lab</th>
<th>Room</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>GameplayKit Lab</td>
<td>Technology Lab G</td>
<td>Fri 9:00AM–12:00PM</td>
</tr>
<tr>
<td>SpriteKit Lab</td>
<td>Technology Lab G</td>
<td>Fri 12:00PM–2:30PM</td>
</tr>
</tbody>
</table>