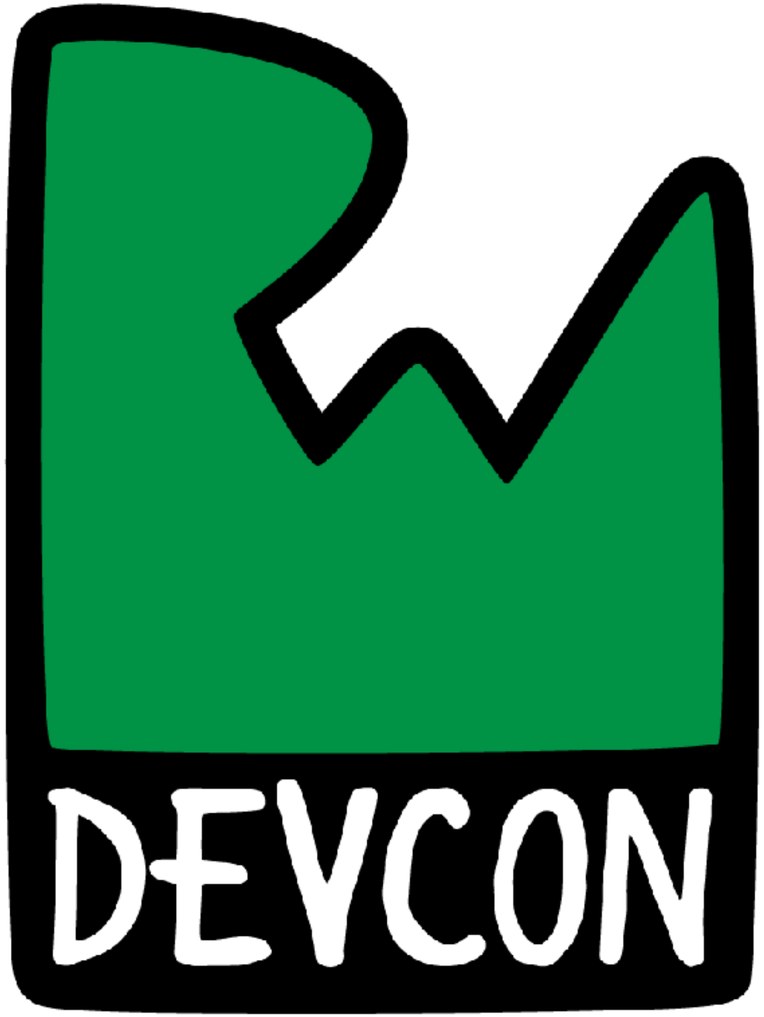
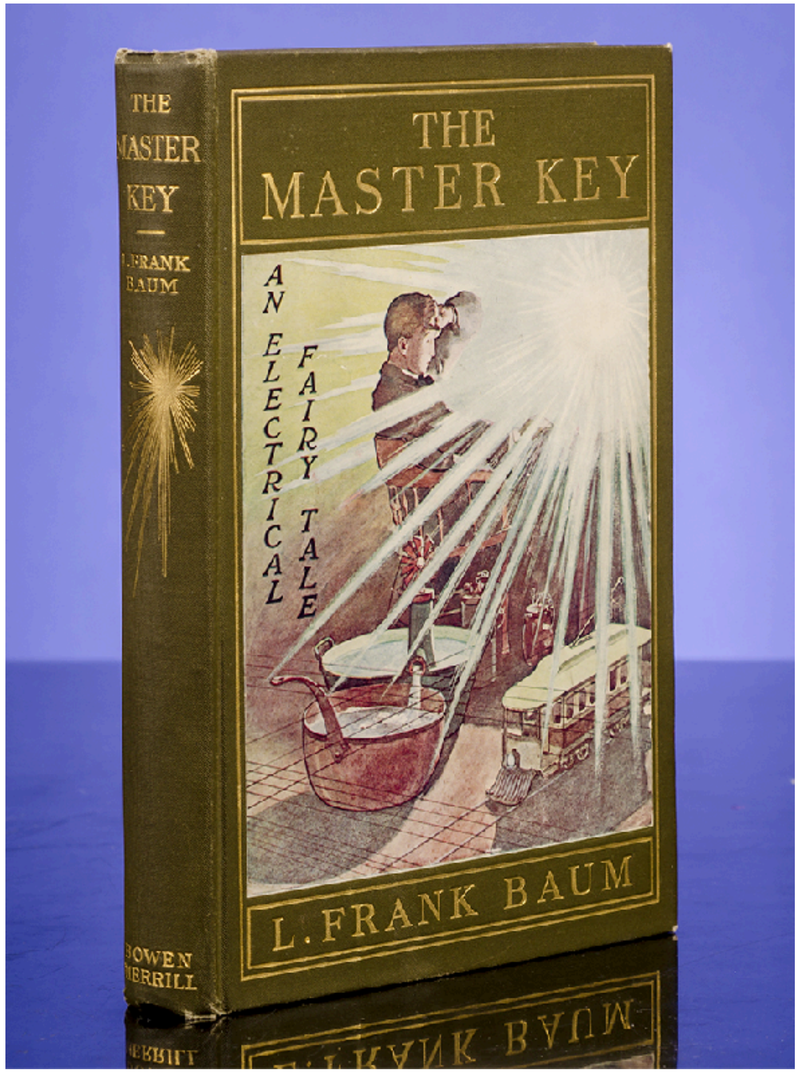
Workshop:
ARKit in Depth



Overview


AR
VR

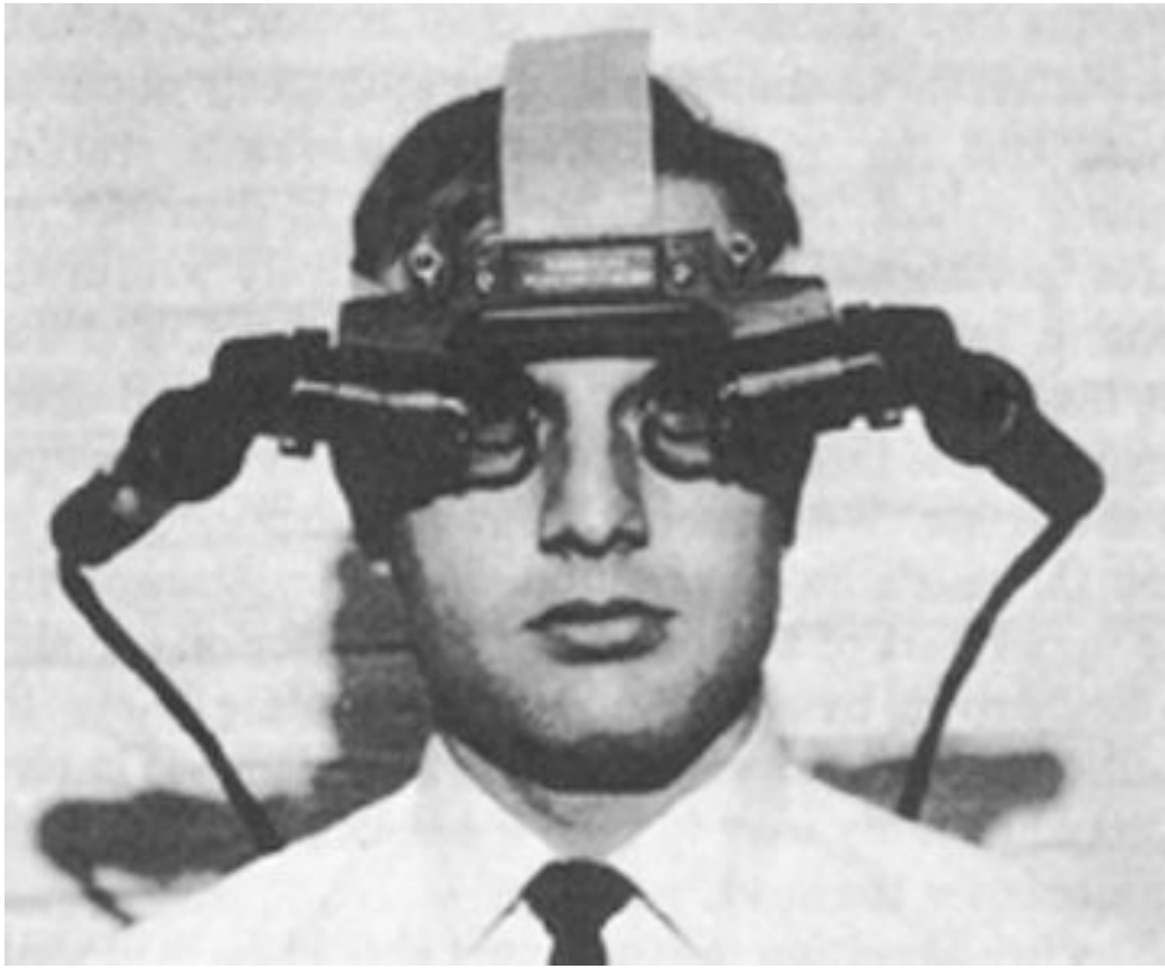
1901: L. Frank Baum



“It consists of this pair of spectacles. While you wear them every one you meet will be marked upon the forehead with a letter indicating his or her character…”

- Good: G
- Evil: E
- Wise: W
- Foolish: F
- Kind: K
- Cruel: C

1968: Ivan Southerland



1990: Tom Caudell

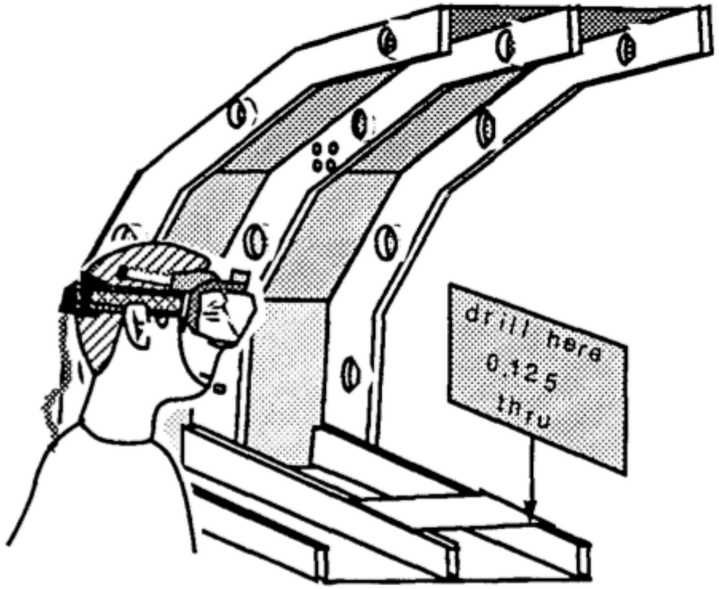
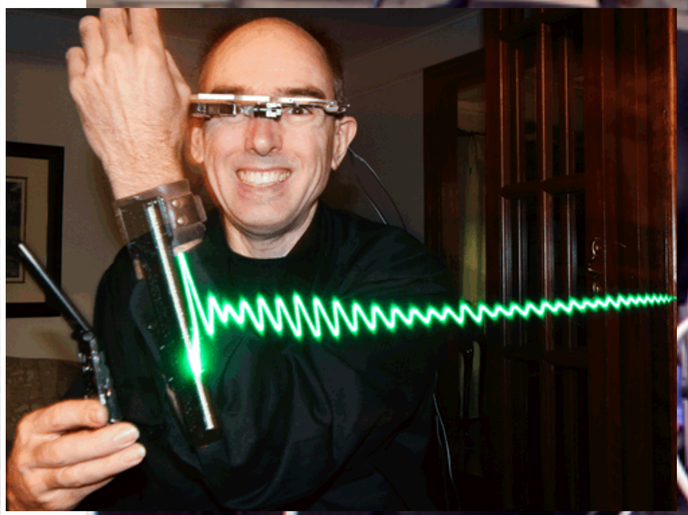
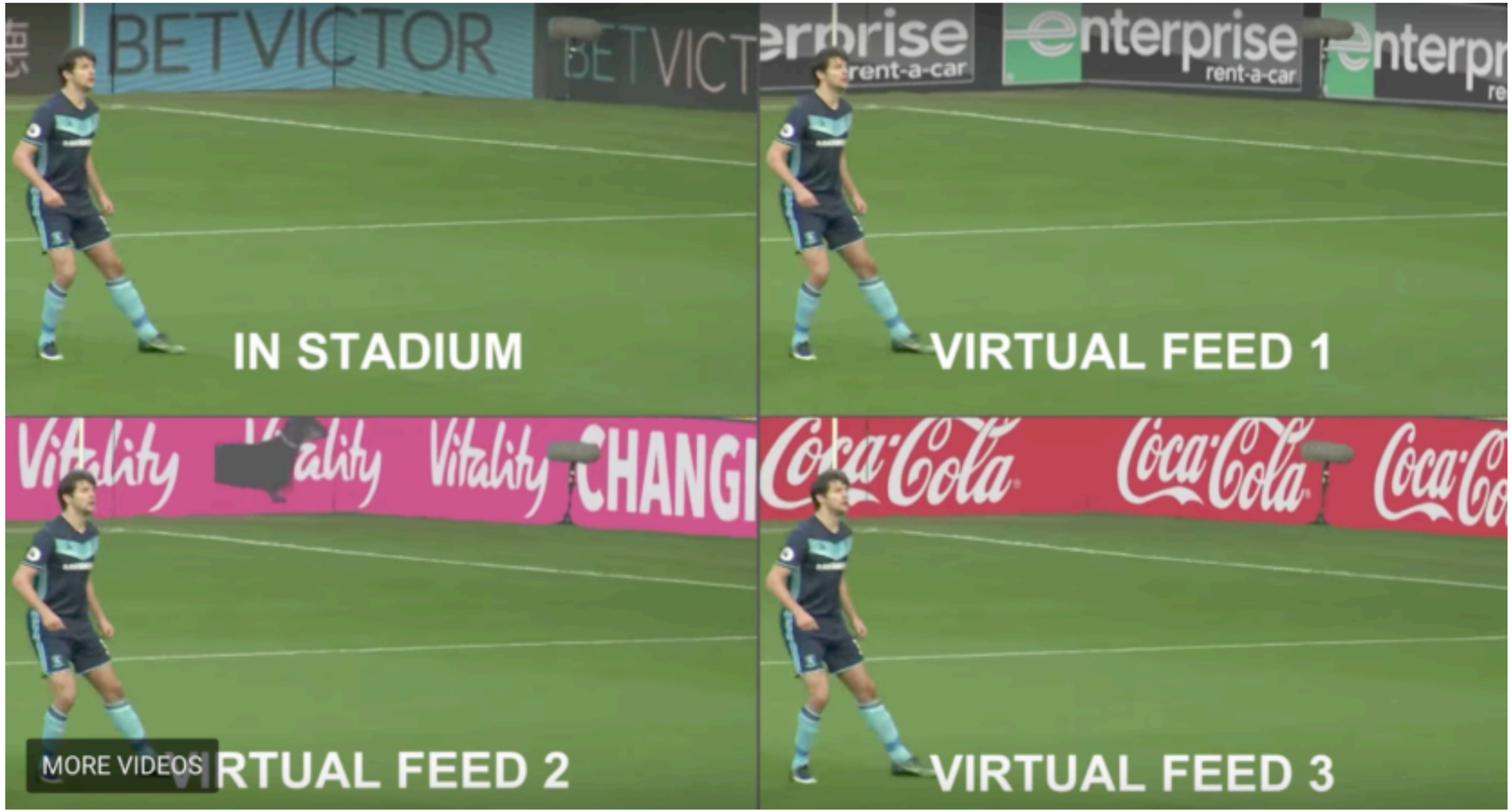


Figure 1. An application where the HUDset is used to dynamically mark the position of a drill/rivet hole inside an aircraft fuselage.

1980s, 1990s and on: Steve Mann

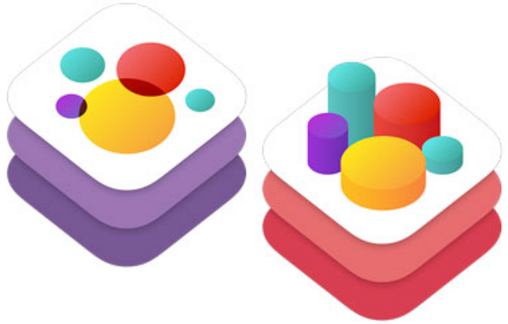


1998: Sports Broadcasting

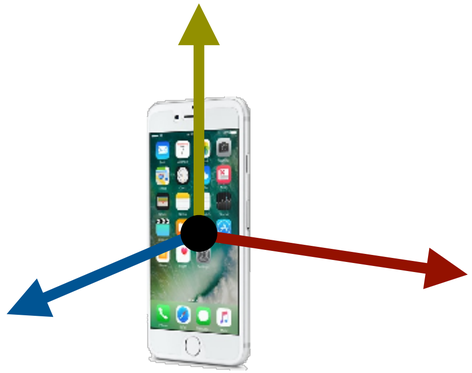


ARKit in a Nutshell

+



+



Real-world images

These come from a camera, and can simply be a backdrop for your app (think Pokémon Go), or can provide input for it (think IKEA Place).

Virtual images

These are 2D or 3D images drawn on top of the real-world images from the camera.

Sensor smarts

This is the AR application's ability to detect its position and orientation, as well as objects and conditions in the real world.

=

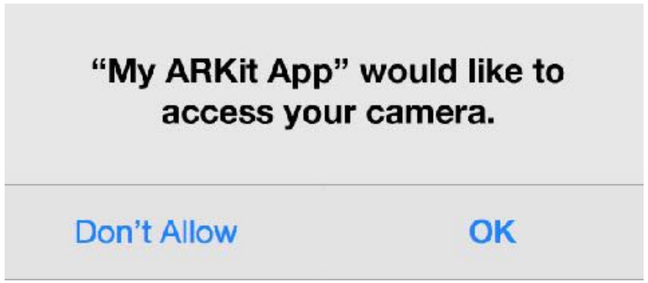
ARKit USER REQUIREMENTS

iDevice with A9 processor or later

- iPhone: 6S / 6S Plus / 7S / 7S Plus / 8 / 8 Plus / X
- iPad Pro: 9.7” / 10.5” / 12.9”
- iPad: 2017 model

iOS 11

The first iOS version that supports ARKit

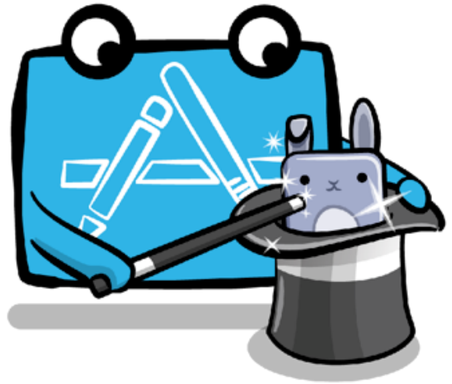


Permission to use the camera

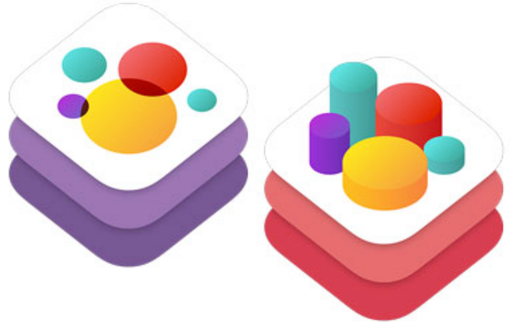
Adequate light

ARKit DEVELOPER REQUIREMENTS

The user requirements



Xcode 9.3 or later



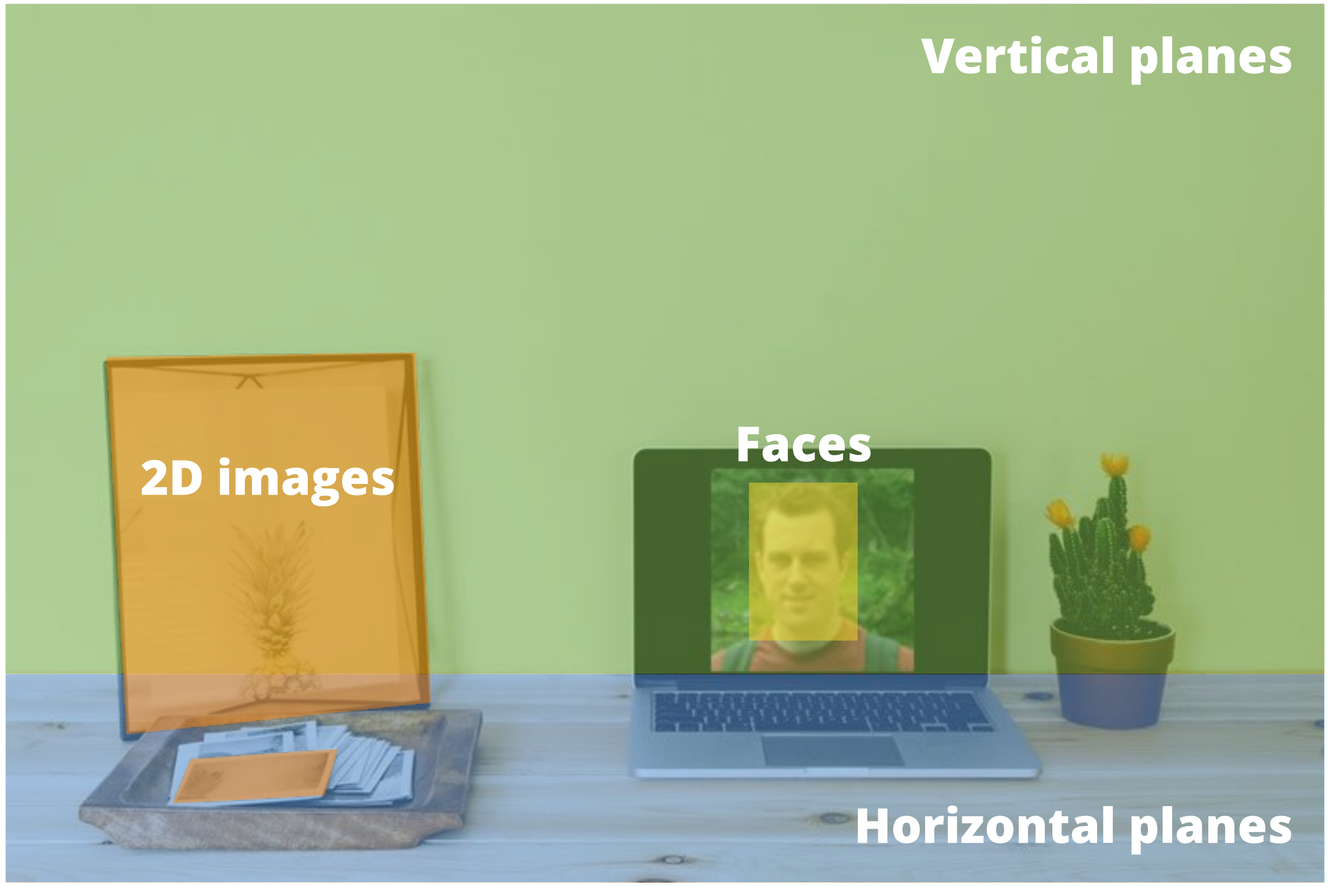
SpriteKit / SceneKit basics

Readiness to walk about and wave your iDevice

Willingness to do at least a little 3D math...

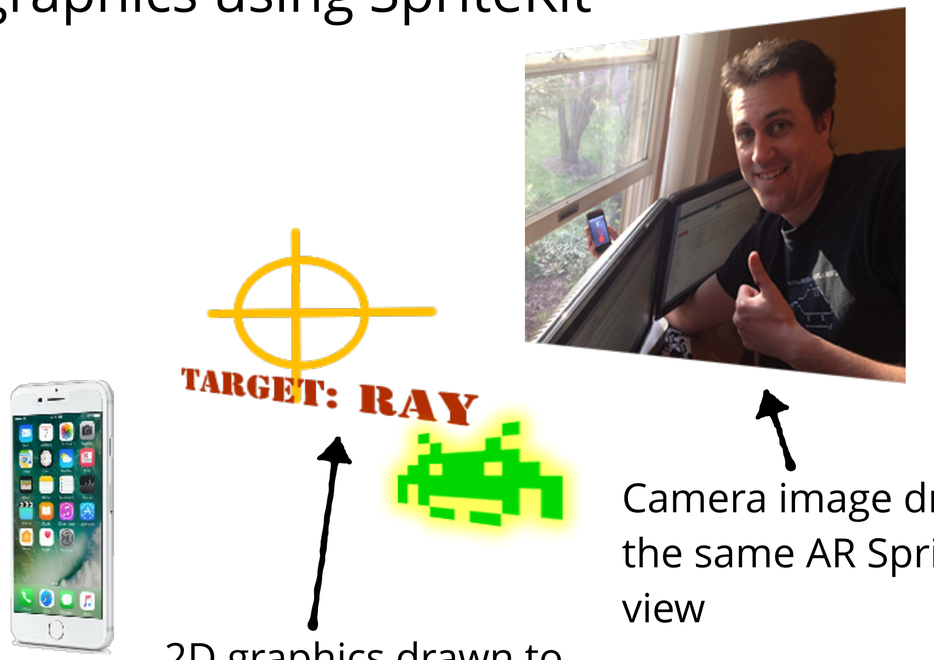
...and deal with changes and upgrades.

What real-world things can ARKit identify?



SpriteKit AR vs. SceneKit AR

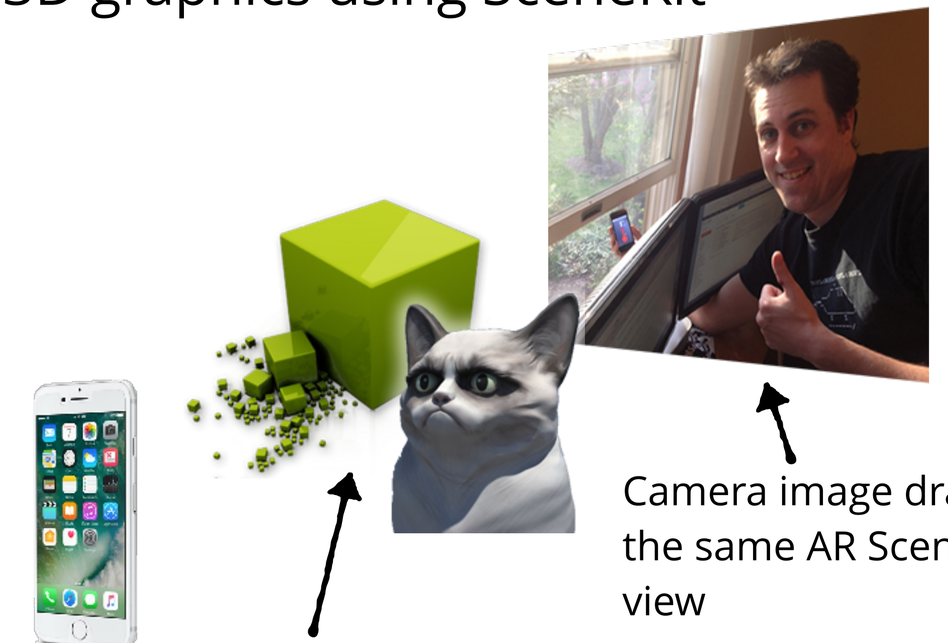
SpriteKit AR:
Overlaying camera images with
2D graphics using SpriteKit



Camera image drawn to the same AR SpriteKit view

2D graphics drawn to an **AR SpriteKit view** (**ARSKView**)

SceneKit AR:
Overlaying camera images with
3D graphics using SceneKit



Camera image drawn to the same AR SceneKit view

3D graphics drawn to an **AR SceneKit view** (**ARSCNView**)

The demos

Demo 1: *Happy AR Painter*

In which you begin your ARKit journey by paying homage to the great Bob Ross by building an app that paints in real-world 3D.

Demo 2: *Raykea*

Let's make our own version of the most popular ARKit app. Why should makers of semi-disposable furniture have all the fun?

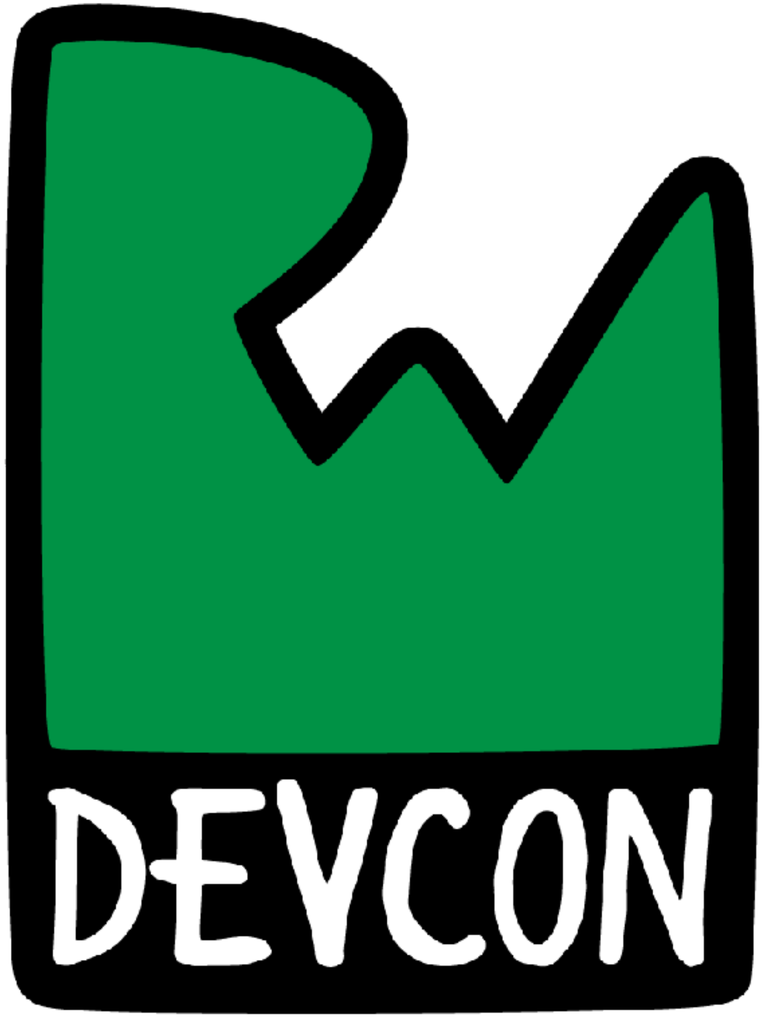
Demo 3: *BaedekAR*

One of my clients is a cruise line company, and they're interested in this app, which is an AR museum guide.

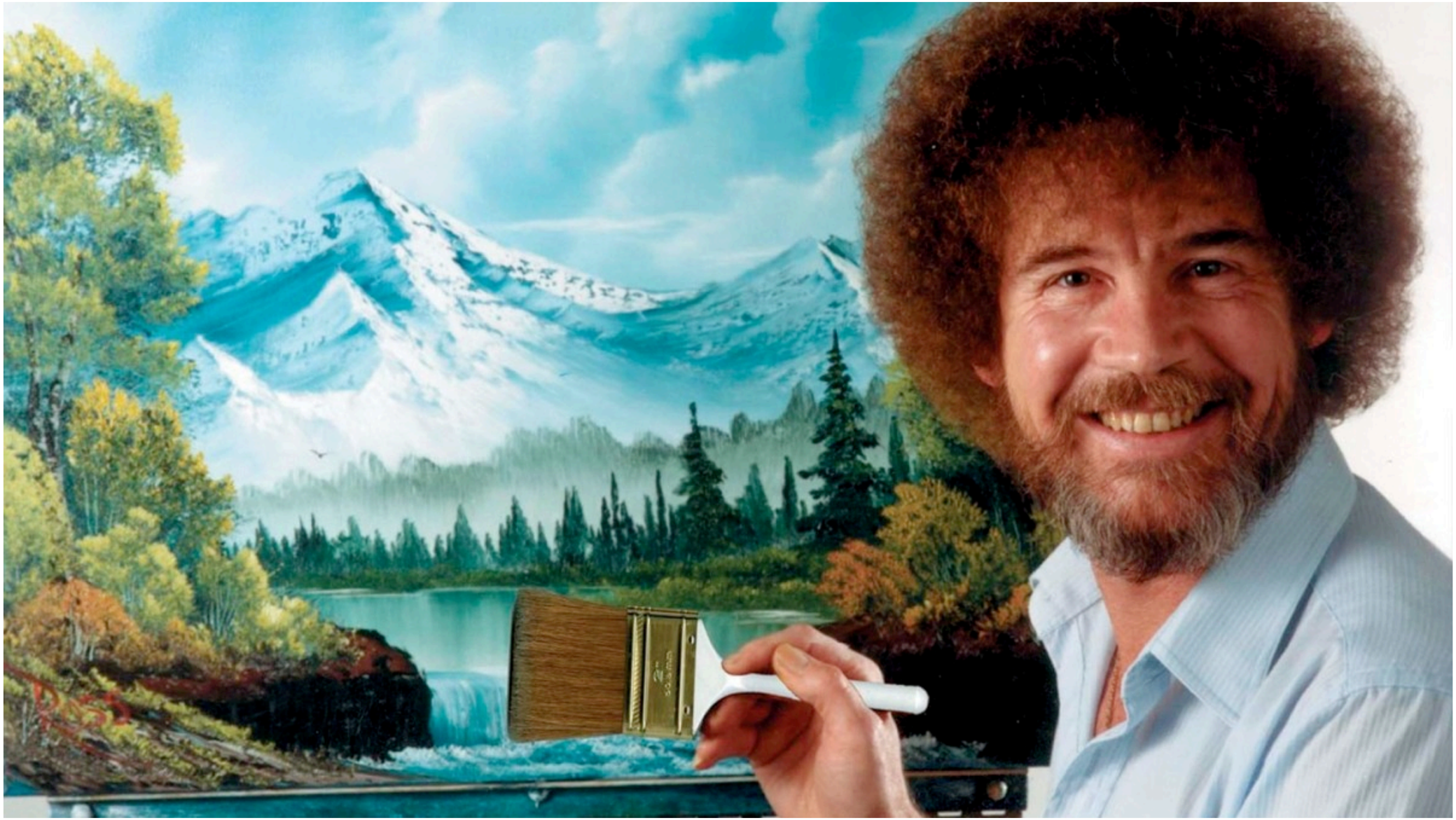
Demo 4: *Vision Quest*

We'll close the workshop either by accidentally creating SkyNet or combine machine learning with ARKit to make an object recognizer.

Workshop:
ARKit in Depth

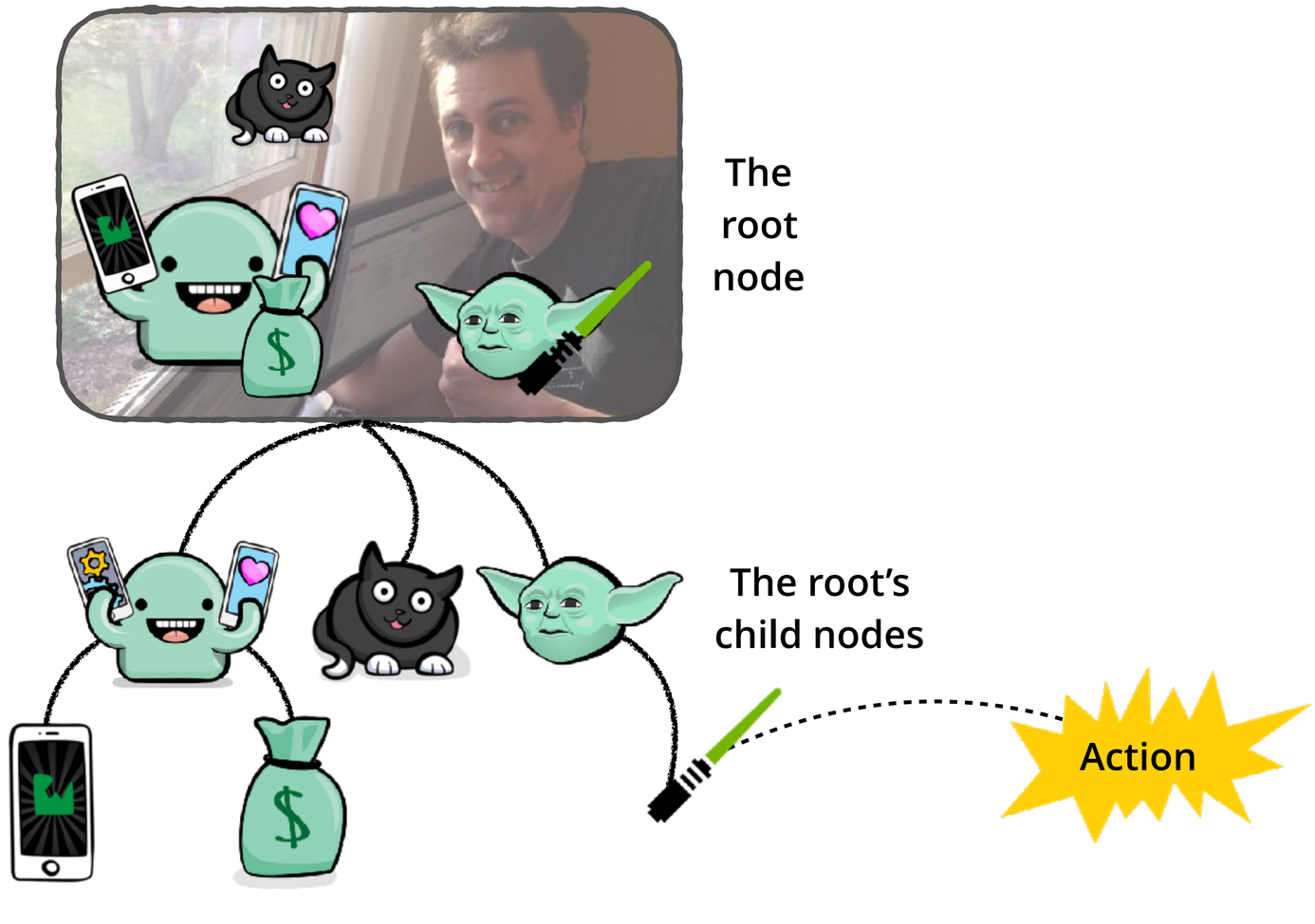


Happy AR Painter



Quick SceneKit Overview

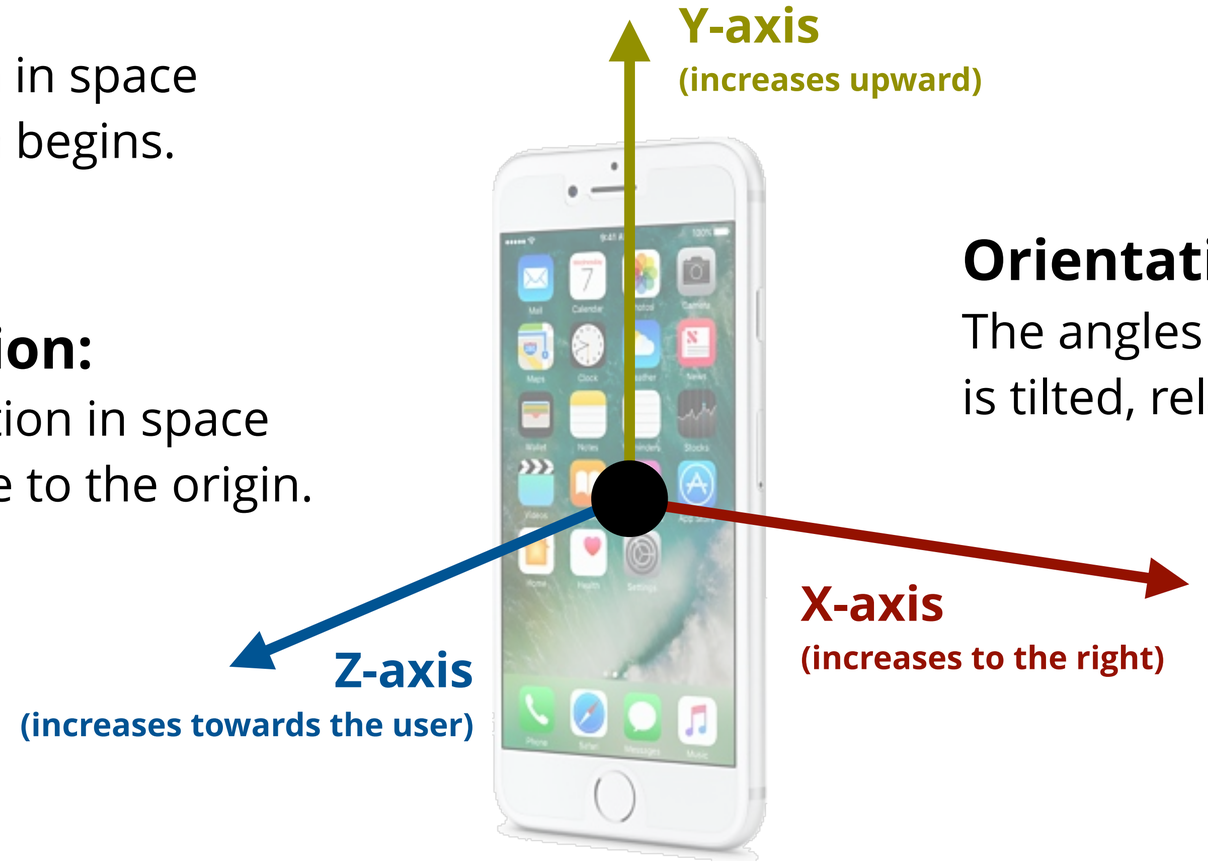
Scene graph



SceneKit's 3D Coordinate System

Origin:
The device's position in space when the AR session begins.

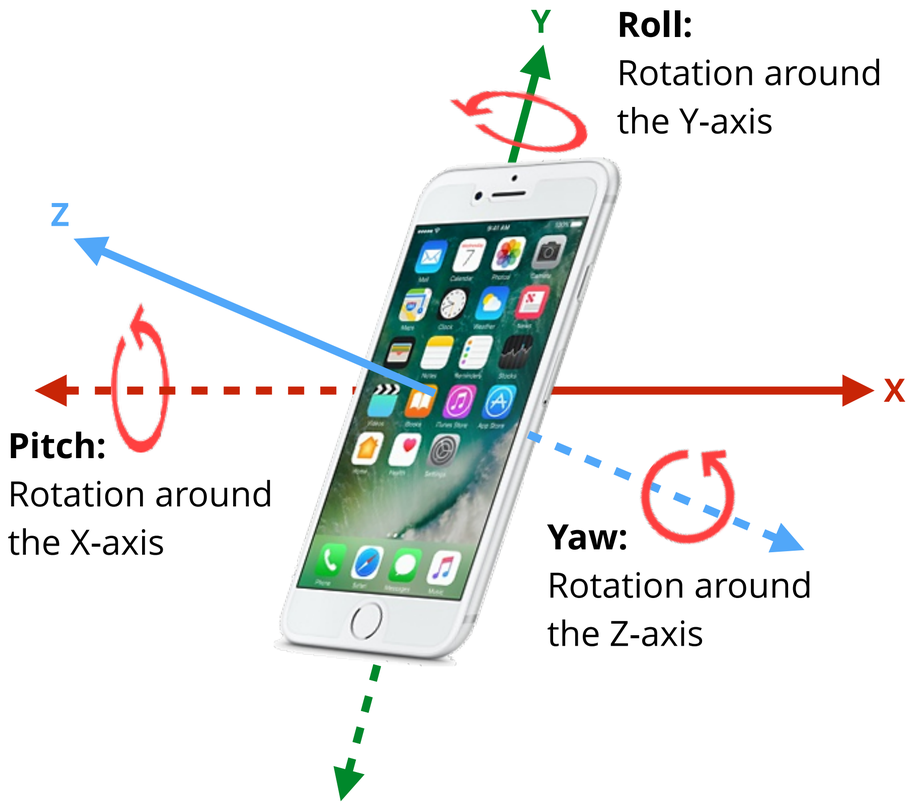
Position:
A position in space relative to the origin.



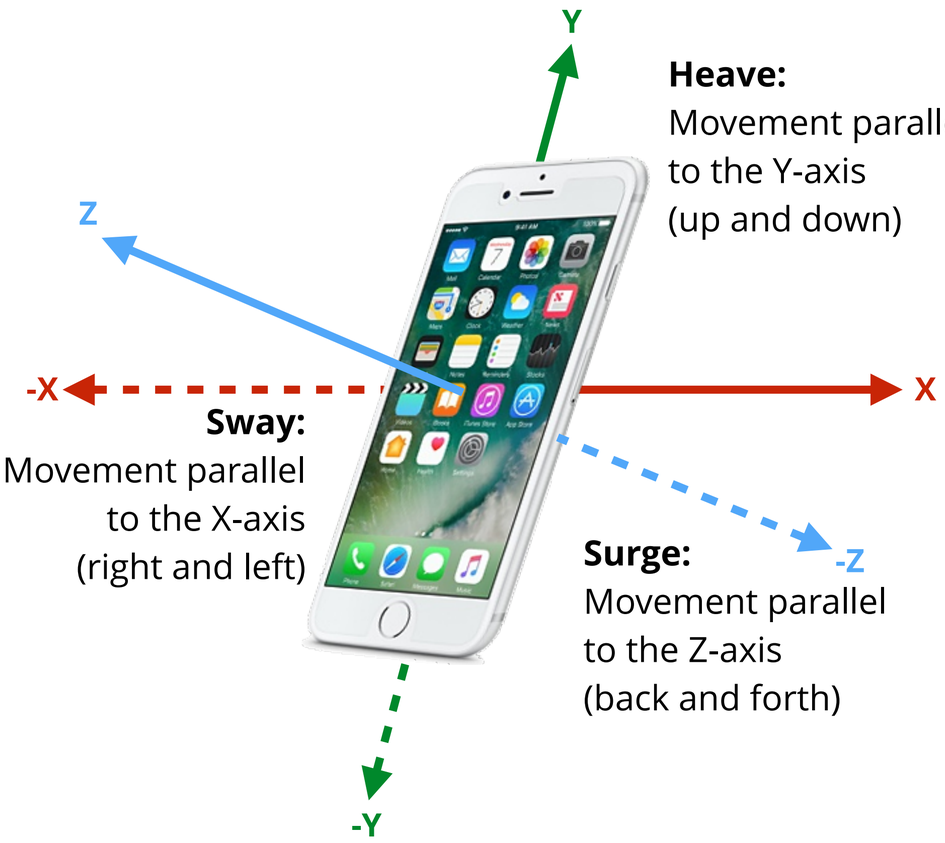
Orientation:
The angles at which something is tilted, relative to x-, y-, and z-axes.

Configuration: 3 or 6 degrees of freedom?

`AROrientationTrackingConfiguration`
`ARWorldTrackingConfiguration`

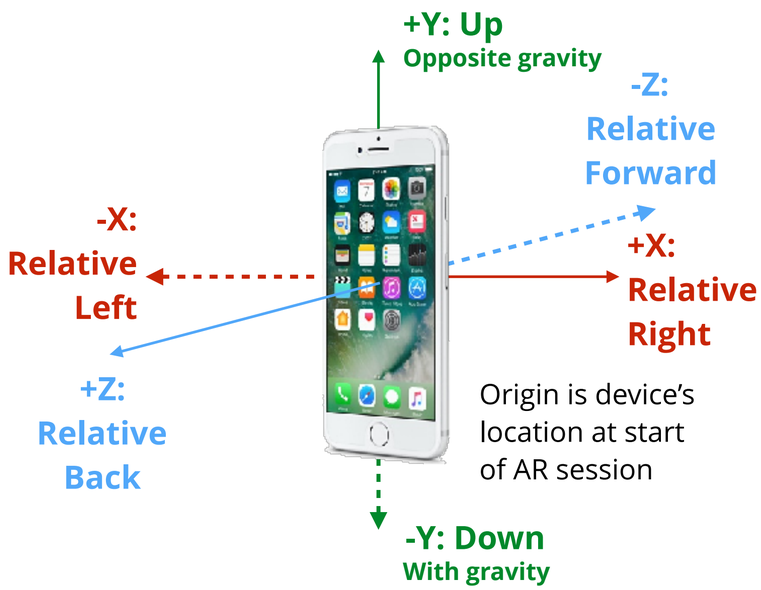


`ARWorldTrackingConfiguration`

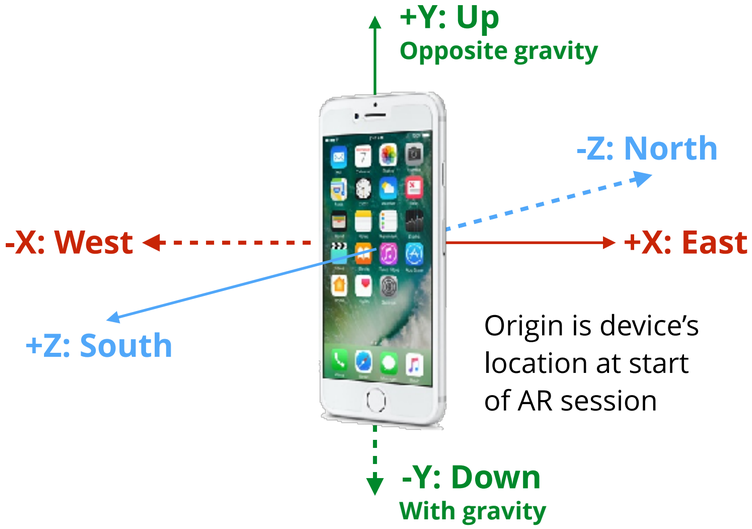


Alignment: Relating to the real world

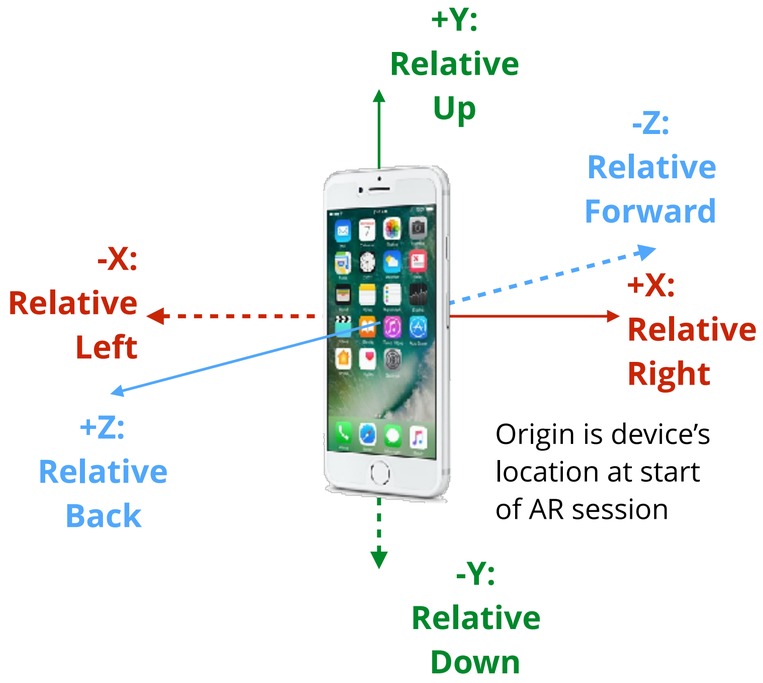
`gravity`



`gravityAndHeading`

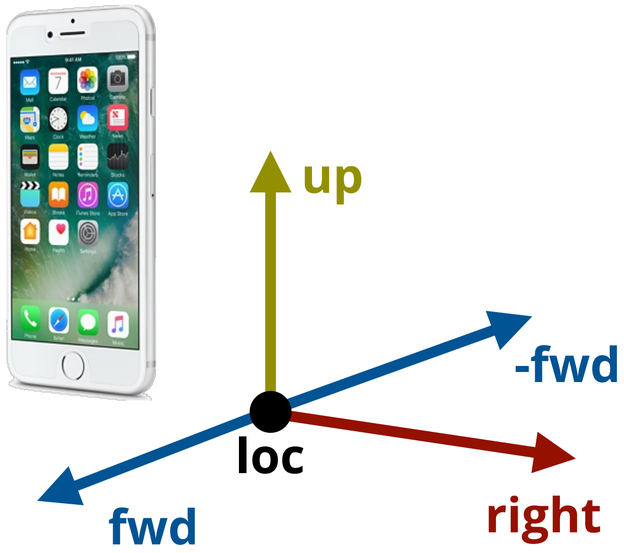


`Camera`



The SceneView's Transformation Matrix

A 4-by-4 matrix representing the scene's ***location*** and ***orientation***:



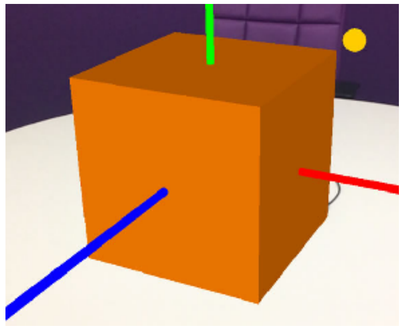
Orientation: Which way the camera's facing

Location: Where the camera is

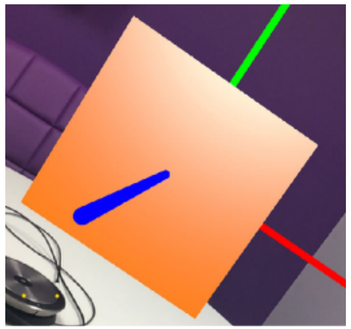
$$\begin{bmatrix} \text{right}_x & \text{up}_x & -\text{fwd}_x & \text{loc}_x \\ \text{right}_y & \text{up}_y & -\text{fwd}_y & \text{loc}_y \\ \text{right}_z & \text{up}_z & -\text{fwd}_z & \text{loc}_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Diffuse vs. Specular Reflection

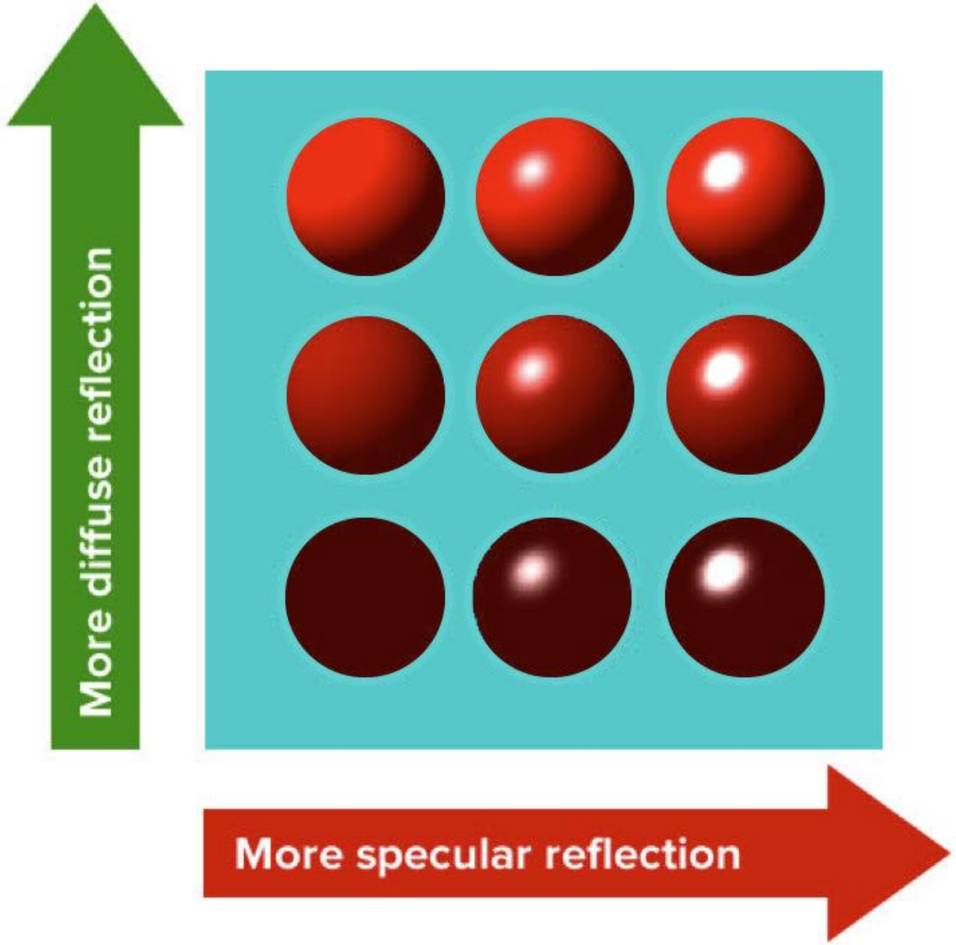
It might be easier to *show* you what these are than to *tell* you about them.



Diffuse reflection



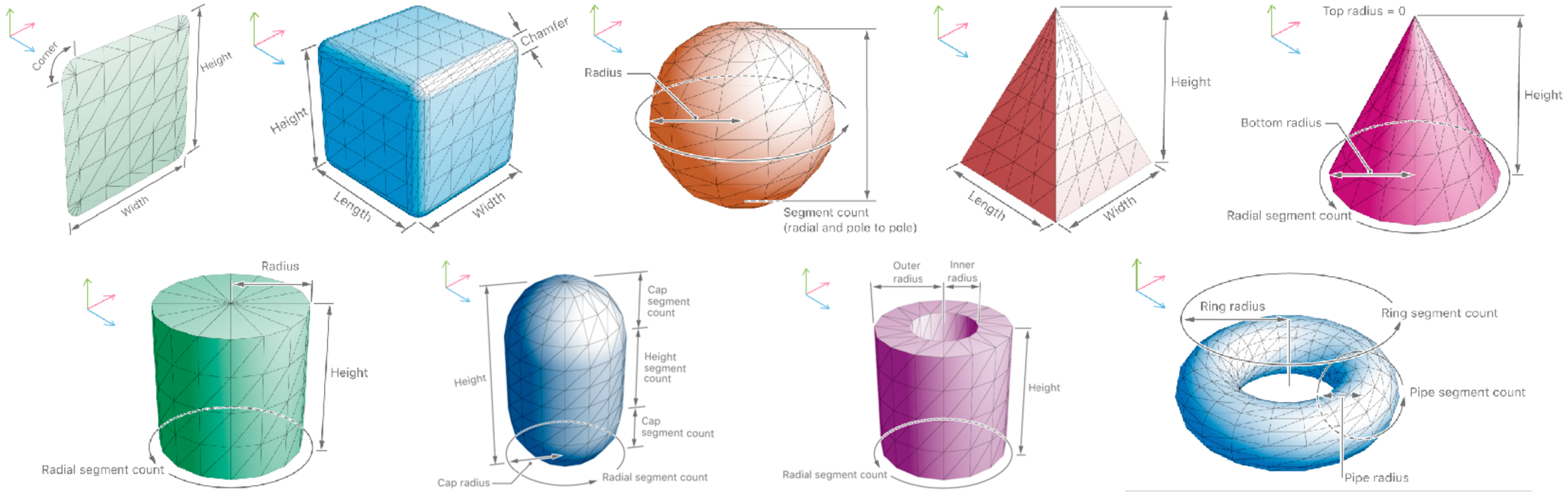
Specular reflection



Demo 1:
Happy AR Painter

How Happy AR Painter works, part 1

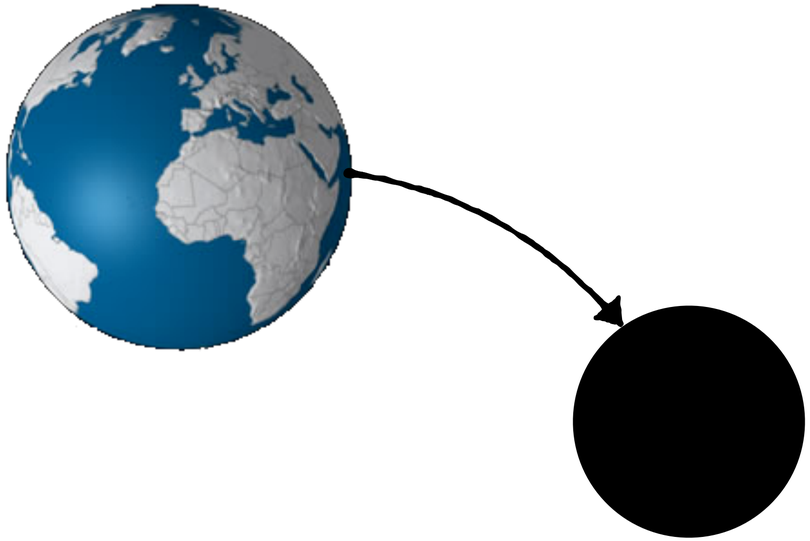
1. Define a SceneKit geometry:



How Happy AR Painter works, part 2

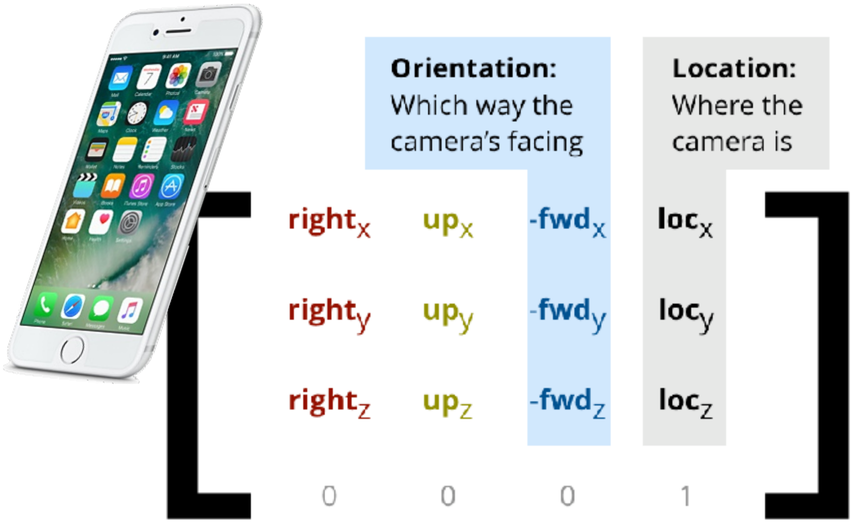
2. Apply reflective properties to the geometry.

3. Assign the geometry to a node.

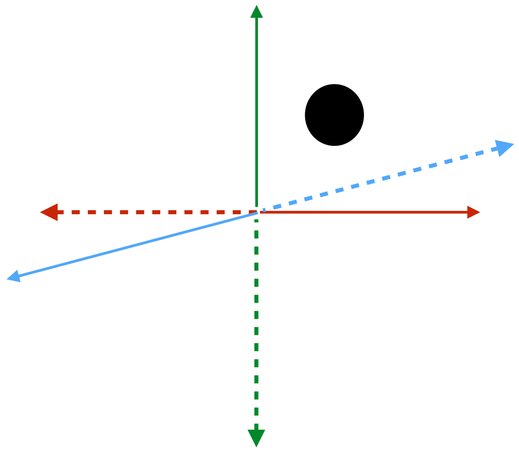


How Happy AR Painter works, part 3

4. Get the device's orientation and position from the SceneView's transform matrix.

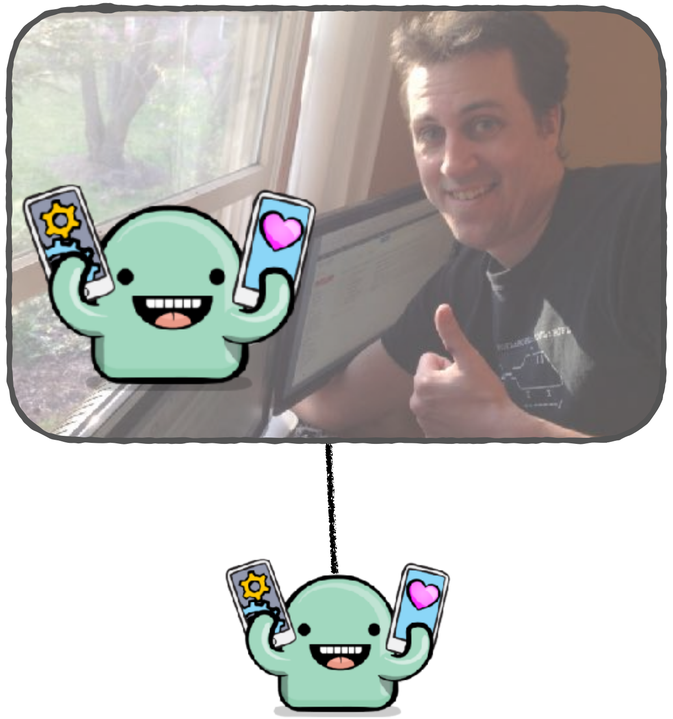


5. Set the node's orientation and position to that of the device's orientation and position.



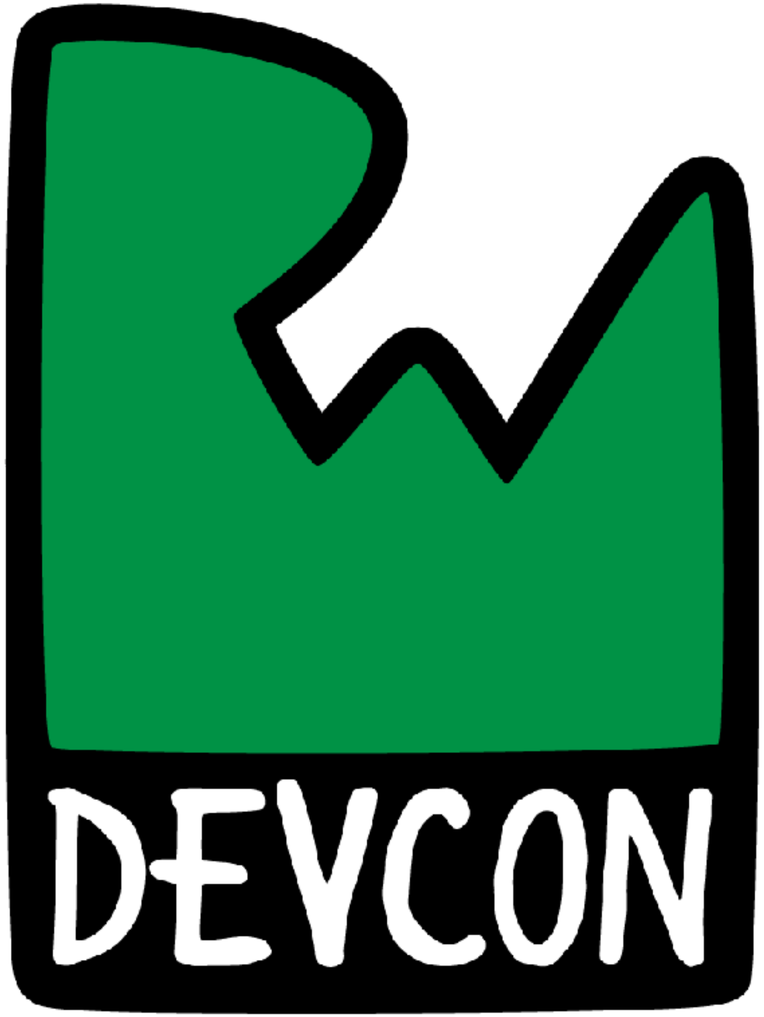
How Happy AR Painter works, part 4

6. Add the node to the scene.



The root node

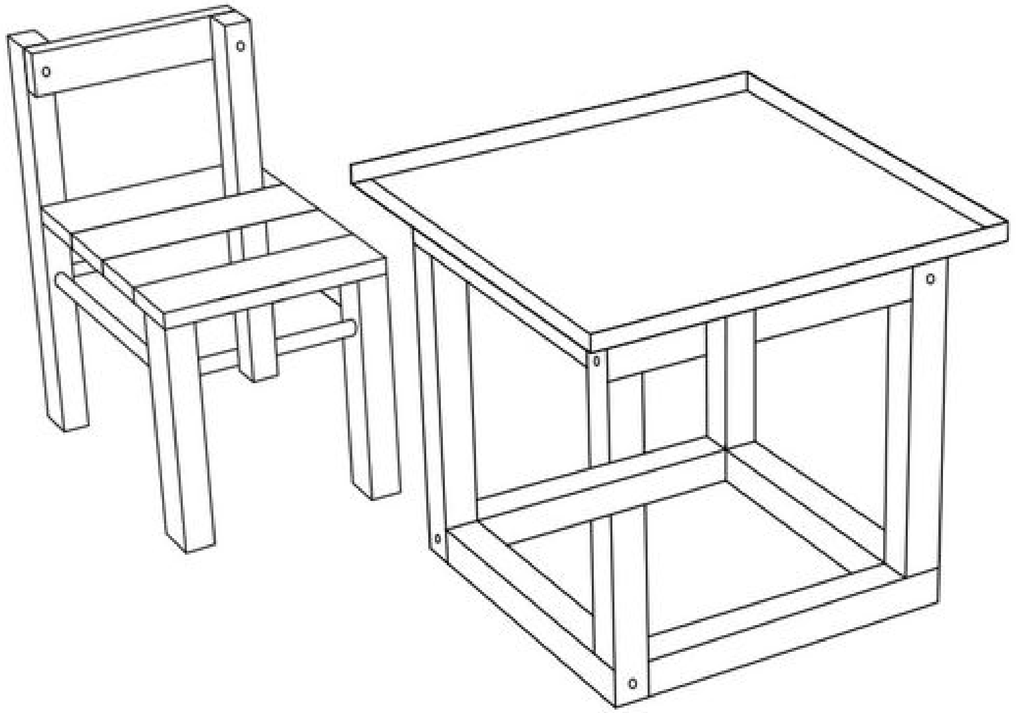
Workshop: ARKit in Depth

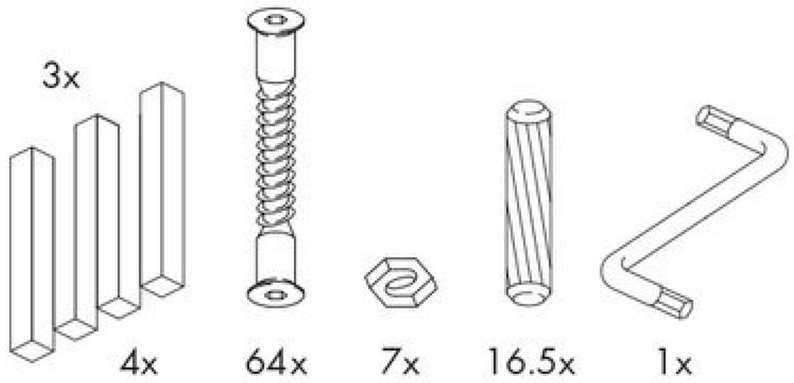


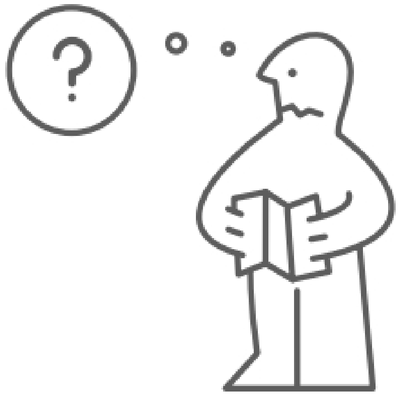
RAYKEA!

PARADÖX

play table and chair







Plane detection

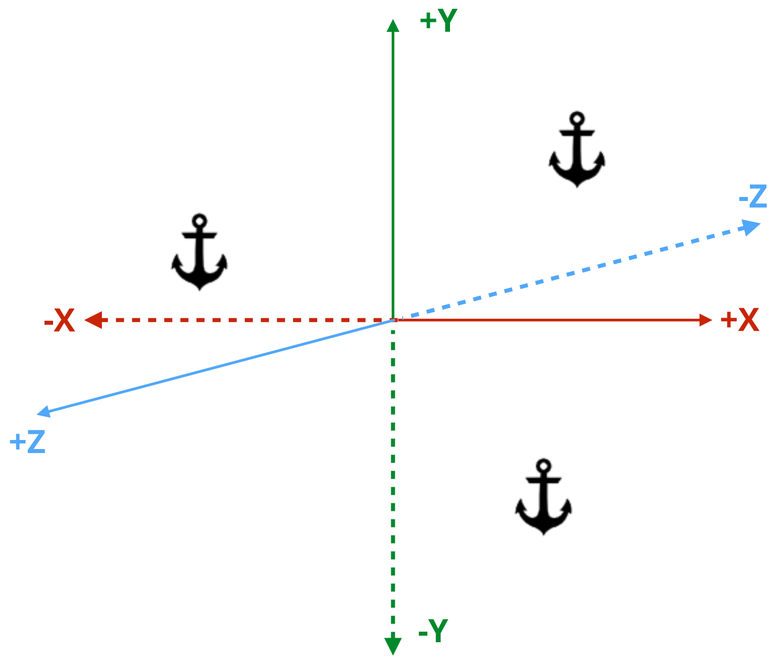
Activating plane detection

These two **`ARWorldTrackingConfiguration`** settings will enable horizontal plane detection:

```
config.worldAlignment = .gravity
// or
config.worldAlignment = .gravityAndHeading

config.planeDetection = .horizontal
// or
config.planeDetection = .vertical
// or
config.planeDetection = [.horizontal,
                         .vertical]
```

AR Anchors



An **`ARAnchor`** is a reference point
in 3D space, marking *position* and *orientation*
in an AR scene.

They can be added to a scene in 2 ways:

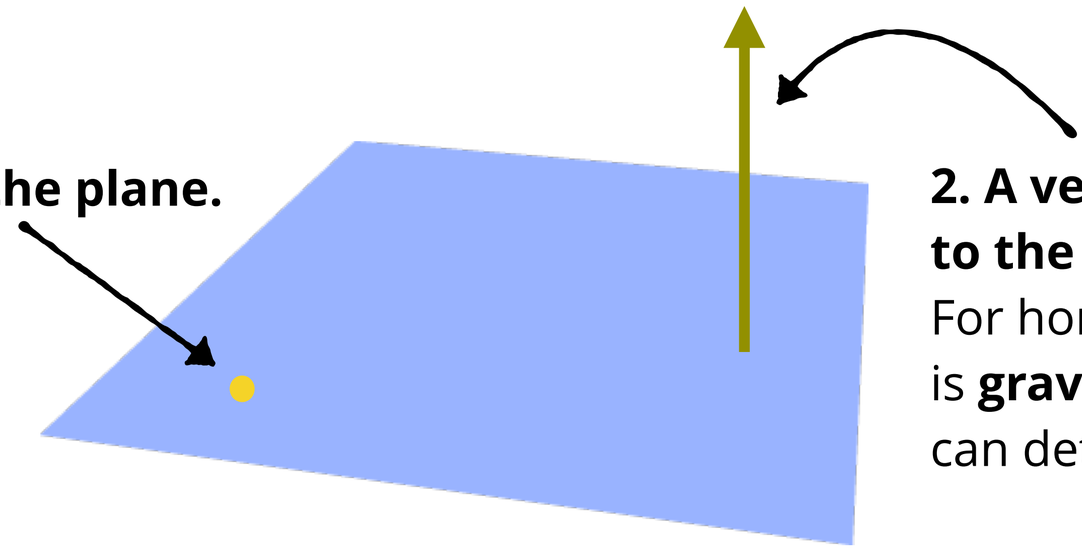
1. ***Programmatically,* by your code.**
   Do this to keep track of specific objects or locations.

2. ***Automatically,* by ARKit.**
   With plane detection turned on, ARKit adds **`ARPlaneAnchors`** (a subclass of **`ARAnchor`**) to the scene.

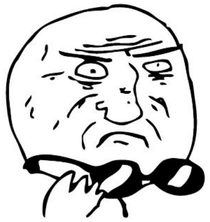
How ARKit detects horizontal planes

The equation for a plane is defined by 2 things...

1. A point on the plane.

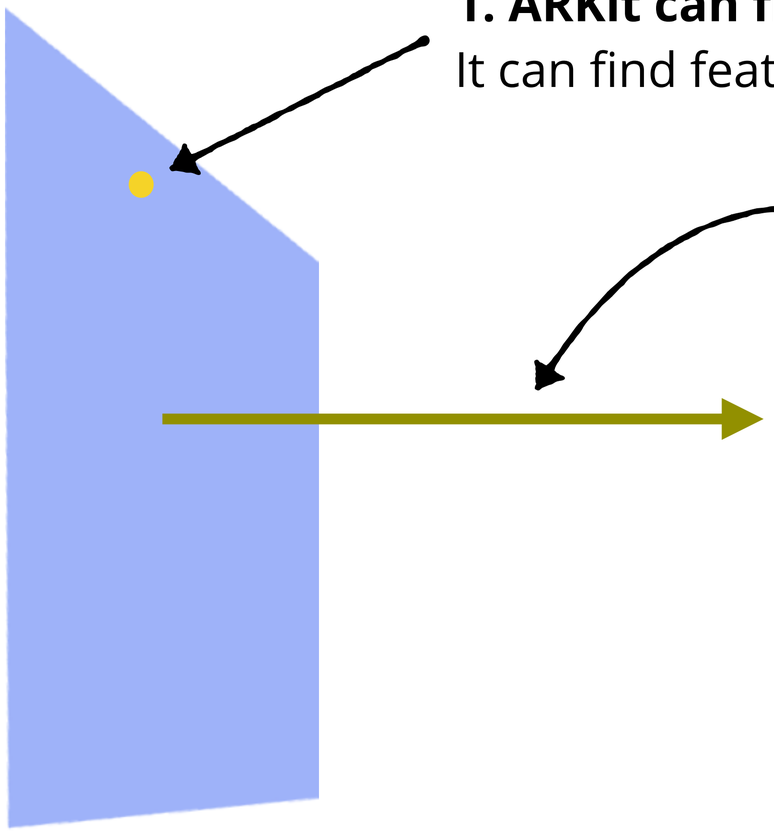


2. A vector *normal* (perpendicular) to the plane.
For horizontal planes, this vector is **gravity,** which the accelerometer can detect.



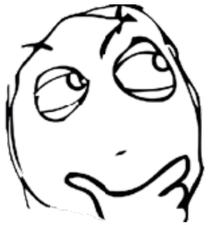
(Don't believe this? Try plane detection with the AR configuration's **`worldAlignment`** property to **`camera`**,
the one setting that doesn't align the Y-axis with gravity.)

Detecting vertical planes is trickier



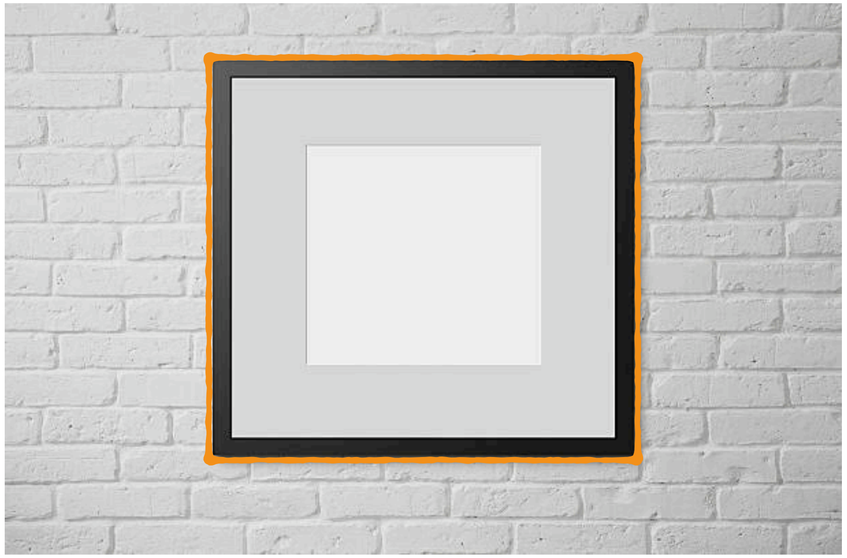
1. ARKit can find points on a vertical plane.
It can find feature points on vertical surfaces easily.

2. The hard part is *finding the plane's normal.*
With vertical planes, we don't have a handy force like gravity to rely on to find the normal.

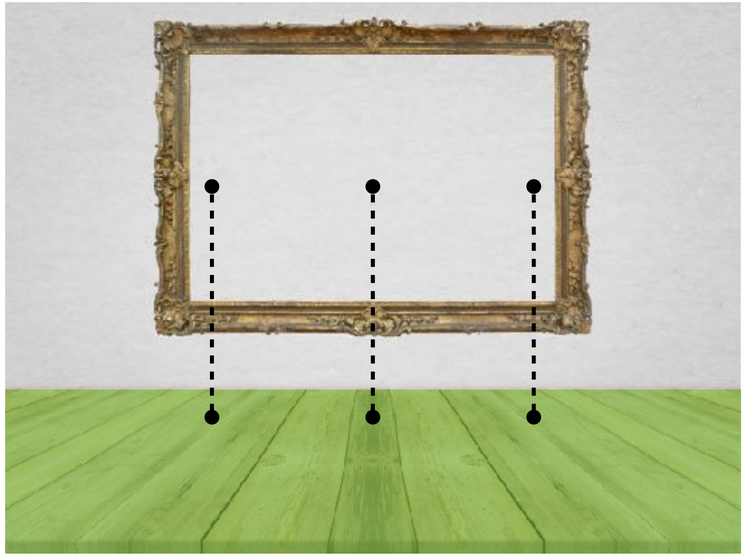


(Special bragging rights to the first person who can answer this question: "Why can't we simply use a vector that's *perpendicular* to gravity as the normal for vertical surfaces?")

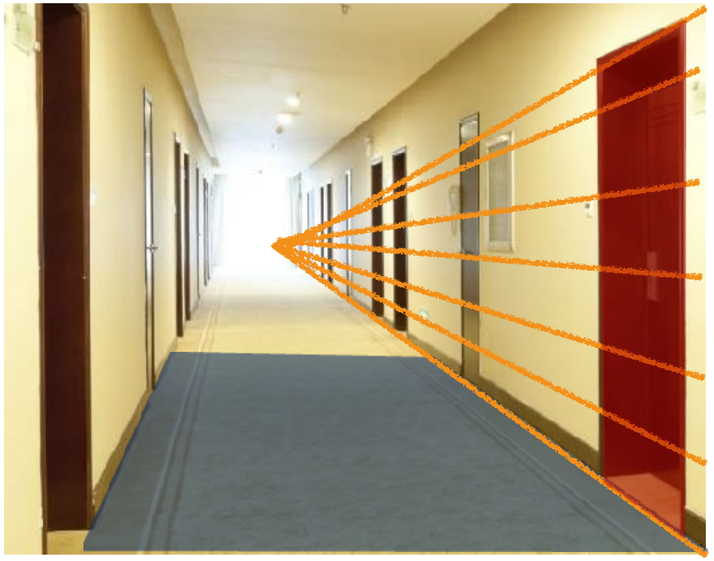
So how does ARKit detect vertical planes?



High-contrast borders



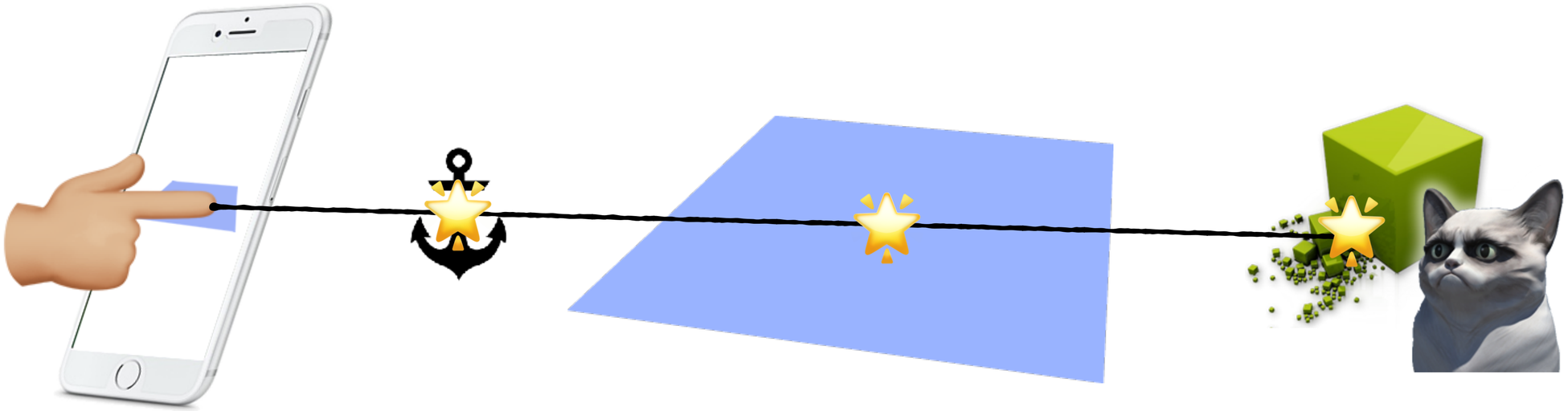
Points and
a known horizontal plane

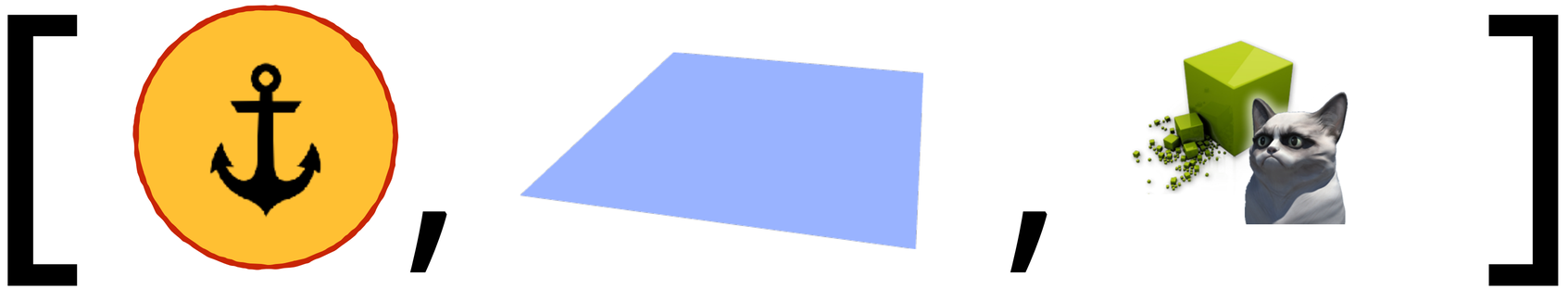


Raycasting

Hit testing

Hit testing answers the question “Does this 2D screen coordinate correspond to the 3D coordinates on an AR anchor, real-world feature, or virtual object?

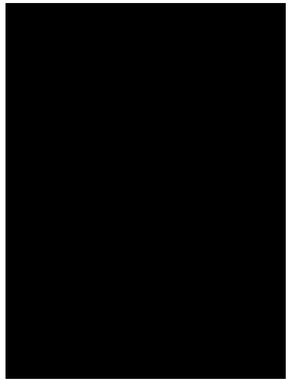




What causes tracking to fail?

Too few features

Too much motion



Too little light

Relocating

How well is the camera tracking?

This ARKit delegate method gets called whenever the camera's tracking state changes...

```
session(_:cameraDidChangeTrackingState:)
```

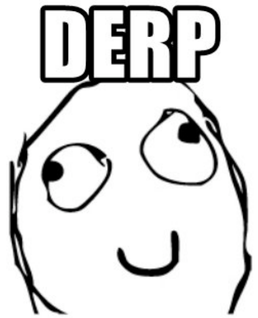
...and it gives us this very handy property...

```
ARCamera.trackingState
```

...which will contain one of three possible values:

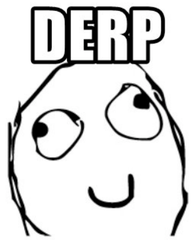
```
.notAvailable
```

```
.normal
```




```
.limited
```

Reasons why tracking is limited



If tracking is limited, you can check the **`reason`** property to see why.

`.initializing`

`.excessiveMotion`

`.insufficientFeatures`

`.relocalizing`

Demo 2:
Raykea

How Raykea works, part 1

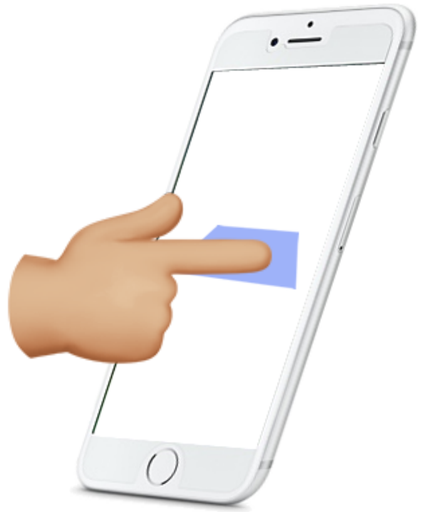
1. Continuously seek horizontal and vertical planes.

2. Cover any detected vertical planes with a poster.

How Raykea works, part 2

3. Cover any detected horizontal planes with a “place furniture here” grid.

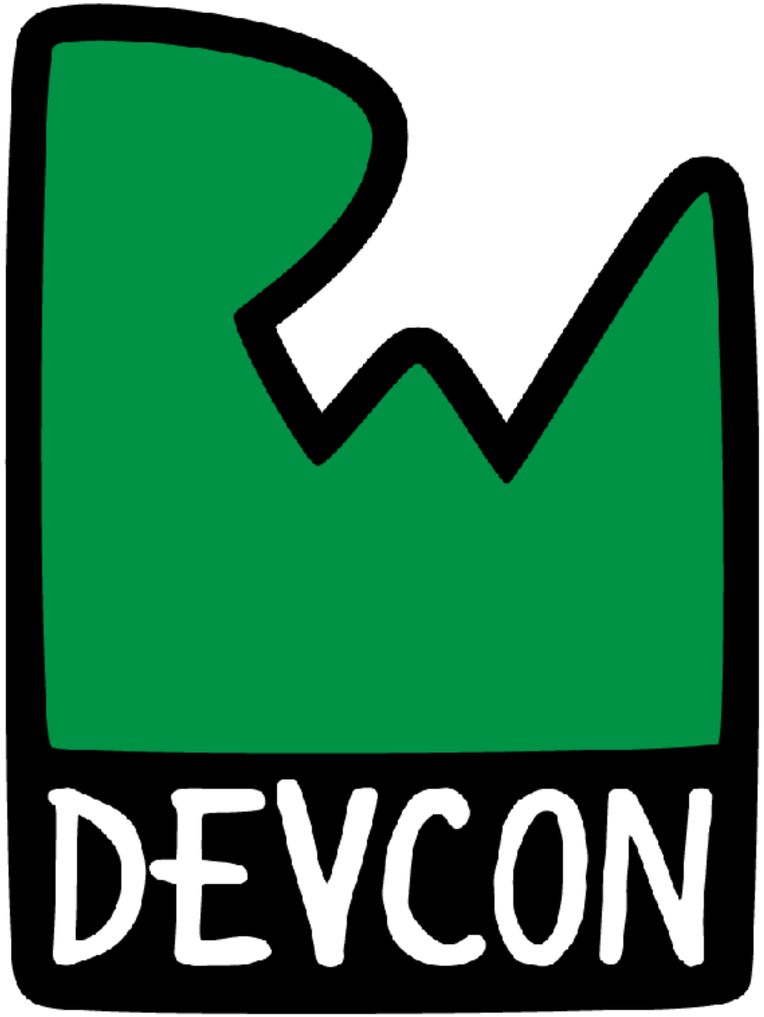
4. When the user taps the screen, perform a hit test to see if that tap corresponds to a detected horizontal plane.



How Raykea works, part 3

5. If the tap corresponds to a detected horizontal plane,
find the real-world coordinates that correspond to that tap,
and draw furniture at those coordinates.

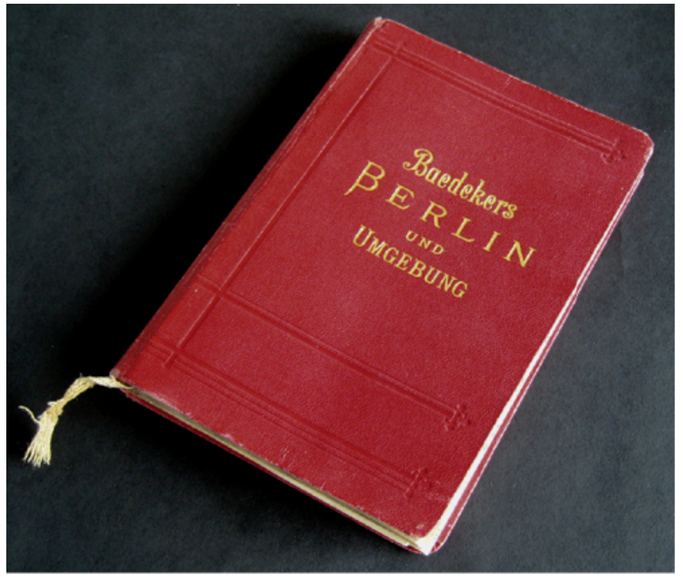
Workshop:
ARKit in Depth



BAEDEKAR

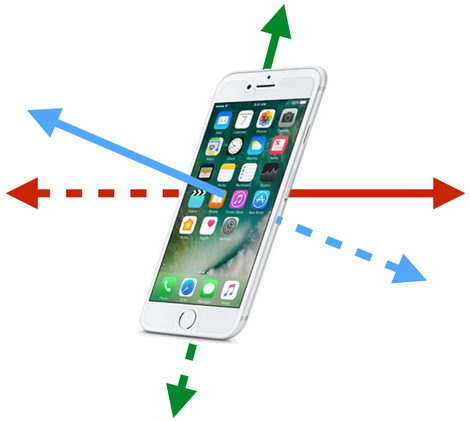
“BaedekAR”?

Baedeker is an old colloquial term for “travel guidebook”. The term comes from Karl Baedeker, whose publishing house started publishing world travel guides under their name in 1827.

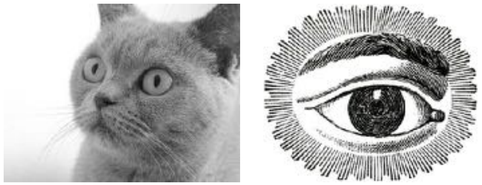


Fun fact: BaedekAR is the basis of an app that I’m actually writing for cruise lines and other tourism-based businesses. You just might run into it on your next vacation!

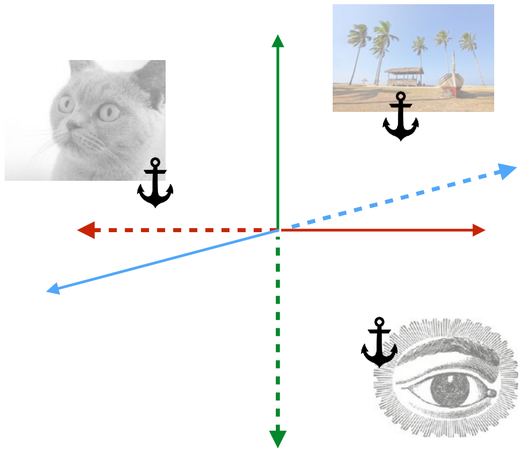
ARKit 1.5 and 2D image recognition



In ARKit 1.5, you can take an **`ARWorldTrackingConfiguration`**...

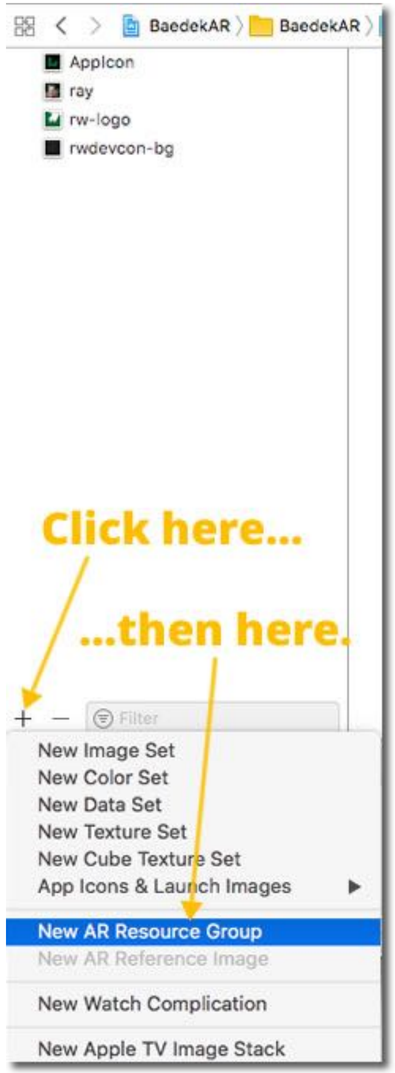


...and pass a set of reference images to its **`detectionImages`** property...



...then ARKit will place **`ARImageAnchors`** (a subclass of **`ARAnchor`**) in the scene wherever it finds any of those reference images.

Storing 2D reference images



There's a new asset type called an **AR Resource Group** that is used to store 2D reference images for image recognition with ARKit.

It's easy to load them into a **`Set`** of **`ARReferenceImage`**s, which you can then pass to the session's configuration via the **`detectionImages`** property:

```
let referenceImages = ARReferenceImage.referenceImages
                          (inGroupNamed: "AR Resources",
                           bundle: nil)
config.detectionImages = referenceImages

// Start the AR session.
sceneView.session.run(config, options: [])
```

The real-world size of reference images

In addition to providing a set of reference images, you also have to provide the real-world size of those images. Why?

It's because objects that are closer appear to be larger...

...than objects that are farther away.

Providing the real-world size of reference images is mandatory. If you don't provide the size for a reference image, ARKit doesn't report its presence in a scene.

Responding to recognized images


```
func renderer(_ renderer: SCNSceneRenderer, didAdd node: SCNNode, for anchor: ARAnchor) {
 // What kind of anchor are we dealing with?
 if let imageAnchor = anchor as? ARImageAnchor {
  // Code to handle recognized 2D images goes here
  }
}
```

This is the same method that we used when detecting planes in *Raykea!*

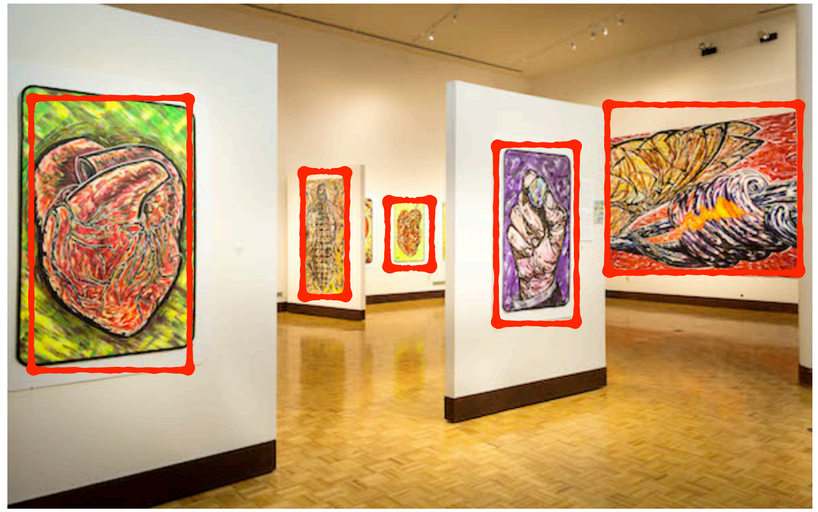
2D IMAGE RECOGNITION TIPS AND TRICKS, PART 1

Partially obscured images

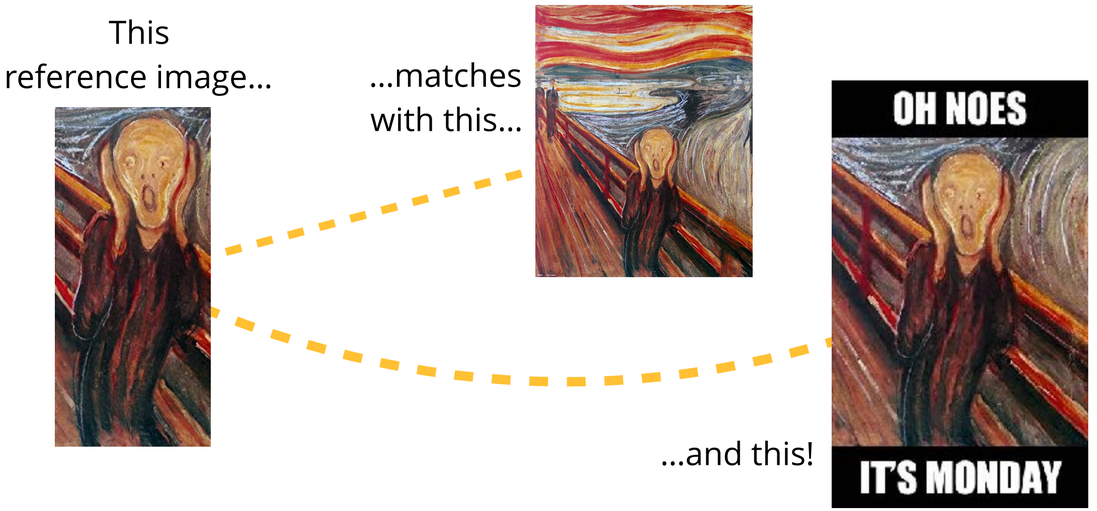
Works best with flat surfaces

Poor lighting conditions
and effects

2D IMAGE RECOGNITION TIPS AND TRICKS, PART 2



“Visual iBeacons”



Experiment with reference images

Drawing 3D text in SceneKit

Define the text content by creating an SCNText instance:

```
let myText = SCNText(string: “Hello, AR world!”, extrusionDepth: 0.01)
```

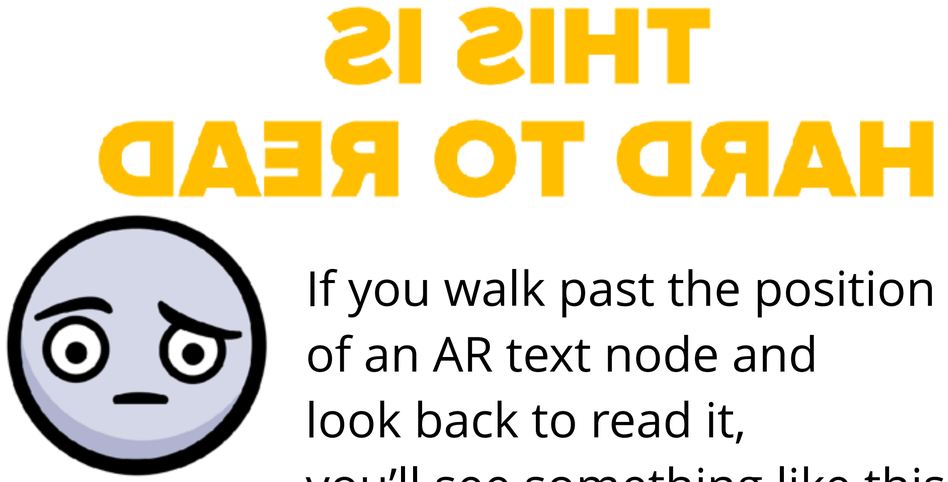
Define the text’s appearance by setting its properties:

```
myText.font = markerFont
myText.alignmentMode = kCAAlignmentCenter
myText.firstMaterial?.diffuse.contents = UIColor.yellow
```

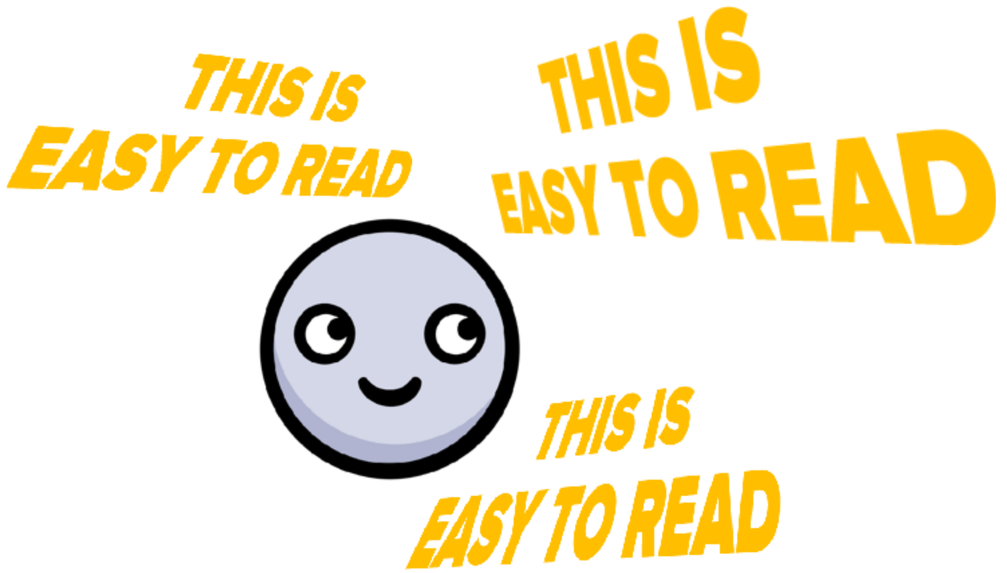
Create a SceneKit node using the text as its geometry, set its position, and add it to the scene:

```
let myTextNode = SCNNode(geometry: myText)
myTextNode.position = SCNVector3(0.0, 0.0, 0.0)
sceneView.scene.rootNode.addChildNode(myTextNode)
```

The problem with 3D text in an AR world



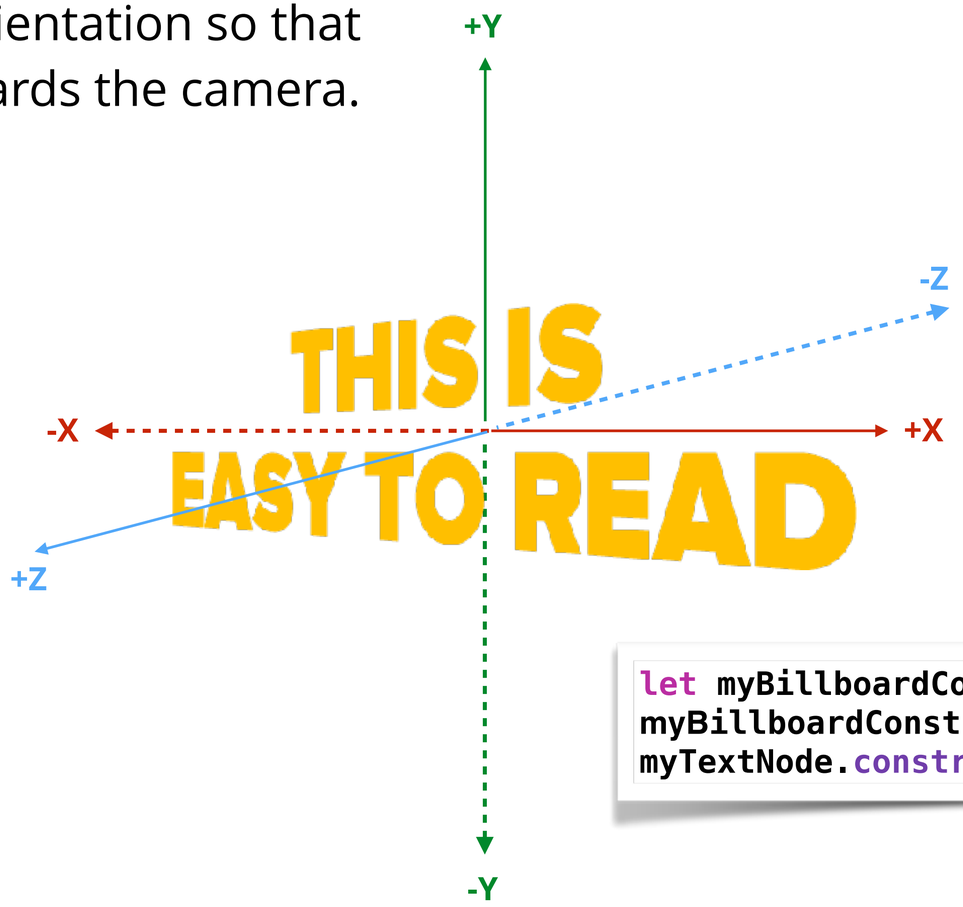
If you walk past the position
of an AR text node and
look back to read it,
you'll see something like this.



We'd like it if 3D text would swing around
to face the user wherever they went
so that it's always easy to read.

Billboard constraints

A **billboard constraint** automatically adjusts a SceneKit node's orientation so that it always points towards the camera.



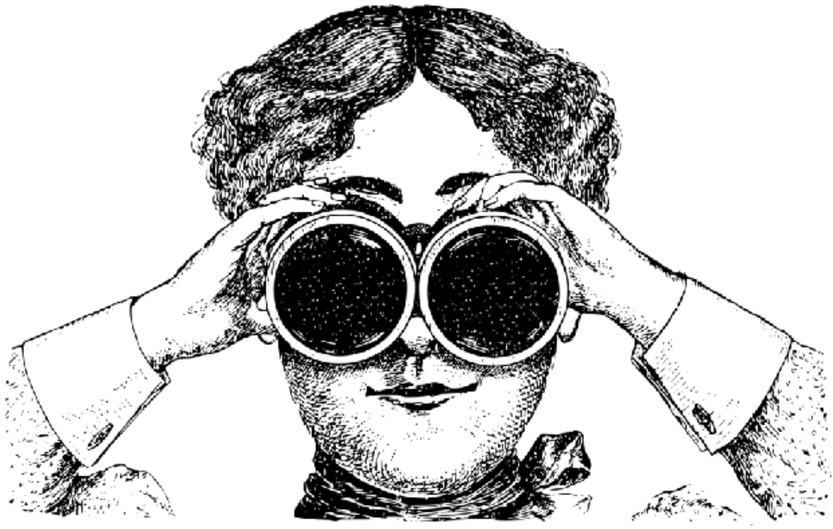

```
let myBillboardConstraint = SCNBillboardConstraint()
myBillboardConstraint.freeAxes = SCNBillboardAxis.Y
myTextNode.constraints = [myBillboardConstraint]
```

Demo 3:
BaedekAR

How Baedekar works, part 1

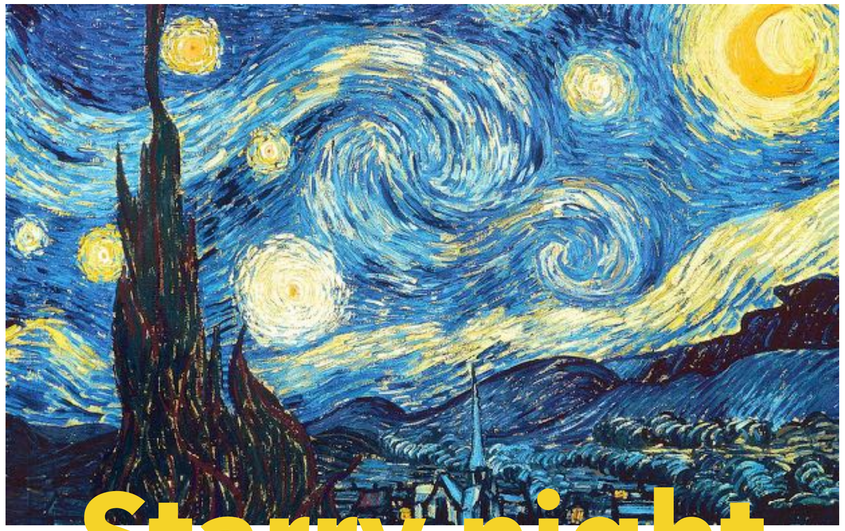
1. Provide the app with reference 2D images.

2. Continuously seek instances of the reference 2D images in the real world.



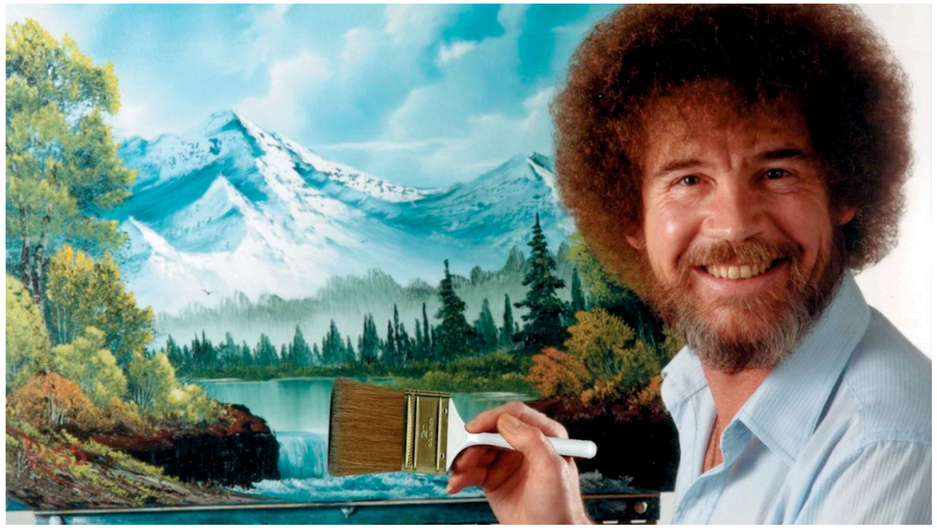
How Raykea works, part 2

3. If the detected 2D image's reference name doesn't begin with "block_this", overlay the image with a tappable hotspot and AR text displaying its name.

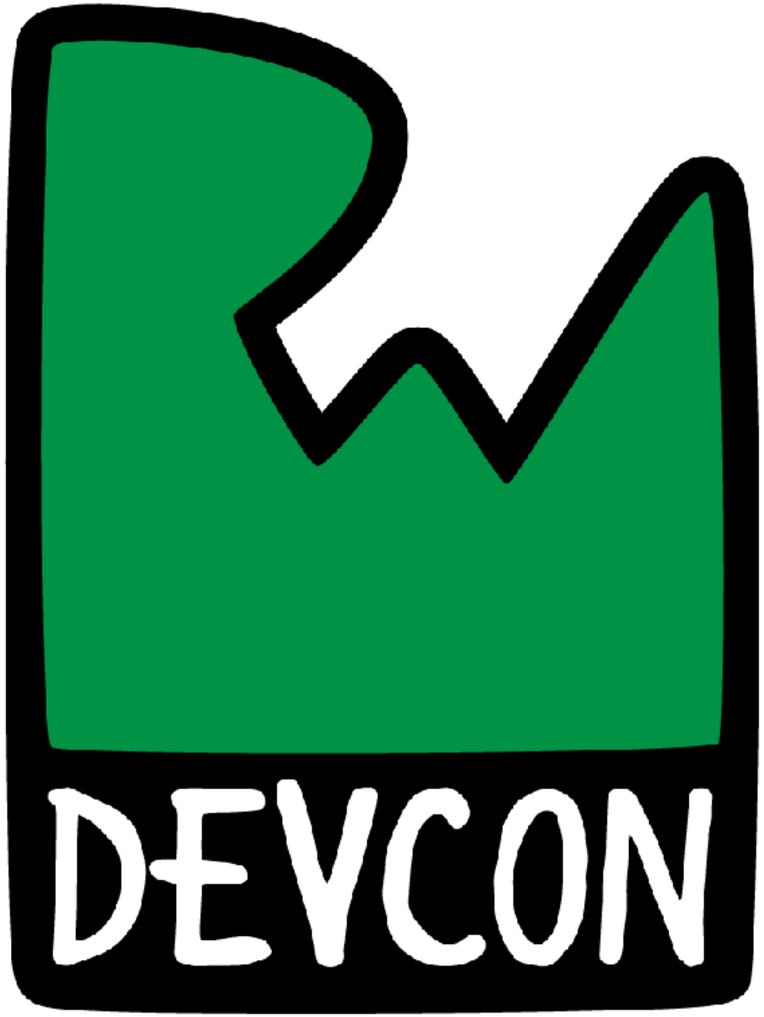


Starry night

4. If the detected 2D image's reference name begins with "block_this", cover up the image with a more comforting one.

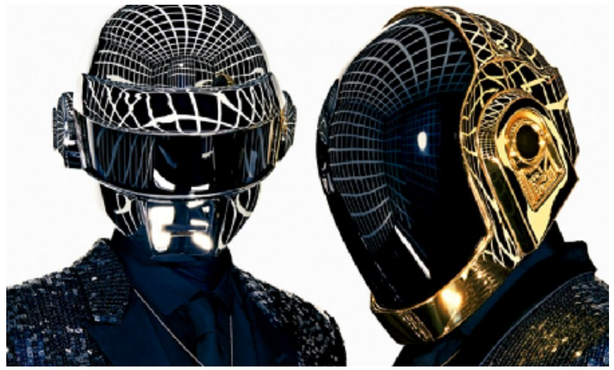


Workshop: ARKit in Depth

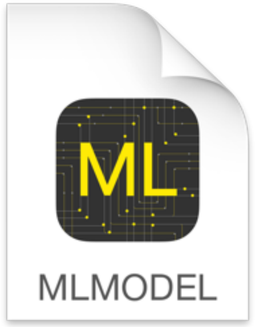


Vision Quest

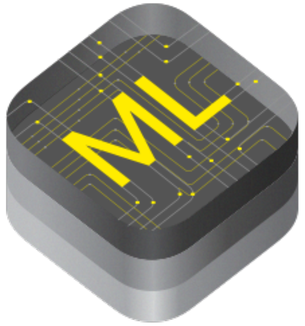
Categorization and machine learning



Models and Core ML



Think of a machine learning model as software...



...and think of Core ML as the hardware on which that software runs.

When you “run” a model on Core ML, you’re harnessing the model’s training to take various inputs and ***categorize*** them.

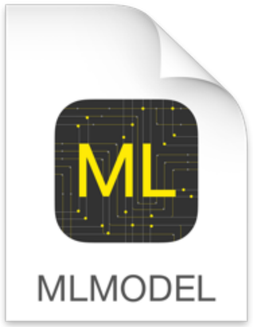
Metal

Pop

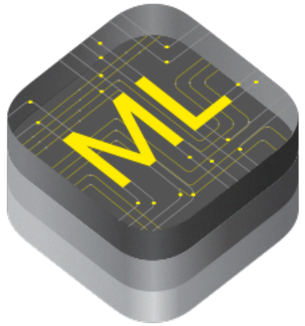
???

Image recognition with Core ML and Vision

Take some **image data**...



...and a
machine learning model
for ***categorizing images***...



...then use **Core ML**
to categorize the image
using the model...

```
Ray Wenderlich
(80% confidence)

Nicolas Cage
(12% confidence)
```

...and get the results,
along with Core ML's
confidence in them.

Sometimes the results
are ***hilariously wrong.***

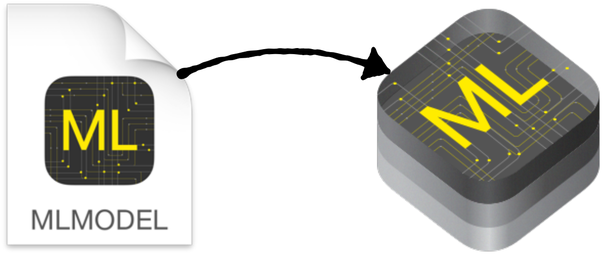
Actual sea urchin

(Hey, it’s pretty close…)

Demo 4:
Vision Quest

How Vision Quest works, part 1

1. Load the appropriate model into Core ML and request that Core ML use the model to classify images.



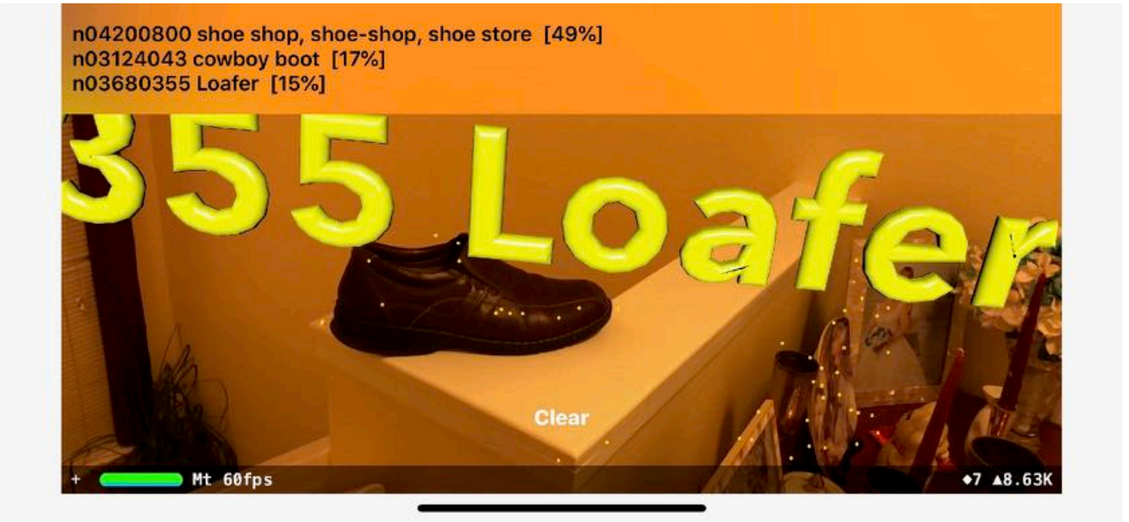
2. Continuously capture the camera image and feed it to Core ML.

How Vision Quest works, part 2

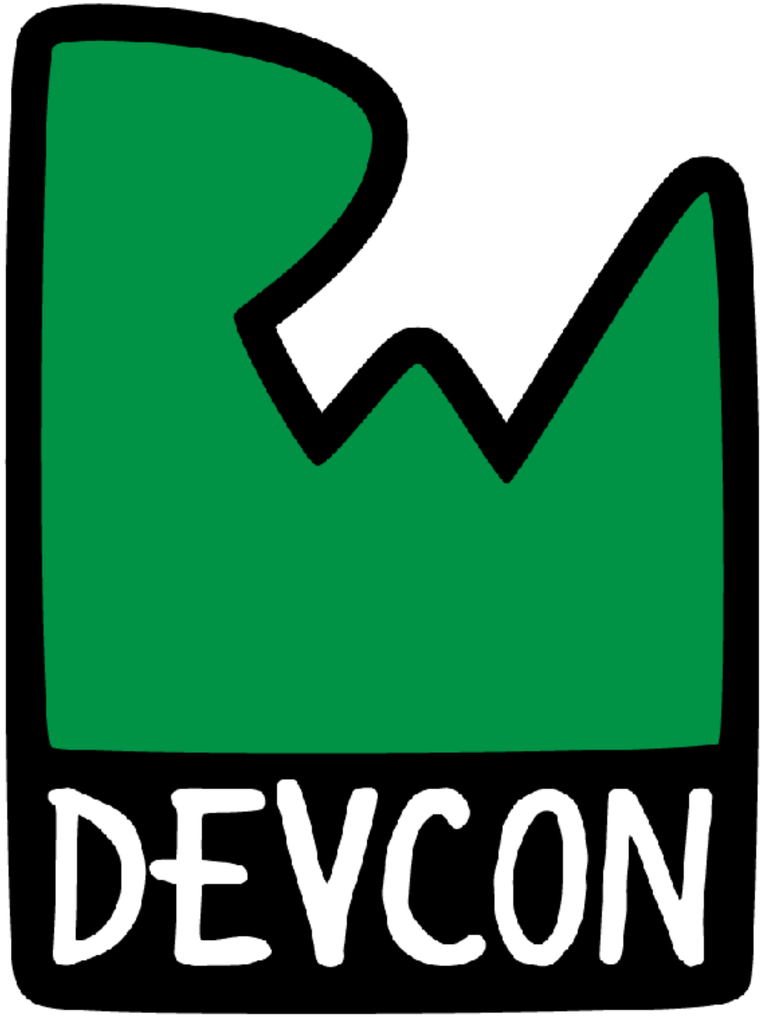
3. Continuously display Core ML's top 3 guesses of what the dominant item in the image is at the top of the screen.

```
Beer mug - 33%
Beaker - 10%
Glass jar - 5%
```

4. When the user taps on a feature point, draw the top guess and its confidence as AR text.



Session #:
Getting started with ARKit



Conclusion

What You Learned

- **Demo 1: *Happy AR Painter***
  Your first AR scene, where you added simple SceneKit geometric shapes to the scene and used the device's position and orientation.

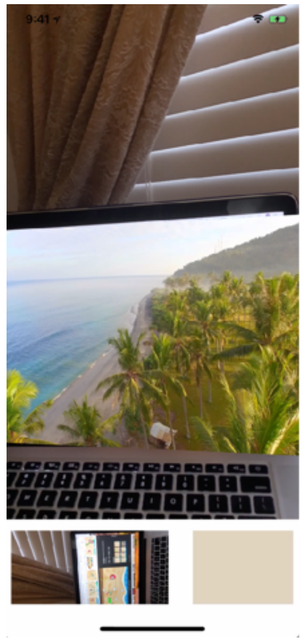
- **Demo 2: *Raykea***
  You took Demo 1's lessons, added 3D models, hit tests, and plane detection, and started interacting with the real world.

- **Demo 3: *BaedekAR***
  You used a new feature in ARKit 1.5 to detect, add hotspots to, and block known 2D images, and tag them with easily-read AR text.

- **Demo 4: *Vision Quest***
  Harnessing the true potential of AR by taking what you learned about ARKit and combining it with machine learning.

What You DIDN't Learn

Even at less than a year old, ARKit already offers so much ground to cover that a single workshop can't cover it all...



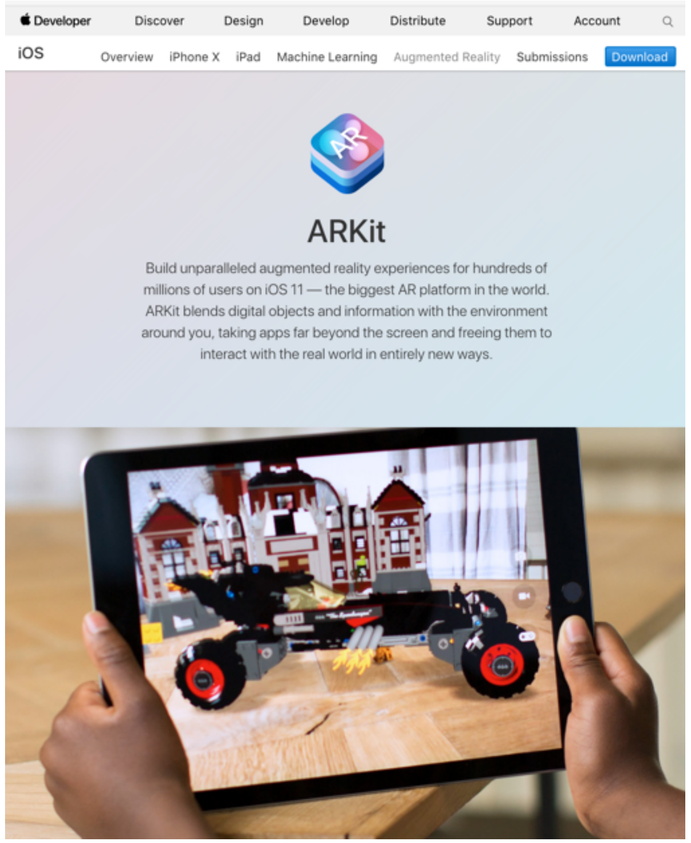
Embedding video in AR scenes

SceneKit physics

SpriteKit and ARKit

ARKit and Core Location

Where To Go From Here?



Apple's ARKit documentation

- Apple developer (developer.apple.com.arkit)
- Apple's human interface guidelines (developer.apple.com/ios/human-interface-guidelines)

"Awesome ARKit" repo on GitHub

github.com/olucurious/Awesome-ARKit

RayWenderlich.com and ARKit

ARKit by Tutorials

Coming soon!

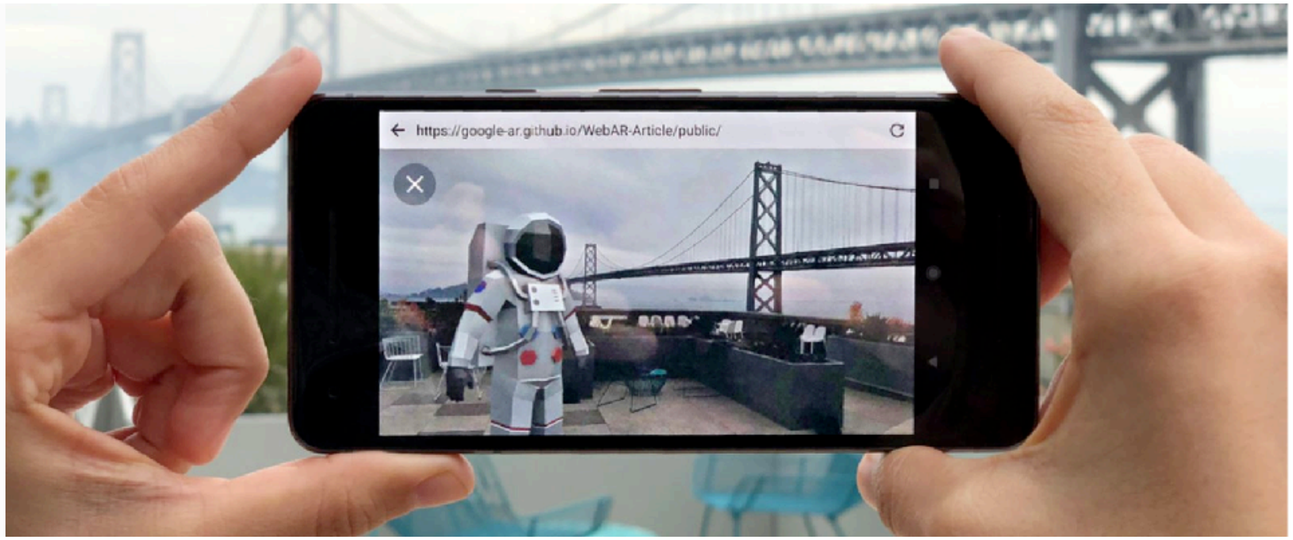
Articles

Coming soon!

Videos

Coming soon!

These are the early days of mobile AR



You have a responsibility

Competitor! I'm going to cover them up.

Totally sketch neighborhood

You don't want to know what this guy's into

Pronounces the word as "nu-cue-ler"

Who made this pile of dirt? *Aliens*.

Tap here to get demographic info on this guy based on his shoes

How to find me online

***Global Nerdy,* my tech blog**
globalnerdy.com

Twitter
@AccordionGuy

LinkedIn
linkedin.com/in/joeydevilla