

TECH 1711 - Mixed Reality Studio



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 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**.

RayCasting

`Physics.Raycast()` is a function built in to Unity. There are many, many different forms it can take. Here is the easiest:

```
Physics.Raycast(Vector3 originOfTheRay, Vector3 directionOfTheRay);
```

All this function actually does is return `true` or `false` to answer “did this Ray hit anything?”

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>

RayCasting

So if wanted to Raycast from a GameObject (for example a Vive tracker or the user's headset POV):

We want to shoot a ray from:

`gameObject.transform.position`

in the direction of:

`gameObject.transform.forward`

(`gameObject.transform.forward` is the local Z-axis of the *object*, which may be different from the *world* Z-axis, which is `Vector3.forward`)

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);  
  
    }  
}
```

RayCasting

the `hit` variable that stores information about the result of the Raycast has a few useful properties:

`hit.point` (The coordinates of the collision as a `Vector3`)

`hit.normal` (A `Vector3` direction that describes the direction coming *straight out* of the face of the hit object)

See the example script on the github for more details:

<https://github.com/ivaylopg/Tech421Tech3706/blob/master/Session17/ScriptExamples/RayCaster.cs>

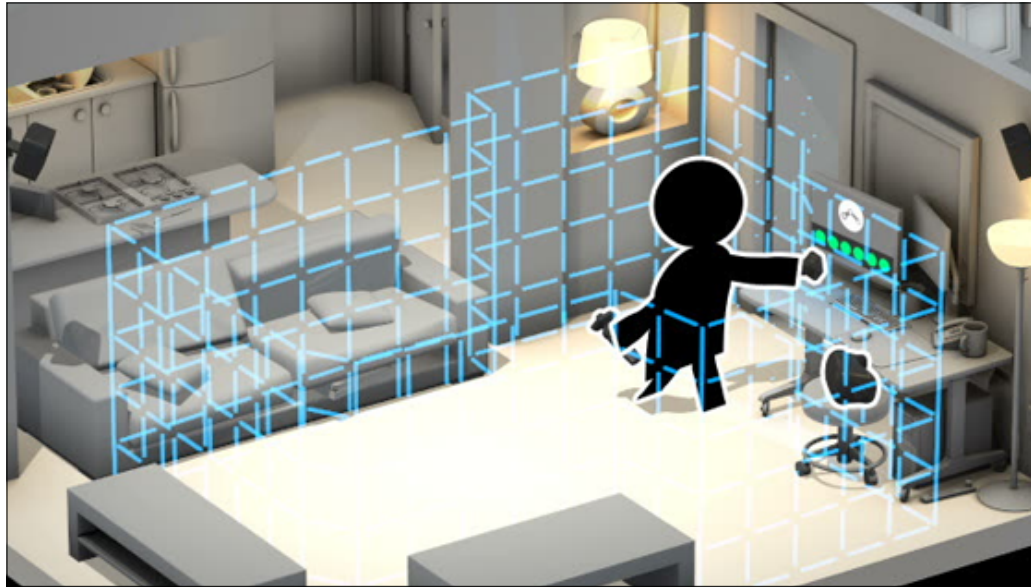
RayCasting

These visual Debug functions help you see what's going on. They will draw lines in your *Editor*, but never in the actual *Game* view:

```
Debug.DrawLine(Vector3 lineStartCoordinate, Vector3 lineEndCoordinate, Color color);  
Debug.DrawRay(Vector3 lineStartCoordinate, Vector3 lineDirection, Color color);
```

See the example script on the github for more details:

<https://github.com/ivaylopg/Tech421Tech3706/blob/master/Session17/ScriptExamples/RayCaster.cs>



Vive uses “Chaperone System” that shows boundaries of available area if you get too close.

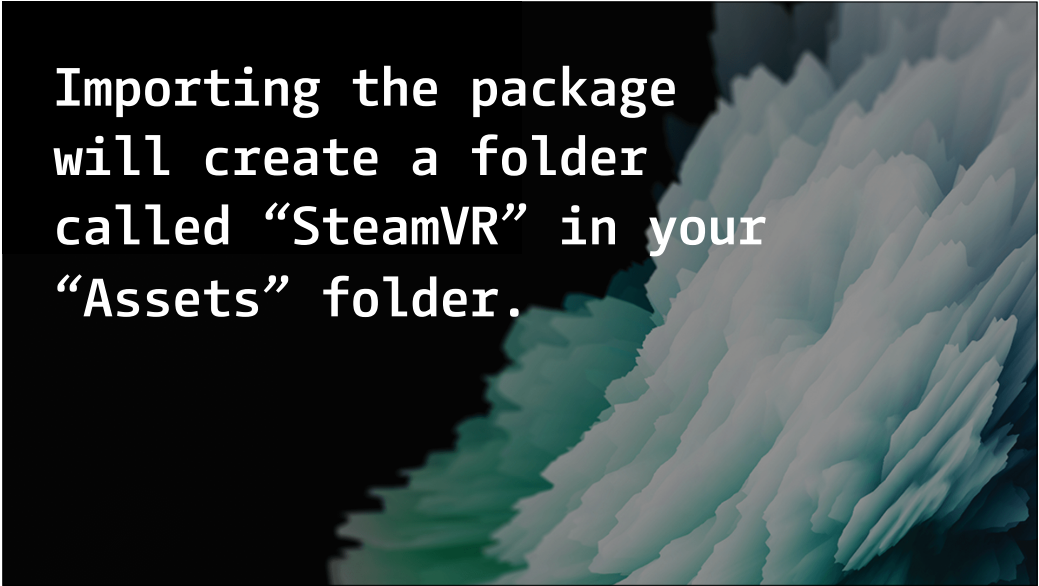


Hand presence and buttons:

- Hair Trigger
- Grip
- Touchpad (detects touch on X,Y Axis as well as press)
- Menu (above Touchpad)
- System button (below Touchpad) - similar to “bloom” gesture as it can’t be used in your app, just to pull up menu.



Search for “Steam VR” in the unity asset store

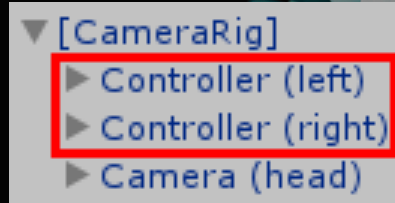


Importing the package
will create a folder
called “SteamVR” in your
“Assets” folder.

Prefabs to include:

- [CameraRig]
 - (Make sure to delete "MainCamera" from your scene)
- [SteamVR]

Nested under [Camera Rig] in your hierarchy, you will see the Vive Controller gameObjects:



Scripts you can use on the controllers:

ViveControllerInput.cs

Reacts to button presses and touchpad input

LaserTeleport.cs

Allows you to teleport by moving the entire Play Area to another part of your scene. Make sure to drag the appropriate prefabs to the public variable slots of this script.

Also, remember to create a new Layer for teleportable area, assign this layer to the floor, and set it in the layer-mask of this script.

(Creating Layers: <https://docs.unity3d.com/Manual/Layers.html>)

**ARx Designers: The Future Kings and
Queens of Silicon Valley**

<https://goo.gl/dfWGeA>

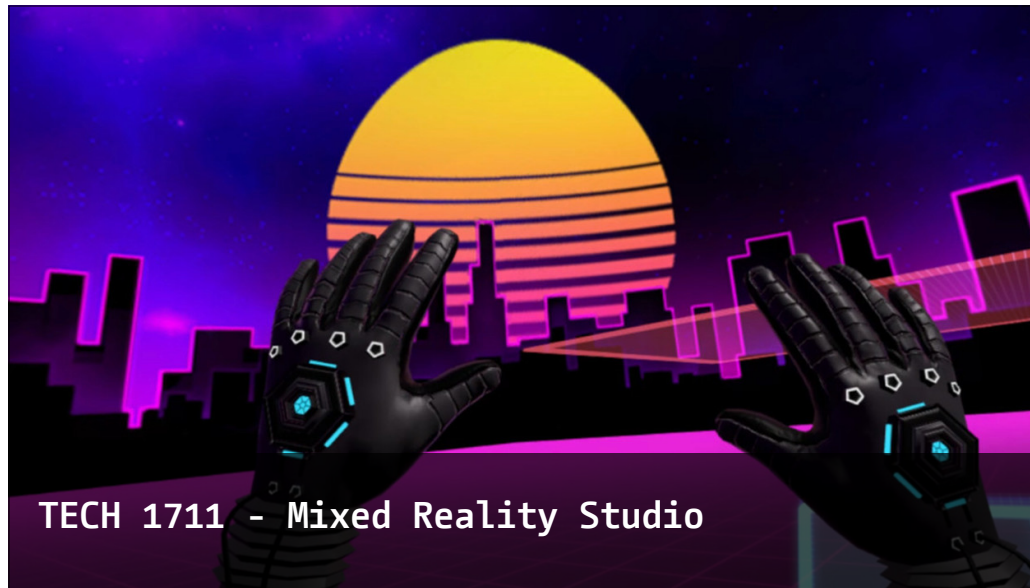
Design For Humanity - Parts 4,5

<http://goo.gl/mWoxTm>

Reading assignment:

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

<https://medium.com/swlh/how-to-design-a-cui-59f1fb3f35fc>



Thank you!