ARKit from a UIKit Developer

In which I talk about something I'm wholey unqualified to.

TL;DR:
There Ain't No Such Thing As a Free Lunch

I promise you this won't be a liberatian rant. It was just harder then I thought.

I'm not a 3D graphics developer, that is a different skill set. When I approached this problem, I assumed Apple abstracted away a lot of problems, much like UIKit does. It doesn't. The scope of ARKit is smaller then you'd thing. A lot of left to the developer.

Which is a crazy complex thing to do with a single camera. It leaves the complex "patent-worthy" work to Apple and leaves you to implement features.

So, if you have done any 3D programming, I'm sorry. This talk will bore and probbly infurate you.

I'll do my best to explain terms as they come up, but if there is something you don't understand just stop me and ask.

What ARKit *Does*

ARKit is good at putting an object in a 3D space that represents the real world.
And keeping it there.

At a high level, ARKit is about creating a X/Y/Z coordinate system for the real world.

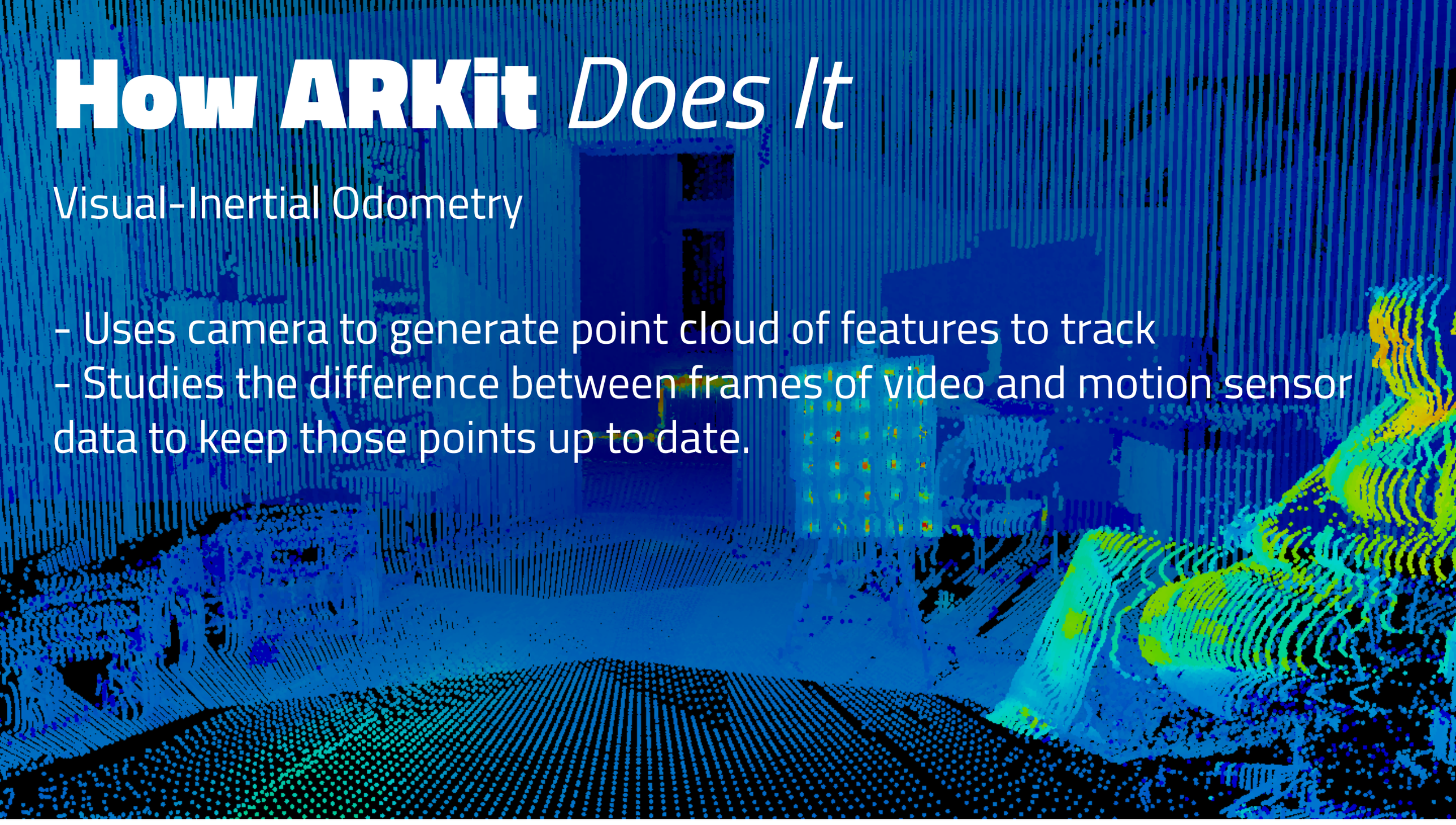
If this room is a box, ARKit keeps a virtual box lined up with it.

The rest is up to you

How ARKit *Does It*

Visual-Inertial Odometry

- Uses camera to generate point cloud of features to track
- Studies the difference between frames of video and motion sensor data to keep those points up to date.



Features are basically just a pixel its determined is relevent.

Its assume a mostly static world. Since its taking it a lot of points, it can close track of a many (occusion, someone exits a room), but keep them up to date based on the data of other points and motion sensors.

But if something is making a lot of movement, it just won't track it.

Getting Started

```
import ARKit
import SceneKit
```

Need to import these to do anything

Getting Started

- ARSCNView
- ARSession
- ARWorldTrackingConfiguration

ARSCNView seems to be the core of ARKit. It is a UIView, to render the camera input. Keeps the "virtual scenekit camera" locked to the movement of the phone, and the "scene" mapped over the camera. But you can ignore all that and pretend its a magic UIView that does AR.

ARSession coordinates everything. All you need to know is you can give it a configuration on what you want to track

ARWorldTrackingConfiguration seems like a sensible default tracking config. Basically gives you everything you'd want.

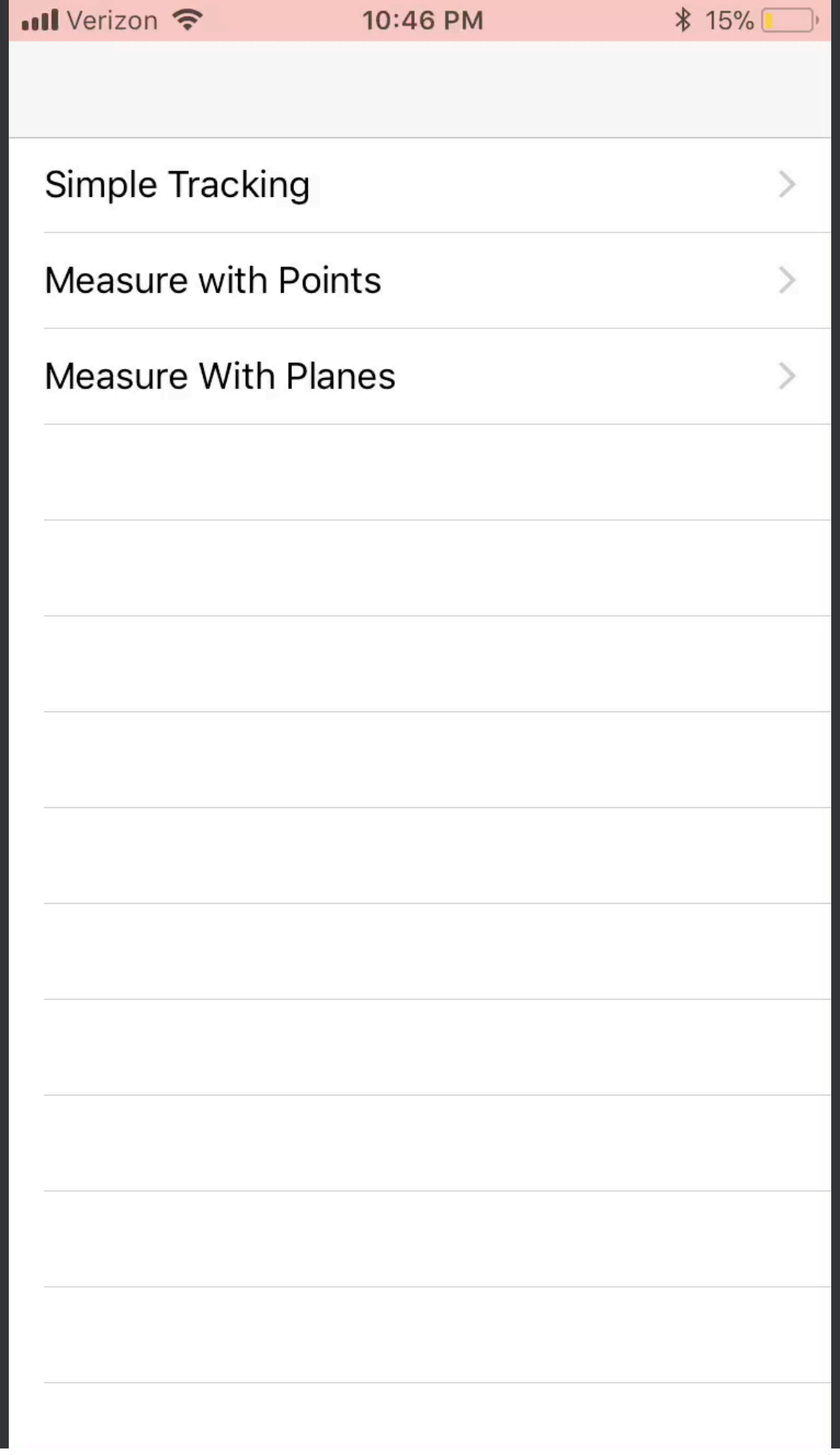
DEMO

Add the view, start the session.

Delegate callback will let you know when the session has gathered enough data to "know" the scene

Transform is position in the "global world coordinate". Not just XYZ, but orientation (ie. which way is it pointing)

A node is a base SceneKit concept for an object in the world, visible or not. Here, just a simple sphere.


Verizon
10:46 PM
15%
Simple Tracking
Measure with Points
Measure With Planes

What Else is in the Box

- Horizontal Plane Detection
- Lighting
- Scene Kit

So, what else does it do? It can detect horizonal planes. Desks, floors. Weirdly, not ceilings. Vertical planes are up to you.

Read the lighting in the room and adjust models automatically, to make it look like they fit in the world and not just float there

Its build on SceneKit, so a lot of libraries/ideas/math carry over. Gravity, collections, etc

Then things got complicated

The original idea was to detect or measure a box, to make shipping easier.

So, where is where my knowledge wasn't enough. I wanted to *know* something about the scene, but that isn't what ARKit was about.

Lets simplify the problem, not measuring a box. Just measuring a line. We have to detect two points, and then the math to figure out how far apart they are is known/trivial.

First Attempt

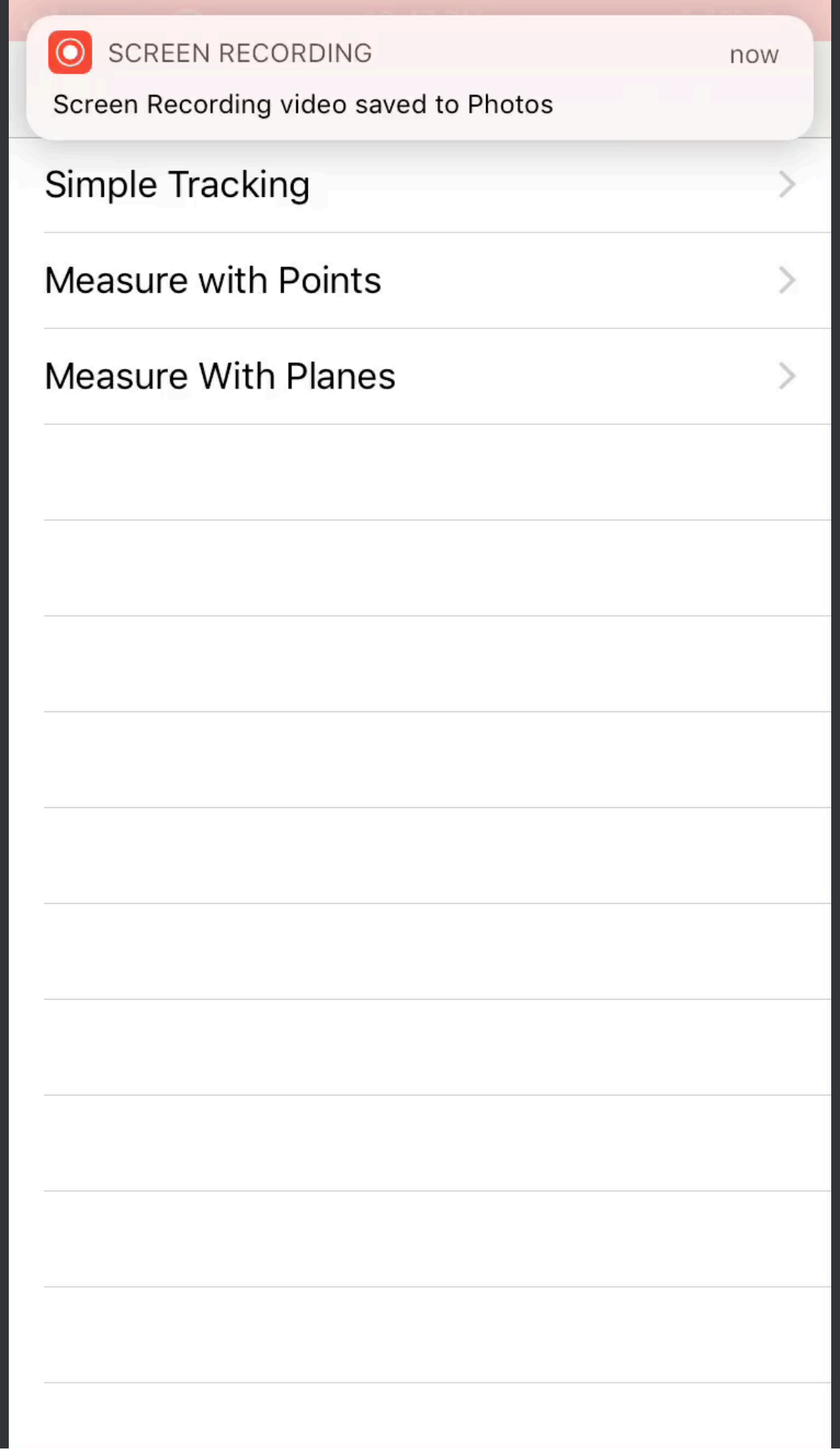
```
sceneView.hitTest(point, types: .featurePoint)
```

To get a point in the world coord space, you perform a hit test. Given this X/Y coordinate, where does that intersect the "known world". Draw a line from the coordinate you give it, where does it intersect with ARKit's known points. Basically, where does it hit something physical.

Since the tracking is imperfect, it was not precise enough. It could get it wrong. I know I'm pointing at a box, but ARKit would put the point between me and the box.

Couldn't guarantee accurate results, so basically useless.

DEMO


SCREEN RECORDING
now
Screen Recording video saved to Photos
Simple Tracking
Measure with Points
Measure With Planes

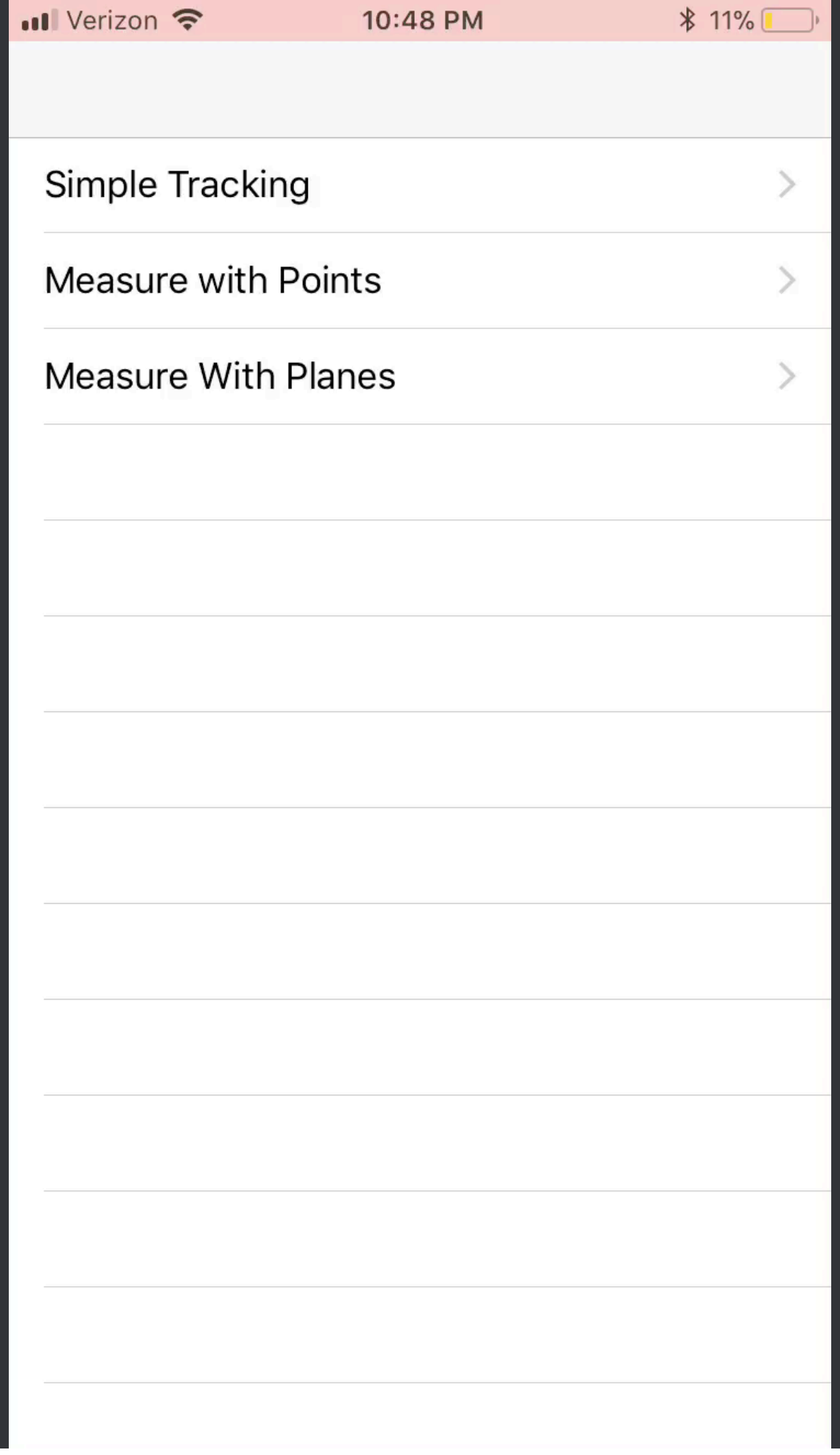
Second Attempt

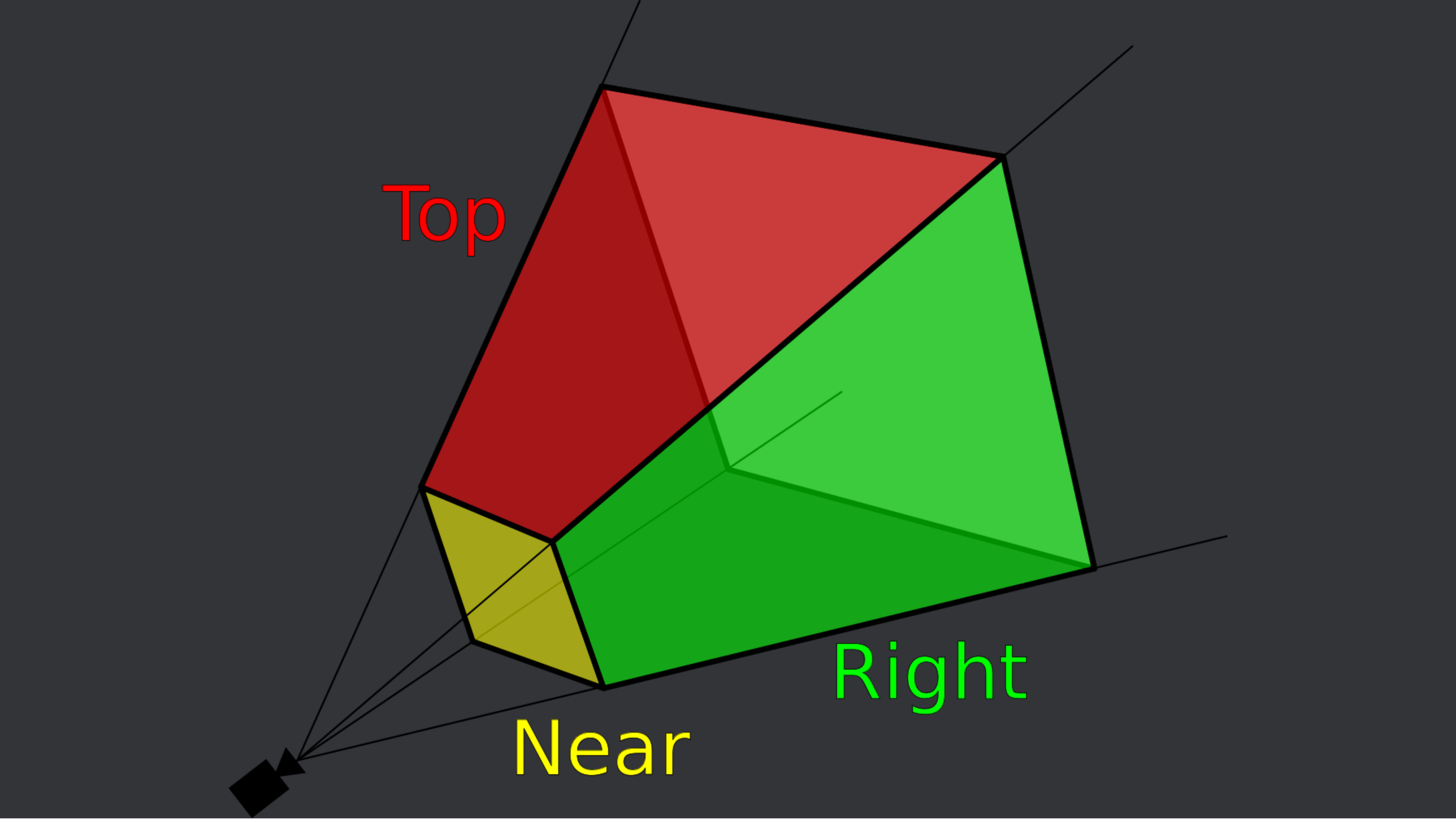
```
sceneView.hitTest(position, types: .existingPlaneUsingExtent)
```

Have it detect planes. More information. A single errent point in the cloud won't throw off our calculation, because they've been normalized/collected into a plane

But ARKit by default only detects horizontal planes. Maybe in an update they will do vertical. So, measuring a desk or a floor gets simpler, but measuring a box is still really hard

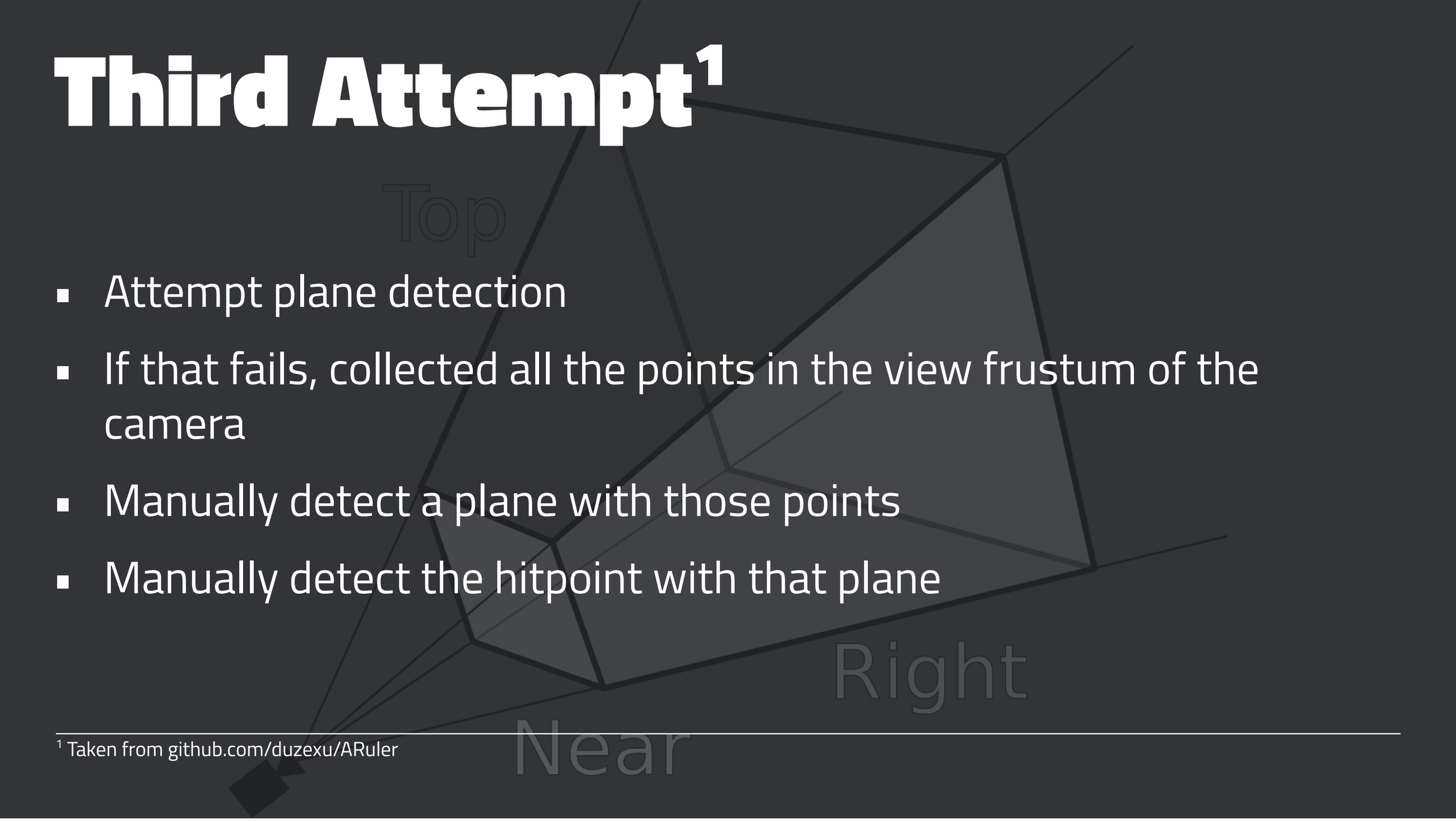
DEMO


Verizon
10:48 PM
11%
Simple Tracking
Measure with Points
Measure With Planes



Some terminology. View Frustum is basically geometric representation of what the camera can see; the field of view. A pyramid extending out of the camera

Third Attempt[1]



- Attempt plane detection
- If that fails, collected all the points in the view frustum of the camera
- Manually detect a plane with those points
- Manually detect the hitpoint with that plane

[1] Taken from github.com/duzexu/ARuler

So, this is a more robust method that applies existing 3D programming techniques to ARKit. Fairly 1:1.

A this would work, we could probably ship this. OpenCV is a C library that can do a lot of this math for you. But, I think the point I'm getting at here is its not free, its not all there for you.

In the end, I think its the start up time that actually killed this feature within our app. Asking the user to move their phone around for a few second to measure a box was just kind of silly

Take Aways

- BYO Features
- Slow start up
- Trust
- UX

Bring your own features. It is giving you a starting point. Its not going to tell you what you are looking at (queue Machine Learning). The rest is up to you.

You have to wave your phone around for 2-5 seconds before anything happens. Personally, I think that is killer for use within a larger app. If its your entire app, then sure, its your startup cost and that's fine. If its a modal you pop up, it has to be pretty damn compelling to get a user to wave their phone around for 5 seconds.

You have to trust it the point cloud it is making. After experimenting with it for awhile, I trust it to be internally consistent. Its good at keeping the world it makes consistent. Its when you take those points *out* of that world there things get hard. (ie. measuring). Take the planes for example, they never line up with the real world. Larger/smaller. Don't cover the entire floor.

The user has to use it right. Which is much harder then an app becuase the possibility space is literally the *entire world*

Apple North Michigan Ave

This is our last meeting at this Apple Store. I still haven't memorized the elevator code. Met a lot of great people because of CocoaHeads and this Apple Store. Got a job that I love because of this Apple Store. New one looks really cool. I won't forget this one.


IT'S ABOUT HOW MUCH COPPER WIRE YOU CAN GET OUT OF THE BUILDING WITH.