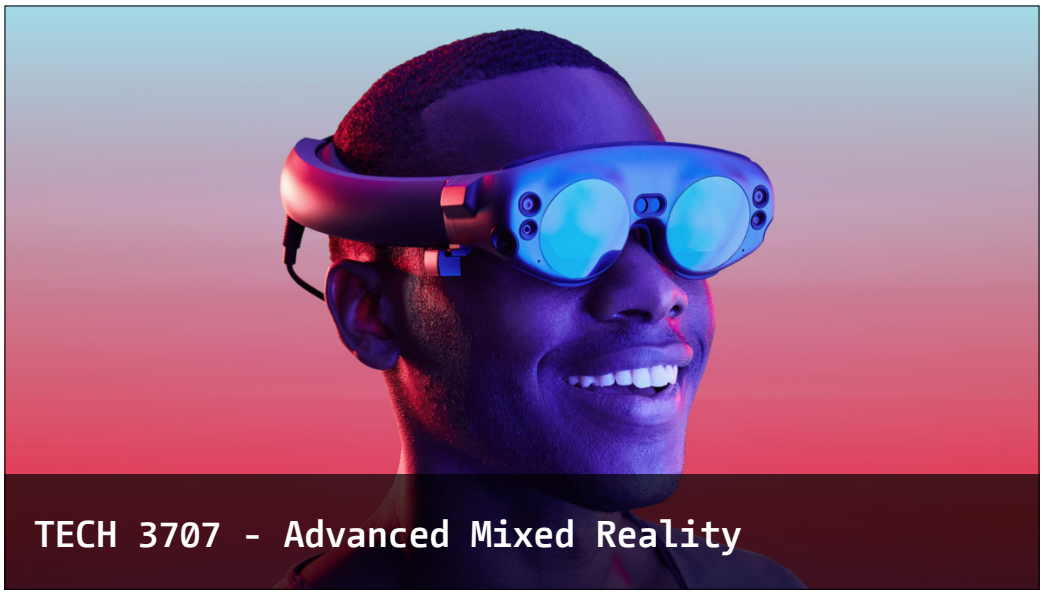
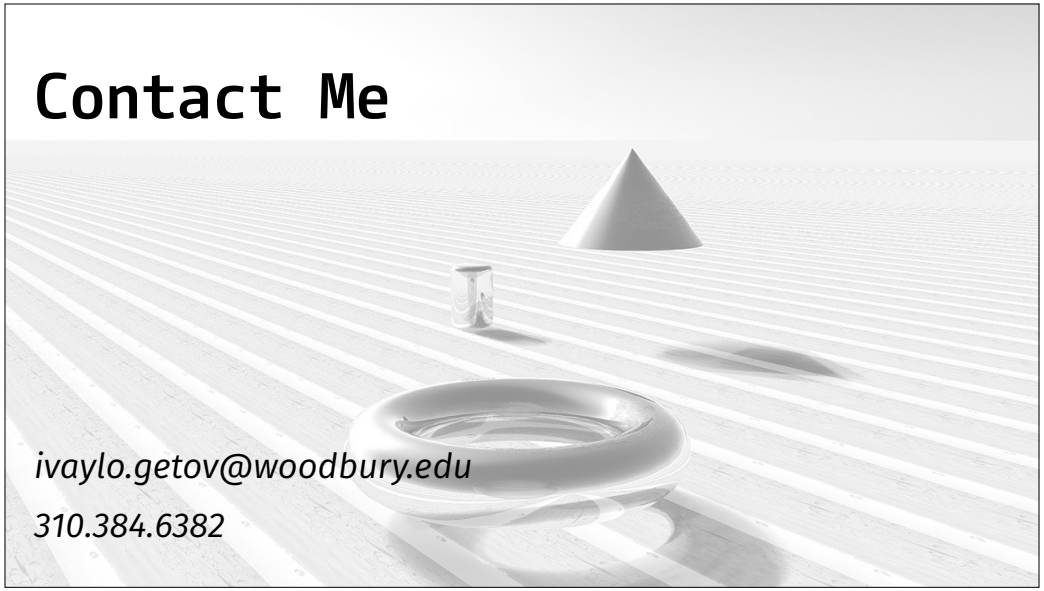

TECH 3707 - Advanced Mixed Reality

You guys all know the drill by now. Syllabus is here but always expect changes!

Github is the latest

I will do my best to keep the Moodle page updated, but I can promise that most up-to-date class materials will be here:
https://github.com/ivaylopg/AdvancedMixedRealityStudio

Contact Me



ivaylo.getov@woodbury.edu

310.384.6382

What do we mean by “ADVANCED” Mixed Reality

First: What is a programmer?

(Intro) Mixed Reality Studio was about covering a lot of different stuff:

Virtual Reality Hardware
Game Design
UX Design
Project Management

We aspired to this…

…but it’s OK if we made this in the end. (It’s still a robot!)

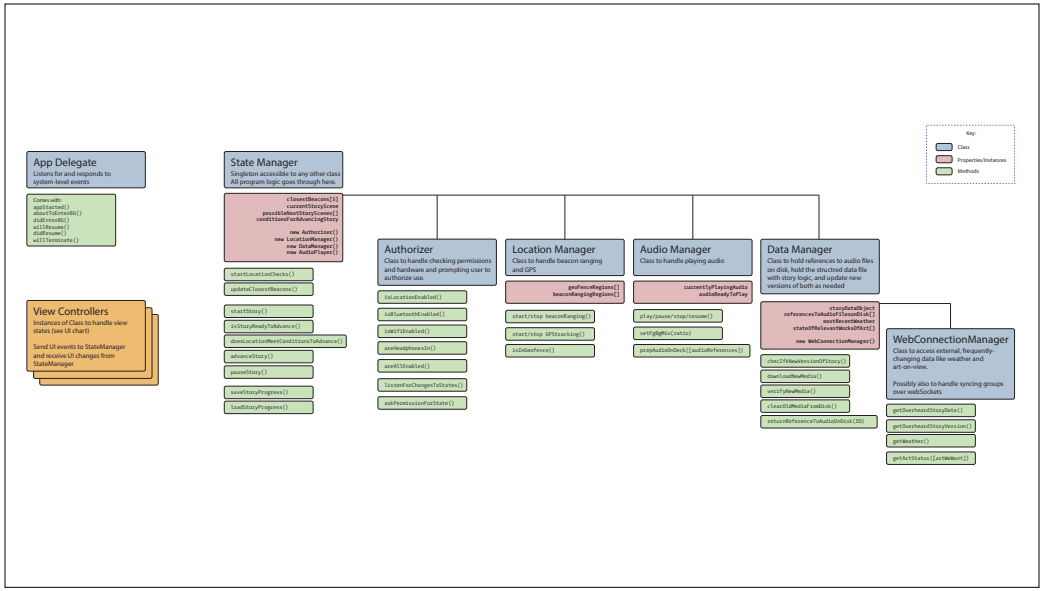
Come up with examples on the board range of things we know a little about:
Virtual Reality Hardware
Game Design
UX Design
Project Management

Now we're going to drill down into the specifics of a slightly narrower field. We're going to take our broad knowledge and apply it to a more constrained area.

The horizontal is still important. It's what makes you *better* at the vertical!

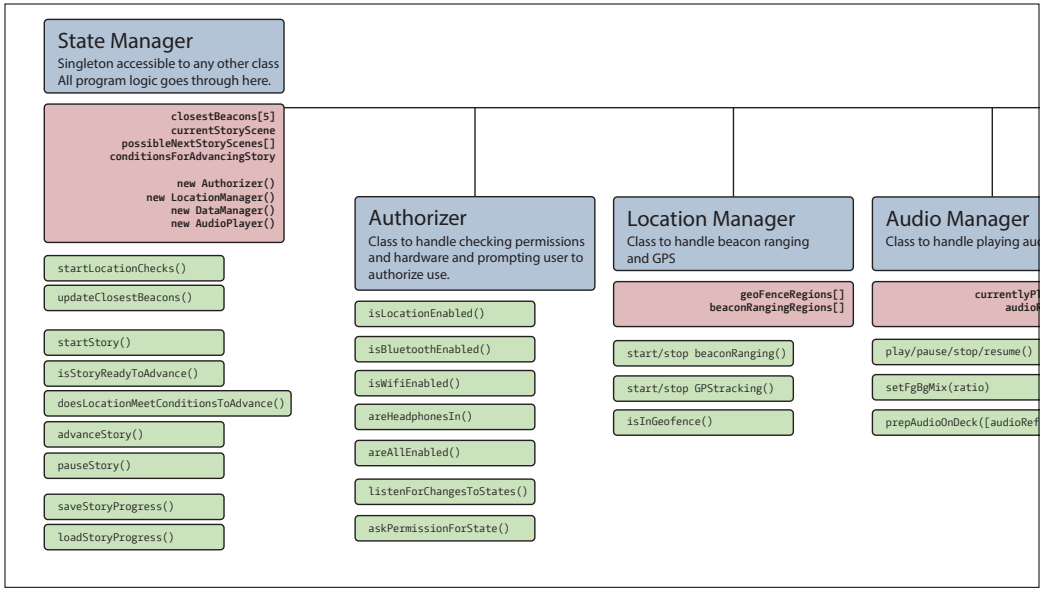
(This is called the T-Shaped Designer/Developer/Engineer/Person)

I’m also going to hold you more accountable for good practices

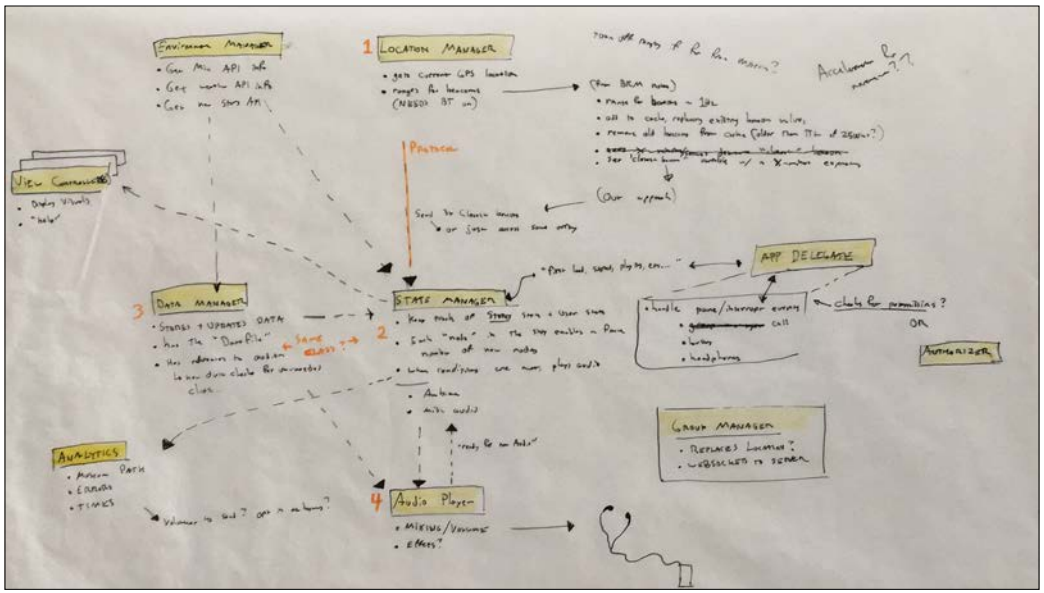


- I’m also going to hold you more accountable for good practices

- PLAN IT OUT
- Here is example of a way to organize an app with lots of components

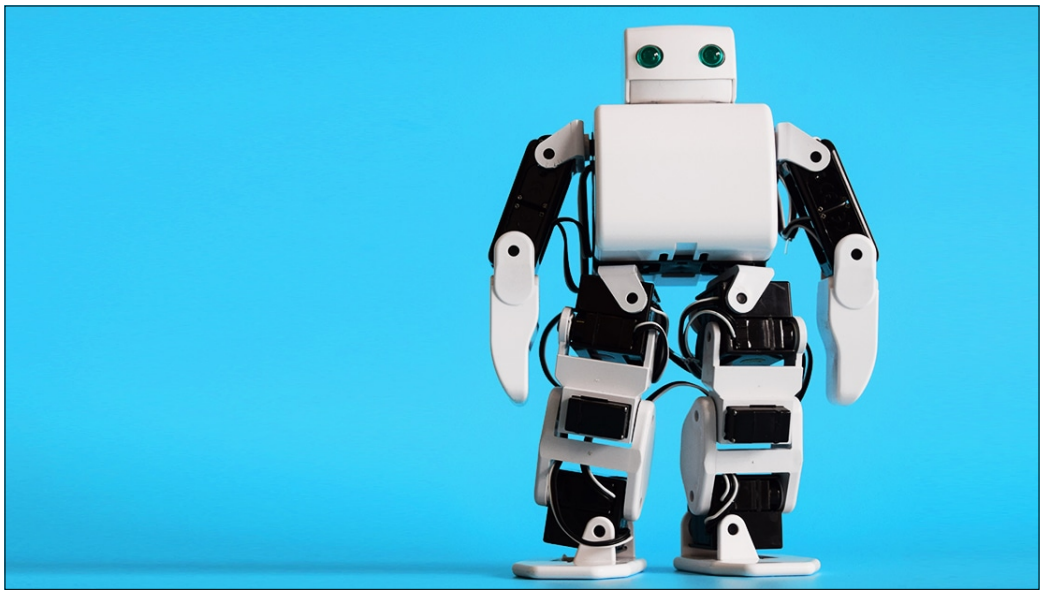


Even for simple programs, helps to organize into well-defined pieces (classes) and the variables & functions that go along with each piece.



- Plan doesn't have to be beautiful.
- Here is the same chart, before it was cleaned up and finalized

So even if we aspire to this and don’t quite make it…



…we should still be landing somewhere around here


What are we focusing on?

Mobile AR

Handling real-world coordinate spaces

Then: adding networking.

This could be inter-device communication…

… or interfacing with IoT

Finally: Publishing an app to the app store.


Assignments
First Project - Interactive AR - 20%
Final Project - Shared AR Experience - 30%
Participation, attendance, and classwork - 50%

Course will be organized into chapters


Format
Chapter 1
Expanding on Fundamentals
A series of class discussions, speakers, and exercises.
Deep dive into specific concepts and explore techniques.

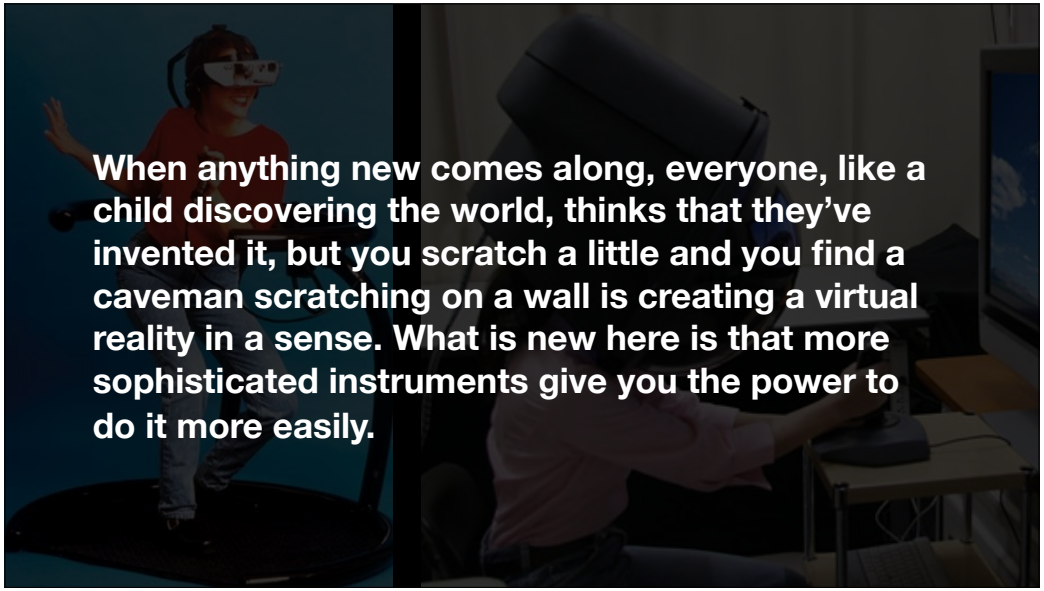

Format
Chapter 2
Interactive AR
We will be developing a mobile AR application
based on the Vuforia framework


Format
Chapter 3
Shared AR Experience
We will build on the previous project
to develop a multi-user mobile AR app

You are not required to have your own computer…

… especially since we'll be doing a considerable amount of work on macOS and iOS once we start building to devices

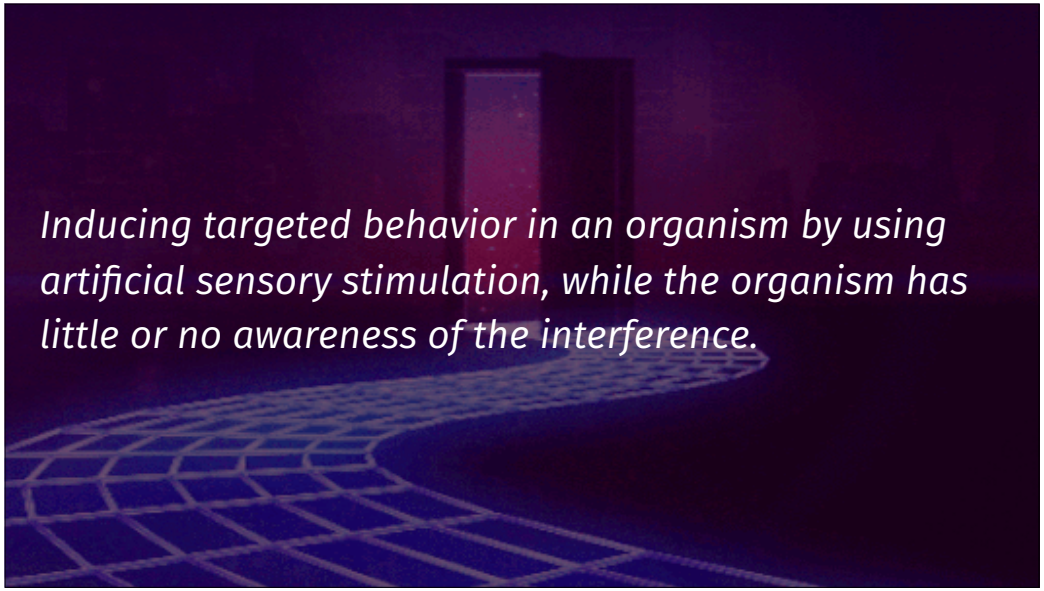
Lots of info coming your way today. Should mostly be a recap though…



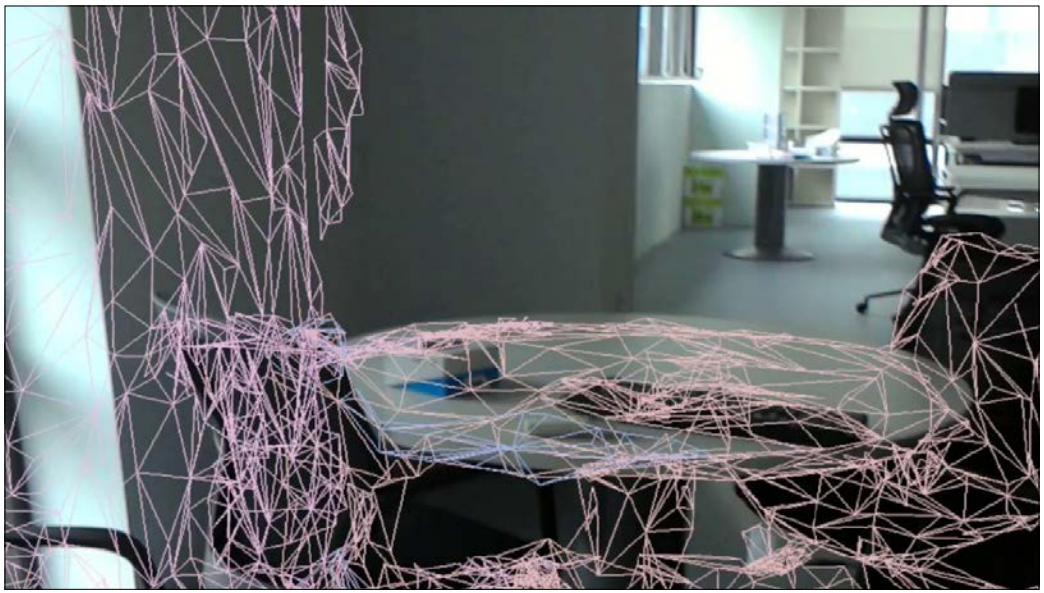
Morton Helig (Sensorama and Telesphere inventor)

With VR we could create everything!
In AR, we have to (get to?) work as a layer on top of the real world (this is actually *harder*!).

Let’s take a look at our old definition again…

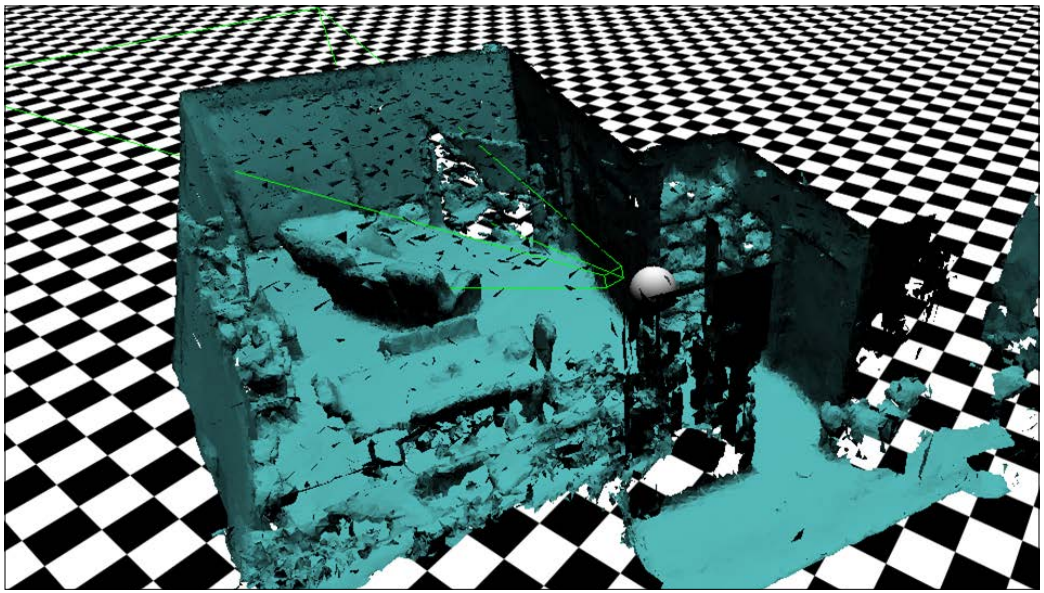


This may need to be revised a bit. AR is generally more actively engaged in that “interference”

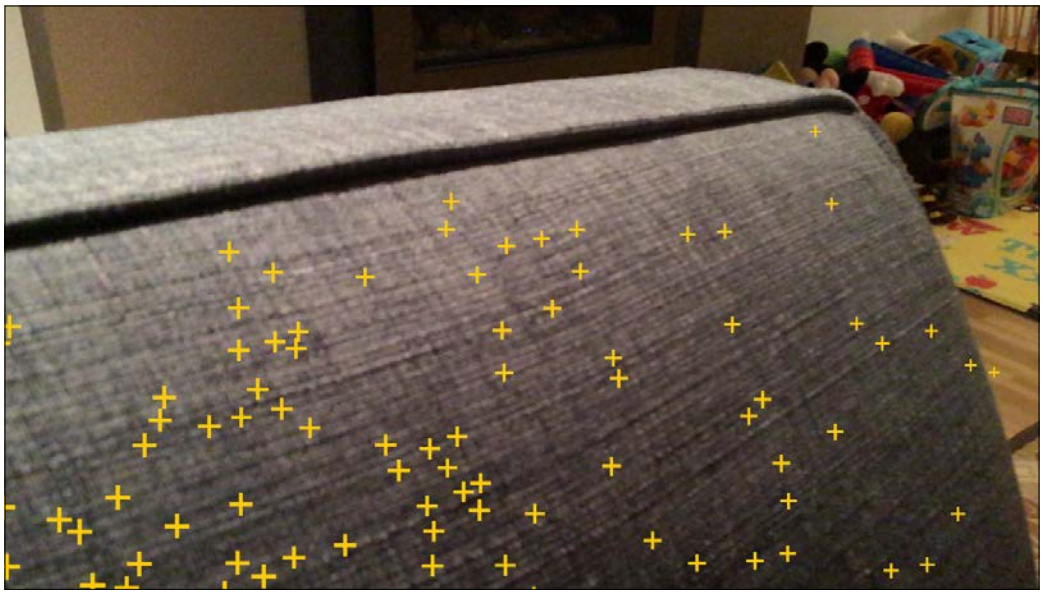


Awareness of the world: 3d Sensors (yay occlusion!)

Similar to what a Kinect does - gets a 3D model of what it can see - but the hololens continues to build that model as you move around.



The hololens creates an increasingly detailed model of your space as you move and look around.



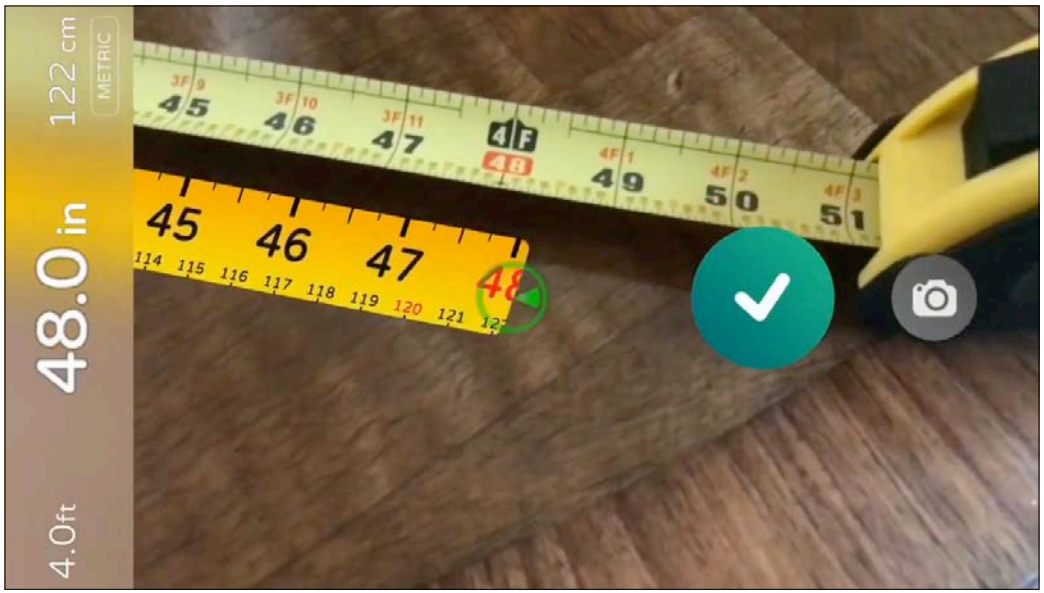
Awareness of the world: camera + feature-points.

ARKit/ARCore devices that don’t have depth sensors try to lock on to details (features) in the images that they see, and then do complex math in-between each frame to see how those feature points have changed.

This lets us do some exciting stuff in/to the world.

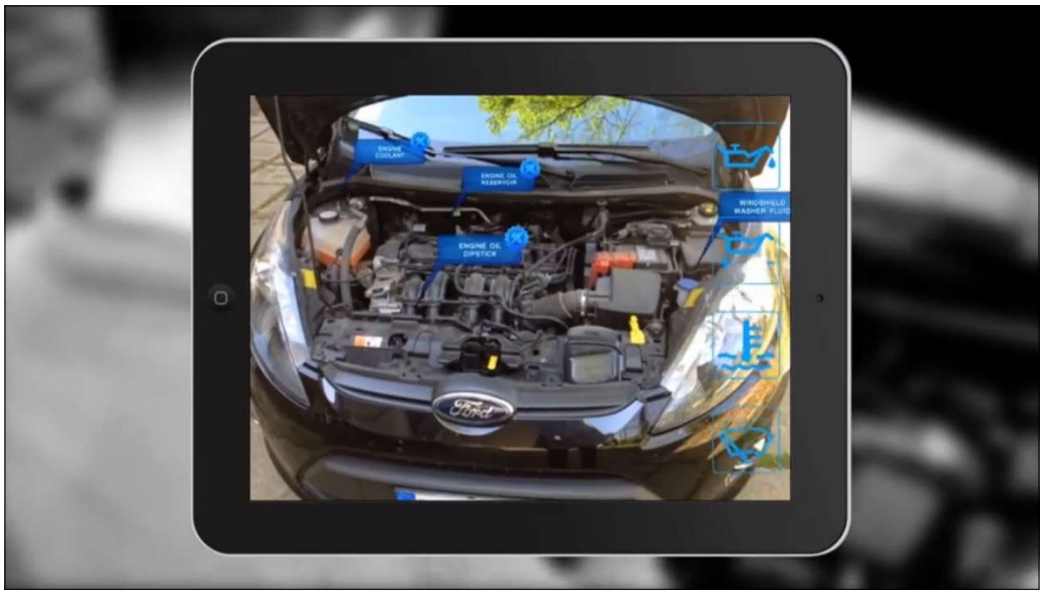
Games

Interactive storytelling.

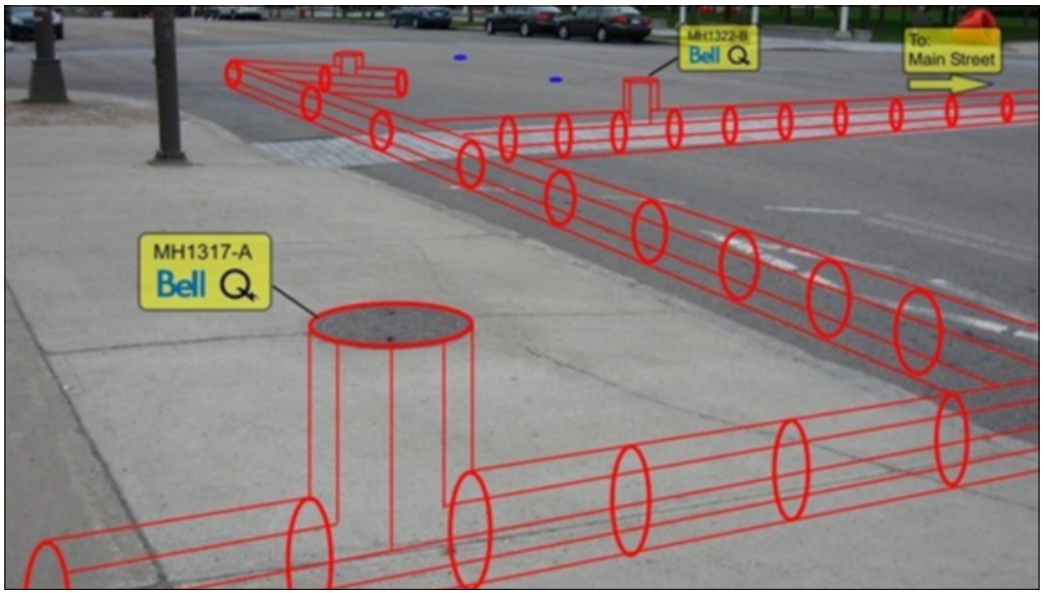


Tools

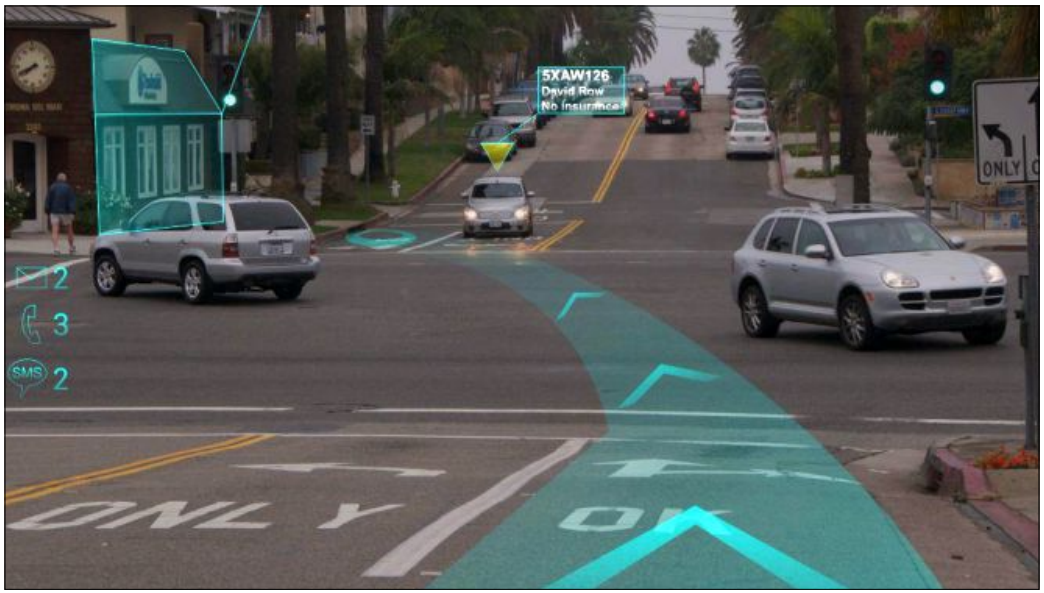
1:1 relationship with real-world scale means AR apps that can measure the environment



Instruction/teaching



Safety/Survey

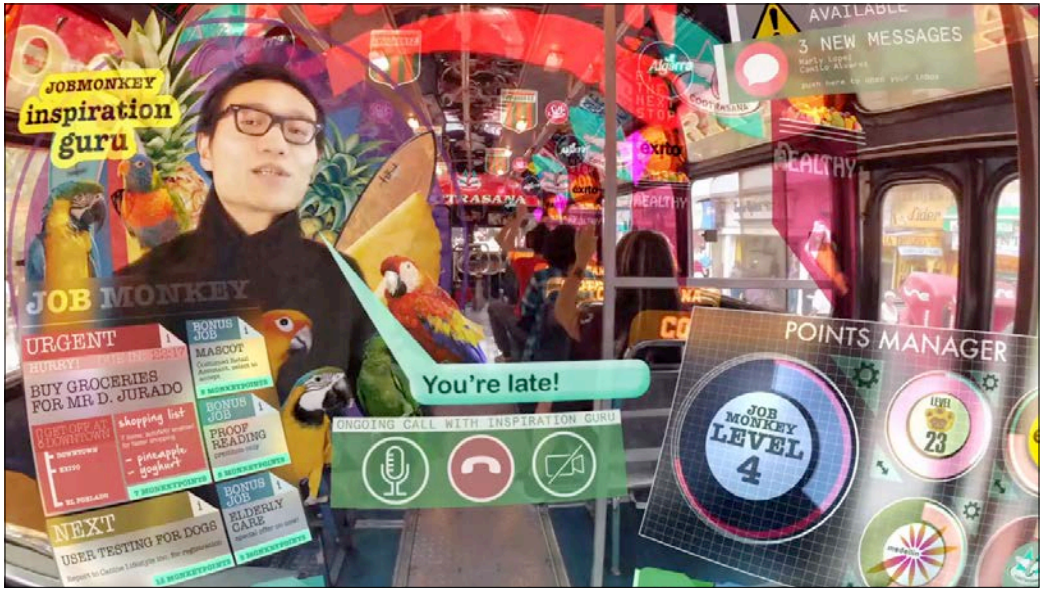


Directions

JPL uses Hololens to visualize 1:1 models of rovers, allowing designers to discover relationships and problems before anything is actually built.

You can use Google’s Translate app to translate this sign in realtime!

Looking at a slightly darker application. What happens when we go from this…



… to this? (i.e. when Times Square follows a user everywhere they go)

———

Keiichi Matsuda

http://km.cx/projects/hyper-reality
https://twitter.com/keiichiban

VR/AR can be seen as an evolution of **content** (what are we looking at) and an evolution of **platform** (how are we looking at it)

Progression from “usable by a small few” to “usable by everyone”

Vive is expensive. Smartphones are ubiquitous.

Disney says their future is in AR, not VR

Even the idea of a keyboard as input had to be invented - that you could interact with computers using an alpha-numeric alphabet

We can invent new types of interaction.

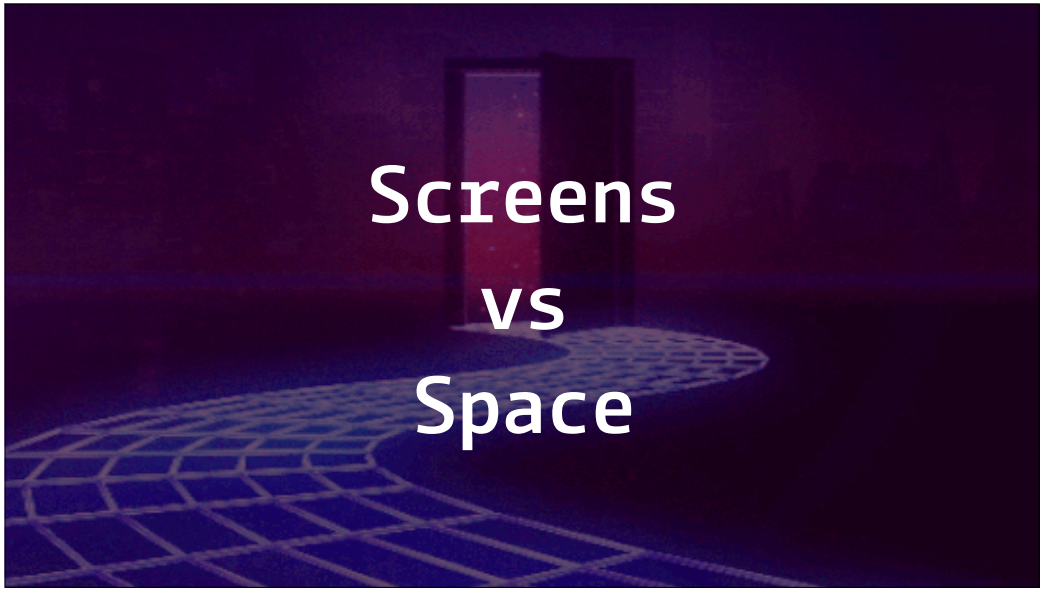
Maybe it’s moving in with your phone while you hold your finger down on screen? Maybe it’s drawing a specific shape on your phone?

Expectations

With new types of interface, we must teach users *how* to perform new kinds of interactions.

Baby trying to pinch/zoom on magazine after using iPad - 2011

We have to be *taught* the possibilities and limitations

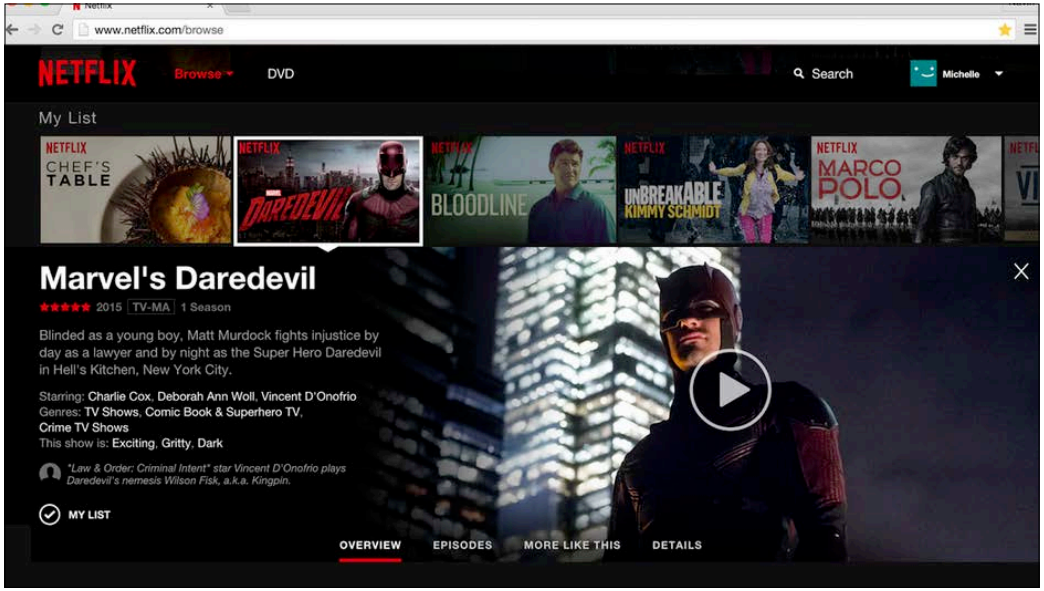


Revisiting some concepts…

Our space just got a lot bigger (and smaller)

All of a sudden we have to think about a spatial context for what we make

We are not limited by pixels or the dimensions of a rectangle, but by the available space.
Designing for Mobile AR, do we have to keep one foot in both?



VR - deals with it by creating new environments

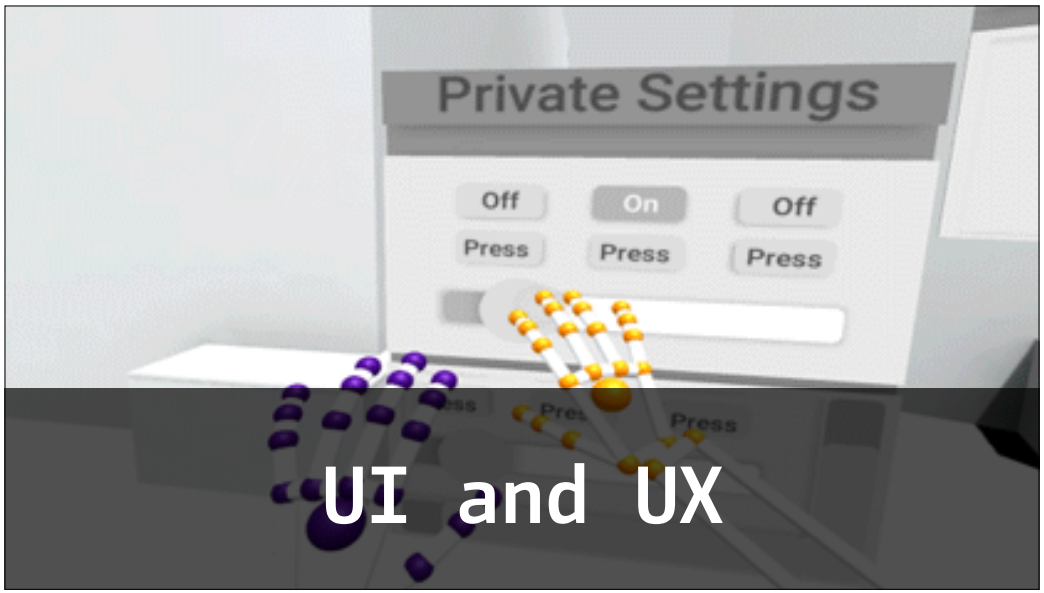
Netflix web interface…

..vs Netflix VR interface

Anticipating a future where we can comfortably sit in VR for a long time, Netflix creates the ideal context in which to watch movies (All of it’s existing content is 2D)

Examples:
Pokemon GO - AR does not need to be a headset. Pokemon GO is a like a “Lens” into another world.

Has both screen (See the UI buttons?) **and** Space



Revisiting some concepts.

What tools do we have open to us? What *don't* we have?

What is the interface?

Moving through an environment to discover content - no direct manipulation.

So what tech will we be using?

…Apple ARKit…

…Google AR Core…

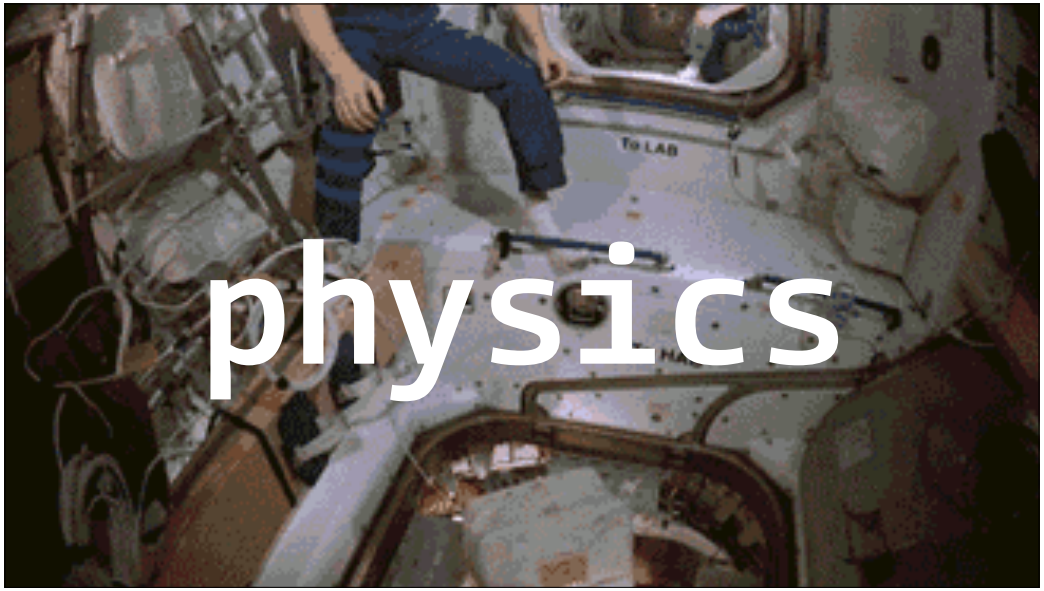
…Vuforia (Built into Unity!)

abstraction

First, let's talk about ***Abstraction:***

In software engineering and computer science, ***abstraction is a technique for arranging complexity of computer systems****. It works by establishing a level of complexity on which a person interacts with the system,* ***suppressing the more complex details below the current level****. The programmer works with an idealized interface (usually well defined) and can add additional levels of functionality that would otherwise be too complex to handle. For an example,* ***a programmer writing code that involves numerical operations may not be interested in the way numbers are represented in the underlying hardware*** *(e.g. whether they're 16 bit or 32 bit integers), and where those details have been suppressed it can be said that they were abstracted away, leaving simply numbers with which the programmer can work. In addition, a task of* ***sending an email message across continents would be extremely complex if the programmer had to start with a piece of fiber optic cable and basic hardware components****. By using layers of complexity that have been created to abstract away the physical cables and network layout, and presenting the programmer with a virtual data channel, this task is manageable.*

https://en.wikipedia.org/wiki/Abstraction_(software_engineering)



Abstracts the math/physics/graphics away for us and we get to work on a higher level.
Knows about: Mass, gravity, inertia, acceleration…

designing for the real world

It's not that we're making games, its that we're designing for the **real world** so it would be helpful not to have to reinvent the wheel every time.

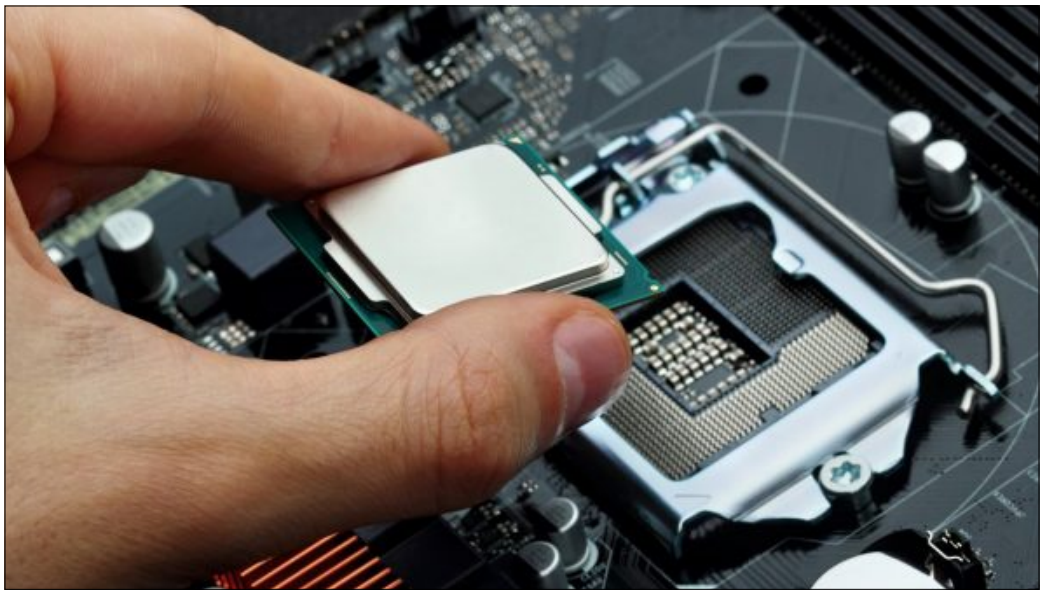
Pros/cons of a**bstraction**

- Find a balance between power and ease
- We'll talk a LOT about this over the next few weeks

abstraction

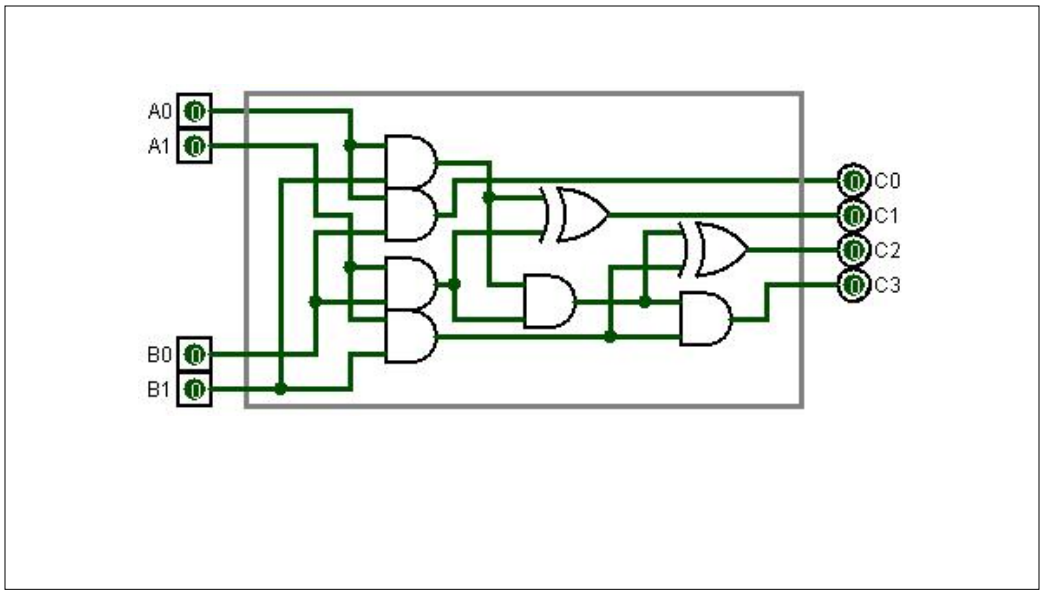
Things are about to get a bit more complex.

Allow me a bit of a tangent…



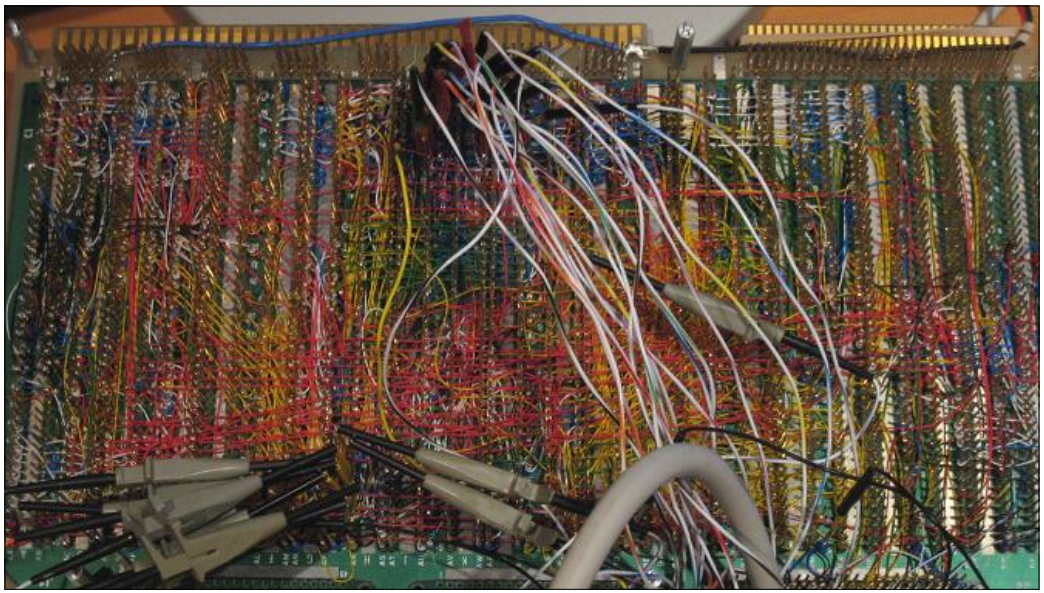
Let’s talk about how computers actually work.

What do we really mean by “Computers only understand 0’s and 1’s”?



Multiply two numbers using **hardware?!**

Bunch of AND gates and a couple of XOR (exclusive OR) gates



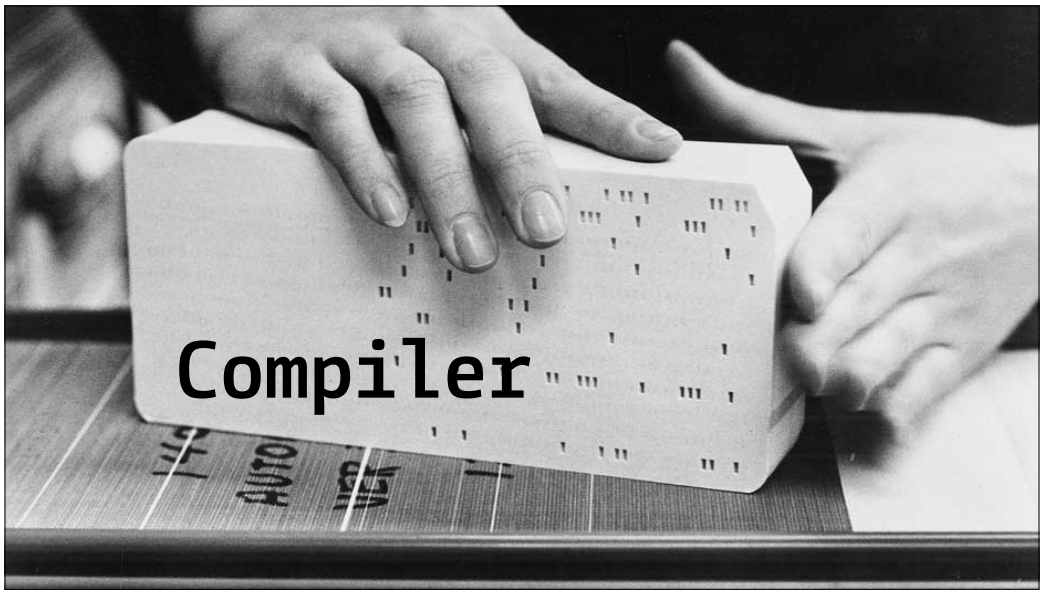
This is a home-built computer called the BMOW (Big Mess of Wires)

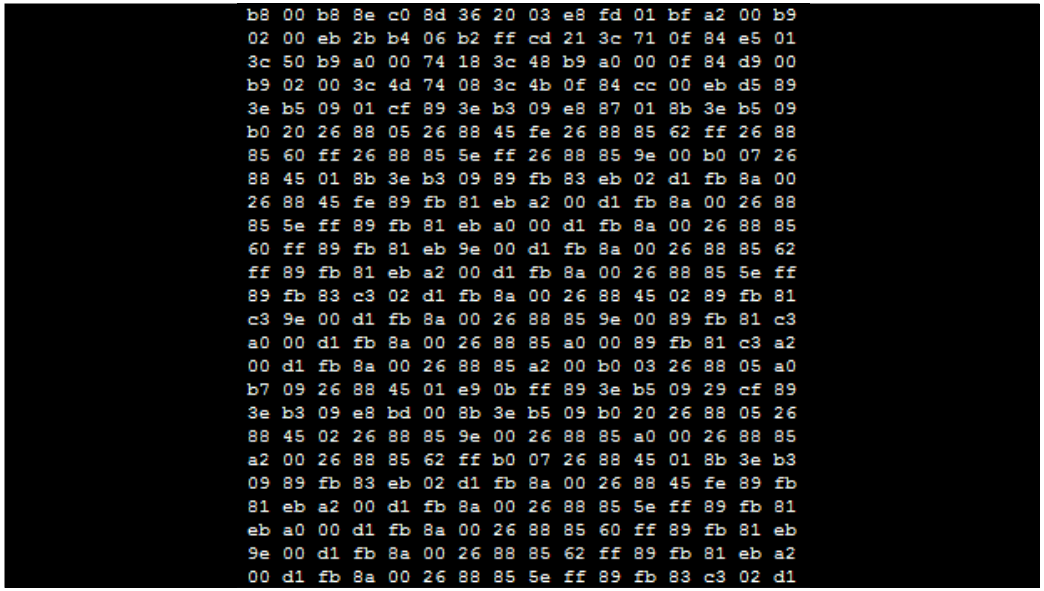
http://www.bigmessowires.com/bmow1/

Have you ever thought about how this chunk of metal and wires knows what to do?

```
// UdpPacket provides packetIO over UDP
public class UDPPacketIO {

  private UdpClient Sender;
  private UdpClient Receiver;
  private bool socketsOpen;
  private string remoteHostName;
  private int remotePort;
  private int localPort;

  private string multicastAddress;
  private bool enableMulticast;



  public UDPPacketIO(string hostIP, int remotePort, int localPort, bool enableMulticast=false, string multicastAddress=null) {
    RemoteHostName = hostIP;
    RemotePort = remotePort;
    LocalPort = localPort;

    EnableMulticast = enableMulticast;
    MulticastAddress = multicastAddress;
    socketsOpen = false;
  }

  ~UDPPacketIO() {
    // latest time for this socket to be closed
    if (IsOpen()) {
      BobRemoteControl.BobDebug.Log("closing udpclient listener on port " + localPort);
      Close();
```

A programming language is just a text file.
Written according to a standard.
It is meant to be HUMAN READABLE. Computer doesn’t care if you use C# or C++ or Java or BrainFuck.

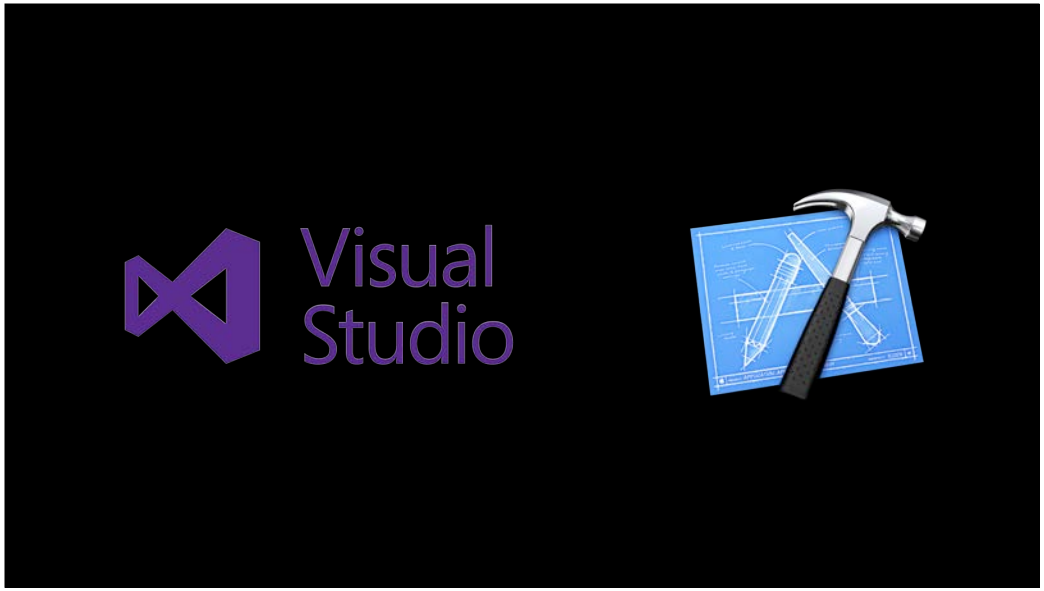

Compiler



Machine-code does not "communicate with the processor".
Rather, the processor "knows how to evaluate" machine-code.

The CPU "looks" at the current instruction. When the instructions are executed side-effects occur such as setting a control flag, putting a value in a register, or jumping to a different index (changing the PC) in the program, etc.

It is the evaluation of each instruction -- as it is encountered -- and the interaction of side-effects that results in the operation of a traditional processor.



Visual Studio and XCode

We can’t just finish in Unity anymore! Because these AR devices are running on different processors than our PC, we have to go through another program that will build the final executable for the device.

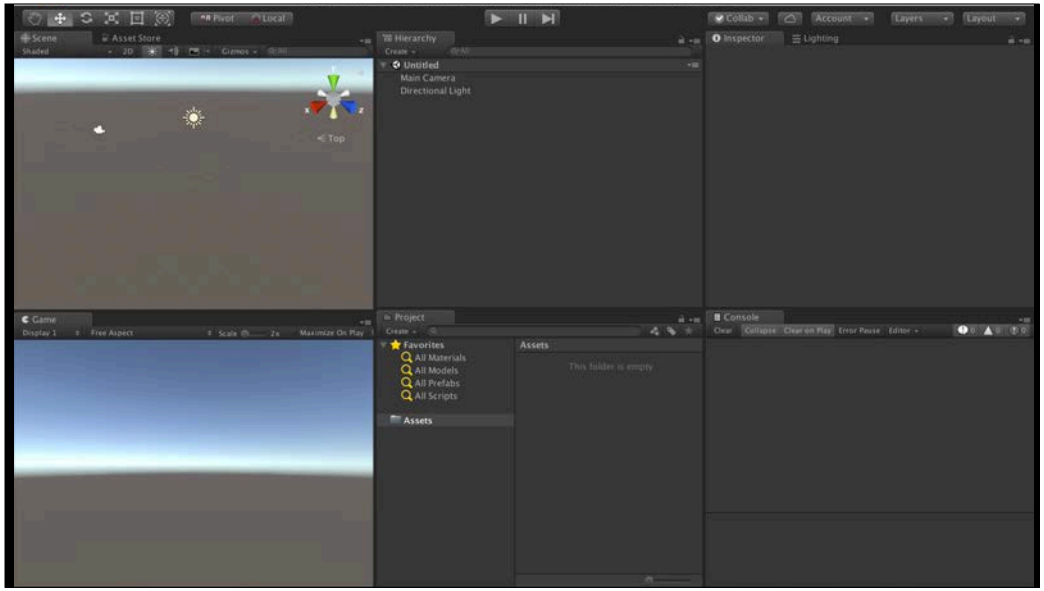
Some extra nerdiness:

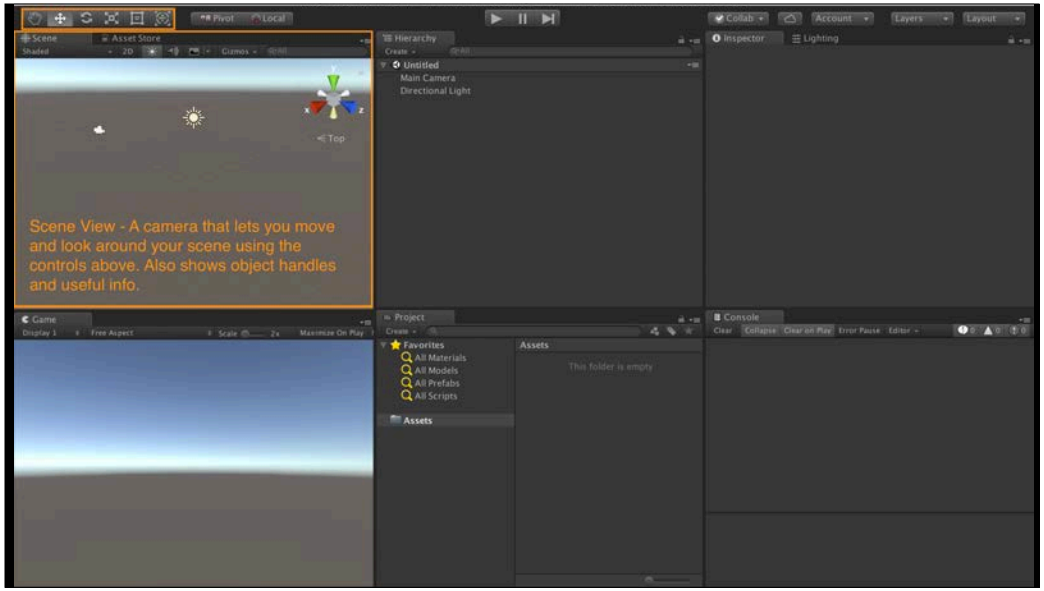
PC/Mac Standalone (and VR):
C# -> IL -> Bytecode -> JIT Compiler -> Machine Code
———

Using IL2CPP bridge:
C# -> IL -> IL2CPP -> C++ -> Anywhere!

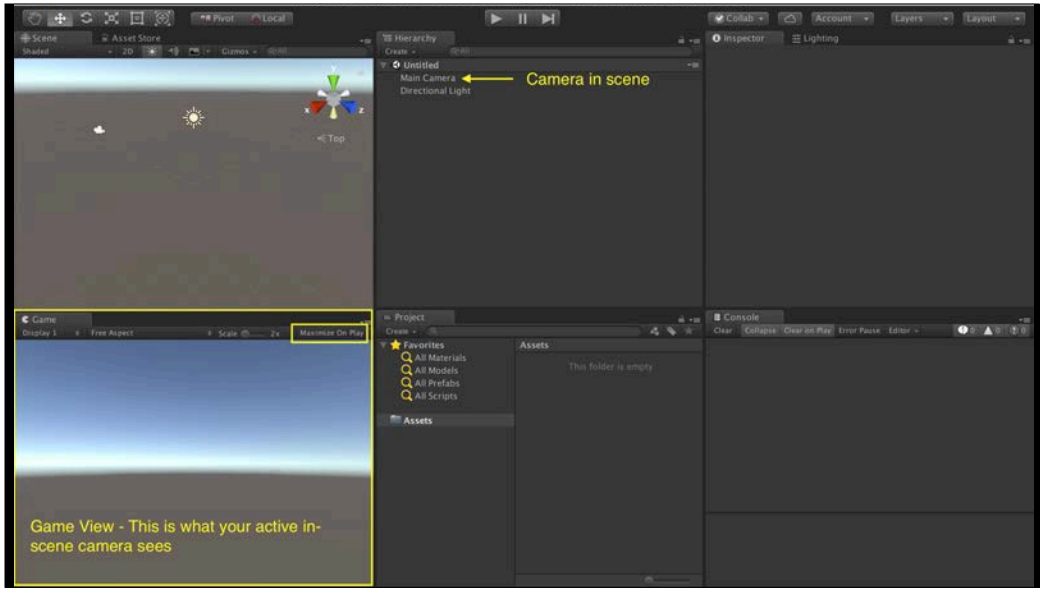
C++ -> XCode -> iOS/Apple A9 processor Machine Code

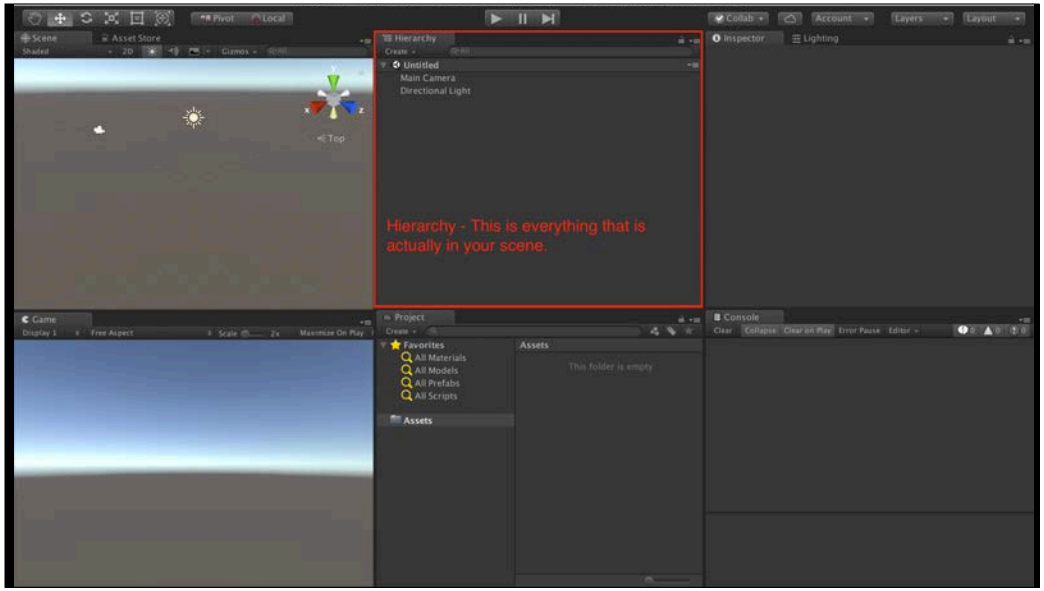
Let’s get in to Unity!

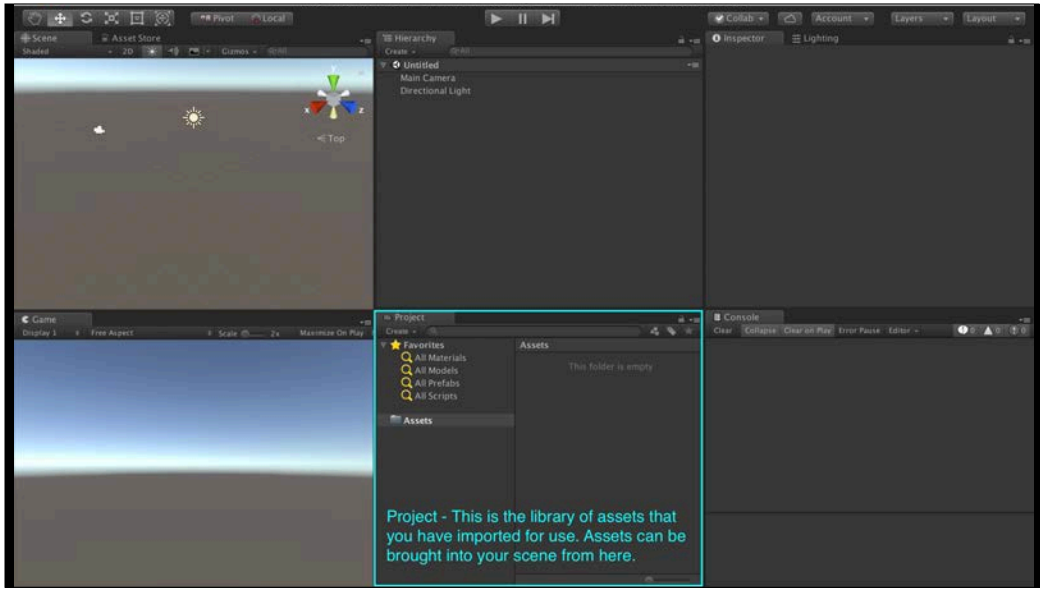

Pivot
Local
Collab
Account
Layers
Layout
Scene
Asset Store
Shaded
2D
Gizmos
Top
Hierarchy
Create
Untitled
Main Camera
Directional Light
Inspector
Lighting
Game
Display 1
Free Aspect
Scale
2x
Maximize On Play
Project
Create
Favorites
All Materials
All Models
All Prefabs
All Scripts
Assets
Assets
This folder is empty
Console
Clear
Collapse
Clear on Play
Error Pause
Editor

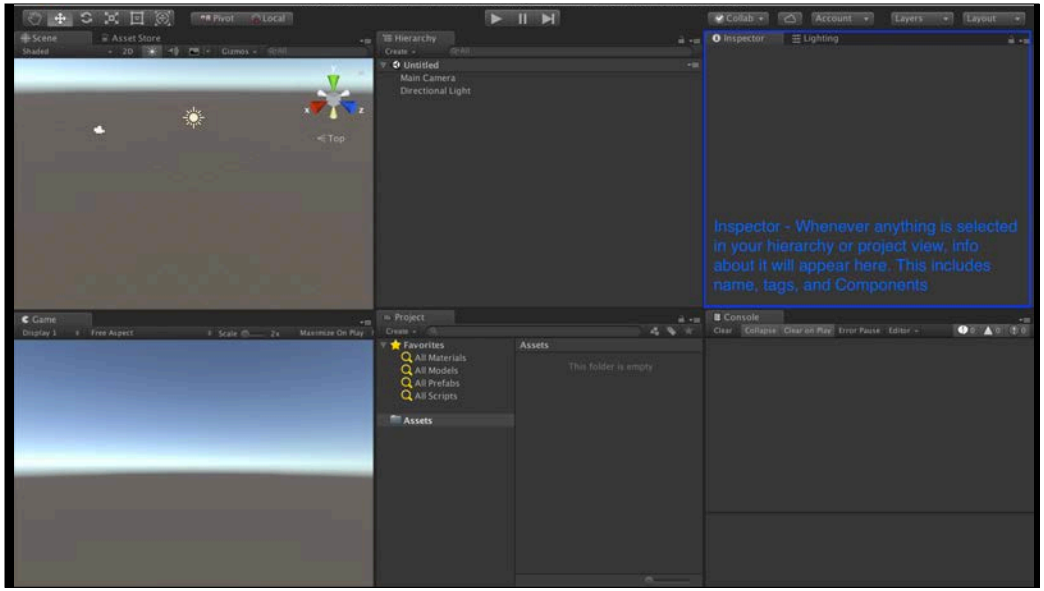


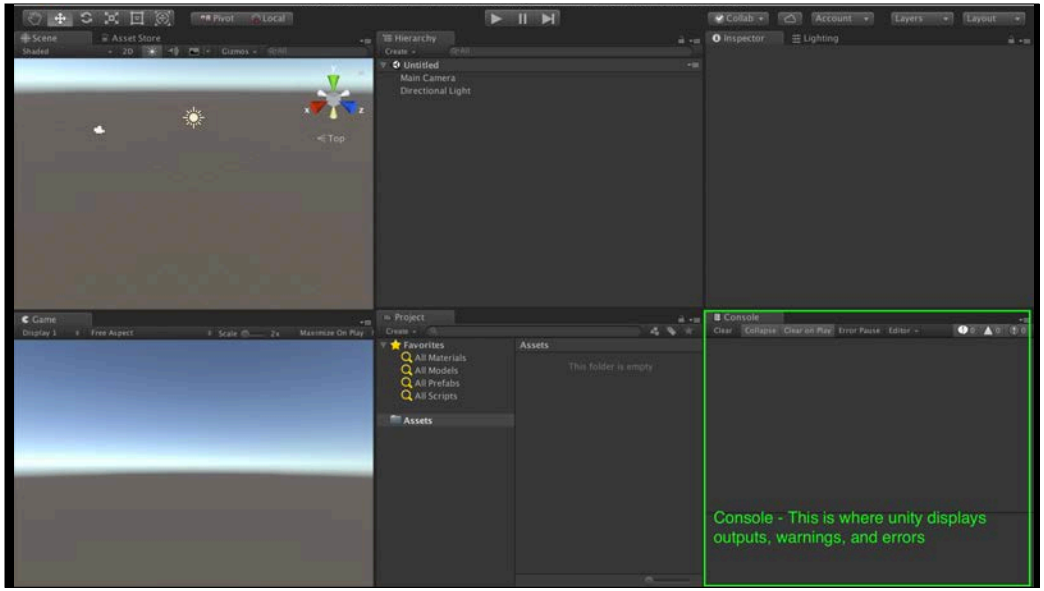
When holding down **Right-Click** inside the scene view, moving mouse along with **W-A-S-D-Q-E** keys will move our point of view.

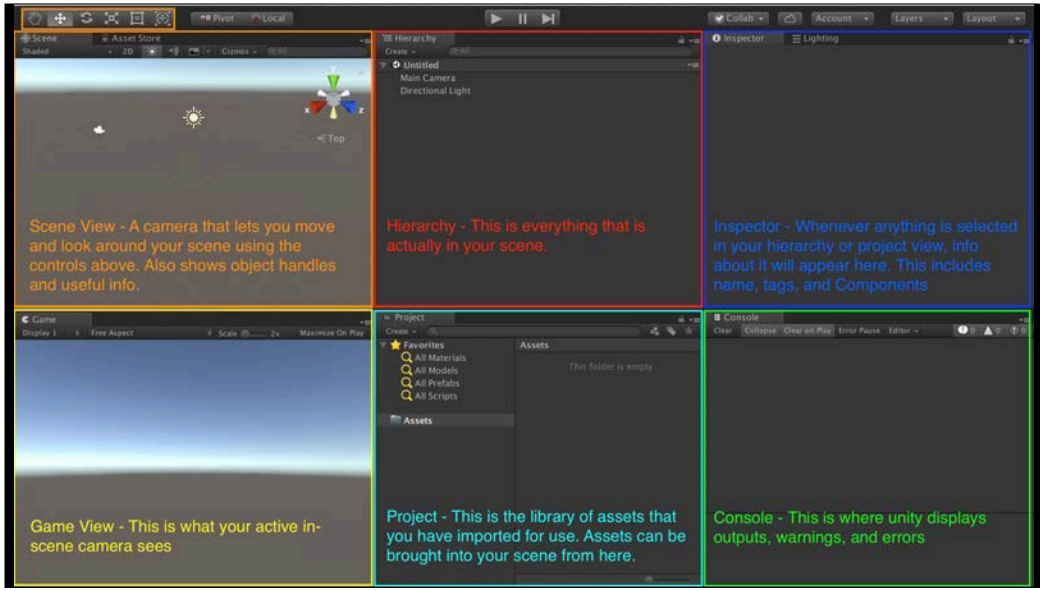

Camera in scene
Game View - This is what your active in-scene camera sees

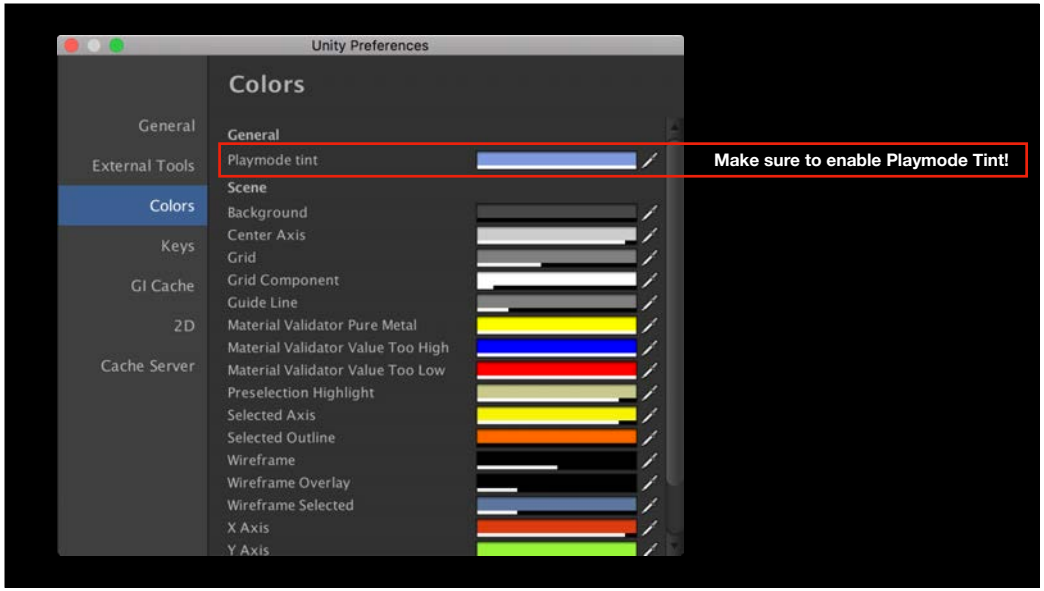

Hierarchy - This is everything that is actually in your scene.


Pivot
Local
Collab
Account
Layers
Layout
Scene
Asset Store
Shaded
2D
Gizmos
Top
Hierarchy
Create
Untitled
Main Camera
Directional Light
Inspector
Lighting
Game
Display 1
Free Aspect
Scale
2x
Maximize On Play
Project
Create
Favorites
All Materials
All Models
All Prefabs
All Scripts
Assets
Assets
This folder is empty
Console
Clear
Collapse
Clear on Play
Error Pause
Editor
Project - This is the library of assets that you have imported for use. Assets can be brought into your scene from here.


Pivot
Local
Collab
Account
Layers
Layout
Scene
Asset Store
Shaded
2D
Gizmos
Top
Hierarchy
Create
Untitled
Main Camera
Directional Light
Inspector
Lighting
Inspector - Whenever anything is selected in your hierarchy or project view, info about it will appear here. This includes name, tags, and Components
Game
Display 1
Free Aspect
Scale
2x
Maximize On Play
Project
Create
Favorites
All Materials
All Models
All Prefabs
All Scripts
Assets
Assets
This folder is empty
Console
Clear
Collapse
Clear on Play
Error Pause
Editor

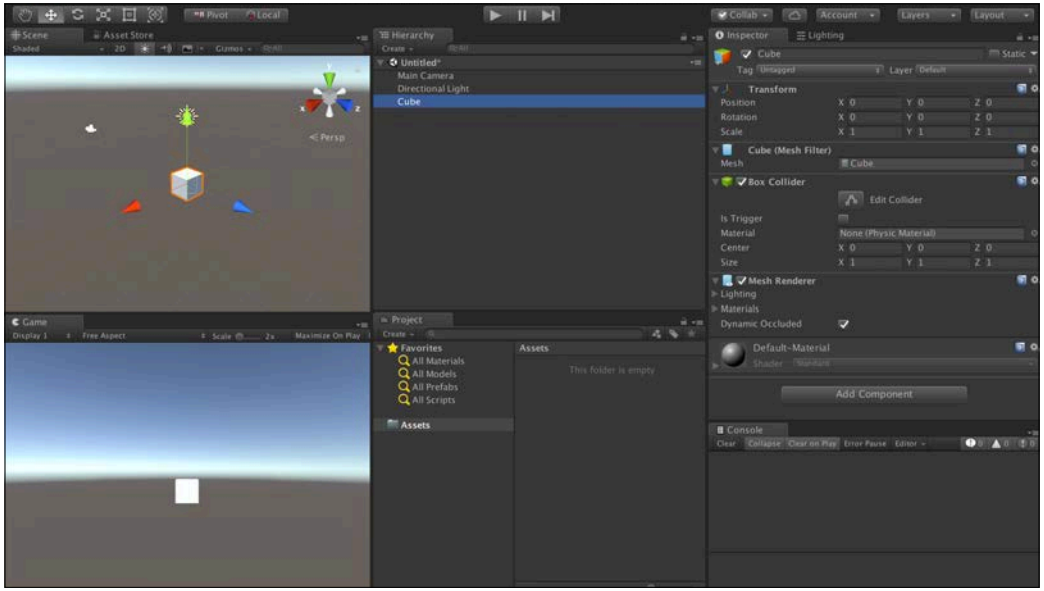

Pivot
Local
Collab
Account
Layers
Layout
Scene
Asset Store
Shaded
2D
Gizmos
Top
Hierarchy
Create
Untitled
Main Camera
Directional Light
Inspector
Lighting
Game
Display 1
Free Aspect
Scale
2x
Maximize On Play
Project
Create
Favorites
All Materials
All Models
All Prefabs
All Scripts
Assets
Assets
This folder is empty
Console
Clear
Collapse
Clear on Play
Error Pause
Editor
Console - This is where unity displays outputs, warnings, and errors


Scene View - A camera that lets you move and look around your scene using the controls above. Also shows object handles and useful info.
Hierarchy - This is everything that is actually in your scene.
Inspector - Whenever anything is selected in your hierarchy or project view, info about it will appear here. This includes name, tags, and Components
Game View - This is what your active in-scene camera sees
Project - This is the library of assets that you have imported for use. Assets can be brought into your scene from here.
Console - This is where unity displays outputs, warnings, and errors

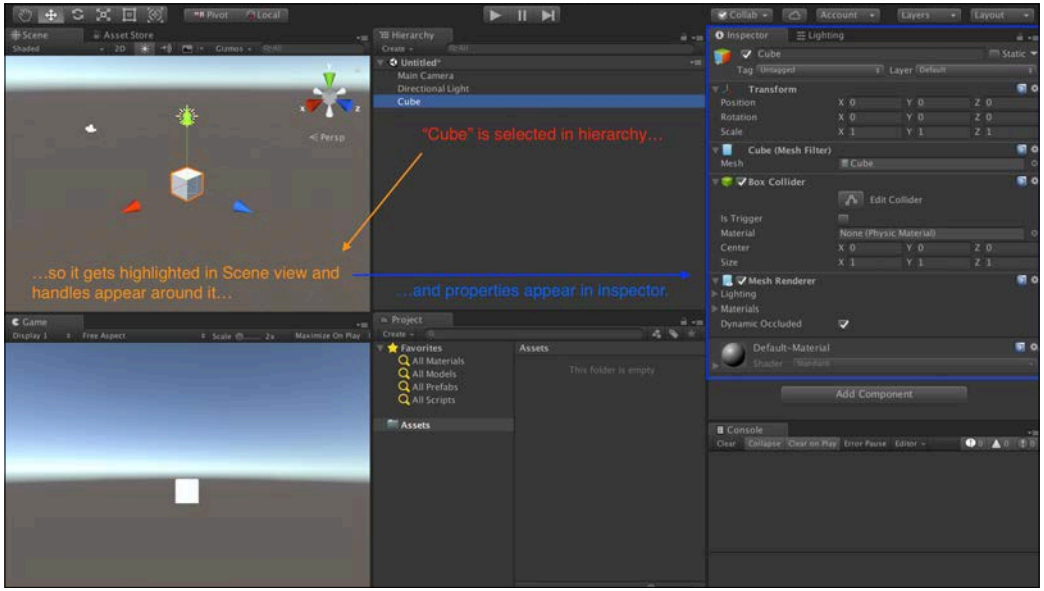


This will save you so much heartache.

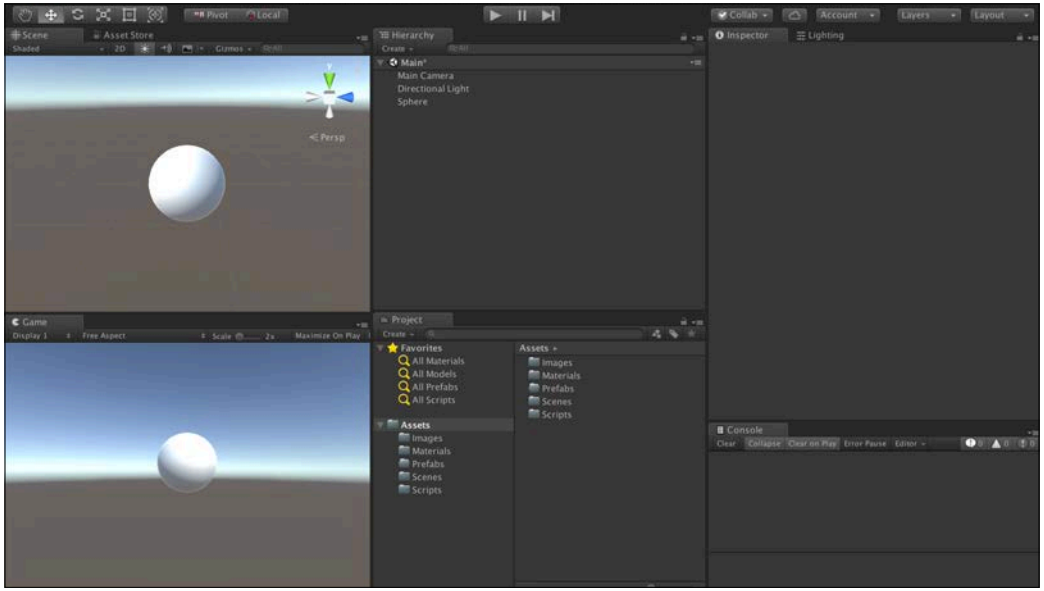
(So so much)



Inspector and **Scene View** change based on what's selected in **Hierarchy**

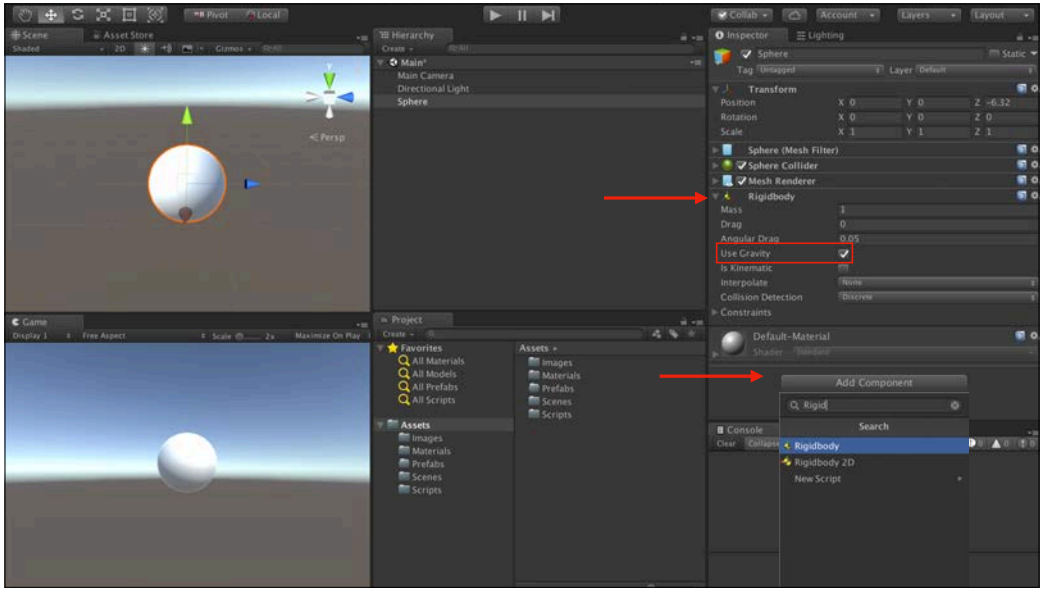


Inspector and **Scene View** change based on what's selected in **Hierarchy**



Working with GameObjects, Components, and Prefabs.

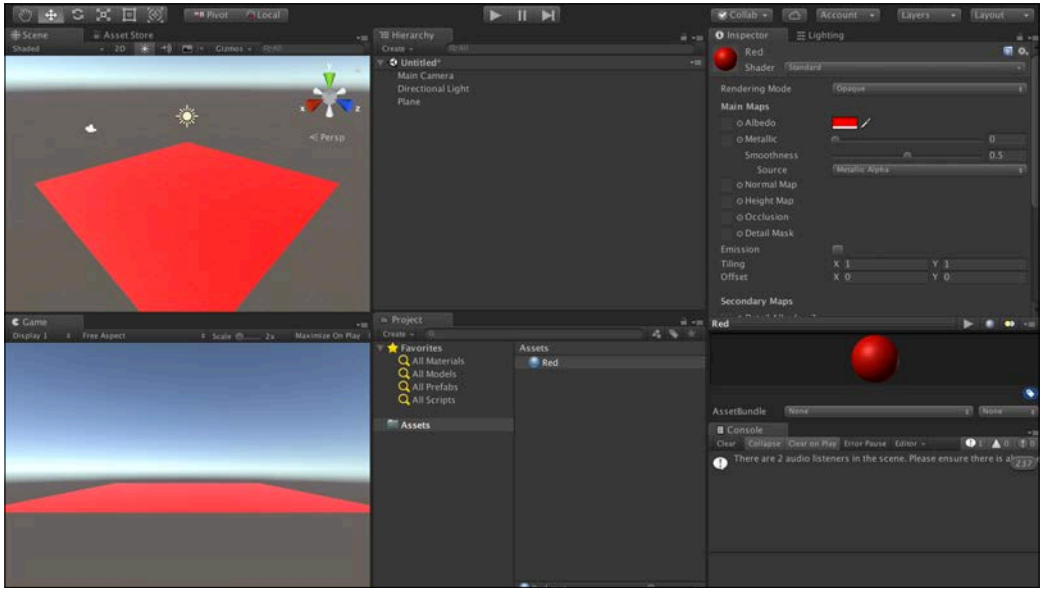
Create a new **Sphere** inside your scene (i.e. create it in the **Hierarchy**)



Back to our **sphere**.

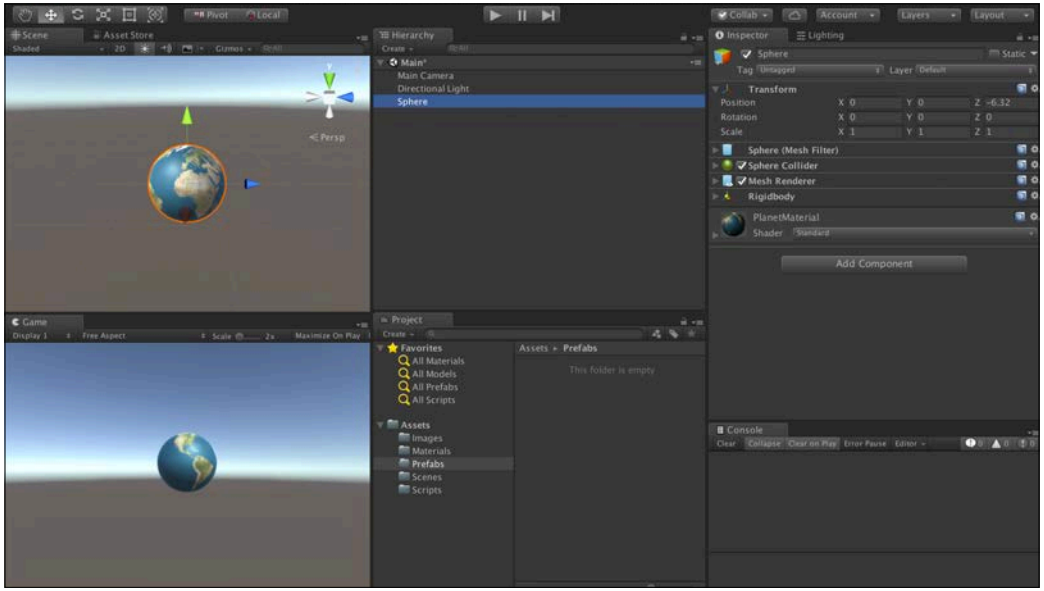
Make sure it is selected and use the “Add Component” button in the inspector to add a **Rigid Body**.

Now you have physics! Yay, thanks Unity!

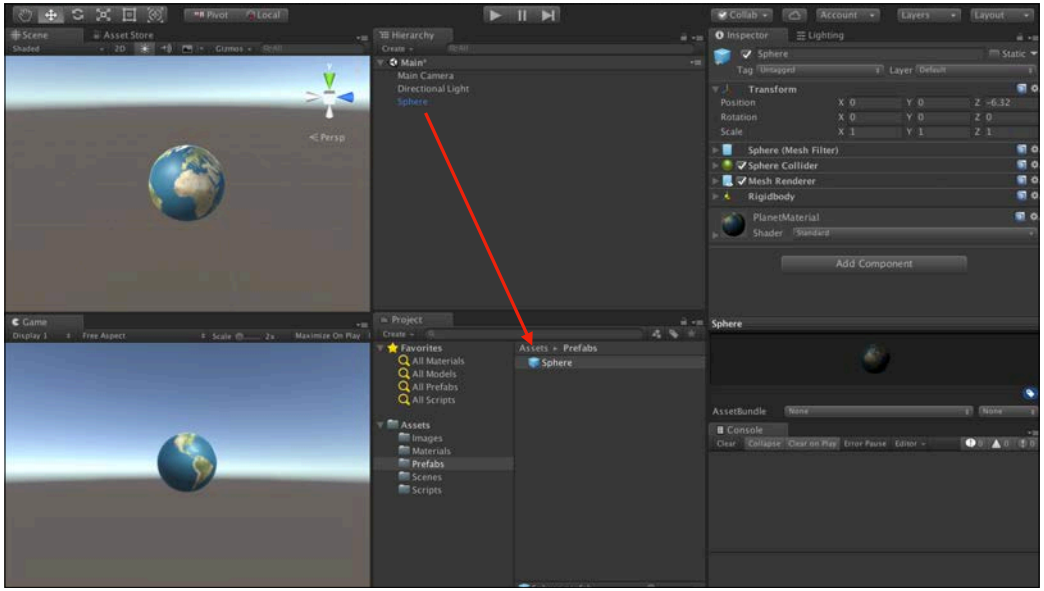


Materials are a bit weird - they're kind of like a *Component* of a Component

Lots of ways to add them but easiest is by dragging. (This actually works for a lot of components, but materials are clearer: easy to see what you're adding to, everything else I prefer adding via the inspector, like we did with RigidBody)

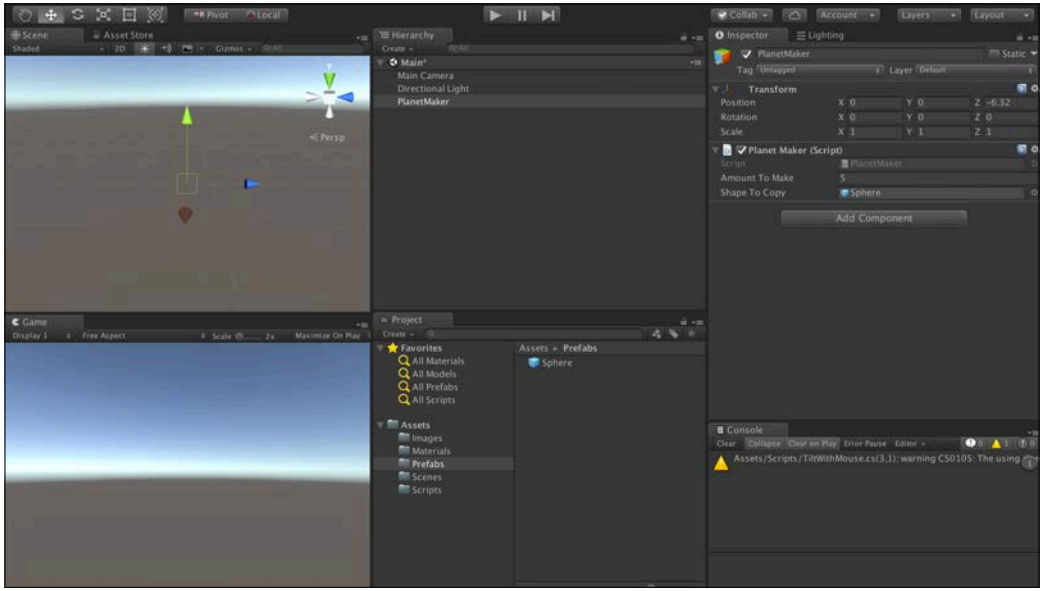


Prefabs! So, we've started with a simple shape, but now we've added stuff to it. Cool textures, useful components.



We can drag things from the hierarchy **back** into the assets in the project view!

This creates a reusable object called a **Prefab.**



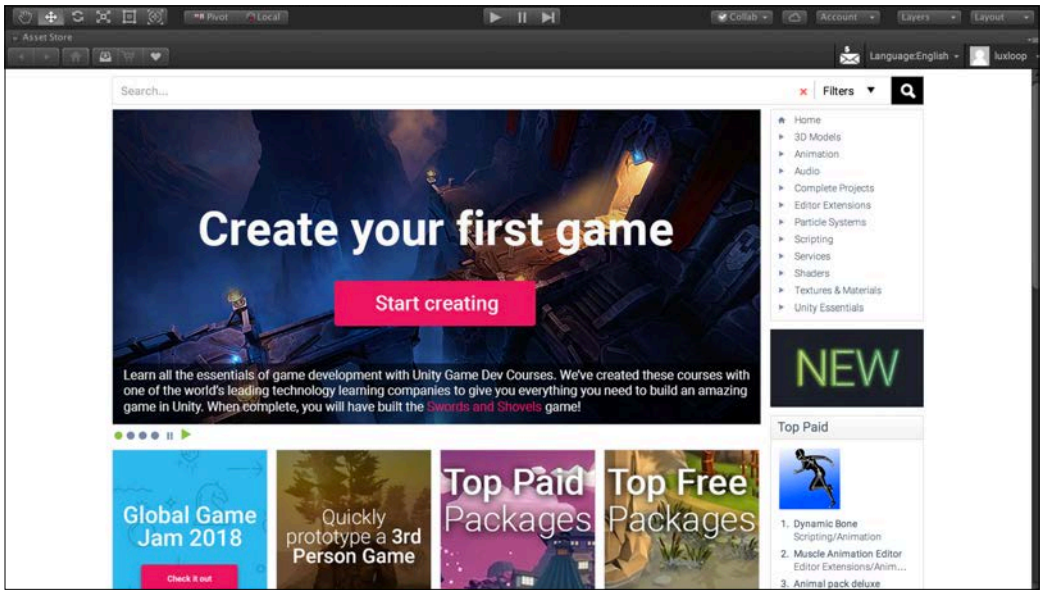
Adding **Scripts** to GameObjects is the same as anything else (We'll cover making our *own* scripts next time)

Scripts have to live somewhere in your Hierarchy (just because they are in the assets does not mean they are active yet).

Create a new **Empty** game object (Just like you created a Sphere) and add the *PlanetMaker* script to it (Just like you added a RigidBody)

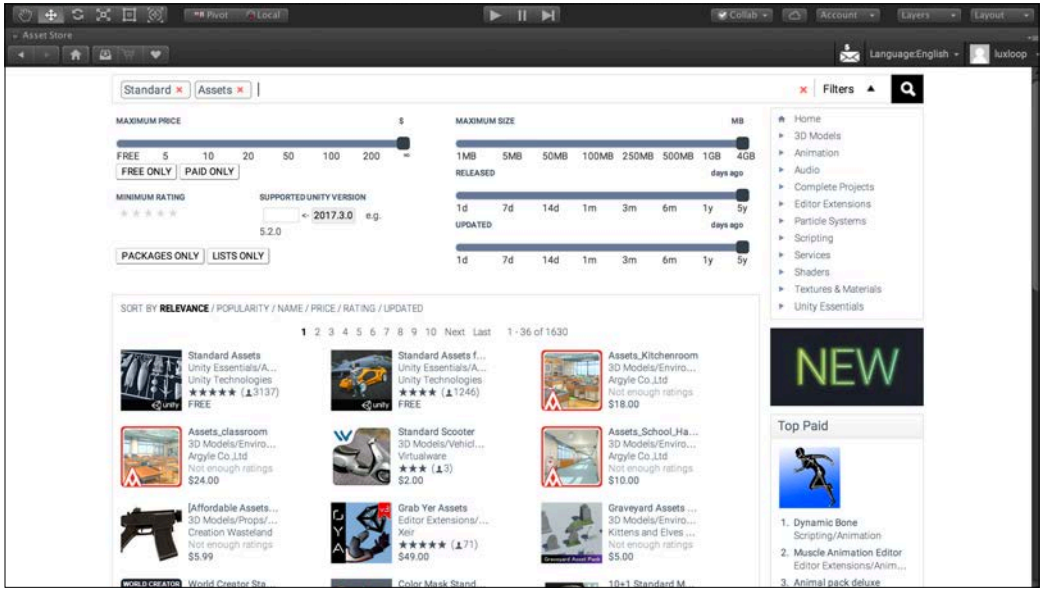
Notice that this particular script has an empty slot in it called "Shape to Copy" and in parenthesis it's saying that this slot takes a **GameObject**. Drag the **Prefab** we made earlier into this slot.

Also notice the "Amount to Make" slot - this is a **public script variable** (more on that later).

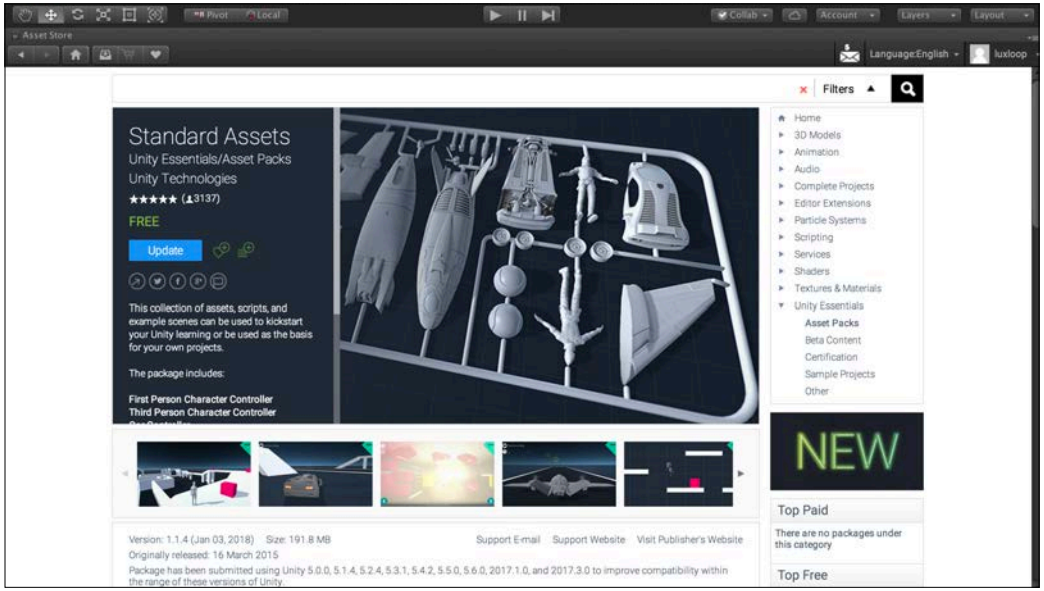


Asset Store

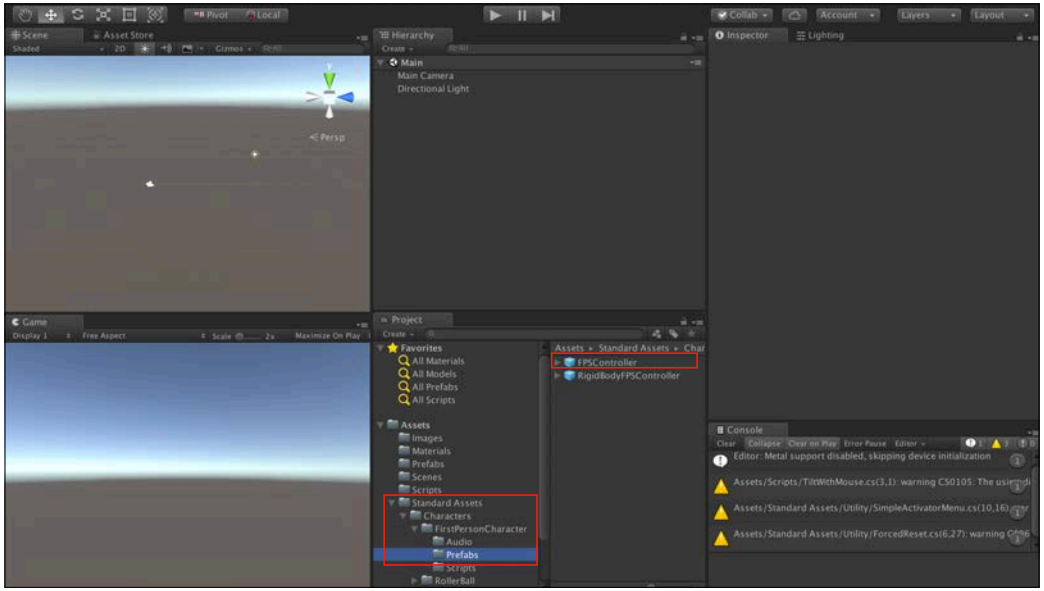
Window > Asset Store



Search for **Standard Assets**



Search for **Standard Assets**



Find the FPS Controller under
Standard Assets > *Characters* > *FirstPersonCharacter* > *Prefabs*

This is the first thing we will use to give ourselves agency inside the scene. Very simple:

- There must be some kind of “floor” for the character to walk on
- The floor must have some sort of **collider** as a component (most Unity primitives already come with collider components

When in play mode, FPSController will move with WASD keys

Back into Unity

- Is everyone on Visual Studio 2017?

Vectors

In programming terms, you can think of Vectors as a way to store 2, 3, or 4 values in one easy-to-use package:

```
Vector2 someNumbers = new Vector2(1.0, 2.2);
Vector3 someOtherNumbers = new Vector3(5.3, 2.6, 12.0);
Vector4 evenMoreNumbers = new Vector4(7.4, 2.1, 12.0, 9.8);
```

Vectors

We can use vectors to:

- Store multiple numbers in one variable
- Describe the position of something in our world
  - For example: (2.1, 8.9, 7.4) represents the point in space 2.1 units along the X-axis, 8.9 units along the Y-axis, and 7.4 units along the Z-axis.

Vectors

We can use vectors to:

- Describe a direction
  - For example: (0.0, 1.0, 0.0) represents a point 1 unit directly above (along Y) the origin.
  - If we a drew an arrow from the origin to this point, it would point straight up.
  - It doesn't matter how long the Vector is:
    - (0.0, 1.0, 0.0) and (0.0, 5.2, 0.0) are different points, but they both describe the same *direction* (straight up).

Vectors

Unity has some built-in direction shorthands:

```
Vector3 example = Vector3.up;
```

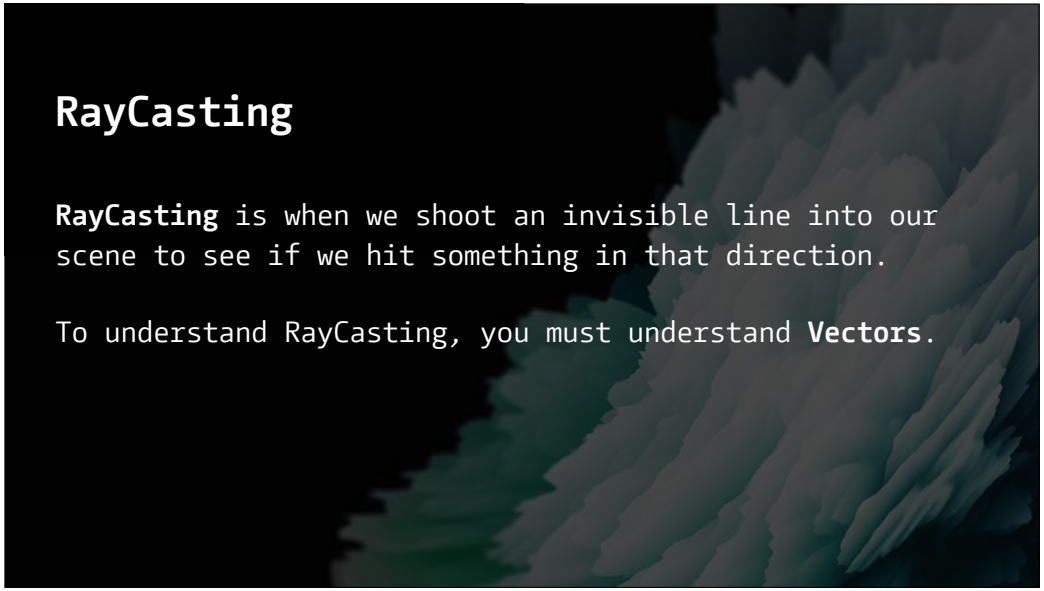
is the same as:

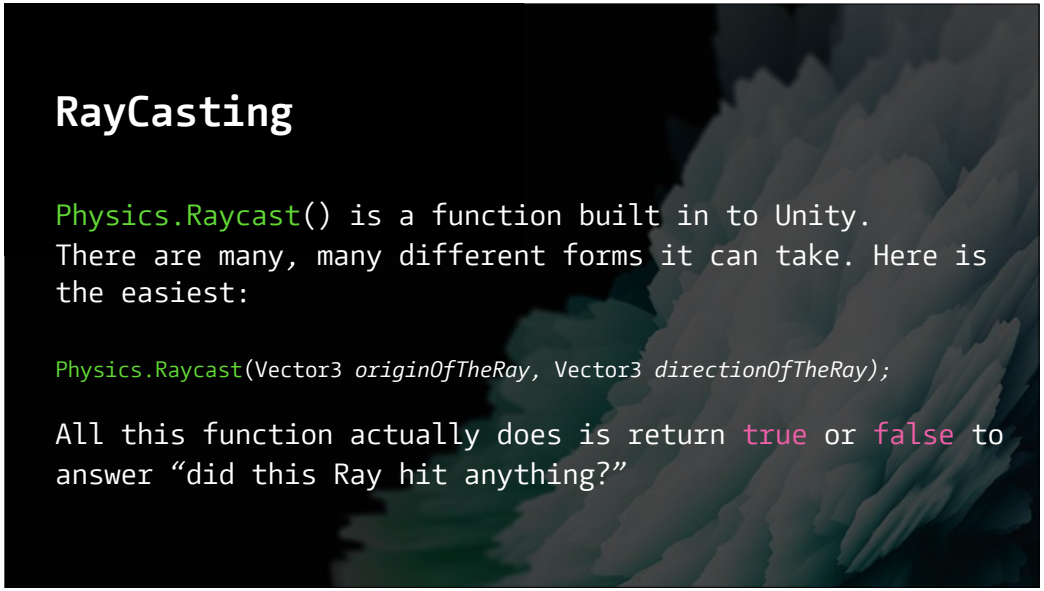
```
Vector3 example = new Vector3(0.0, 1.0, 0.0);
```

Vectors

Other shorthands:

- `Vector3.up` (pointing along Y-axis)
- `Vector3.forward` (pointing along Z-axis)
- `Vector3.right` (pointing along X-axis)
- `Vector3.one` (Equal to (1.0, 1.0, 1.0)


RayCasting
RayCasting is when we shoot an invisible line into our scene to see if we hit something in that direction.
To understand RayCasting, you must understand Vectors.



See the example script on the github for more details:
https://github.com/ivaylopg/Tech421Tech3706/blob/master/Session17/ScriptExamples/RayCaster.cs

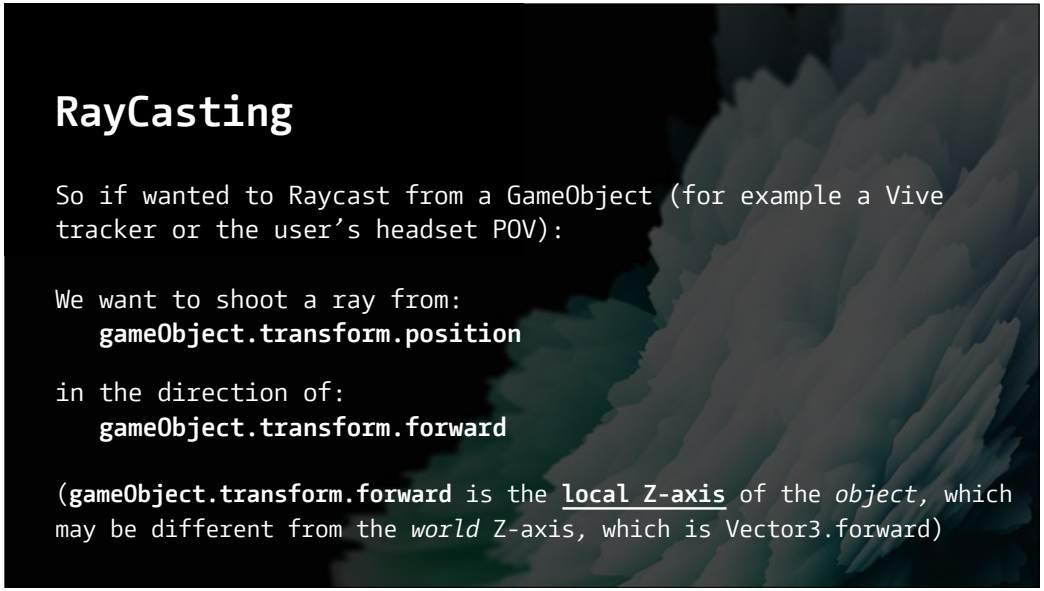
RayCasting

To store information about *what* was hit, and more importantly *where* the hit is in space, we have to do two things:

1. Declare a variable of the type `RaycastHit` to store the information about the hit point.
2. Use a slightly different version of `Physics.Raycast()` to pass the hit info out of it:

```
RaycastHit hitInfoVariable
Physics.Raycast(Vector3 originOfTheRay, Vector3 directionOfTheRay, out hitInfoVariable)
```

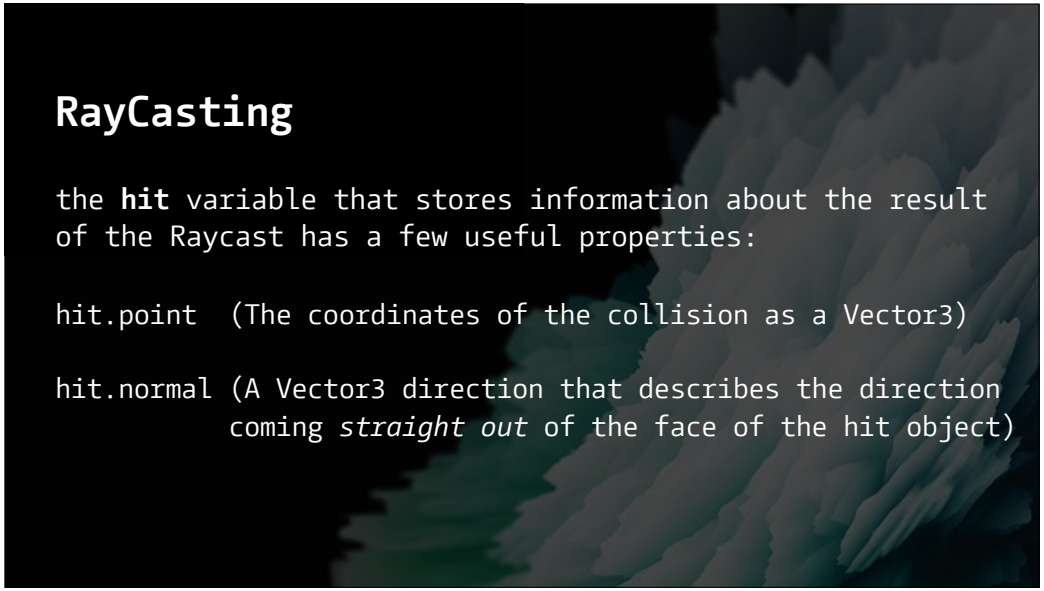
See the example script on the github for more details:
https://github.com/ivaylopg/Tech421Tech3706/blob/master/Session17/ScriptExamples/RayCaster.cs



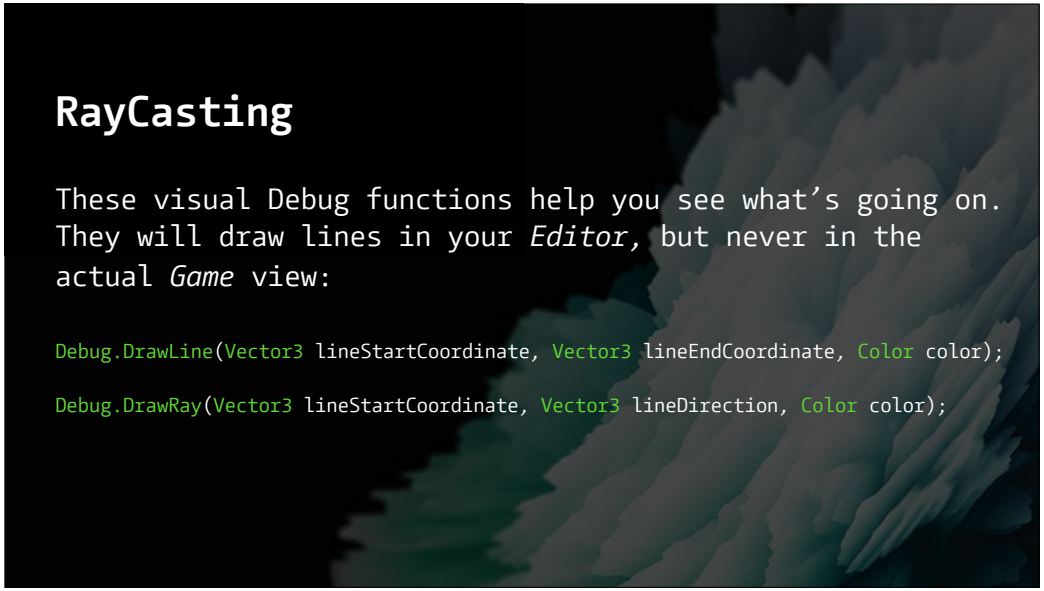
See the example script on the github for more details:
https://github.com/ivaylopg/Tech421Tech3706/blob/master/Session17/ScriptExamples/RayCaster.cs

RayCasting

```
void Update() {
    RaycastHit hit;
    if ( Physics.Raycast(gameObject.transform.position, gameObject.transform.forward, out hit) ) {

        Debug.DrawLine(gameObject.transform.position, hit.point, Color.red);
        Debug.DrawRay(hit.point, hit.normal, Color.green);

    }
}
```



See the example script on the github for more details:
https://github.com/ivaylopg/Tech421Tech3706/blob/master/Session17/ScriptExamples/RayCaster.cs



See the example script on the github for more details:
https://github.com/ivaylopg/Tech421Tech3706/blob/master/Session17/ScriptExamples/RayCaster.cs

ARx Designers: The Future Kings and Queens of Silicon Valley

https://goo.gl/dfWGeA

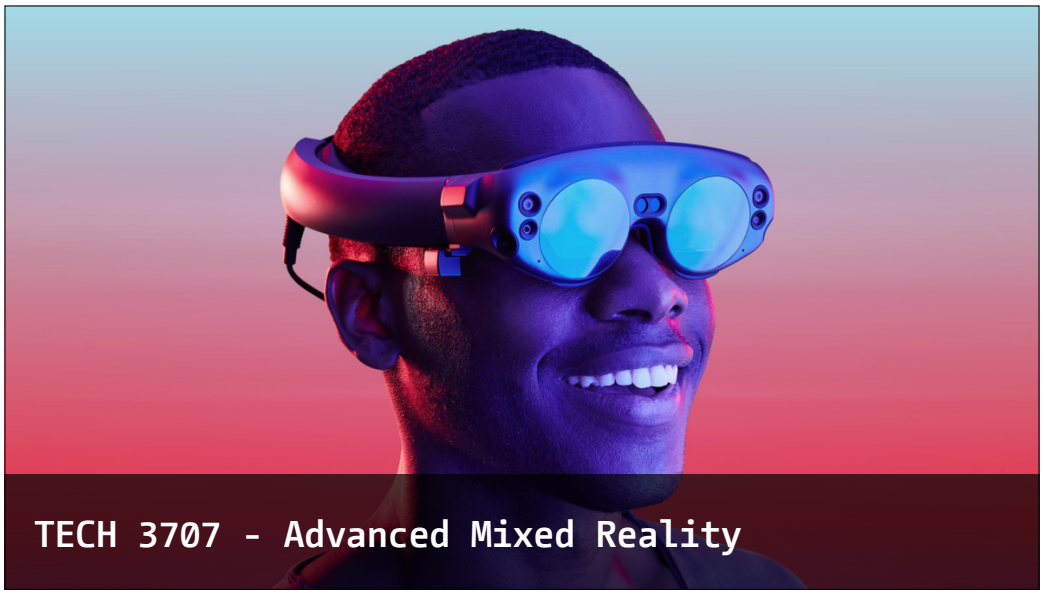
Design For Humanity - Parts 4,5

https://goo.gl/Jv5EFZ

Reading assignment:

https://medium.com/the-mission/arx-designers-the-future-kings-and-queens-of-silicon-valley-9c2a7a208fdb

https://augmented.reality.news/news/bob-iger-says-ar-not-vr-is-way-future-for-disney-parks-0176733/


TECH 3707 - Advanced Mixed Reality