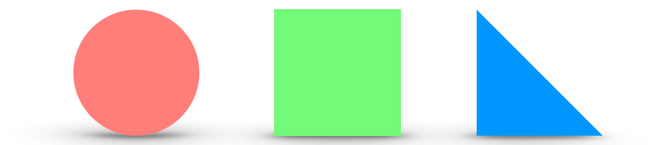
SPRITEKIT, SCENEKIT, ARKIT—OH MY!

Justin Miller • Swift By Northwest

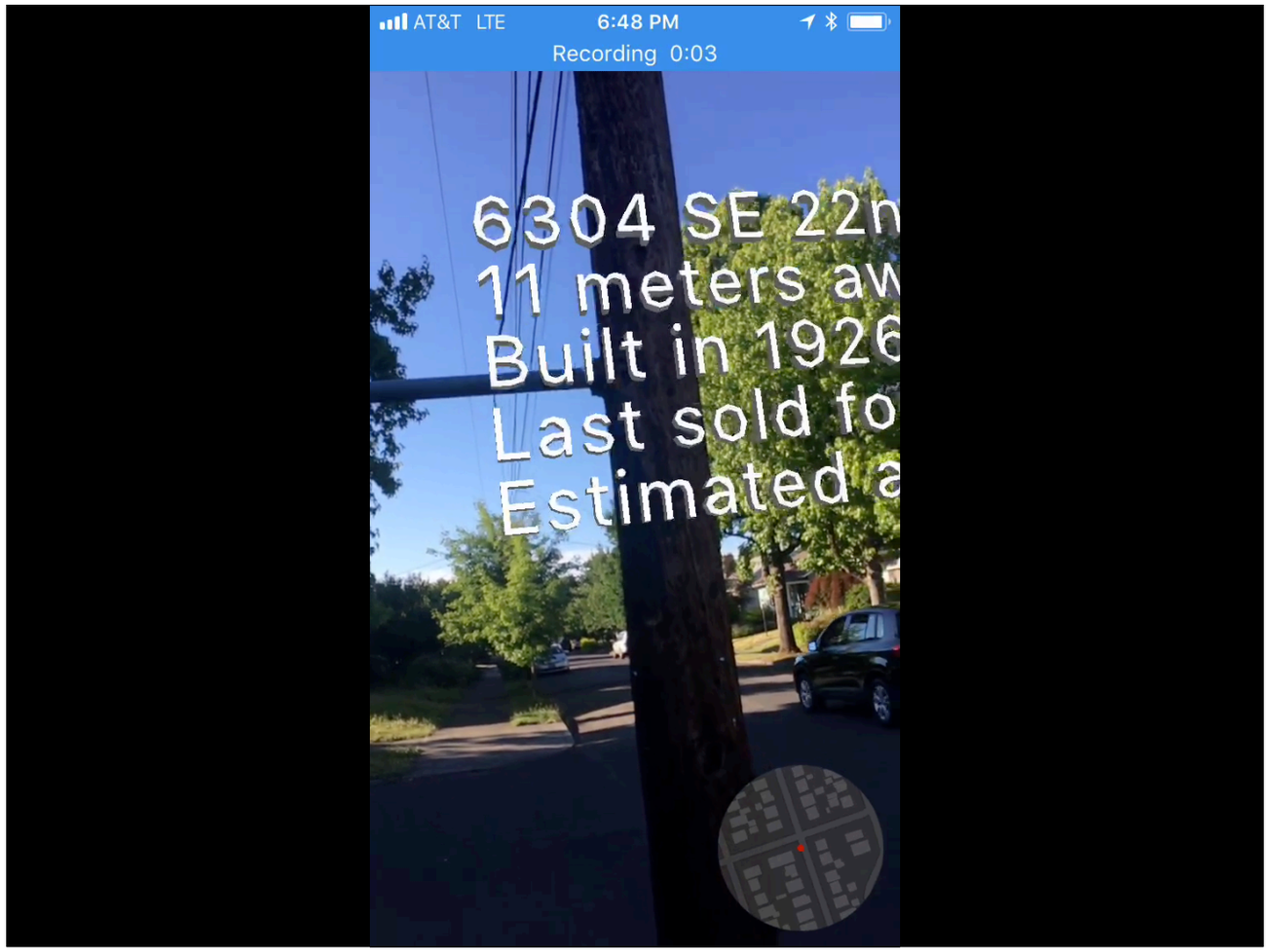


AUGMENTED REALITY!

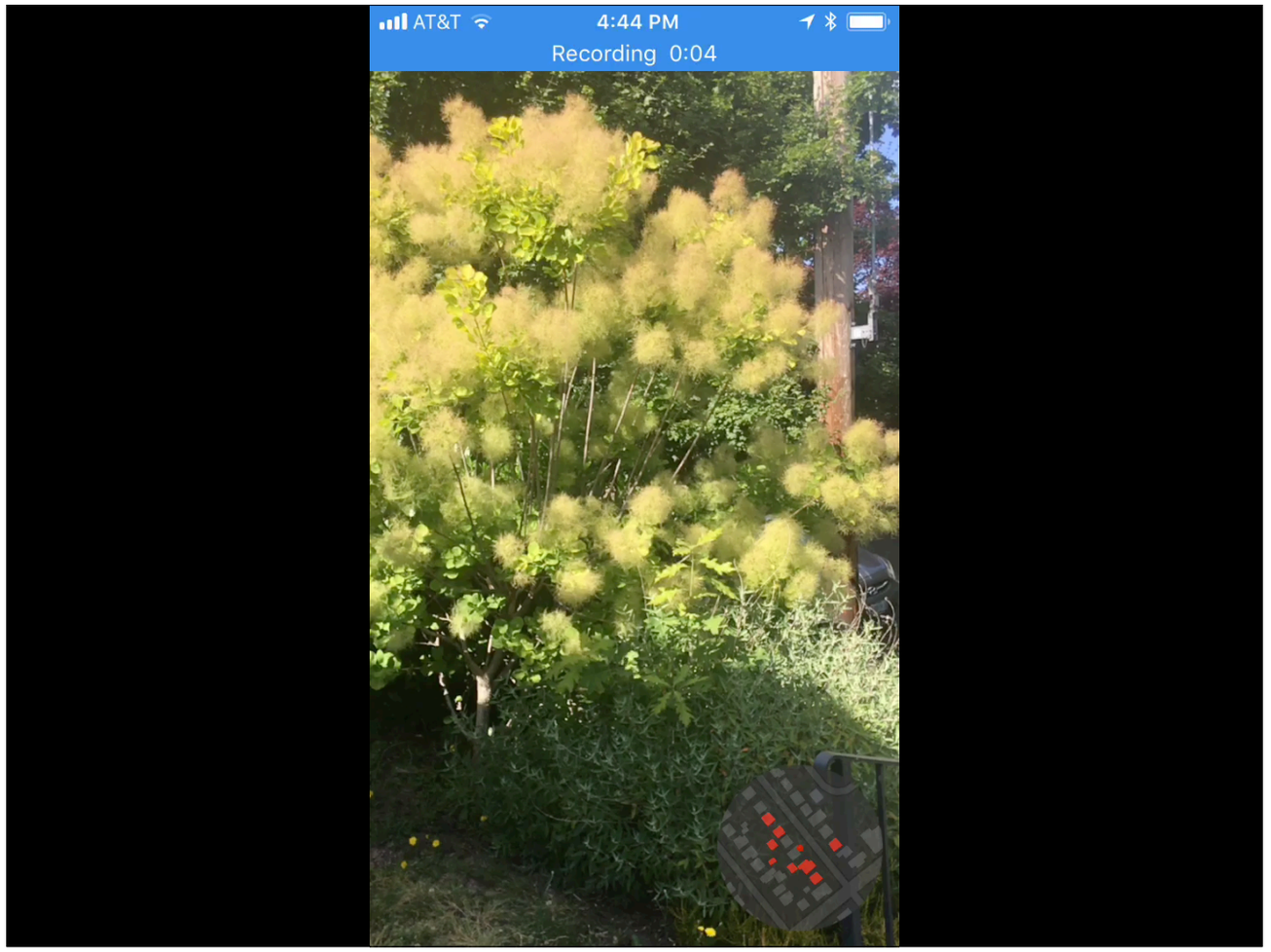
- I'm going to guess that you have heard some buzz about this
- Jonathan talked about the heavy lifting that Apple does for us
- I'm going to talk about the intersection of three kits:
  - SpriteKit: 2D content (sprites)
  - SceneKit: 3D content (solids)
  - ARKit: rendering graphics content into the real world
- This is not about Swift specifically
  - But Swift's elegant syntax makes the code pretty clear
  - So I've got a good number of jump-right-in code examples

SCOPE OF THIS TALK

- This is a broad overview
  - There are many, many more facets to each of these kits than we can get into today
- You'll leave here knowing the working parts and where to go for more
  - This is a talk about the higher-level *how*
- You'll see examples in each of the rendering kits separately, then put into AR
- I am an interested dabbler, not (yet) building production AR apps


AT&T LTE
6:48 PM
Recording 0:03
6304 SE 22n
11 meters aw
Built in 1926
Last sold fo
Estimated a

Video A



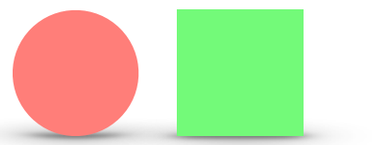
Video B

QUICK INTERLUDE ABOUT AR

- AR has been explored and researched for over 20 years
- I highly encourage you to check out the work of Mark Billinghurst of the University of South Australia
  - Academic papers, Twitter, Medium, books(?!)
- The irony here is that the iOS device exists *between* us and the real world for AR currently
  - Revolution of smartphones was *removing* abstraction
- But there are still, I think, many compelling use cases
  - Archaeology & history of place
  - Surgical & medical
  - Mechanical dissection

COMMON CONCEPTS FOR BOTH SPRITEKIT & SCENEKIT

- First of all, let's get our confusing prefixes out of the way
  - SK: SpriteKit
  - SCN: SceneKit
- Both support external assets (images, textures, models)
- Both support physics modeling (gravity, collisions, inertia, particles)
- Both build up scenes *(sigh…)*
- Both rely on a *node graph*
  - Spatial relationships between objects
  - All nodes descend from the *root node*
  - Position specified in 2D or 3D vectors

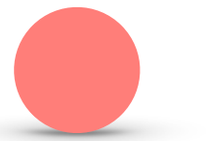


SPRITEKIT

SPRITEKIT: 2D SCENES

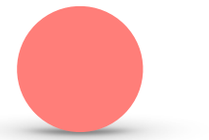
- Base view class: SKView
  - Can be layered with other UIView derivatives
  - Or can be built up in complexity
    - Example: SKView-based buttons that have physics properties when tapped
- Can be interrelated with physics or even joints
  - Think about old-school Mario jumping & squashing
  - Good use case for joints is a body with literal joints
- Supports positional audio (see SKAudioNode)
  - Example: character moving in from right

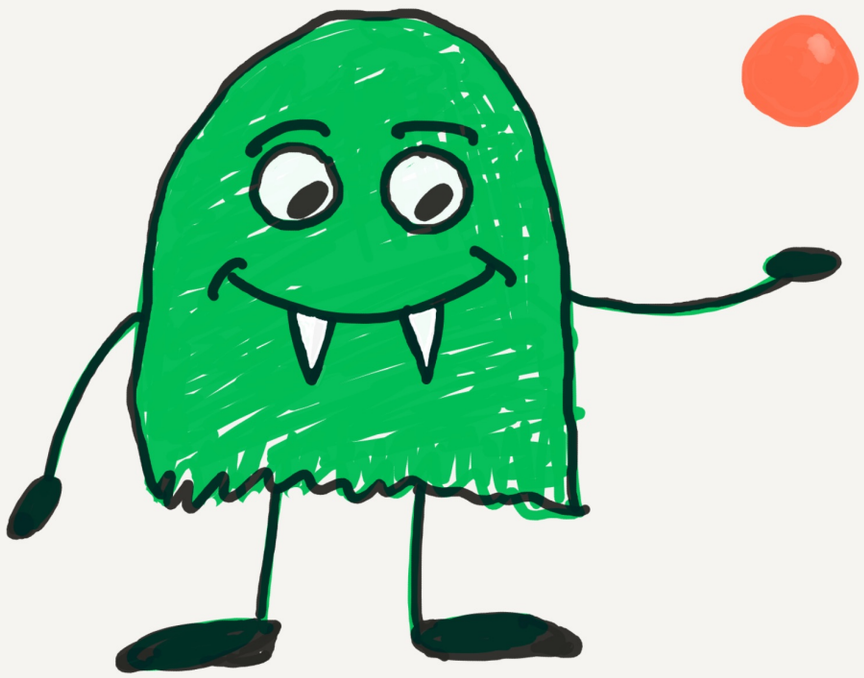
SPRITEKIT BY EXAMPLE



SPRITEKIT BY EXAMPLE: THE BUILD

1. Create the view (SKView)
2. Load an initial, empty scene (SKScene)
3. Add sprites using the node graph (SKSpriteNode)
4. Add some physics behavior (SKPhysicsBody)
5. Add interactivity to kick off physics (UIGestureRecognizer)





```swift
import UIKit
import SpriteKit

class ViewController: UIViewController {

    var spriteView: SKView!
    var monster: SKNode!
    var ball: SKNode!

    override func viewDidLoad() {
        super.viewDidLoad()
        setupSprites()
    }
```

```
func setupSprites() {
    spriteView = SKView()
    spriteView.frame = view.bounds
    view.addSubview(spriteView)
```

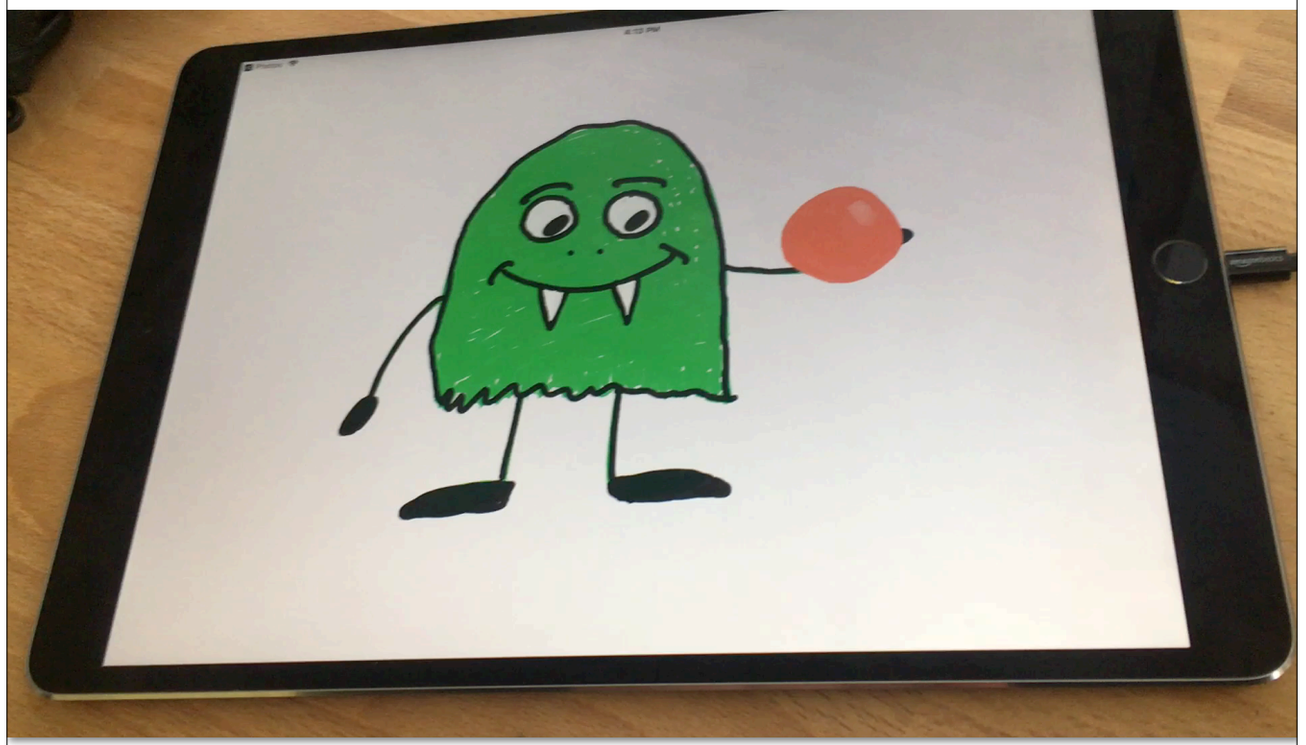
```swift
let scene = SKScene(size: spriteView.bounds.size)
scene.backgroundColor = .white
scene.anchorPoint = CGPoint(x: 0.5, y: 0.5)
spriteView.presentScene(scene)
```

```
monster = SKSpriteNode(imageNamed: "monster.png")
monster.position = CGPoint(x: 0, y: 0)
monster.setScale(0.5)
scene.addChild(monster)

ball = SKSpriteNode(imageNamed: "ball.png")
ball.position = CGPoint(x: 250, y: 70)
ball.setScale(0.75)
scene.addChild(ball)
```

```
ball.physicsBody = SKPhysicsBody(circleOfRadius:
    max(ball.frame.size.width, ball.frame.size.height) / 2)
ball.physicsBody!.affectedByGravity = true
ball.physicsBody!.isDynamic = false
```

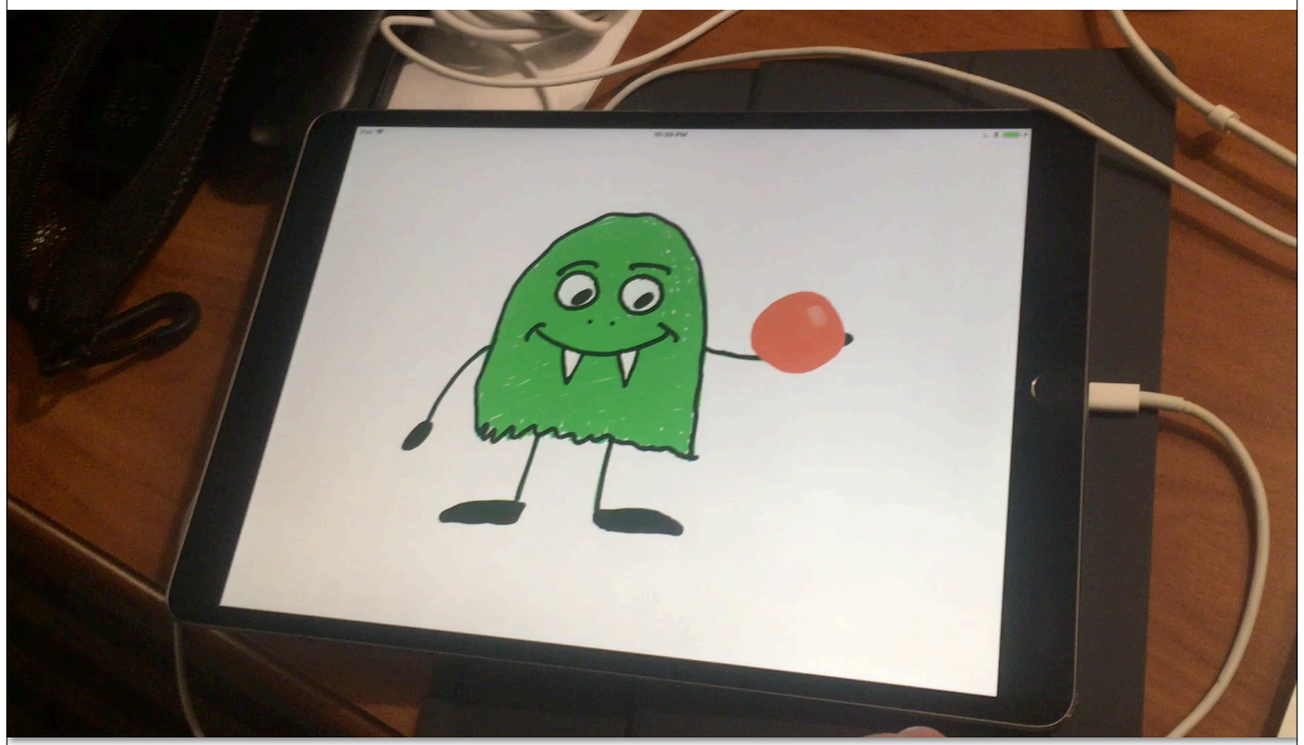
```
spriteView.addGestureRecognizer(UITapGestureRecognizer(target: self, action:
    #selector(handleTap(_:))))
```

```
@objc func handleTap(_ tap: UITapGestureRecognizer) {
    let spin = SKAction.applyAngularImpulse(-0.1, duration: 0.5)
    ball.run(spin)
    ball.physicsBody!.isDynamic = true
}
```



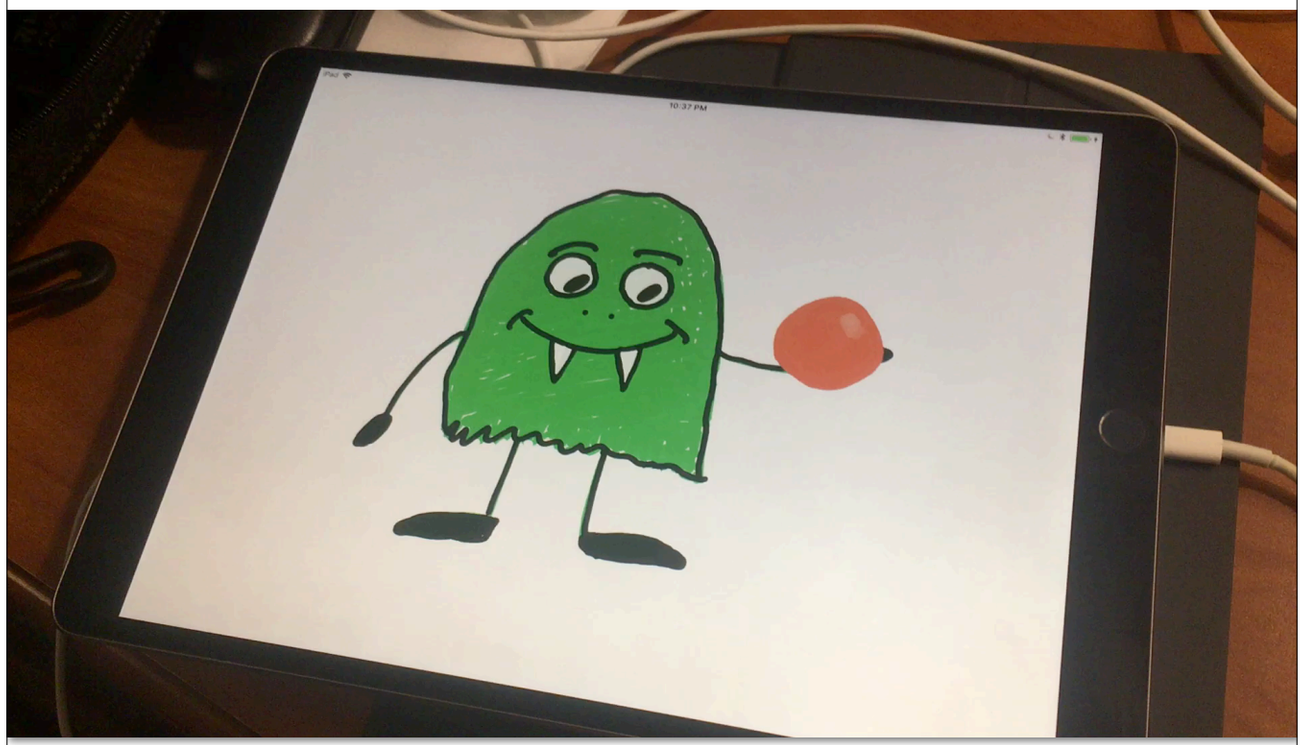
Video C

```swift
let floor = SKSpriteNode()
floor.size = CGSize(width: 5000, height: 1)
floor.position = CGPoint(x: ball.position.x - 500, y: ball.position.y - 350)
spriteView.scene!.addChild(floor)

floor.physicsBody = SKPhysicsBody(rectangleOf: floor.size)
floor.physicsBody!.isDynamic = false
```



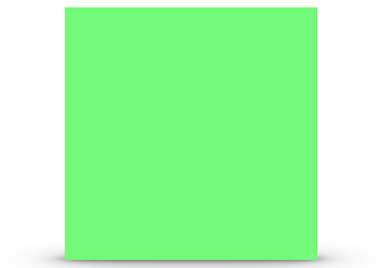
Video D

```
ball.physicsBody = SKPhysicsBody(circleOfRadius:
    max(ball.frame.size.width, ball.frame.size.height) / 2)
ball.physicsBody!.affectedByGravity = true
ball.physicsBody!.usesPreciseCollisionDetection = true
ball.physicsBody!.restitution = 0.75
ball.physicsBody!.isDynamic = false
```



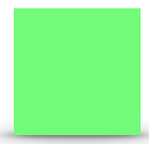
Video E

SCENEKIT

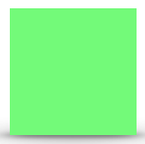


SCENEKIT: 3D SCENES

- Base view class: SCNView
  - Can also be layered with other UIView derivatives
- Can be interrelated with physics
  - Example: ping pong ball projectile striking a target
- Also supports positional audio (see SCNAudioSource)
  - Example: hearing a monster behind you

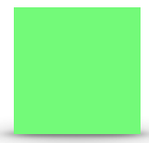


SCENEKIT BY EXAMPLE



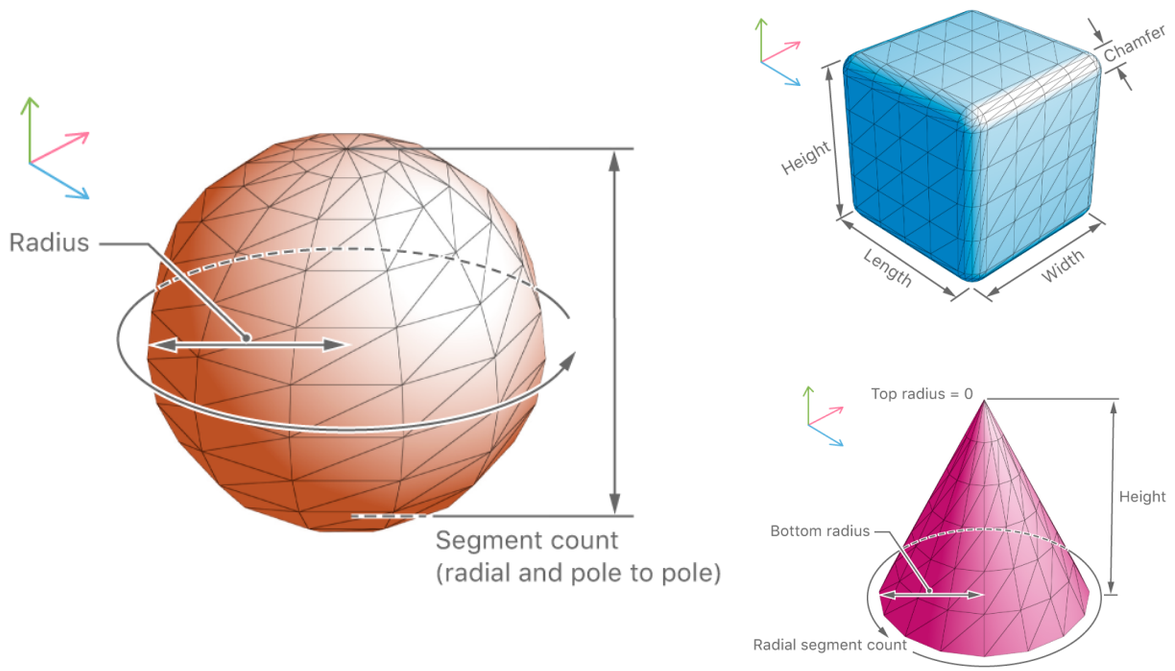
SCENEKIT BY EXAMPLE: THE BUILD

1. Create the view (SCNView)
2. Load an initial, empty scene (SCNScene)
3. Add geometries using the node graph (SCNNode, SCNGeometry & subclasses)
4. Add some animation (CABasicAnimation)
5. Add interactivity to stop & start animation (UIGestureRecognizer)

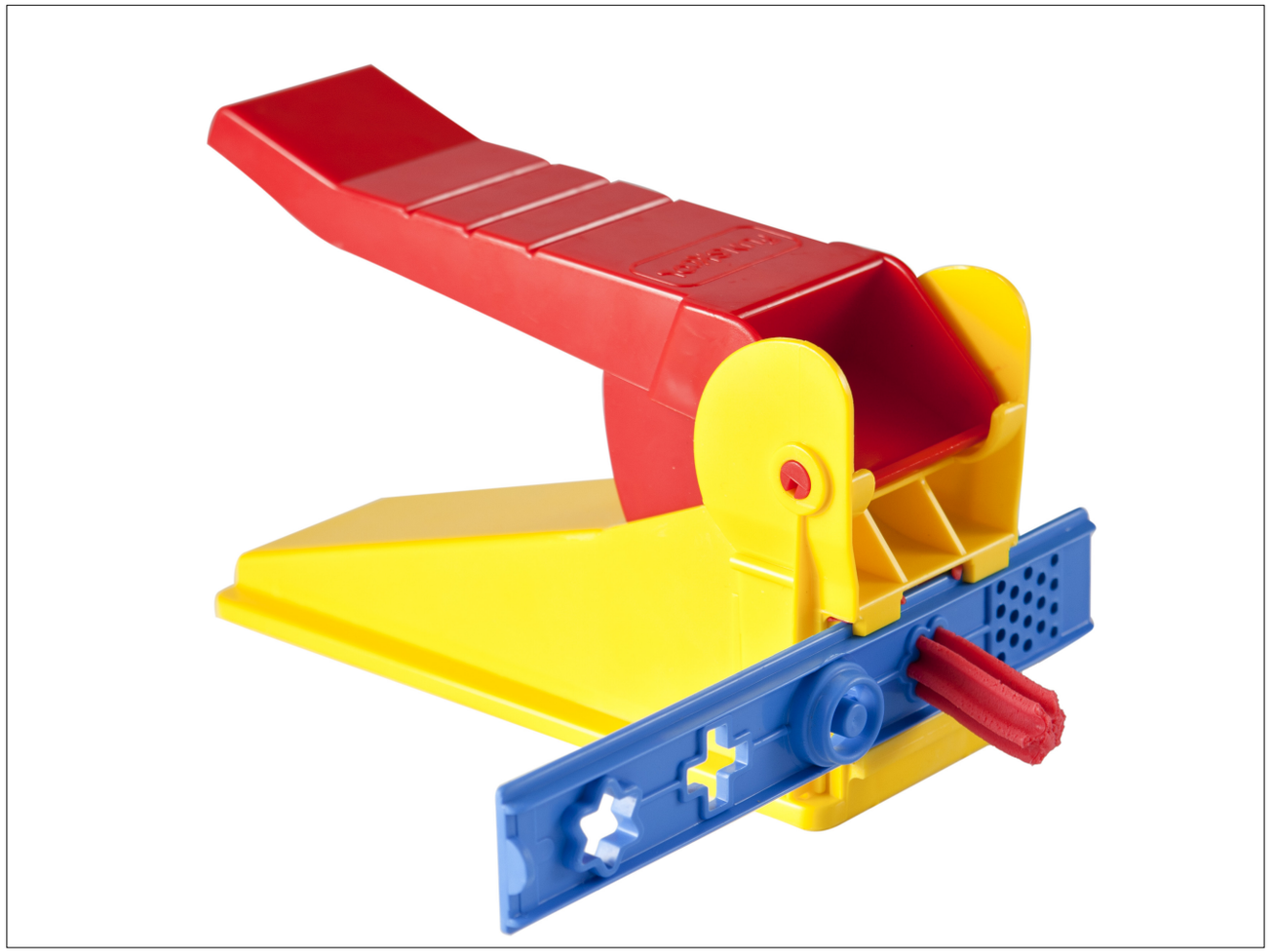


SCENEKIT GEOMETRIES

- All of your favorite shapes are there
  - (SCN)Plane, Box, Sphere, Pyramid, Cone, Cylinder, Capsule, Tube, Torus
- You can extrude your own 2D shapes
  - SCNShape, SCNText
- You can also bring in external models!
  - Load a model into an SCNScene, then fetch its root node



SceneKit built-in shapes

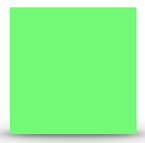



TEXT

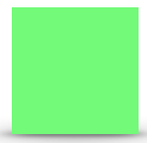
External 3D models (in many formats)

```swift
import UIKit
import SceneKit

class ViewController: UIViewController {

    var sceneView: SCNView!
    var earth: SCNNode!
    var moon: SCNNode!

    override func viewDidLoad() {
        super.viewDidLoad()
        setupScene()
    }
```

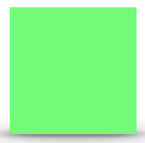
```swift
func setupScene() {
    sceneView = SCNView()
    sceneView.frame = view.bounds
    sceneView.backgroundColor = .black
    sceneView.allowsCameraControl = true
    view.addSubview(sceneView)

    let scene = SCNScene()
    sceneView.scene = scene
```



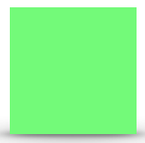
```
let earthGeometry = SCNSphere(radius: 10)
earth = SCNNode(geometry: earthGeometry)
earth.position = SCNVector3Make(0, 0, 0)
scene.rootNode.addChildNode(earth)
```



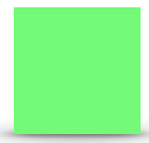
```
let moonRotator = SCNNode()
moonRotator.position = SCNVector3Make(0, 0, 0)
scene.rootNode.addChildNode(moonRotator)
```



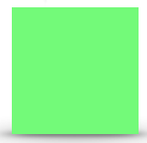
```
let moonGeometry = SCNSphere(radius: 2.5)
moon = SCNNode(geometry: moonGeometry)
moon.position = SCNVector3Make(50, 0, 0)
moonRotator.addChildNode(moon)
```



```swift
let light = SCNNode()
light.position = SCNVector3Make(-500, 0, 0)
light.light = SCNLight()
light.light!.type = .omni
scene.rootNode.addChildNode(light)
```

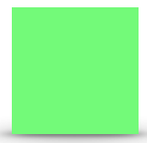


```swift
let moonOrbit = CABasicAnimation(keyPath: "rotation")
moonOrbit.fromValue = moonRotator.rotation
moonOrbit.toValue = SCNVector4Make(0, 1, 0, 2 * .pi)
moonOrbit.duration = 3 * 28
moonOrbit.repeatCount = .infinity
moonRotator.addAnimation(moonOrbit, forKey: "rotation")

let earthRotation = CABasicAnimation(keyPath: "rotation")
earthRotation.fromValue = earth.rotation
earthRotation.toValue = SCNVector4Make(0, 1, 0, 2 * .pi)
earthRotation.duration = 3
earthRotation.repeatCount = .infinity
earth.addAnimation(earthRotation, forKey: "rotation")
```

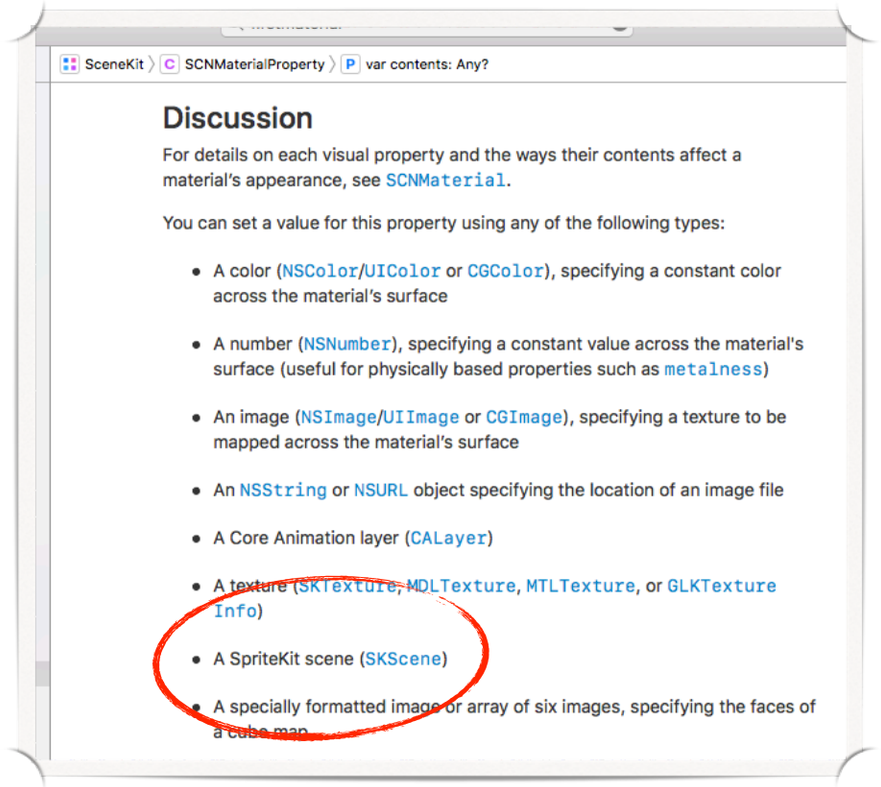


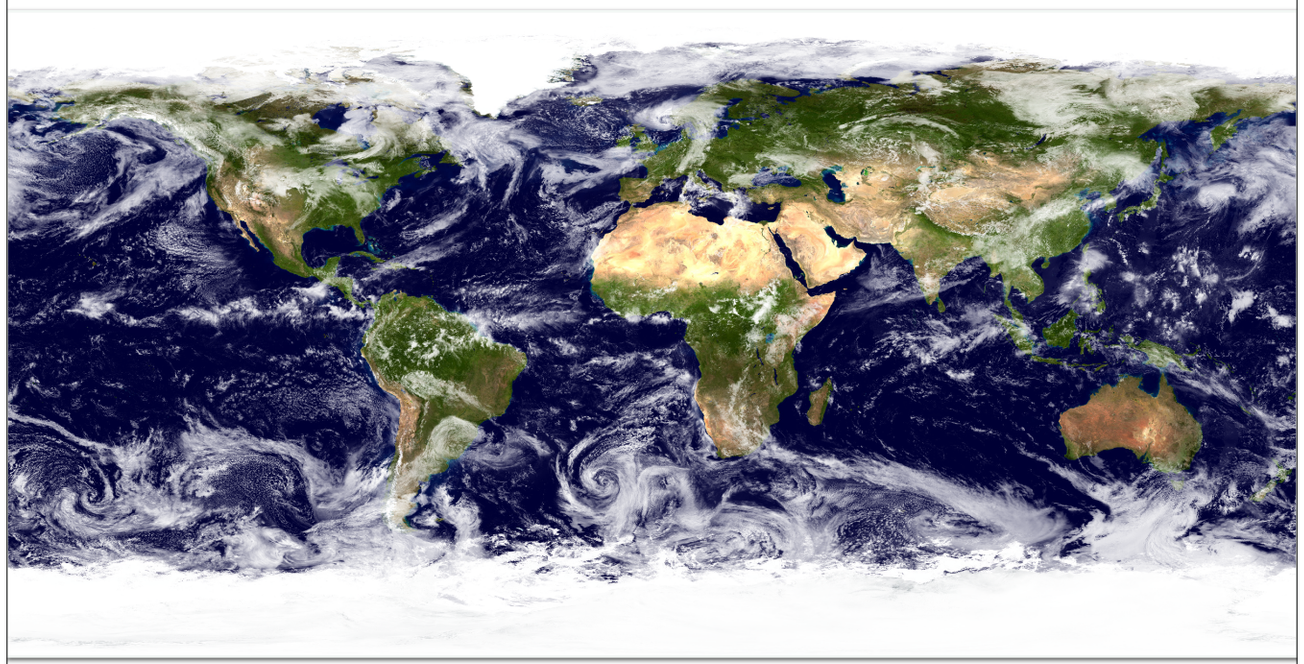
```
sceneView.addGestureRecognizer(UITapGestureRecognizer(target: self, action:
    #selector(handleTap(_:))))
```

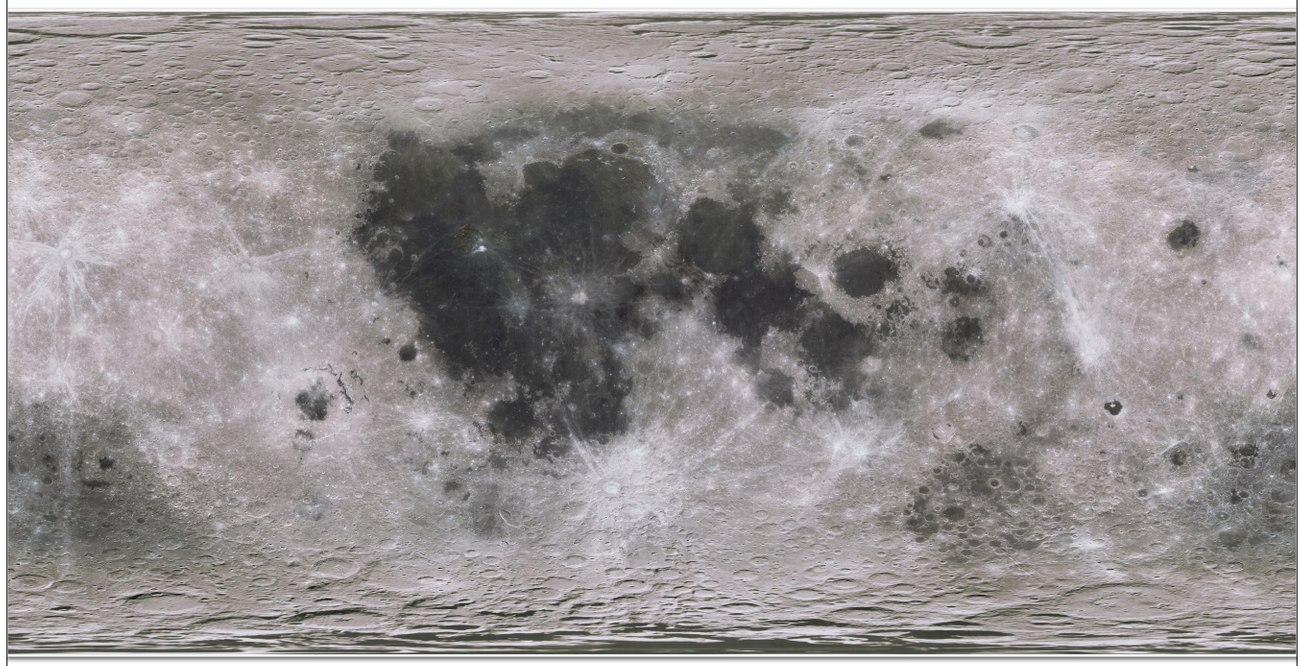
```
@objc func handleTap(_ tap: UITapGestureRecognizer) {
    sceneView.scene!.isPaused = !sceneView.scene!.isPaused
}
```



Video F

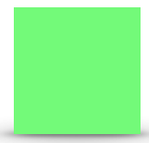

SceneKit
SCNMaterialProperty
var contents: Any?
Discussion
For details on each visual property and the ways their contents affect a material's appearance, see SCNMaterial.
You can set a value for this property using any of the following types:
A color (NSColor/UIColor or CGColor), specifying a constant color across the material's surface
A number (NSNumber), specifying a constant value across the material's surface (useful for physically based properties such as metalness)
An image (NSImage/UIImage or CGImage), specifying a texture to be mapped across the material's surface
An NSString or NSURL object specifying the location of an image file
A Core Animation layer (CALayer)
A texture (SKTexture, MDLTexture, MTLTexture, or GLKTexture Info)
A SpriteKit scene (SKScene)
A specially formatted image or array of six images, specifying the faces of a cube map

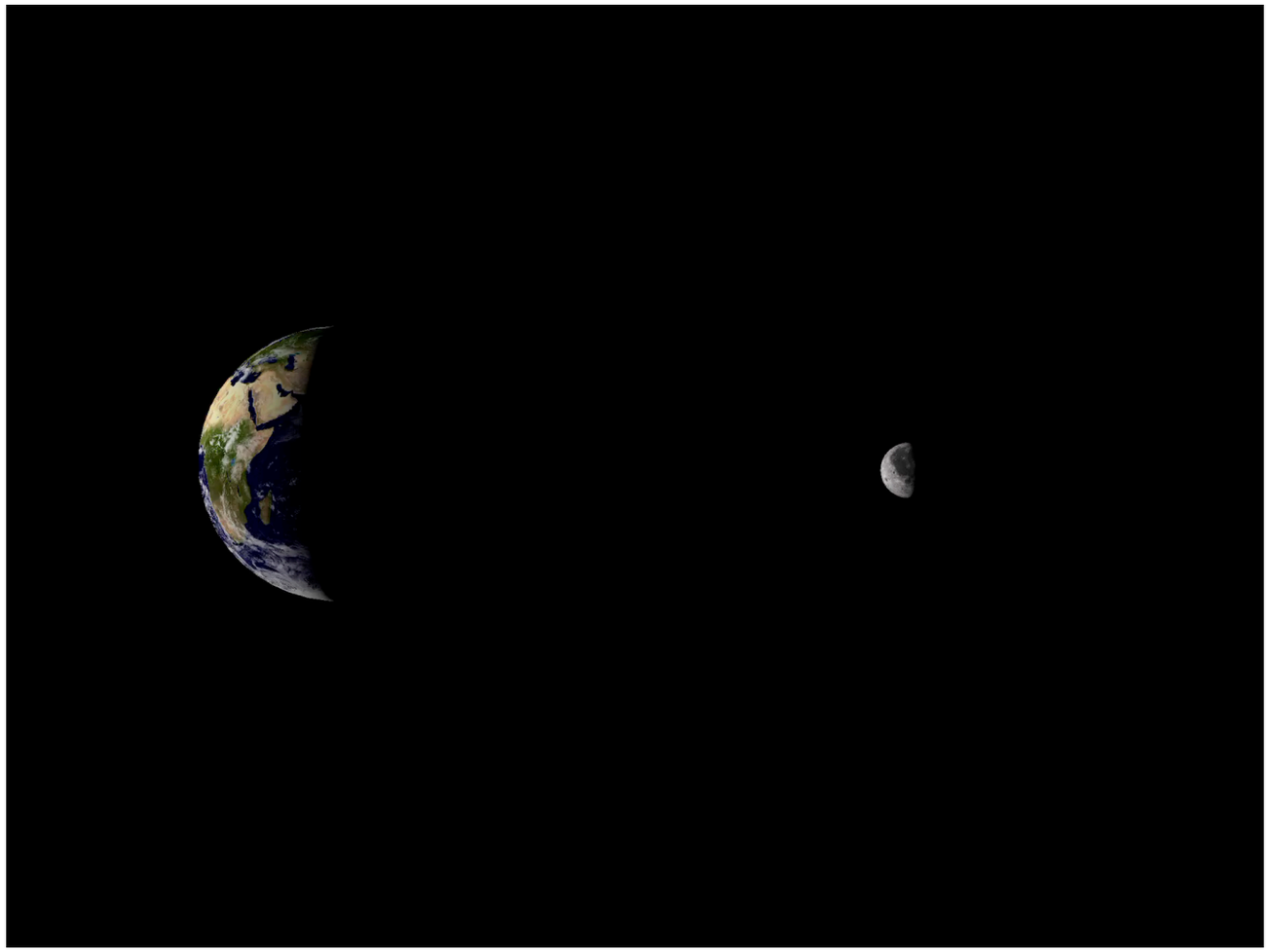




```swift
let earthGeometry = SCNSphere(radius: 10)
earthGeometry.materials.first!.diffuse.contents = UIImage(named: "earth.png")
earth = SCNNode(geometry: earthGeometry)
earth.position = SCNVector3Make(0, 0, 0)
scene.rootNode.addChildNode(earth)
```

```swift
let moonGeometry = SCNSphere(radius: 2.5)
moonGeometry.materials.first!.diffuse.contents = UIImage(named: "moon.jpg")
moon = SCNNode(geometry: moonGeometry)
moon.position = SCNVector3Make(50, 0, 0)
moonRotator.addChildNode(moon)
```





Video G

ARKIT

ARKIT IS NOT A LARGE FRAMEWORK

- There aren't a ton of classes (currently about a dozen)
  - Session, configuration, anchors, rendering views, hit testing, and camera details (frames, lighting estimates)
- Magic ✨: rendering coordinate systems are now in meters!
- More magic ✨✨: the combination of ARKit's camera processing and *your content*
  - Which is a lot like most apps
  - *How* you build is not as difficult as *what* you build

ARKIT BY EXAMPLE

ARKIT BY EXAMPLE: THE BUILD

1. Create the view (ARSKView or ARSCNView)
2. Load an initial, empty scene (SKScene or SCNScene)
3. Add objects using the node graph (sprites or geometries)
4. Add animations (SKAction or CABasicAnimation)
5. Add interactivity (UIGestureRecognizer)

UH, HOW DO WE PLACE ITEMS IN THE RIGHT PLACES?

https://www.flickr.com/photos/para_llm/33389226121

ANCHORS!

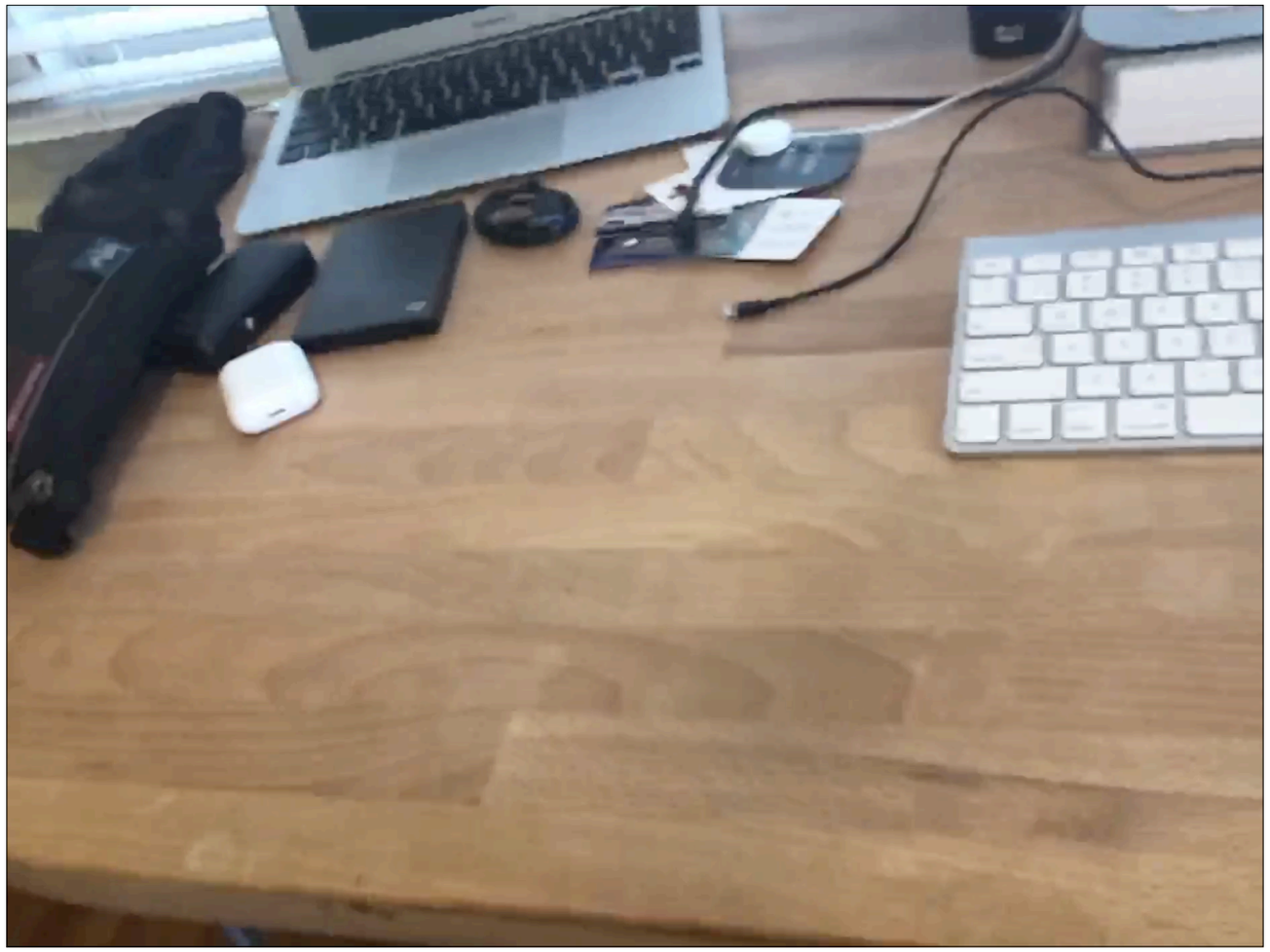
- ARAnchor is the key to proper AR content placement
  - Can be auto-detected (currently horizontal planes as ARPlaneAnchor or faces as ARFaceAnchor)
  - Can be manually added
    - By acting on hit testing query results
    - By externally determining placement (computer vision?)
- For Apple-provided, position & transform get updated & refined
- Use delegation pattern to provide content & respond to anchor add, update, and remove
  - Think of this like table cell delegation

```swift
import UIKit
import ARKit

class ViewController: UIViewController, ARSCNViewDelegate {

    var arView: ARSCNView!

    override func viewDidLoad() {
        super.viewDidLoad()
        setupARSession()
    }
```

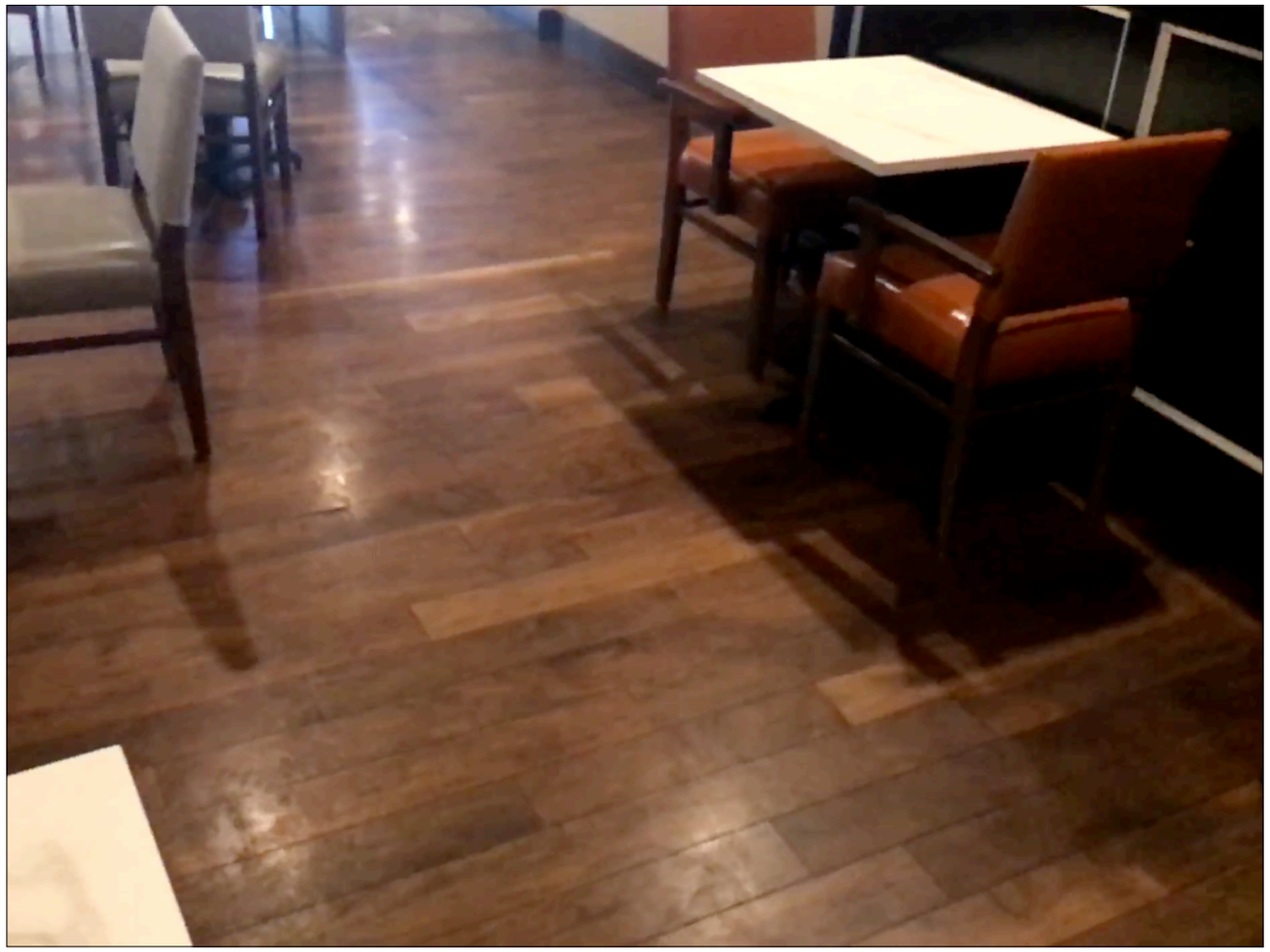
```swift
func setupARSession() {
    guard ARWorldTrackingConfiguration.isSupported else {
        fatalError("ARKit is not available on this device.")
    }

    arView = ARSCNView(frame: view.bounds)
    arView.autoenablesDefaultLighting = true
    arView.delegate = self
    view.addSubview(arView)
    let configuration = ARWorldTrackingConfiguration()
    arView.session.run(configuration, options: [])
```

```
    arView.addGestureRecognizer(UITapGestureRecognizer(target: self, action:
        #selector(handleTap(_:))))
}

@objc func handleTap(_ tap: UITapGestureRecognizer) {
    if let hit = arView.hitTest(tap.location(in: tap.view), types:
        [.estimatedHorizontalPlane]).first {
        let transform = hit.worldTransform
        let anchor = ARAnchor(transform: transform)
        arView.session.add(anchor: anchor)
    }
}
```

```swift
func renderer(_ renderer: SCNSceneRenderer, nodeFor anchor: ARAnchor) -> SCNNode? {
    let cube = SCNBox(width: 0.05, height: 0.05, length: 0.05, chamferRadius: 0)
    cube.firstMaterial!.diffuse.contents = UIColor.blue
    let node = SCNNode(geometry: cube)
    node.simdTransform = anchor.transform
    return node
}
```



Video H



Video I

LET'S RECAP

- Choose SpriteKit for 2D content and SceneKit for 3D content
  - Content can be cross-loaded between kits!
    - SpriteKit scenes can be applied as SceneKit textures
    - SceneKit geometries can be rendered into 2D SpriteKit content
- Learn and understand the node graph
- Understand AR anchors
  - And watch this space! Apple's really going to innovate here
- *How* you put things in AR isn't nearly as difficult as *what* you put there.
  - Go make some great apps!

THANK YOU!

CONTACT & RESOURCES

- http://justinmiller.io
  - I blog about tech & non-tech stuff
- http://github.com/incanus
  - I'll be putting source code from this talk online
- Twitter: @incanus77
- Apple's *Building Your First AR Experience* sample app
  - Great for visually understanding horizontal plane detection & anchor refinement over time