Enhance your world with ARKit

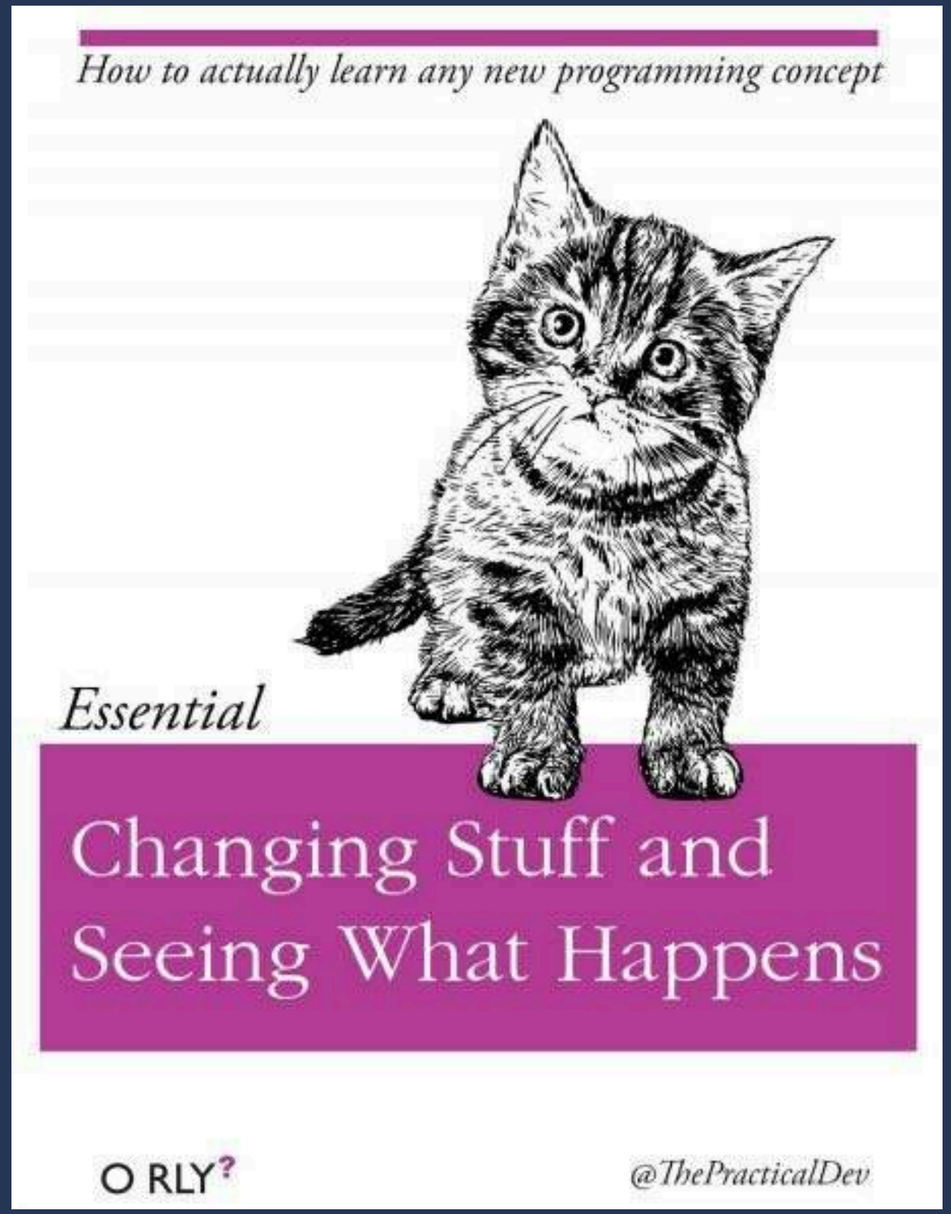
an introduction

Marius Constantinescu

@marius_const

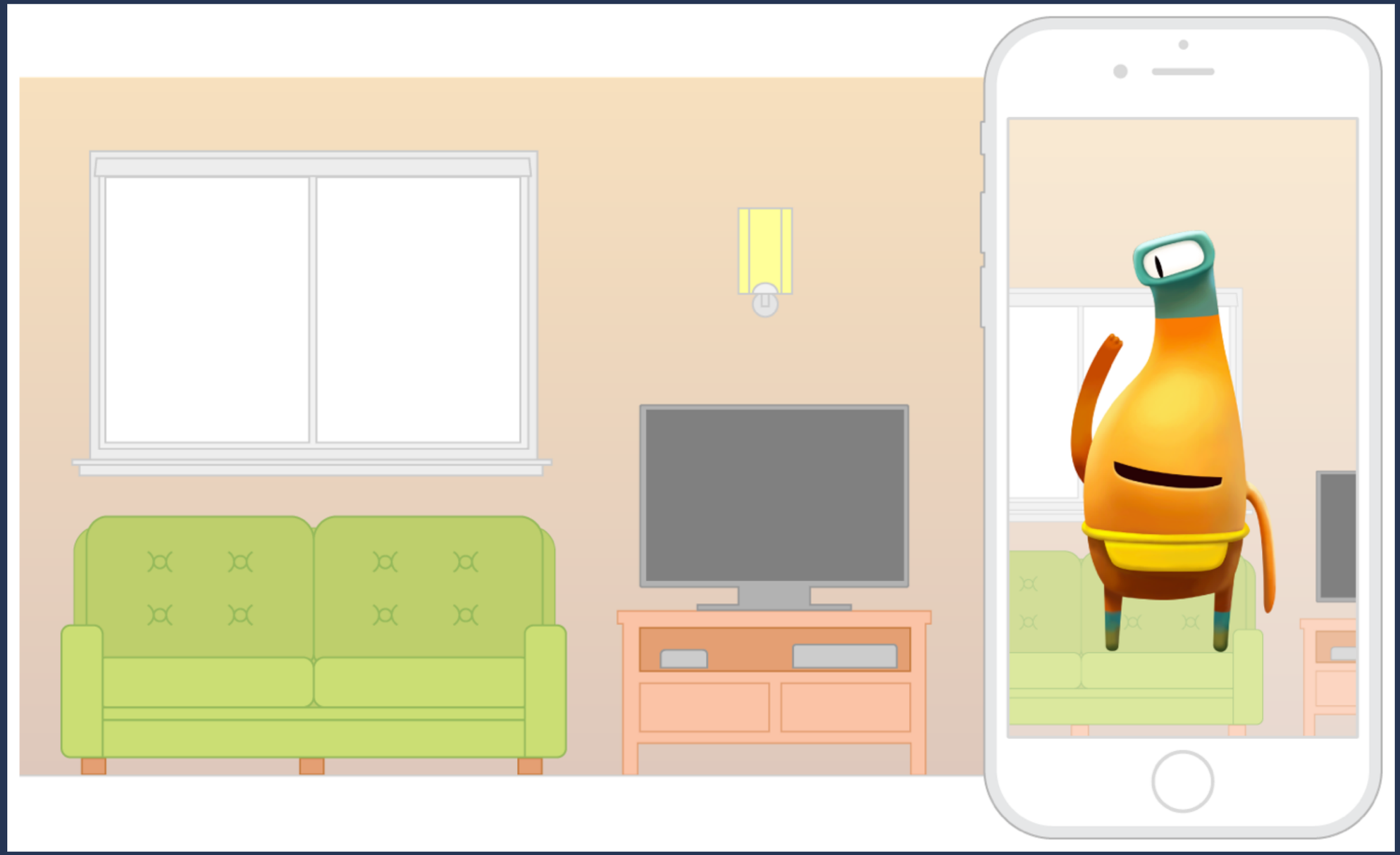
Greener Pastures

iOS Goodies

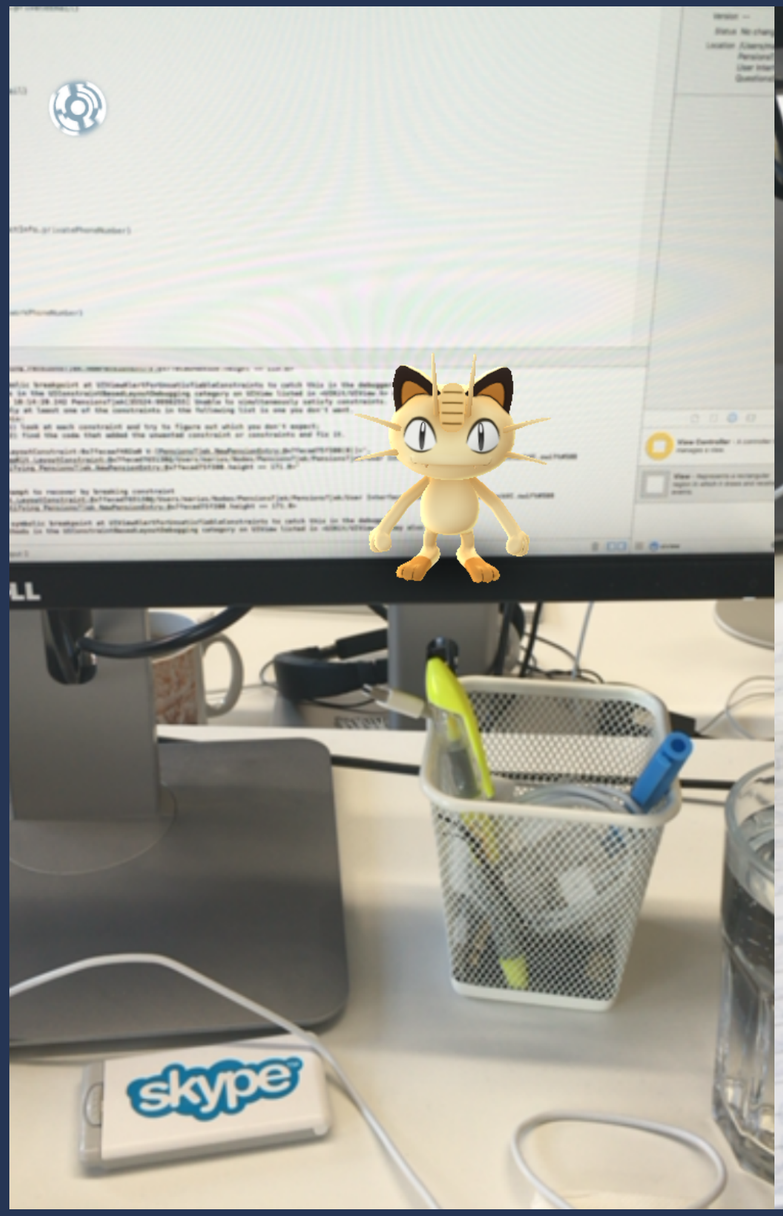


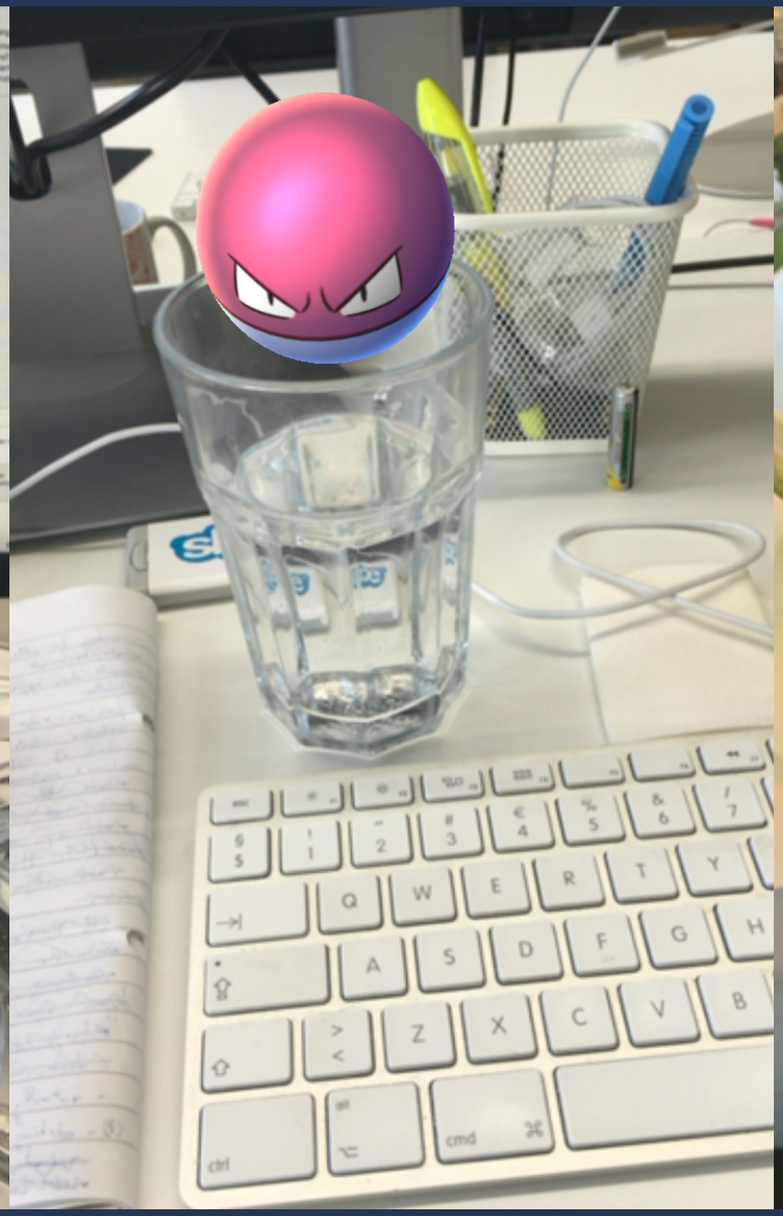
https://www.facebook.com/nixcraft/photos/a.431194973560553.114666.126000117413375/1856451711034865/?type=3&theater

What is Augmented Reality?



https://developer.apple.com/documentation/arkit/about_augmented_reality_and_arkit


skype


ctrl
cmd

What is ARKit?

Why I think it's huge?

from the WWDC 2017 Keynote

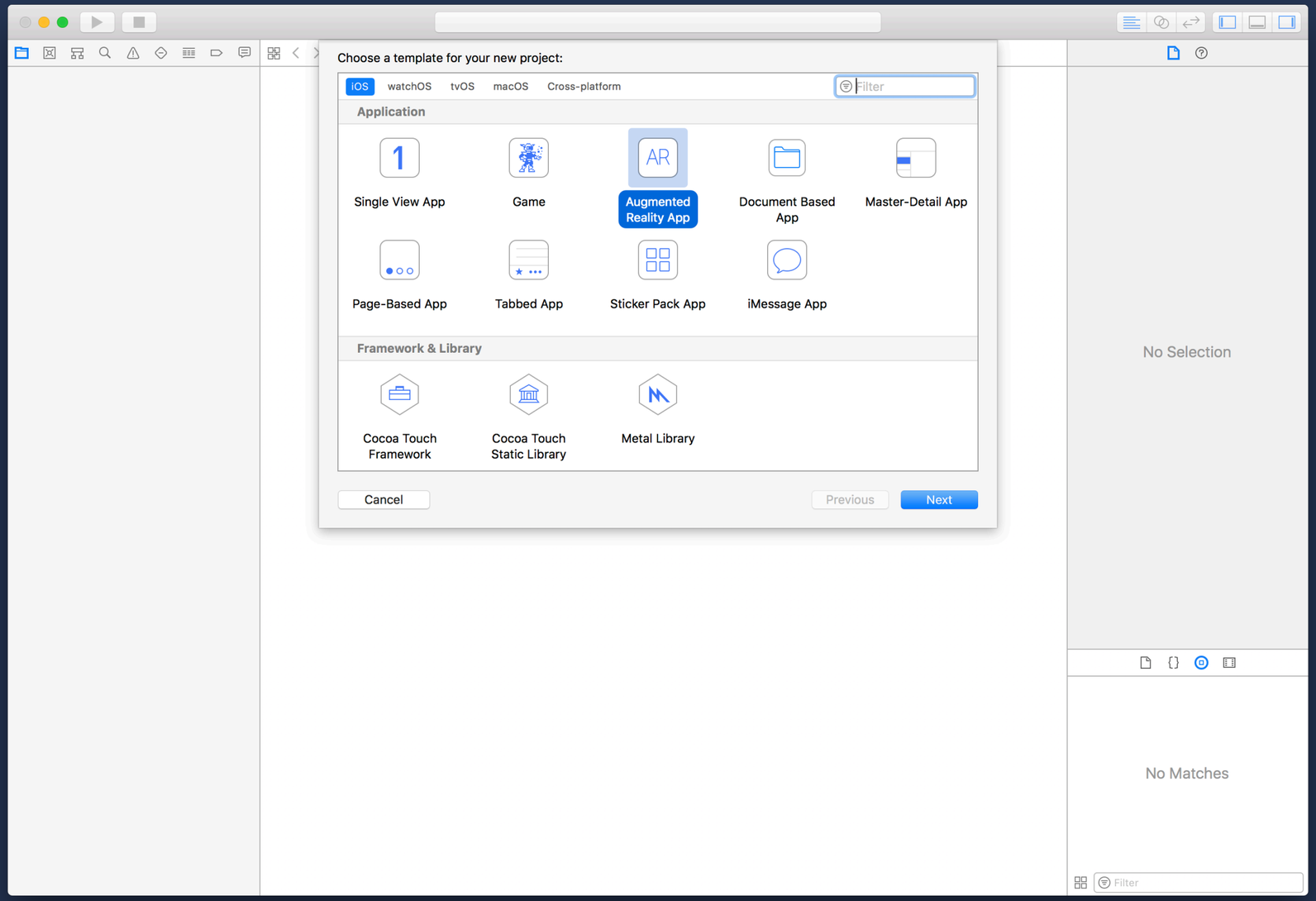
What does ARKit do?

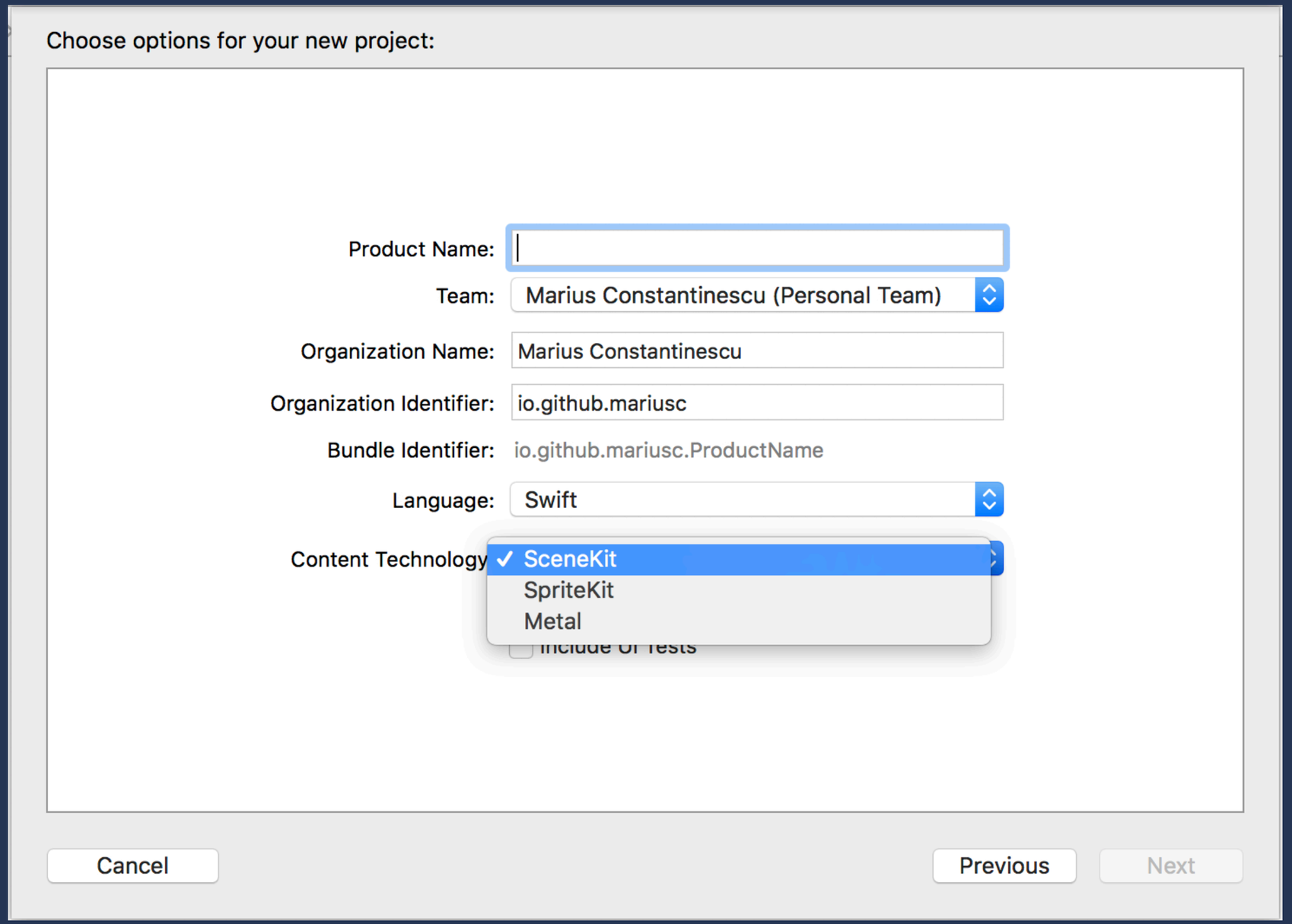
- tracking
- scene understanding
  - plane detection
  - hit test
  - light estimate
- rendering support for SpriteKit, SceneKit and Metal

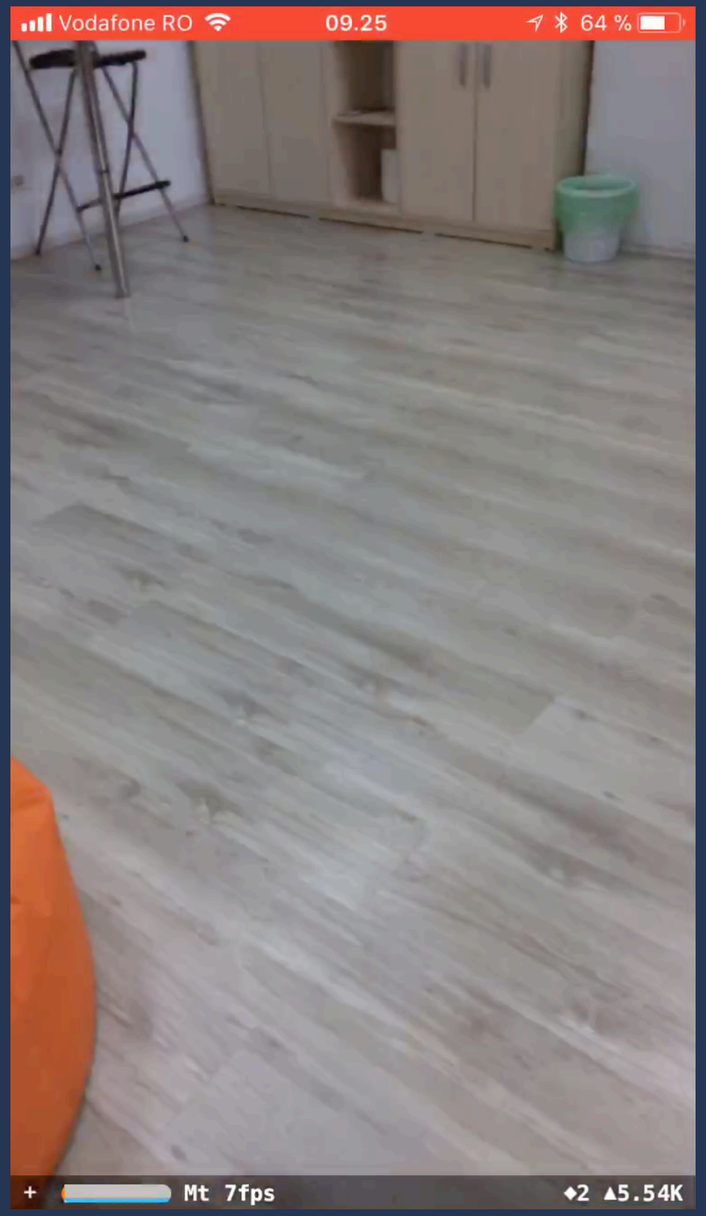
How does it do it?

- Visual Inertial Odometry system
- AVFoundation
- CoreMotion

How do we start?


Choose a template for your new project:
iOS
watchOS
tvOS
macOS
Cross-platform
Filter
Application
Single View App
Game
AR
Augmented Reality App
Document Based App
Master-Detail App
Page-Based App
Tabbed App
Sticker Pack App
iMessage App
Framework & Library
Cocoa Touch Framework
Cocoa Touch Static Library
Metal Library
Cancel
Previous
Next
No Selection
No Matches
Filter


Choose options for your new project:
Product Name:
Team:
Marius Constantinescu (Personal Team)
Organization Name:
Marius Constantinescu
Organization Identifier:
io.github.mariusc
Bundle Identifier:
io.github.mariusc.ProductName
Language:
Swift
Content Technology
SceneKit
SpriteKit
Metal
Cancel
Previous
Next


Vodafone RO
09.25
64 %
+
Mt 7fps
◆2 ▲5.54K

Let's look at the code a bit

```swift
class ViewController: UIViewController, ARSCNViewDelegate {

    @IBOutlet var sceneView: ARSCNView!

    override func viewDidLoad() {
        super.viewDidLoad()

        // Create a new scene
        let scene = SCNScene(named: "art.scnassets/ship.scn")!

        // Set the scene to the view
        sceneView.scene = scene
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)

        // Create a session configuration
        let configuration = ARWorldTrackingConfiguration()

        // Run the view's session
        sceneView.session.run(configuration)
    }
}
```

```swift
class ViewController: UIViewController, ARSCNViewDelegate {

    @IBOutlet var sceneView: ARSCNView!

    override func viewDidLoad() {
        super.viewDidLoad()

        // Create a new scene
        let scene = SCNScene(named: "art.scnassets/ship.scn")!

        // Set the scene to the view
        sceneView.scene = scene
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)

        // Create a session configuration
        let configuration = ARWorldTrackingConfiguration()

        // Run the view's session
        sceneView.session.run(configuration)
    }
}
```

```swift
class ViewController: UIViewController, ARSCNViewDelegate {

    @IBOutlet var sceneView: ARSCNView!

    override func viewDidLoad() {
        super.viewDidLoad()

        // Create a new scene
        let scene = SCNScene(named: "art.scnassets/ship.scn")!

        // Set the scene to the view
        sceneView.scene = scene
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)

        // Create a session configuration
        let configuration = ARWorldTrackingConfiguration()

        // Run the view's session
        sceneView.session.run(configuration)
    }
}
```

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

        // Set the view's delegate
        sceneView.delegate = self

        // Show statistics such as fps and timing information
        sceneView.showsStatistics = true

        // Create a new scene
        let scene = SCNScene(named: "art.scnassets/ship.scn")!

        // Set the scene to the view
        sceneView.scene = scene
    }
```

```swift
// Implement to create and configure nodes for anchors added to the view's session.
func renderer(_ renderer: SCNSceneRenderer, nodeFor anchor: ARAnchor) -> SCNNode?
func renderer(_ renderer: SCNSceneRenderer, didAdd node: SCNNode, for anchor: ARAnchor)
func renderer(_ renderer: SCNSceneRenderer, willUpdate node: SCNNode, for anchor: ARAnchor)
func renderer(_ renderer: SCNSceneRenderer, didUpdate node: SCNNode, for anchor: ARAnchor)
func renderer(_ renderer: SCNSceneRenderer, didRemove node: SCNNode, for anchor: ARAnchor)
```

Let's take some snapshots

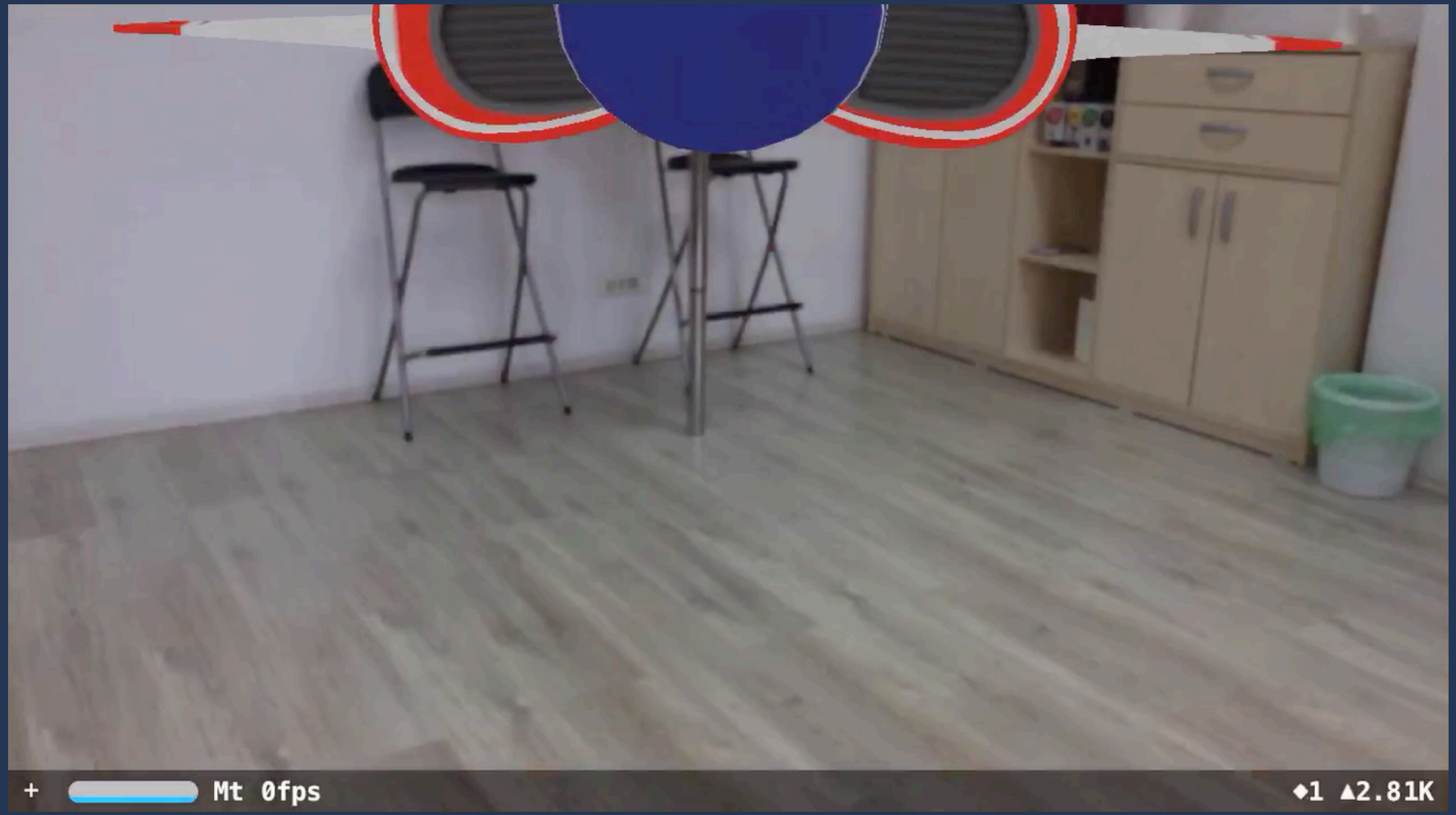
```swift
//...
let tgr = UITapGestureRecognizer(target: self, action: #selector(ViewController.handleTap(gestureRecognizer:)))
view.addGestureRecognizer(tgr)
//...

@objc func handleTap(gestureRecognizer: UITapGestureRecognizer) {

    guard let currentFrame = sceneView.session.currentFrame else { return }
    // Take a snapshot of the view
    let imagePlane = SCNPlane(width: sceneView.bounds.width/6000, height: sceneView.bounds.height/6000)
    imagePlane.firstMaterial?.diffuse.contents = sceneView.snapshot()
    imagePlane.firstMaterial?.lightingModel = .constant

    // Add the snapshot to the scene
    let planeNode = SCNNode(geometry: imagePlane)
    sceneView.scene.rootNode.addChildNode(planeNode)

    // Move the snapshot 10 cm in fromt of the user
    var translation = matrix_identity_float4x4
    translation.columns.3.z = -0.1
    planeNode.simdTransform = matrix_multiply(currentFrame.camera.transform, translation)
}
```

```swift
//...
let tgr = UITapGestureRecognizer(target: self, action: #selector(ViewController.handleTap(gestureRecognizer:)))
view.addGestureRecognizer(tgr)
//...

@objc func handleTap(gestureRecognizer: UITapGestureRecognizer) {

    guard let currentFrame = sceneView.session.currentFrame else { return }
    // Take a snapshot of the view
    let imagePlane = SCNPlane(width: sceneView.bounds.width/6000, height: sceneView.bounds.height/6000)
    imagePlane.firstMaterial?.diffuse.contents = sceneView.snapshot()
    imagePlane.firstMaterial?.lightingModel = .constant

    // Add the snapshot to the scene
    let planeNode = SCNNode(geometry: imagePlane)
    sceneView.scene.rootNode.addChildNode(planeNode)

    // Move the snapshot 10 cm in fromt of the user
    var translation = matrix_identity_float4x4
    translation.columns.3.z = -0.1
    planeNode.simdTransform = matrix_multiply(currentFrame.camera.transform, translation)
}
```

```swift
//...
let tgr = UITapGestureRecognizer(target: self, action: #selector(ViewController.handleTap(gestureRecognizer:)))
view.addGestureRecognizer(tgr)
//...

@objc func handleTap(gestureRecognizer: UITapGestureRecognizer) {

    guard let currentFrame = sceneView.session.currentFrame else { return }
    // Take a snapshot of the view
    let imagePlane = SCNPlane(width: sceneView.bounds.width/6000, height: sceneView.bounds.height/6000)
    imagePlane.firstMaterial?.diffuse.contents = sceneView.snapshot()
    imagePlane.firstMaterial?.lightingModel = .constant

    // Add the snapshot to the scene
    let planeNode = SCNNode(geometry: imagePlane)
    sceneView.scene.rootNode.addChildNode(planeNode)

    // Move the snapshot 10 cm in fromt of the user
    var translation = matrix_identity_float4x4
    translation.columns.3.z = -0.1
    planeNode.simdTransform = matrix_multiply(currentFrame.camera.transform, translation)
}
```

```swift
//...
let tgr = UITapGestureRecognizer(target: self, action: #selector(ViewController.handleTap(gestureRecognizer:)))
view.addGestureRecognizer(tgr)
//...

@objc func handleTap(gestureRecognizer: UITapGestureRecognizer) {

    guard let currentFrame = sceneView.session.currentFrame else { return }
    // Take a snapshot of the view
    let imagePlane = SCNPlane(width: sceneView.bounds.width/6000, height: sceneView.bounds.height/6000)
    imagePlane.firstMaterial?.diffuse.contents = sceneView.snapshot()
    imagePlane.firstMaterial?.lightingModel = .constant

    // Add the snapshot to the scene
    let planeNode = SCNNode(geometry: imagePlane)
    sceneView.scene.rootNode.addChildNode(planeNode)

    // Move the snapshot 10 cm in fromt of the user
    var translation = matrix_identity_float4x4
    translation.columns.3.z = -0.1
    planeNode.simdTransform = matrix_multiply(currentFrame.camera.transform, translation)
}
```

```
//...
let tgr = UITapGestureRecognizer(target: self, action: #selector(ViewController.handleTap(gestureRecognizer:)))
view.addGestureRecognizer(tgr)
//...

@objc func handleTap(gestureRecognizer: UITapGestureRecognizer) {

    guard let currentFrame = sceneView.session.currentFrame else { return }
    // Take a snapshot of the view
    let imagePlane = SCNPlane(width: sceneView.bounds.width/6000, height: sceneView.bounds.height/6000)
    imagePlane.firstMaterial?.diffuse.contents = sceneView.snapshot()
    imagePlane.firstMaterial?.lightingModel = .constant

    // Add the snapshot to the scene
    let planeNode = SCNNode(geometry: imagePlane)
    sceneView.scene.rootNode.addChildNode(planeNode)

    // Move the snapshot 10 cm in fromt of the user
    var translation = matrix_identity_float4x4
    translation.columns.3.z = -0.1
    planeNode.simdTransform = matrix_multiply(currentFrame.camera.transform, translation)
}
```


+
Mt 0fps
◆1 ▲2.81K

Let's see how it detects planes

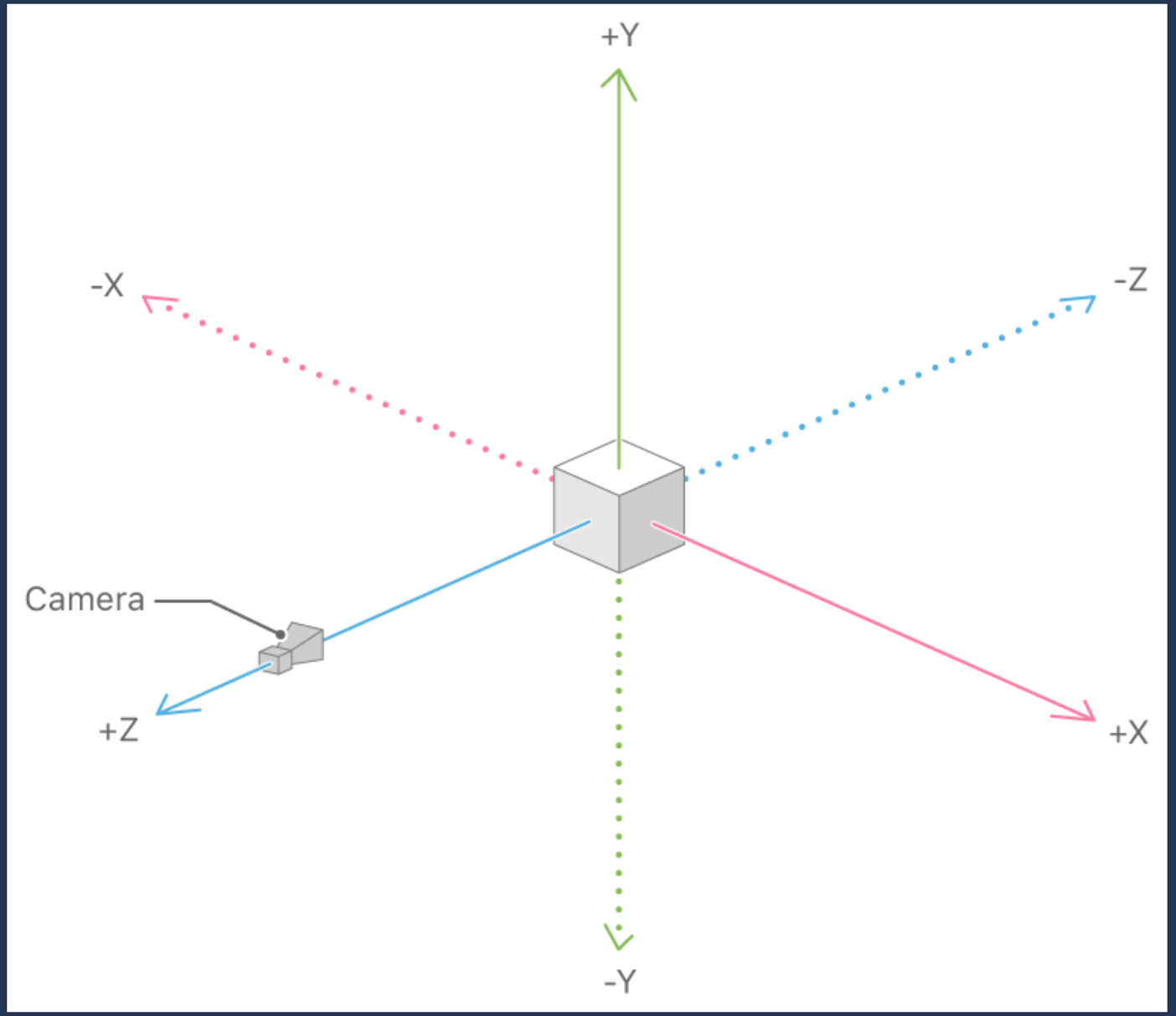
```
let configuration = ARWorldTrackingConfiguration()
configuration.planeDetection = .horizontal
sceneView.session.run(configuration)
```

ARKit 〉 ARConfiguration 〉 horizontal

```
26  extension ARWorldTrackingConfiguration {
27
28
29      /**
30       Option set indicating the type of planes to detect.
31       */
32      @available(iOS 11.0, *)
33      public struct PlaneDetection : OptionSet {
34
35          public init(rawValue: UInt)
36
37
38          /** Plane detection determines horizontal planes in the scene. */
39          public static var horizontal: ARWorldTrackingConfiguration.PlaneDetection { get }
40      }
41  }
42
```

```swift
func renderer(_ renderer: SCNSceneRenderer, didAdd node: SCNNode, for anchor: ARAnchor) {
    // Place content only for anchors found by plane detection.
    guard let planeAnchor = anchor as? ARPlaneAnchor else { return }

    // Create a SceneKit plane to visualize the plane anchor using its position and extent.
    let plane = SCNPlane(width: CGFloat(planeAnchor.extent.x), height: CGFloat(planeAnchor.extent.z))
    let planeNode = SCNNode(geometry: plane)
    planeNode.simdPosition = float3(planeAnchor.center.x, 0, planeAnchor.center.z)

    /*
     `SCNPlane` is vertically oriented in its local coordinate space, so
     rotate the plane to match the horizontal orientation of `ARPlaneAnchor`.
    */
    planeNode.eulerAngles.x = -.pi / 2

    // Make the plane visualization semitransparent to clearly show real-world placement.
    planeNode.opacity = 0.55
    plane.firstMaterial?.diffuse.contents = #imageLiteral(resourceName: "tiles")

    /*
     Add the plane visualization to the ARKit-managed node so that it tracks
     changes in the plane anchor as plane estimation continues.
    */
    node.addChildNode(planeNode)
}
```

```swift
func renderer(_ renderer: SCNSceneRenderer, didAdd node: SCNNode, for anchor: ARAnchor) {
    // Place content only for anchors found by plane detection.
    guard let planeAnchor = anchor as? ARPlaneAnchor else { return }

    // Create a SceneKit plane to visualize the plane anchor using its position and extent.
    let plane = SCNPlane(width: CGFloat(planeAnchor.extent.x), height: CGFloat(planeAnchor.extent.z))
    let planeNode = SCNNode(geometry: plane)
    planeNode.simdPosition = float3(planeAnchor.center.x, 0, planeAnchor.center.z)

    /*
     `SCNPlane` is vertically oriented in its local coordinate space, so
     rotate the plane to match the horizontal orientation of `ARPlaneAnchor`.
    */
    planeNode.eulerAngles.x = -.pi / 2

    // Make the plane visualization semitransparent to clearly show real-world placement.
    planeNode.opacity = 0.55
    plane.firstMaterial?.diffuse.contents = #imageLiteral(resourceName: "tiles")

    /*
     Add the plane visualization to the ARKit-managed node so that it tracks
     changes in the plane anchor as plane estimation continues.
    */
    node.addChildNode(planeNode)
}
```



https://developer.apple.com/documentation/scenekit/organizing_a_scene_with_nodes

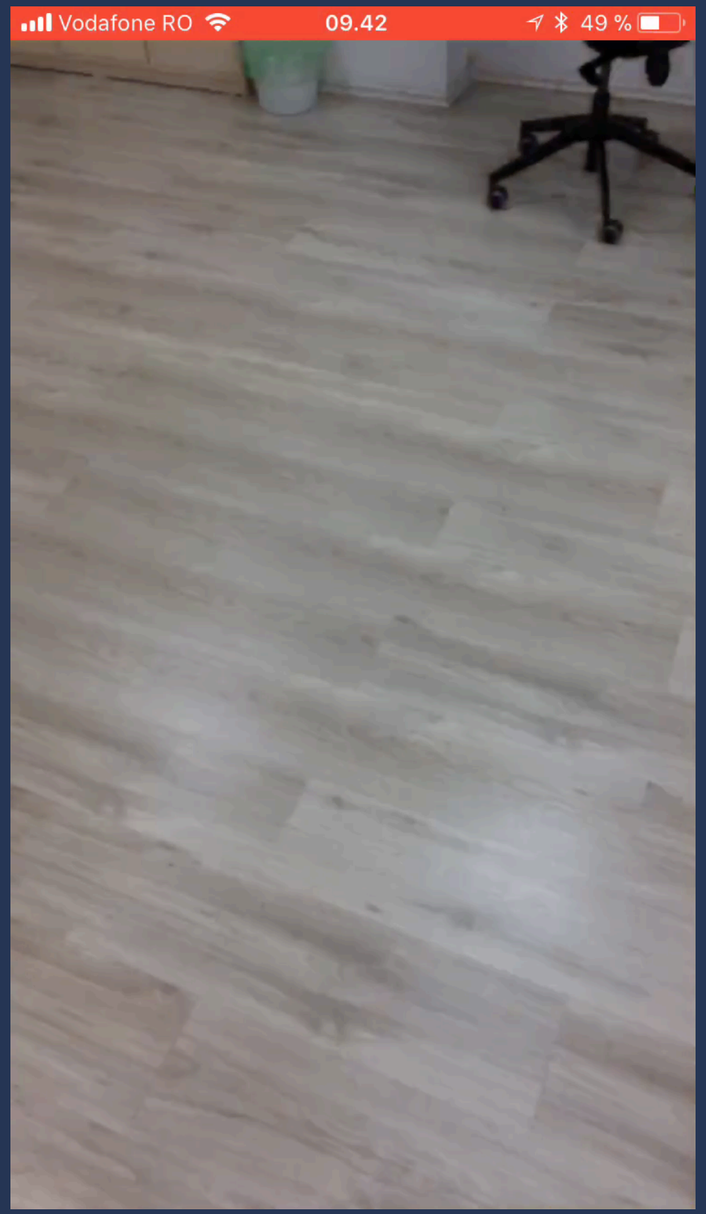
```swift
func renderer(_ renderer: SCNSceneRenderer, didAdd node: SCNNode, for anchor: ARAnchor) {
    // Place content only for anchors found by plane detection.
    guard let planeAnchor = anchor as? ARPlaneAnchor else { return }

    // Create a SceneKit plane to visualize the plane anchor using its position and extent.
    let plane = SCNPlane(width: CGFloat(planeAnchor.extent.x), height: CGFloat(planeAnchor.extent.z))
    let planeNode = SCNNode(geometry: plane)
    planeNode.simdPosition = float3(planeAnchor.center.x, 0, planeAnchor.center.z)

    /*
     `SCNPlane` is vertically oriented in its local coordinate space, so
     rotate the plane to match the horizontal orientation of `ARPlaneAnchor`.
    */
    planeNode.eulerAngles.x = -.pi / 2

    // Make the plane visualization semitransparent to clearly show real-world placement.
    planeNode.opacity = 0.55
    plane.firstMaterial?.diffuse.contents = #imageLiteral(resourceName: "tiles")

    /*
     Add the plane visualization to the ARKit-managed node so that it tracks
     changes in the plane anchor as plane estimation continues.
    */
    node.addChildNode(planeNode)
}
```

```swift
func renderer(_ renderer: SCNSceneRenderer, didAdd node: SCNNode, for anchor: ARAnchor) {
    // Place content only for anchors found by plane detection.
    guard let planeAnchor = anchor as? ARPlaneAnchor else { return }

    // Create a SceneKit plane to visualize the plane anchor using its position and extent.
    let plane = SCNPlane(width: CGFloat(planeAnchor.extent.x), height: CGFloat(planeAnchor.extent.z))
    let planeNode = SCNNode(geometry: plane)
    planeNode.simdPosition = float3(planeAnchor.center.x, 0, planeAnchor.center.z)

    /*
     `SCNPlane` is vertically oriented in its local coordinate space, so
     rotate the plane to match the horizontal orientation of `ARPlaneAnchor`.
    */
    planeNode.eulerAngles.x = -.pi / 2

    // Make the plane visualization semitransparent to clearly show real-world placement.
    planeNode.opacity = 0.55
    plane.firstMaterial?.diffuse.contents = #imageLiteral(resourceName: "tiles")

    /*
     Add the plane visualization to the ARKit-managed node so that it tracks
     changes in the plane anchor as plane estimation continues.
    */
    node.addChildNode(planeNode)
}
```

```swift
func renderer(_ renderer: SCNSceneRenderer, didAdd node: SCNNode, for anchor: ARAnchor) {
    // Place content only for anchors found by plane detection.
    guard let planeAnchor = anchor as? ARPlaneAnchor else { return }

    // Create a SceneKit plane to visualize the plane anchor using its position and extent.
    let plane = SCNPlane(width: CGFloat(planeAnchor.extent.x), height: CGFloat(planeAnchor.extent.z))
    let planeNode = SCNNode(geometry: plane)
    planeNode.simdPosition = float3(planeAnchor.center.x, 0, planeAnchor.center.z)

    /*
     `SCNPlane` is vertically oriented in its local coordinate space, so
     rotate the plane to match the horizontal orientation of `ARPlaneAnchor`.
    */
    planeNode.eulerAngles.x = -.pi / 2

    // Make the plane visualization semitransparent to clearly show real-world placement.
    planeNode.opacity = 0.55
    plane.firstMaterial?.diffuse.contents = #imageLiteral(resourceName: "tiles")

    /*
     Add the plane visualization to the ARKit-managed node so that it tracks
     changes in the plane anchor as plane estimation continues.
    */
    node.addChildNode(planeNode)
}
```

```swift
func renderer(_ renderer: SCNSceneRenderer, didAdd node: SCNNode, for anchor: ARAnchor) {
    // Place content only for anchors found by plane detection.
    guard let planeAnchor = anchor as? ARPlaneAnchor else { return }

    // Create a SceneKit plane to visualize the plane anchor using its position and extent.
    let plane = SCNPlane(width: CGFloat(planeAnchor.extent.x), height: CGFloat(planeAnchor.extent.z))
    let planeNode = SCNNode(geometry: plane)
    planeNode.simdPosition = float3(planeAnchor.center.x, 0, planeAnchor.center.z)

    /*
     `SCNPlane` is vertically oriented in its local coordinate space, so
     rotate the plane to match the horizontal orientation of `ARPlaneAnchor`.
    */
    planeNode.eulerAngles.x = -.pi / 2

    // Make the plane visualization semitransparent to clearly show real-world placement.
    planeNode.opacity = 0.55
    plane.firstMaterial?.diffuse.contents = #imageLiteral(resourceName: "tiles")

    /*
     Add the plane visualization to the ARKit-managed node so that it tracks
     changes in the plane anchor as plane estimation continues.
    */
    node.addChildNode(planeNode)
}
```

```swift
func renderer(_ renderer: SCNSceneRenderer, didUpdate node: SCNNode, for anchor: ARAnchor) {
        // Update content only for plane anchors and nodes matching the setup created in `renderer(_:didAdd:for:)`.
        guard let planeAnchor = anchor as?  ARPlaneAnchor,
            let planeNode = node.childNodes.first,
            let plane = planeNode.geometry as? SCNPlane
            else { return }

        // Plane estimation may shift the center of a plane relative to its anchor's transform.
        planeNode.simdPosition = float3(planeAnchor.center.x, 0, planeAnchor.center.z)

        /*
         Plane estimation may extend the size of the plane, or combine previously detected
         planes into a larger one. In the latter case, `ARSCNView` automatically deletes the
         corresponding node for one plane, then calls this method to update the size of
         the remaining plane.
        */
        plane.width = CGFloat(planeAnchor.extent.x)
        plane.height = CGFloat(planeAnchor.extent.z)
    }
```


Vodafone RO
09.42
49 %

Let's add an object on a plane

```swift
class ViewController: UIViewController, ARSCNViewDelegate, ARSessionDelegate {

    @IBOutlet weak var sceneView: ARSCNView!

    var lampScene = SCNScene()
    var lamp: SCNNode!

    override func viewDidLoad() {
        super.viewDidLoad()

        sceneView.scene = lampScene

        let node =  SCNReferenceNode(url: Bundle.main.url(forResource: "Models.scnassets/lamp/lamp.scn", withExtension: nil)!)!
        node.load()
        lampScene.rootNode.addChildNode(node)
        node.isHidden = true
        lamp = node

        let tgr = UITapGestureRecognizer(target: self, action: #selector(didTap(_:)))
        view.addGestureRecognizer(tgr)
    }
```

```
class ViewController: UIViewController, ARSCNViewDelegate, ARSessionDelegate {

    @IBOutlet weak var sceneView: ARSCNView!

    var lampScene = SCNScene()
    var lamp: SCNNode!

    override func viewDidLoad() {
        super.viewDidLoad()

        sceneView.scene = lampScene

        let node =  SCNReferenceNode(url: Bundle.main.url(forResource: "Models.scnassets/lamp/lamp.scn", withExtension: nil)!)!
        node.load()
        lampScene.rootNode.addChildNode(node)
        node.isHidden = true
        lamp = node

        let tgr = UITapGestureRecognizer(target: self, action: #selector(didTap(_:)))
        view.addGestureRecognizer(tgr)
    }
```

```
class ViewController: UIViewController, ARSCNViewDelegate, ARSessionDelegate {

    @IBOutlet weak var sceneView: ARSCNView!

    var lampScene = SCNScene()
    var lamp: SCNNode!

    override func viewDidLoad() {
        super.viewDidLoad()

        sceneView.scene = lampScene

        let node =  SCNReferenceNode(url: Bundle.main.url(forResource: "Models.scnassets/lamp/lamp.scn", withExtension: nil)!)!
        node.load()
        lampScene.rootNode.addChildNode(node)
        node.isHidden = true
        lamp = node

        let tgr = UITapGestureRecognizer(target: self, action: #selector(didTap(_:)))
        view.addGestureRecognizer(tgr)
    }
```

```swift
class ViewController: UIViewController, ARSCNViewDelegate, ARSessionDelegate {

    @IBOutlet weak var sceneView: ARSCNView!

    var lampScene = SCNScene()
    var lamp: SCNNode!

    override func viewDidLoad() {
        super.viewDidLoad()

        sceneView.scene = lampScene

        let node =  SCNReferenceNode(url: Bundle.main.url(forResource: "Models.scnassets/lamp/lamp.scn", withExtension: nil)!)!
        node.load()
        lampScene.rootNode.addChildNode(node)
        node.isHidden = true
        lamp = node

        let tgr = UITapGestureRecognizer(target: self, action: #selector(didTap(_:)))
        view.addGestureRecognizer(tgr)
    }
```

```swift
class ViewController: UIViewController, ARSCNViewDelegate, ARSessionDelegate {

    @IBOutlet weak var sceneView: ARSCNView!

    var lampScene = SCNScene()
    var lamp: SCNNode!

    override func viewDidLoad() {
        super.viewDidLoad()

        sceneView.scene = lampScene

        let node =  SCNReferenceNode(url: Bundle.main.url(forResource: "Models.scnassets/lamp/lamp.scn", withExtension: nil)!)!
        node.load()
        lampScene.rootNode.addChildNode(node)
        node.isHidden = true
        lamp = node

        let tgr = UITapGestureRecognizer(target: self, action: #selector(didTap(_:)))
        view.addGestureRecognizer(tgr)
    }
```

```swift
@IBAction func didTap(_ recognizer: UITapGestureRecognizer) {
    let location = recognizer.location(in: sceneView)

    // When tapped on a plane, reposition the content
    let arHitTestResult = sceneView.hitTest(location, types: .existingPlane)
    if !arHitTestResult.isEmpty {
        let hit = arHitTestResult.first!
        lamp.simdTransform = hit.worldTransform

        if lamp.isHidden {
            lamp.isHidden = false
        }
    }
}
```

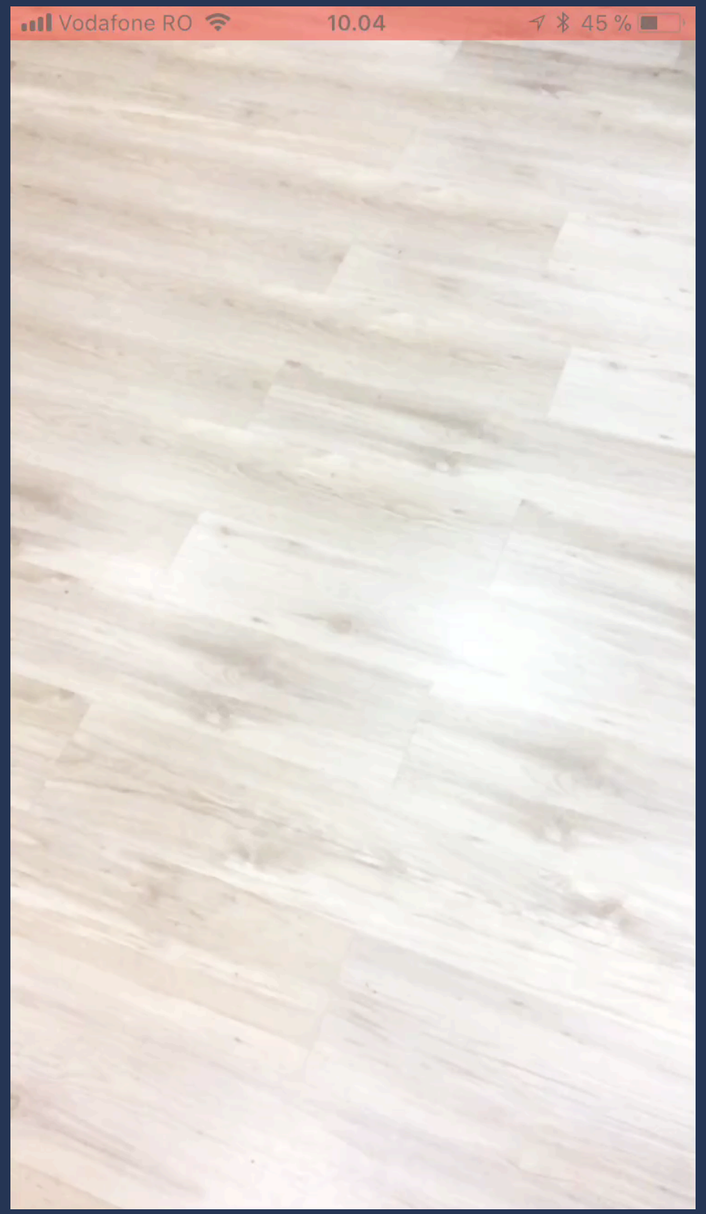
```
@IBAction func didTap(_ recognizer: UITapGestureRecognizer) {
    let location = recognizer.location(in: sceneView)

    // When tapped on a plane, reposition the content
    let arHitTestResult = sceneView.hitTest(location, types: .existingPlane)
    if !arHitTestResult.isEmpty {
        let hit = arHitTestResult.first!
        lamp.simdTransform = hit.worldTransform

        if lamp.isHidden {
            lamp.isHidden = false
        }
    }
}
```

```
@IBAction func didTap(_ recognizer: UITapGestureRecognizer) {
    let location = recognizer.location(in: sceneView)

    // When tapped on a plane, reposition the content
    let arHitTestResult = sceneView.hitTest(location, types: .existingPlane)
    if !arHitTestResult.isEmpty {
        let hit = arHitTestResult.first!
        lamp.simdTransform = hit.worldTransform

        if lamp.isHidden {
            lamp.isHidden = false
        }
    }
}
```

```
@IBAction func didTap(_ recognizer: UITapGestureRecognizer) {
    let location = recognizer.location(in: sceneView)

    // When tapped on a plane, reposition the content
    let arHitTestResult = sceneView.hitTest(location, types: .existingPlane)
    if !arHitTestResult.isEmpty {
        let hit = arHitTestResult.first!
        lamp.simdTransform = hit.worldTransform

        if lamp.isHidden {
            lamp.isHidden = false
        }
    }
}
```

```
@IBAction func didTap(_ recognizer: UITapGestureRecognizer) {
    let location = recognizer.location(in: sceneView)

    // When tapped on a plane, reposition the content
    let arHitTestResult = sceneView.hitTest(location, types: .existingPlane)
    if !arHitTestResult.isEmpty {
        let hit = arHitTestResult.first!
        lamp.simdTransform = hit.worldTransform

        if lamp.isHidden {
            lamp.isHidden = false
        }
    }
}
```

```
@IBAction func didTap(_ recognizer: UITapGestureRecognizer) {
    let location = recognizer.location(in: sceneView)

    // When tapped on a plane, reposition the content
    let arHitTestResult = sceneView.hitTest(location, types: .existingPlane)
    if !arHitTestResult.isEmpty {
        let hit = arHitTestResult.first!
        lamp.simdTransform = hit.worldTransform

        if lamp.isHidden {
            lamp.isHidden = false
        }
    }
}
```


Vodafone RO
10.04
45 %

Let's move it around

```
let pgr = UIPanGestureRecognizer(target: self, action: #selector(didPan(_:)))
view.addGestureRecognizer(pgr)
```

```
@IBAction func didPan(_ recognizer: UIPanGestureRecognizer) {
    let location = recognizer.location(in: sceneView)

    // Drag the object on an infinite plane
    let arHitTestResult = sceneView.hitTest(location, types: .existingPlane)
    if !arHitTestResult.isEmpty {
        let hit = arHitTestResult.first!
        lamp.simdTransform = hit.worldTransform
    }
}
```

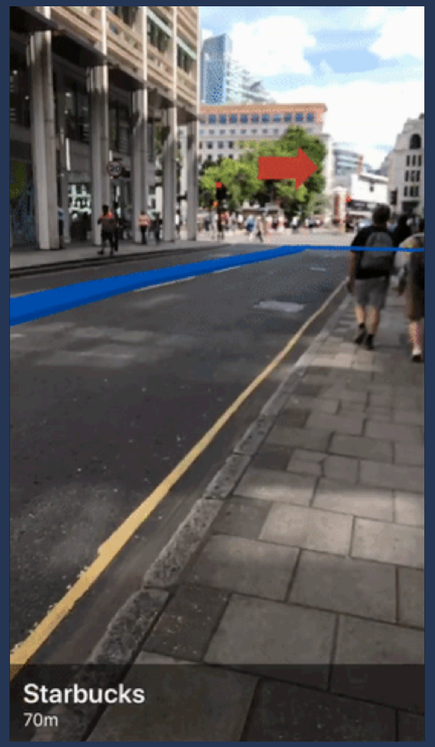

vodafone RO
09:43
46 %

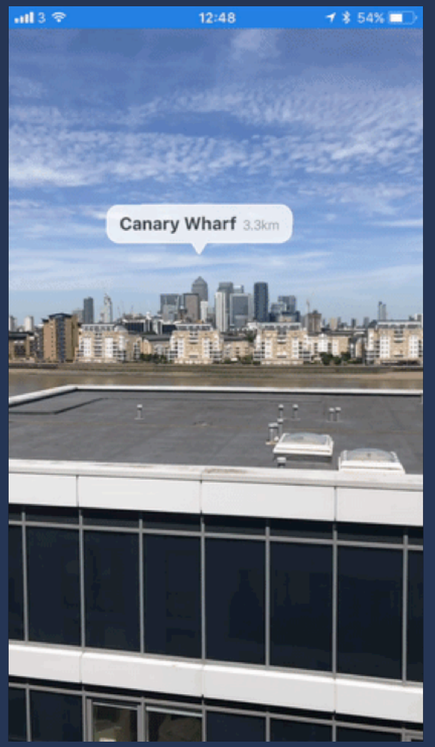
Usecases

IKEA Place

Conduct AR

ARKit-CoreLocation





https://github.com/ProjectDent/ARKit-CoreLocation

Measurements

Starting from the basic plane detection example

```
var measurementNodes: [SCNNode] = []
```

```swift
@IBAction func didTap(gestureRecognizer: UITapGestureRecognizer) {

     //make a hit test to see where we should place it
    let point = gestureRecognizer.location(in: self.view)

    if let hit = sceneView.hitTest(point, types: [.existingPlaneUsingExtent]).first {
        let radius: CGFloat = 0.01
        let sphere = SCNSphere(radius:radius)
        sphere.firstMaterial?.diffuse.contents = UIColor.green
        let newNode = SCNNode(geometry: sphere)

        //if we already have 2 or more nodes, we start a new measurement
        if measurementNodes.count >= 2 {
            clearMeasurement()
        }

        //we add the measuring point to the scene
        measurementNodes.append(newNode)
        self.sceneView.scene.rootNode.addChildNode(newNode)
        newNode.simdTransform = hit.worldTransform

        //if we have two measuring nodes here, we draw a line and show the measuring result
        if measurementNodes.count == 2 {
            computeDistance()
        }
    }
}
```

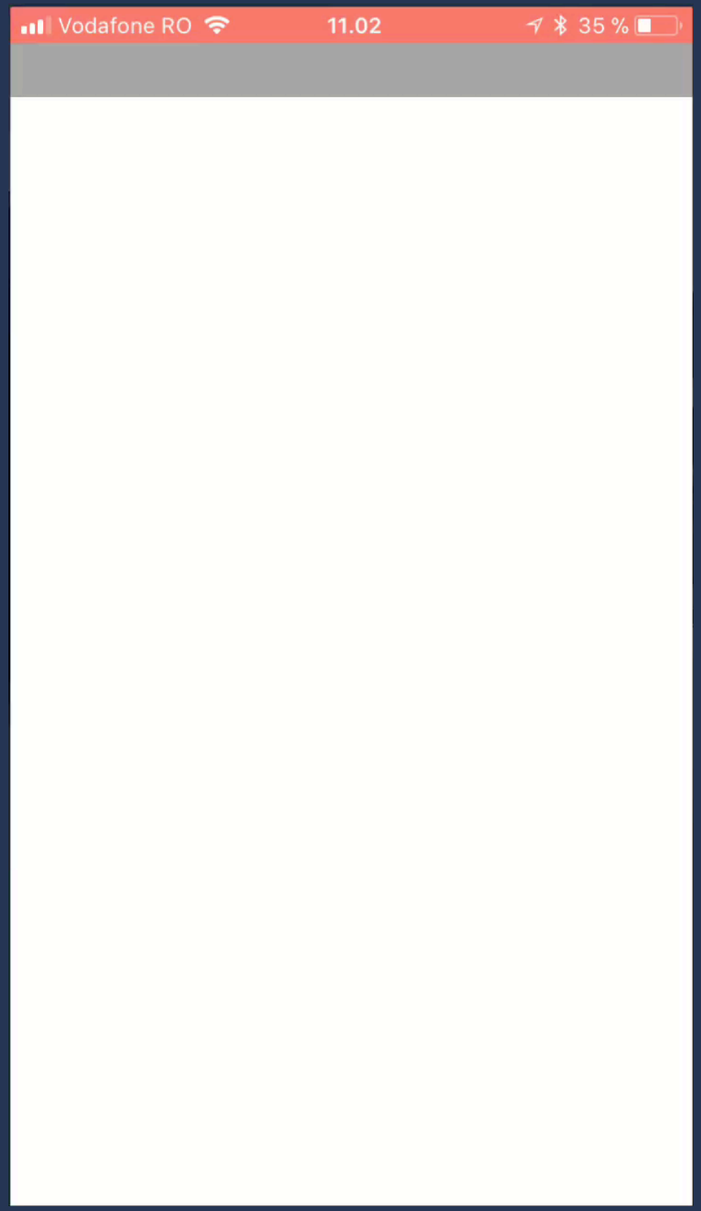
```swift
@IBAction func didTap(gestureRecognizer: UITapGestureRecognizer) {

     //make a hit test to see where we should place it
    let point = gestureRecognizer.location(in: self.view)

    if let hit = sceneView.hitTest(point, types: [.existingPlaneUsingExtent]).first {
        let radius: CGFloat = 0.01
        let sphere = SCNSphere(radius:radius)
        sphere.firstMaterial?.diffuse.contents = UIColor.green
        let newNode = SCNNode(geometry: sphere)

        //if we already have 2 or more nodes, we start a new measurement
        if measurementNodes.count >= 2 {
            clearMeasurement()
        }

        //we add the measuring point to the scene
        measurementNodes.append(newNode)
        self.sceneView.scene.rootNode.addChildNode(newNode)
        newNode.simdTransform = hit.worldTransform

        //if we have two measuring nodes here, we draw a line and show the measuring result
        if measurementNodes.count == 2 {
            computeDistance()
        }
    }
}
```

```swift
@IBAction func didTap(gestureRecognizer: UITapGestureRecognizer) {

     //make a hit test to see where we should place it
    let point = gestureRecognizer.location(in: self.view)

    if let hit = sceneView.hitTest(point, types: [.existingPlaneUsingExtent]).first {
        let radius: CGFloat = 0.01
        let sphere = SCNSphere(radius:radius)
        sphere.firstMaterial?.diffuse.contents = UIColor.green
        let newNode = SCNNode(geometry: sphere)

        //if we already have 2 or more nodes, we start a new measurement
        if measurementNodes.count >= 2 {
            clearMeasurement()
        }

        //we add the measuring point to the scene
        measurementNodes.append(newNode)
        self.sceneView.scene.rootNode.addChildNode(newNode)
        newNode.simdTransform = hit.worldTransform

        //if we have two measuring nodes here, we draw a line and show the measuring result
        if measurementNodes.count == 2 {
            computeDistance()
        }
    }
}
```

```swift
@IBAction func didTap(gestureRecognizer: UITapGestureRecognizer) {

     //make a hit test to see where we should place it
    let point = gestureRecognizer.location(in: self.view)

    if let hit = sceneView.hitTest(point, types: [.existingPlaneUsingExtent]).first {
        let radius: CGFloat = 0.01
        let sphere = SCNSphere(radius:radius)
        sphere.firstMaterial?.diffuse.contents = UIColor.green
        let newNode = SCNNode(geometry: sphere)

        //if we already have 2 or more nodes, we start a new measurement
        if measurementNodes.count >= 2 {
            clearMeasurement()
        }

        //we add the measuring point to the scene
        measurementNodes.append(newNode)
        self.sceneView.scene.rootNode.addChildNode(newNode)
        newNode.simdTransform = hit.worldTransform

        //if we have two measuring nodes here, we draw a line and show the measuring result
        if measurementNodes.count == 2 {
            computeDistance()
        }
    }
}
```

```swift
@IBAction func didTap(gestureRecognizer: UITapGestureRecognizer) {

     //make a hit test to see where we should place it
    let point = gestureRecognizer.location(in: self.view)

    if let hit = sceneView.hitTest(point, types: [.existingPlaneUsingExtent]).first {
        let radius: CGFloat = 0.01
        let sphere = SCNSphere(radius:radius)
        sphere.firstMaterial?.diffuse.contents = UIColor.green
        let newNode = SCNNode(geometry: sphere)

        //if we already have 2 or more nodes, we start a new measurement
        if measurementNodes.count >= 2 {
            clearMeasurement()
        }

        //we add the measuring point to the scene
        measurementNodes.append(newNode)
        self.sceneView.scene.rootNode.addChildNode(newNode)
        newNode.simdTransform = hit.worldTransform

        //if we have two measuring nodes here, we draw a line and show the measuring result
        if measurementNodes.count == 2 {
            computeDistance()
        }
    }
}
```

```swift
@IBAction func didTap(gestureRecognizer: UITapGestureRecognizer) {

     //make a hit test to see where we should place it
    let point = gestureRecognizer.location(in: self.view)

    if let hit = sceneView.hitTest(point, types: [.existingPlaneUsingExtent]).first {
        let radius: CGFloat = 0.01
        let sphere = SCNSphere(radius:radius)
        sphere.firstMaterial?.diffuse.contents = UIColor.green
        let newNode = SCNNode(geometry: sphere)

        //if we already have 2 or more nodes, we start a new measurement
        if measurementNodes.count >= 2 {
            clearMeasurement()
        }

        //we add the measuring point to the scene
        measurementNodes.append(newNode)
        self.sceneView.scene.rootNode.addChildNode(newNode)
        newNode.simdTransform = hit.worldTransform

        //if we have two measuring nodes here, we draw a line and show the measuring result
        if measurementNodes.count == 2 {
            computeDistance()
        }
    }
}
```


Vodafone RO
11.02
35 %

Things you should know

- Plane detection only for horizontal planes
- Take care of your users
- Add `arkit` to `UIRequiredDeviceCapabilities`

What I learned

- Lighting is super important
- Being familiar with SceneKit is a huge advantage
- Don't expect the same result every time

Questions?