ODIN II

Engineering Logbook

TEAM 11

TODD TOWNSEND, ABRAM FOUTS, KARLA BARRAZA LOPEZ, WENHAO CHENG, XUENING JIA

Sponsor: Brad Smith - Intel Corporation

Advisor: Fu Li

PROJECT DESCRIPTION

ODIN II uses a VR headset to rotate an image to match the alignment of a person's eyes, one image for each the left and the right eye. This will allow the user to see the VR image as it was intended to be rendered. Our project's purpose is to serve patients with Strabismus: a condition which causes a person's eyes to not align parallel to each other.

LOG

Touch Controller

3/6/2020

The hardware has been set up in the special projects lab, and all of the software has been installed. Next steps are getting comfortable with SteamVR within Unity and learn the basic controls.

3/13/2020

After failed attempts at getting the controllers custom input to work we've learned that from the time we have started the project SteamVR came out with a major API change that is still being documented. SteamVR 2.2 was released between our initial research and today's attempted development, so the currently written code is syntactically incorrect and is why we have problems.

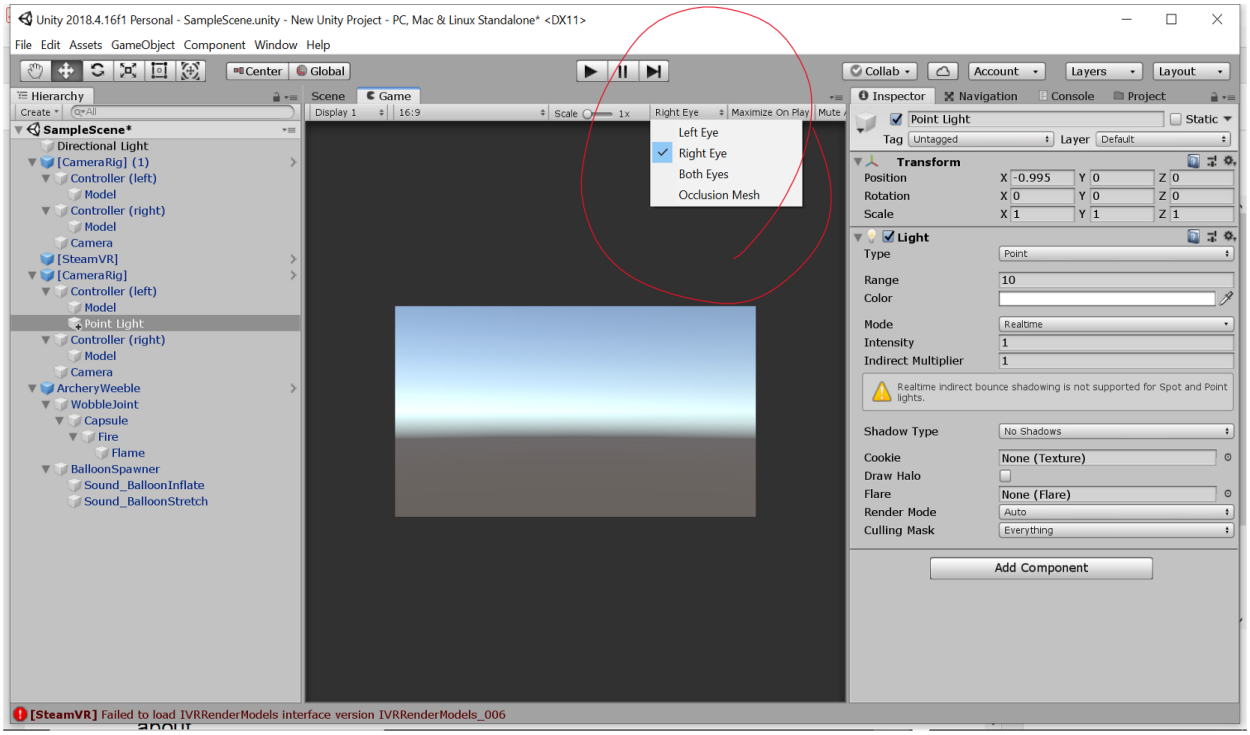
4/6/2020

- SteamVR has changed the way inputs are registered -- into actions and then inputs
- From SteamVR Unity Plugin PDF:
    - Several types of inputs:
        - Boolean - buttons
        - Single - values on triggers (0 and 1)
        - Vector2 - touchpads
        - Vector3
        - Pose
        - Skeleton
    - You can edit action sets
- Go in Unity to window --> SteamVR Input --> default (to see the action sets that are listed under default)
    - The action paths are in: /actions/default/in
    - Create new action, and set to Vector 2 for touchpad touch
        - Give it a name

- Set as type: maybe Vector 2 (because touchpad is vector 2) or Vector 3, or single for the button
- Once in binding UI, expand Trigger and set it as a button and choose grab

Feedback from Ryan
Graph two points and turn in unison
Graph one point as a reference with one controller, and rotate the scene about that point with the other
Would like option to switch between eyes in a menu (left, right, both)
○ Bring up menu with standard action ○ Go to "Rotate" or "Translate" ○ Choose left or right eye ○ Would like a guide prompt with instructions for how to proceed ○ Expecting to use BUTTONS
A micro and a macro control option to fine tune rotation

Ideas
When button is pushed, we are able to access FOV, control with controller rotation
When button is pushed, we are able to access FOV, control with touchpad rotation



- We can set each camera rig controller input
    - Camera rig --> controller (left/right)
- Change pose action to: \actions\default\in\Skeleton(Right/Left)Hand to link it to whichever controller we want (for each camera we can disable one hand and then set the other)
- !!!! There's a frustrum.cs under steamvr --> scripts --> steamvr_frustrum
    - Also look at: https://forum.unity.com/threads/how-to-rotate-a-vr-camera-through-code-with-steam-vr-plugin-camera-rig.444461/
    - Camera Follow Player Position & Rotation in Unity 3D

GOAL FOR NEXT SESSION: find the graphics interface!!!!!

Next session:

We can control the camera rotation just like we can it's position -- but maybe not dynamically (easily) as I'm hearing that a lot of systems are designed explicitly to keep this from happening

- Look at these:
    - https://forum.unity.com/threads/steam-vr-how-to-rotate-independently-left-right-cameras.415491/

        - https://docs.unity3d.com/ScriptReference/Camera-onPreRender.html?_ga=2.183401830.1316330623.1586323790-1004963644.1583728633
    - Read comment from bgolus,
        - https://forum.unity.com/threads/how-to-make-an-object-child-parent-of-another-one-at-runtime.31988/
    - How to create parent/child objects at runtime
        - https://docs.unity3d.com/Manual/InstantiatingPrefabs.html
    - How to instantiate objects at run time

Possible solution: make the camera a child of another object, rotate the parent through code

- How to rotate object based on rotation of steamvr controller: https://answers.unity.com/questions/1570927/rotate-object-based-on-rotation-of-steamvr-control.html
    - We could get the object's Transform (https://docs.unity3d.com/ScriptReference/Transform.html) and then manipulate it with the Vector3.Euler function (https://docs.unity3d.com/ScriptReference/Quaternion.Euler.html) which rotates about the z axis

NOTE: QUATERNION IS THE DATA TYPE USED TO REPRESENT ROTATIONS IN UNITY

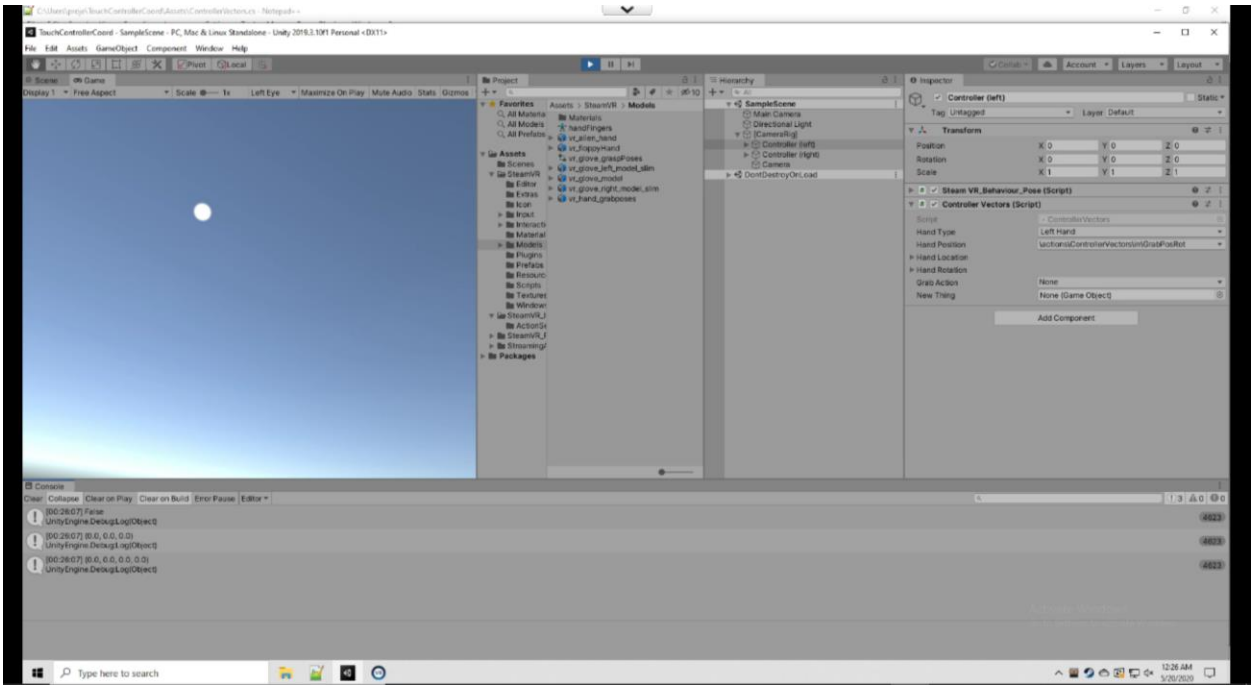
Cool tutorial that I should try out to get rotation from Pose action, not skeleton action
https://lawlorcode.wordpress.com/2019/03/24/steamvr-2-2-input-system-tiny-example/

4/29/2020

Need to make sure to use SteamVR_Action_Skeleton as it has location and rotation members. SteamVR_Action_Pose seems similar but it doesn't have anything for location and rotation.

5/26/2020

Success! I needed to activate the bindings and that allowed me to get input from the controller.



FOV

4/9/2020

https://docs.unity3d.com/ScriptReference/Camera-fieldOfView.html

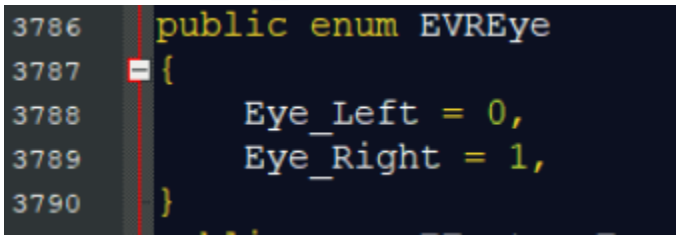
- Numerical value in degrees of field of view held in the camera
  - Camera.fieldofview
  - Bool Usephysicalproperties - enable [usephysicalproperties]to use physical camera properties to compute the field of view and the frustrum
- https://www.youtube.com/watch?v=IrSh0GLo3As
  - How to manipulate a camera FOV in unity

In order to be able to connect a script like this to the SteamVR API, you need to connect it to an action. Check out this video to learn how to do that: https://www.youtube.com/watch?v=bn8eMxBcI70 , this is also a good resource: *https://www.raywenderlich.com/9189-htc-vive-tutorial-for-unity*

SteamVR works off of different types of actions, you can learn about what kind of actions there are here: https://valvesoftware.github.io/steamvr_unity_plugin/articles/SteamVR-Input.html

Gaze Vectors

1/18/20:

- Researched Unity API
  - Described today that we will be needing to find the gaze vector hooks for Intel for future development. They will use our research for future possible strabismus correction.
  - Looking in Unity API to find the variables responsible for holding the value of the gaze vectors
  - URL: https://github.com/ValveSoftware/steamvr_unity_plugin/blob/master/Assets/SteamVR/Plugins/openvr_api.cs
  - 
```
3786  public enum EVREye
3787  {
3788      Eye_Left = 0,
3789      Eye_Right = 1,
3790  }
```
  - Line 3786 in Unity API
- Researched VIVE developers site
  - Find how to develop new functions for VIVE Pro Eye virtual reality headset
  - URL: https://developer.vive.com/eu/
  - I found that by importing the “assets” into unity the entire VIVE SDK will be available in Unity
  - How do do this for all the SDK’s needed will be essential

1/25/20:

- Researched Unity eye tracking
  - Looking at previous examples of how other developers accomplished eye tracking
  - URL: https://www.youtube.com/watch?v=wDRAKcKz9rk
  - I found that I will need the SteamVR SDK imported into Unity
  - Head mounted display (HMD) needs to be initialized
  - Packages are not the same in this source
  - Will need to find another example that is the same as VIVE Pro Eye SDK, otherwise the variables for the gaze vectors will not be the same
- Researched SteamVR
  - Looking at what this really is and what is needed
  - URL: https://stackoverflow.com/questions/56786269/unity3d-steamvr-tutorial-vive-pro-source-code
  - I found that this is a program that is necessary for the head mounted display (VIVE Pro Eye) to run.
  - The program runs in the background and is responsible for all of the interaction between Unity and the head mounted display

2/2/20:

- Researched HMD source code
    - I need to find out the similarities and differences between the code in the Unity API and the HMD
    - URL: https://github.com/ViveSoftware/VIVE-Wave-SDK-Unreal
    - I found that the HMD source code is modified to match SteamVR API
- Researched SteamVR source code
    - I need find the correct variable types to match the SteamVR API
    - URL: https://valvesoftware.github.io/steamvr_unity_plugin/api/index.html
    - I found that I need to match the variables in SteamVR API

2/7/20:

- Researched and installed Unity
    - I am beginning to develop my source code
    - URL: https://unity3d.com/get-unity/download
    - I found that the installation is simple and I now am setup
- Researched the needed plugins for Unity, so HMD can work
    - I need the SteamVR plugin for sure
    - URL: https://github.com/ValveSoftware/steamvr_unity_plugin/blob/master/Assets/SteamVR/Plugins/openvr_api.cs

```
namespace Valve.VR
{

[StructLayout(LayoutKind.Sequential)]
public struct IVRSystem
{
        [UnmanagedFunctionPointer(CallingConvention.StdCall)]
        internal delegate void _GetRecommendedRenderTargetSize(ref uint pnWidth, ref uint pnHeight);
        [MarshalAs(UnmanagedType.FunctionPtr)]
        internal _GetRecommendedRenderTargetSize GetRecommendedRenderTargetSize;
```

- I found that I will need to install this in Unity and learn more C# coding
    - It is possible I can write a function in C or Python to call the SteamVR plugin

2/18/20:

- Researched and took C# tuterorials
    - I need to learn C# better to understand how the SteamVR plugin works
    - URL: https://www.udemy.com/course/unitycourse/?utm_source=adwords&utm_medium=udemyads&utm_campaign=Unity_v.PROF_la.EN_cc.US_ti.8154&utm_content=deal4584&utm_term=_.ag_83329501114_.ad_436603255162_.kw__.de_c_.dm__.pl__.ti_dsa-774930034729_.li_9032912_.pd__&matchtype=b&gclid=CjwKCAjwiMj2BRBFEiwAYfTbCsAHwfe4rc_w2T5qEDs9kPMO13MpLszwqWtbpCBP2Tj5_7sIs4QqxRoCjccQAvD_BwE
    - I found that C# is very similar to C, looks like Python used some of their practices from C#

- Researched Unity API
    - This will be a constant place of research until I find the placement of the x, y, z three dimensional vector
    - URL: https://github.com/ValveSoftware/steamvr_unity_plugin/blob/master/Assets/SteamVR/Plugins/openvr_api.cs
    - I found a three dimensional vector, I'm not sure if this is the right one, so I will need to start coding to find out

```
5033    public Vector3 GetPosition()
5034    {
5035        return new Vector3(m3, m7, -m11);
5036    }
5037
```

3/2/20:

- Researched previous coding solutions for obtaining vectors in Unity
    - I need to find a similar solution so I can get an idea of what I need to do to lock the value of the x, y, z vector in place
    - URL: https://forum.unity.com/threads/vive-pro-eye-unity-retina-tracking-examples.707000/
    - I found that previous examples, you can write an output script that will tie into the SteamVR API
    - You can have the underlying SteamVR API send the output to the Unity command line
- Developed my first draft of code
    - Code has compiled in IDE but I still cannot get compilation in the Unity editor
    - The errors exist in the types of variables
    - "Variable does not exist in this space"
    - For C# it is "using" instead of "#include"
    - I thought that I got all of my "using" statements and I made a new instance of HMDVector3_t but it is still not compiling

This is my first script, it is not working

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;
using System.Diagnostics;

public class Command_Line : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        namespace runGnomeTerminal
        {
            class MainClass
            {
                public static void ExecuteCommand(string Vector3)
                {
                    Process proc = new System.Diagnostics.Process ();
                    proc.StartInfo.FileName = "/bin/bash";
                    proc.StartInfo.Arguments = "-c \" " + Vector3 + " \"";
                    proc.StartInfo.UseShellExecute = false;
                    proc.StartInfo.RedirectStandardOutput = true;
                    proc.Start ();

                    while (!proc.StandardOutput.EndOfStream) {
                        Console.WriteLine (proc.StandardOutput.ReadLine ());
                    }
                }

                public static void Main (string[] args)
                {
                    ExecuteCommand("gnome-terminal -x bash -ic 'cd $HOME; ls; bash'");
                }


            }
        }
    }
```

This is my second script, it is not working, but now I'm making a call to the console.writeline on every clock cycle. This is the right method, but I think I need windows command line not Linux

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;
using System.Diagnostics;

public class Command_Line : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        namespace runGnomeTerminal
        {
            class MainClass
            {
                public static void ExecuteCommand(string Vector3)
                {
                    Process proc = new System.Diagnostics.Process ();
                    proc.StartInfo.FileName = "/bin/bash";
                    proc.StartInfo.Arguments = "-c \" " + Vector3 + " \"";
                    proc.StartInfo.UseShellExecute = false;
                    proc.StartInfo.RedirectStandardOutput = true;
                    proc.Start ();

                    while (!proc.StandardOutput.EndOfStream) {
                        Console.WriteLine (proc.StandardOutput.ReadLine ());
                    }
                }

                public static void Main (string[] args)
                {
                    ExecuteCommand("gnome-terminal -x bash -ic 'cd $HOME; ls; bash'");
                }


            }
        }
    }
```

3/7/20:

- Researched and wrote another script that will be compatible in windows
    - I need to find out if my current method will work
    - URL: https://forum.unity.com/threads/vive-pro-eye-unity-retina-tracking-examples.707000/

- I found that my new script still is having a variable reference error, now that i have switched the command line output to a windows host method
- I need to come up with a new plan, either modify a existing script or modify the SteamVR API to produce the output that I desire

This is my third script, it is still not working, I'm making a call to the console.writeline on every clock cycle. This is now in the windows command line format, but the variable reference is still throwing a type and value error

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;
using System.Diagnostics;

public class Command_Line : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
        // Open command Line Prompt ************************************************************//
        Process cmd = new Process();
        cmd.StartInfo.FileName = "cmd.exe";
        cmd.StartInfo.RedirectStandardInput = true;
        cmd.StartInfo.RedirectStandardOutput = true;
        cmd.StartInfo.CreateNoWindow = true;
        cmd.StartInfo.UseShellExecute = false;
        cmd.Start();

    }

    // Update is called once per frame *************************************************//
    void Update()
    {
        // GAZE VECTORS - PRINT ************************************************************//
        cmd.StandardInput.WriteLine("============== X-axis Vector =" + HmdVector3_t.v0 + " ==============");
        cmd.StandardInput.WriteLine("============== Y-axis Vector =" + HmdVector3_t.v1 + " ==============");
        cmd.StandardInput.WriteLine("============== Z-axis Vector =" + HmdVector3_t.v3 + " ==============");
        sleep(2);

        // Eye Origin and destinition - Print ******************************************//
        cmd.StandardInput.WriteLine("============== Eye Position Origin =" + VROverlayIntersectionParams_t.eOrigin + " ==============");
        sleep(2);
        // Flush Buffer
        cmd.StandardInput.Flush();
        cmd.StandardInput.Close();
        cmd.WaitForExit();
        Console.WriteLine(cmd.StandardOutput.ReadToEnd());
```

3/15/20:

- Researched new methods for fixing my variable type error
  - I need to find a way to make the variable values come over from the StaemVR API
  - Preserving their value
  - URL: https://support.getfove.com/hc/en-us/community/posts/360029033073-Unity-how-to-get-raw-x-y-position-of-gaze-as-a-function-of-the-screen-
  - I found that there are many ways to produce the gaze vectors i am looking for, but the stand alone script may not be the answer
- Researched how to write command line display without making a stand alone script
  - Investigated how to modify the default scripts within Unity
  - I need to find a new method as my first three scripts are not working
  - URL: https://github.com/ValveSoftware/steamvr_unity_plugin/tree/master/Assets/SteamVR
  - I found that it is possible to modify the SteamVR API I will try this

3/21/20:

- Researched and coded new solution
  - I need to find another way to accomplish this, as the SteamVR API and the corresponding scripts are very complex

- URL: https://valvesoftware.github.io/steamvr_unity_plugin/api/index.html
- I have now attempted to modify the SteamVR API near the x, y, z coordinates of the gaze vector
- I have almost certainly found the correct vector now
- I did not have the correct vector before
- I found that I now have far less errors
- The errors that i do create though, send me syntax errors in the corresponding scripts
- I have now reverse engineered the syntax errors and have found the calling scripts! This is a huge discovery
- I will continue to create more syntax errors in the API so I can find the scripts that I need to modify

This is the vector I need for the gaze vectors, “Vector3”

```
37
38              var corners = new Vector3[] {
39              new Vector3(lsin * nearZ / lcos, tsin * nearZ / tcos, nearZ), //tln
40              new Vector3(rsin * nearZ / rcos, tsin * nearZ / tcos, nearZ), //trn
41              new Vector3(rsin * nearZ / rcos, bsin * nearZ / bcos, nearZ), //brn
42              new Vector3(lsin * nearZ / lcos, bsin * nearZ / bcos, nearZ), //bln
43              new Vector3(lsin * farZ  / lcos, tsin * farZ  / tcos, farZ ), //tlf
44              new Vector3(rsin * farZ  / rcos, tsin * farZ  / tcos, farZ ), //trf
45              new Vector3(rsin * farZ  / rcos, bsin * farZ  / bcos, farZ ), //brf
46              new Vector3(lsin * farZ  / lcos, bsin * farZ  / bcos, farZ ), //blf
47          };
```

This is my fourth attempt, now modifying the SteamVR API, it is not working but I have found that causing syntax errors in the SteamVR API, I can reverse engineer the errors and find the script that i need to modify.

```
7030        public CVRSpatialAnchors VRSpatialAnchors()
7031        {
7032            CheckClear();
7033            if (m_pVRSpatialAnchors == null)
7034            {
7035                var eError = EVRInitError.None;
7036                var pInterface = OpenVRInterop.GetGenericInterface(FnTable_Prefix + IVRSpatialAnchors_Version, ref eError);
7037                if (pInterface != IntPtr.Zero && eError == EVRInitError.None)
7038                    m_pVRSpatialAnchors = new CVRSpatialAnchors(pInterface);
7039
7040                // Gaze Vector Frustrum Print in SteamVR_Frustrum Line 83
7041                Debug.Log("============== X-axis Vector Frustrum =" + m_pVRSpatialAnchors + " ==============");
7042                Debug.Log("============== Y-axis Vector Frustrum =" + m_pVRSpatialAnchors + " ==============");
7043                Debug.Log("============== Z-axis Vector Frustrum =" + m_pVRSpatialAnchors + " ==============");
7044            }
7045            return m_pVRSpatialAnchors;
7046        }
```

3/30/20:

- Researched and coded new script solution
  - I now know what vector I need and I also know where I need to put my modifications
  - I need to find out more about the corresponding scripts
  - The scripts are responsible for sending updates and real-time code insertions
  - URL: https://valvesoftware.github.io/steamvr_unity_plugin/api/index.html
  - I found that I need to modify the existing scripts
  - I am down to two errors and i have the variables transferring

4/5/20:

- Researched and coded new script solution

- I need to insert another variable type and I am stuck figuring out how the value comes back to unity
- URL: https://valvesoftware.github.io/steamvr_unity_plugin/api/index.html
- I found that inserting the code into the scripts may not be the entire solution, only part of the solution.

This is my fifth attempt, now modifying the SteamVR_Frustum.cs in the SteamVR API, it is not working yet, but I have found that this is the correct operation and the triangulation of the vector is the one that I need

```
68          {
69              var a = corners[triangles[i * 3 + 0]];
70              var b = corners[triangles[i * 3 + 1]];
71              var c = corners[triangles[i * 3 + 2]];
72              var n = Vector3.Cross(b - a, c - a).normalized;
73              normals[i * 3 + 0] = n;
74              normals[i * 3 + 1] = n;
75              normals[i * 3 + 2] = n;
76              vertices[i * 3 + 0] = a;
77              vertices[i * 3 + 1] = b;
78              vertices[i * 3 + 2] = c;
79              triangles[i * 3 + 0] = j++;
80              triangles[i * 3 + 1] = j++;
81              triangles[i * 3 + 2] = j++;
82
83              // Gaze Vector Frustrum Print in SteamVR_Frustrum Line 83
84              Debug.Log("============== X-axis Vector Frustrum =" + a + " ==============");
85              Debug.Log("============== Y-axis Vector Frustrum =" + b + " ==============");
86              Debug.Log("============== Z-axis Vector Frustrum =" + c + " ==============");
87          }
```

4/12/20:

- Researched and coded new script solution
  - I need to find the way to bring the value of the vectors x, y, z components back into the Unity command line.
  - I have achieved compilation and have verified that this is the correct vector
  - URL: https://developer.tobii.com/community/forums/topic/interpreting-the-tobii-htc-vive-eye-tracking-data/
  - I have found that I may need this modification plus a stand alone script

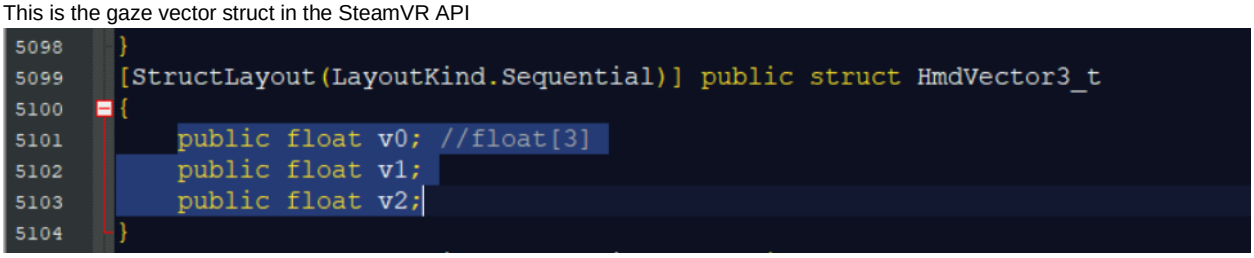
4/12/20:

- Researched and coded new script solution for the field of view now as well
  - Teammembers are having trouble with their part of the project so I have now taken on trying to get the hooks for the gaze vectors and the field of view vectors
  - URL: https://developer.tobii.com/pc-gaming/unity-sdk/scripting-api/
  - I have found that I can write the field of view code just like I wrote the gaze vector code, but I still need a secondary solution
  - I will need to write a stand alone script for both the gaze vectors and the field of view vectors

4/20/20:

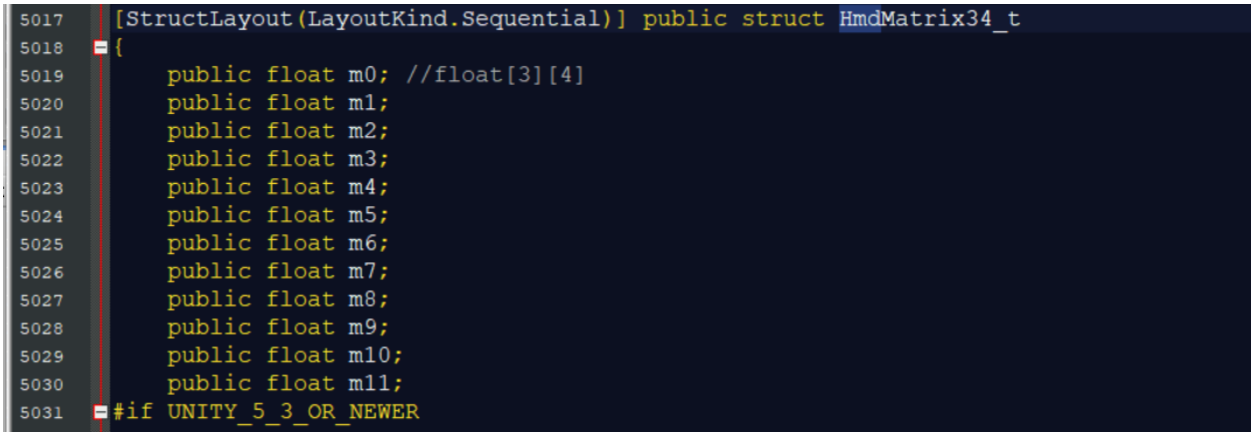
- Researched and coded new script solution for gaze vector and field of view vector
  - I have successfully found and verified the field of view vector
  - URL: https://valvesoftware.github.io/steamvr_unity_plugin/api/index.html
  - I have found both sets of vectors for the field of view vectors and for the gaze vector

This is the gaze vector struct in the SteamVR API



```
5098    }
5099    [StructLayout(LayoutKind.Sequential)] public struct HmdVector3_t
5100    {
5101        public float v0; //float[3]
5102        public float v1;
5103        public float v2;
5104    }
```

This is the field of view vector struct in the SteamVR API



```
5017    [StructLayout(LayoutKind.Sequential)] public struct HmdMatrix34_t
5018    {
5019        public float m0; //float[3][4]
5020        public float m1;
5021        public float m2;
5022        public float m3;
5023        public float m4;
5024        public float m5;
5025        public float m6;
5026        public float m7;
5027        public float m8;
5028        public float m9;
5029        public float m10;
5030        public float m11;
5031    #if UNITY_5_3_OR_NEWER
```

4/29/20:

- Researched and coded new script solution for gaze vector and field of view vector
    - I need to now find what my secondary solution is, I have achieved compilation with both of my code insertions into the scripts of the SteamVR API
    - URL: https://medium.com/@sarthakghosh/a-complete-guide-to-the-steamvr-2-0-input-system-in-unity-380e3b1b3311
    - I have found that I need to write a stand alone script that I can call the functions that I have modified in the default scripts within the SteamVR API

This is the code insertion for the field of view vector in SteamVR_Utils.cs. This is the correct operation as I have achieved compilation, but I still need a stand alone script to call this function

```
350         public HmdMatrix34_t ToHmdMatrix34()
351         {
352             var m = Matrix4x4.TRS(pos, rot, Vector3.one);
353             var pose = new HmdMatrix34_t();
354 
355             pose.m0  =  m[0, 0];
356             pose.m1  =  m[0, 1];
357             pose.m2  = -m[0, 2];
358             pose.m3  =  m[0, 3];
359 
360             pose.m4  =  m[1, 0];
361             pose.m5  =  m[1, 1];
362             pose.m6  = -m[1, 2];
363             pose.m7  =  m[1, 3];
364 
365             pose.m8  = -m[2, 0];
366             pose.m9  = -m[2, 1];
367             pose.m10 =  m[2, 2];
368             pose.m11 = -m[2, 3];
369 
370             // Inserted Field of View Print
371             Debug.Log ("============== FOV Vector HMD Matrix 0 = " + pose.m0 + " ==============");
372             Debug.Log ("============== FOV Vector HMD Matrix 1 = " + pose.m1 + " ==============");
373             Debug.Log ("============== FOV Vector HMD Matrix 2 = " + pose.m2 + " ==============");
374             Debug.Log ("============== FOV Vector HMD Matrix 3 = " + pose.m3 + " ==============");
375             Debug.Log ("============== FOV Vector HMD Matrix 4 = " + pose.m4 + " ==============");
376             Debug.Log ("============== FOV Vector HMD Matrix 5 = " + pose.m5 + " ==============");
377             Debug.Log ("============== FOV Vector HMD Matrix 6 = " + pose.m6 + " ==============");
378             Debug.Log ("============== FOV Vector HMD Matrix 7 = " + pose.m7 + " ==============");
379             Debug.Log ("============== FOV Vector HMD Matrix 8 = " + pose.m8 + " ==============");
380             Debug.Log ("============== FOV Vector HMD Matrix 9 = " + pose.m9 + " ==============");
381             Debug.Log ("============== FOV Vector HMD Matrix 10 = " + pose.m10 + " ==============");
382             Debug.Log ("============== FOV Vector HMD Matrix 11 = " + pose.m11 + " ==============");
383 
384             return pose;
385         }
```

5/5/20:

- Researched and coded new script solution for gaze vector and field of view vector
    - I have now coded the stand alone scripting solution for the gaze vectors
    - I need this solution to call the function that is holding my modifications in the SteamVR API script
    - URL: https://valvesoftware.github.io/steamvr_unity_plugin/api/index.html
    - I have found that I still have compilation errors as the value of the vector variables are not recognized in this new script

This is the stand alone code solution for the gaze vectors. It seems to be correct but I am still receiving compilation errors. The value of the vector is not coming into Unity as it is not recognized.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;
using System.Diagnostics;

public class Command_Line : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
        // Open command Line Prompt ************************************************************//
        Process cmd = new Process();
        cmd.StartInfo.FileName = "cmd.exe";
        cmd.StartInfo.RedirectStandardInput = true;
        cmd.StartInfo.RedirectStandardOutput = true;
        cmd.StartInfo.CreateNoWindow = true;
        cmd.StartInfo.UseShellExecute = false;
        cmd.Start();

    }

    // Update is called once per frame *****************************************************//
    void Update()
    {
        // GAZE VECTORS - PRINT ****************************************************************//
        cmd.StandardInput.WriteLine("============== X-axis Vector =" + HmdVector3_t.v0 + " ==============");
        cmd.StandardInput.WriteLine("============== Y-axis Vector =" + HmdVector3_t.v1 + " ==============");
        cmd.StandardInput.WriteLine("============== Z-axis Vector =" + HmdVector3_t.v3 + " ==============");
        sleep(2);

        // Eye Origin and destinition - Print ***************************************************//
        cmd.StandardInput.WriteLine("============== Eye Position Origin =" + VROverlayIntersectionParams_t.eOrigin + " ==============");
        sleep(2);
        // Flush Buffer
        cmd.StandardInput.Flush();
        cmd.StandardInput.Close();
        cmd.WaitForExit();
        Console.WriteLine(cmd.StandardOutput.ReadToEnd());
```

5/10/20:

- Researched and coded new script solution for gaze vector and field of view vector
    - I have written the stand alone script for the field of view vector
    - URL: https://valvesoftware.github.io/steamvr_unity_plugin/api/index.html
    - I have found that I still have compilation errors in the stand alone field of view script, just as the stand alone gaze vector script, as the value of the vector variables are not recognized in this new script
    - The errors are almost identical

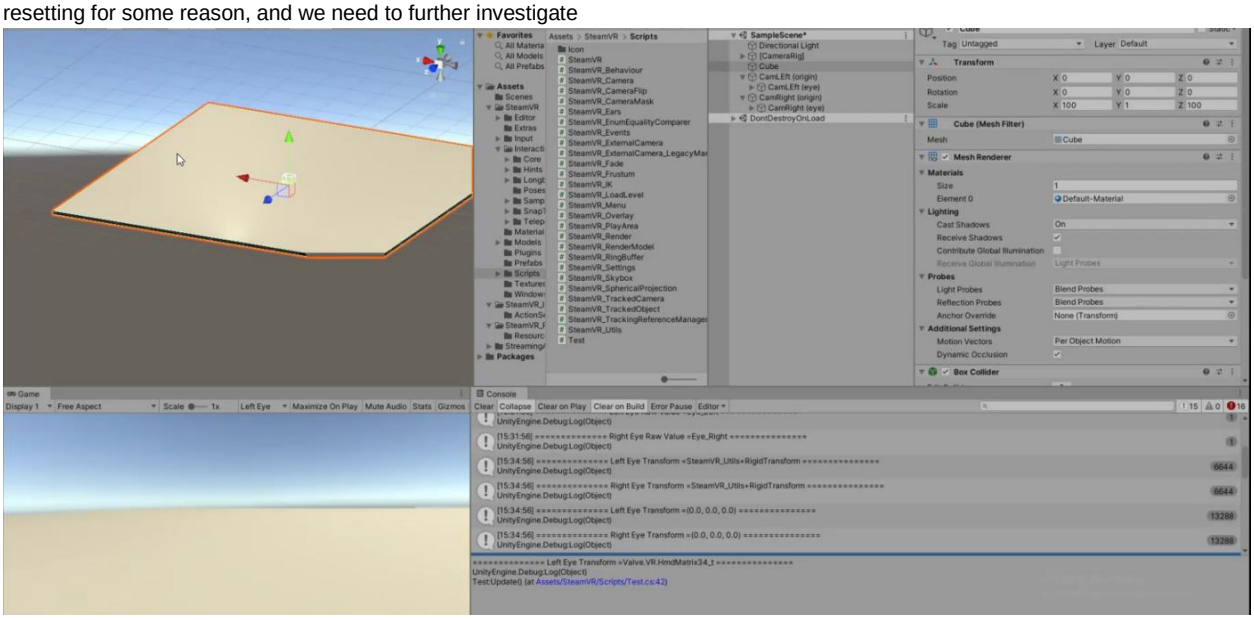
This is the stand alone code solution for the field of view vectors. It seems to be correct but I am still receiving compilation errors. The value of the vector is not coming into Unity as it is not recognized.

```
public class Command_Line_FOV : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
        // Open command Line Prompt ************************************************************//
        Process cmd = new Process();
        cmd.StartInfo.FileName = "cmd.exe";
        cmd.StartInfo.RedirectStandardInput = true;
        cmd.StartInfo.RedirectStandardOutput = true;
        cmd.StartInfo.CreateNoWindow = true;
        cmd.StartInfo.UseShellExecute = false;
        cmd.Start();

    }

    // Update is called once per frame
    void Update()
    {
        // FOV Vertex Data - PRINT *************************************************************//
        cmd.StandardInput.WriteLine("============== FOV Vertex Data = " + *RenderModel_t.rVertexData + " ==============");
        cmd.StandardInput.WriteLine("============== FOV Vertex Count = " + RenderModel_t.unVertexCount + " ==============");
        cmd.StandardInput.WriteLine("============== FOV Index Data = " + *RenderModel_t.rIndexData + " ==============");
        cmd.StandardInput.WriteLine("============== FOV Vertex Triangulation Count = " + RenderModel_t.unTriangleCount + " ==============");
        cmd.StandardInput.WriteLine("============== FOV Texture Identification = " + RenderModel_t.diffuseTextureId + " ==============");
        sleep(2);

        // FOV Vectors in HMD Matrix - PRINT ***************************************************//
        cmd.StandardInput.WriteLine("============== FOV Vector HMD Matrix 0 = " + HmdMatrix34_t.m0 + " ==============");
        cmd.StandardInput.WriteLine("============== FOV Vector HMD Matrix 1 = " + HmdMatrix34_t.m1 + " ==============");
        cmd.StandardInput.WriteLine("============== FOV Vector HMD Matrix 2 = " + HmdMatrix34_t.m2 + " ==============");
        cmd.StandardInput.WriteLine("============== FOV Vector HMD Matrix 3 = " + HmdMatrix34_t.m3 + " ==============");
        cmd.StandardInput.WriteLine("============== FOV Vector HMD Matrix 4 = " + HmdMatrix34_t.m4 + " ==============");
        cmd.StandardInput.WriteLine("============== FOV Vector HMD Matrix 5 = " + HmdMatrix34_t.m5 + " ==============");
        cmd.StandardInput.WriteLine("============== FOV Vector HMD Matrix 6 = " + HmdMatrix34_t.m6 + " ==============");
        cmd.StandardInput.WriteLine("============== FOV Vector HMD Matrix 7 = " + HmdMatrix34_t.m7 + " ==============");
        cmd.StandardInput.WriteLine("============== FOV Vector HMD Matrix 8 = " + HmdMatrix34_t.m8 + " ==============");
        cmd.StandardInput.WriteLine("============== FOV Vector HMD Matrix 9 = " + HmdMatrix34_t.m9 + " ==============");
        cmd.StandardInput.WriteLine("============== FOV Vector HMD Matrix 10 = " + HmdMatrix34_t.m10 + " ==============");
        cmd.StandardInput.WriteLine("============== FOV Vector HMD Matrix 11 = " + HmdMatrix34_t.m11 + " ==============");
        sleep(2);

```

5/15/20:

- Researched and coded new script solution for gaze vector and field of view vector
    - I have successfully achieved the gaze vector hooks as I have locked down the position of the vector and the stand alone function is calling the function in the SteamVR API
    - The output seems to be getting reset as the data is transferred to Unity
    - URL: https://www.tobii.com/
    - I learned that I need to import Tobii API settings and the SDK units into Unity

This is the gaze vector hook displaying the vectors in the Unity command line. The vectors from the head mounted display are resetting for some reason, and we need to further investigate



5/17/20:

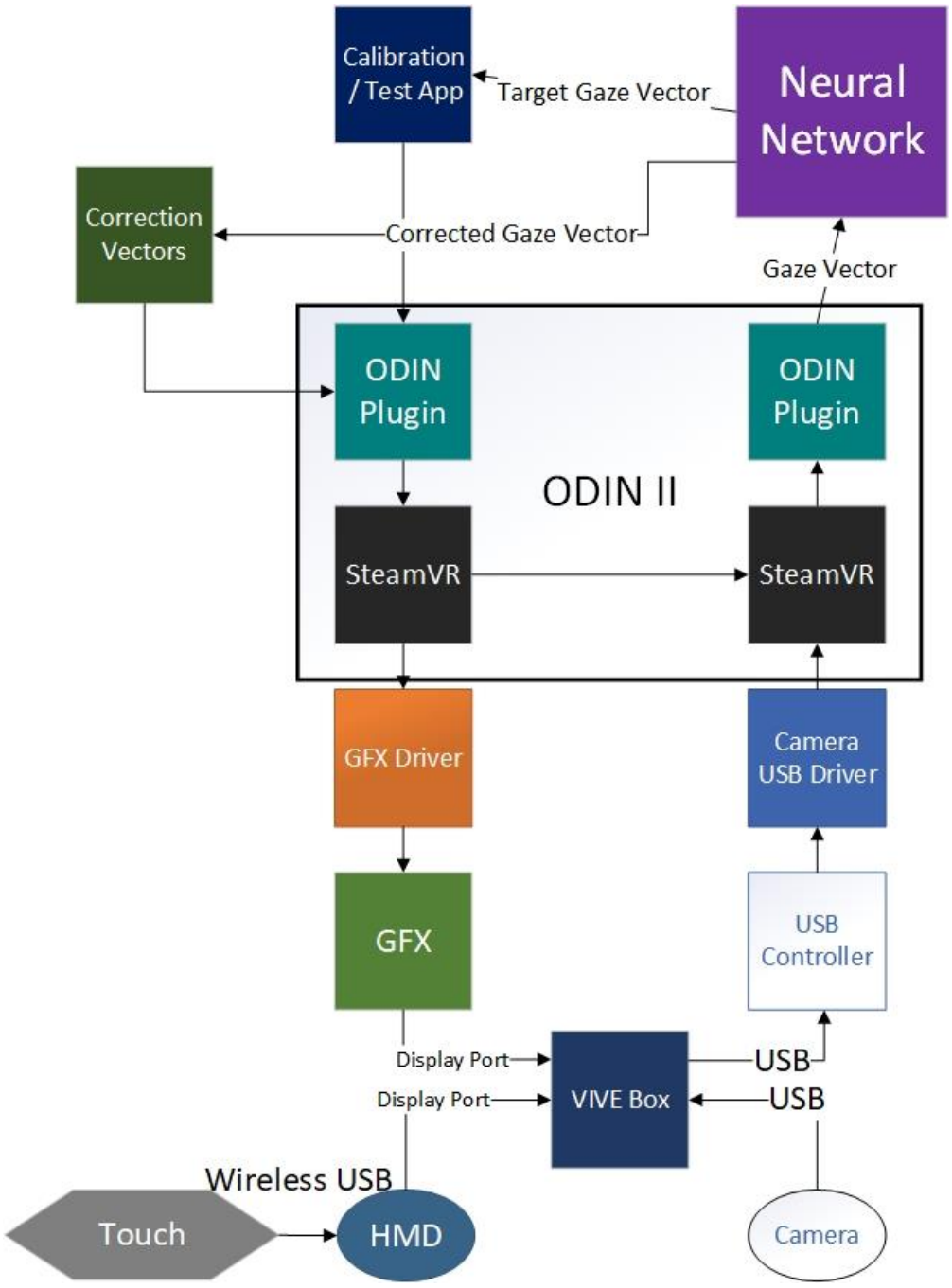
- Researched and coded new script solution for gaze vector and field of view vector
    - I have successfully imported Tobii API settings and the SDK units into Unity
    - The output is holding a rotational value
    - I had to modify the code to match the variable types
    - URL: https://valvesoftware.github.io/steamvr_unity_plugin/api/index.html
    - I learned that the Tobii variables are included in the Tobii SDK
    - URL: https://www.tobii.com/

This is the second version of the stand alone code solution for the gaze vectors.There are no compilation errors and the code has hooked the gaze vectors, during run-time the vales a continually cleared for each clock cycle

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

using Valve.VR;
using System.IO;
using System.Linq;

#if UNITY_2017_2_OR_NEWER
    using UnityEngine.XR;
#else
using XRSettings = UnityEngine.VR.VRSettings;
using XRDevice = UnityEngine.VR.VRDevice;
#endif

public class Test : MonoBehaviour
{
    // Start is called before the first frame update
    UnityEngine.Vector3 eyes1, eyes2;
    //hmd = OpenVR.System;
    public CVRSystem hmd { get; private set; }
    void Start()
    {
            //Vector3 eyes[2];

            hmd = OpenVR.System;
            eyes1 = new SteamVR_Utils.RigidTransform(hmd.GetEyeToHeadTransform(EVREye.Eye_Left));
            eyes2 = new SteamVR_Utils.RigidTransform(hmd.GetEyeToHeadTransform(EVREye.Eye_Right));
    }

    // Update is called once per frame
    void Update()
    {
            // Update eye offsets to account for IPD changes.
            eyes1 = new SteamVR_Utils.RigidTransform(hmd.GetEyeToHeadTransform(EVREye.Eye_Left));
            eyes2 = new SteamVR_Utils.RigidTransform(hmd.GetEyeToHeadTransform(EVREye.Eye_Right));

            // Print Left and right Eye Transform
            Debug.Log("============== Left Eye Transform =" +  eyes1 + " ==============");
            Debug.Log("============== Right Eye Transform =" +  eyes2 + " ==============");
    }
}
```

```
[13:06:23] ============== Left Eye Rotation =(0.0, 0.0, 0.0, 1.0) ==============  3118
UnityEngine.Debug:Log(Object)
[13:06:23] ============== Right Eye Rotation =(0.0, 0.0, 0.0, 1.0) ==============  3118
UnityEngine.Debug:Log(Object)
[13:06:23] ============== Left Eye Position =(0.0, 0.0, 0.0) ==============  3118
UnityEngine.Debug:Log(Object)
[13:06:23] ============== Right Eye Position =(0.0, 0.0, 0.0) ==============  3118
UnityEngine.Debug:Log(Object)
```

BLOCK DIAGRAM



MEETING NOTES

1/22/20

- Weekly Check-Ins, not working meeting
    - If we have big problems or things we can't solve, then bring them to the check-in
    - Each teammate should have an assignment
    - Each teammate should have a deadline
    - Weekly report to Dr. Li
        - Note: we can let Dr. Li when we want him in a meeting
    - Github
- Using:
    - Vibe Pro
        - Write a request to the chair for $$ in case Intel can't get it
        - Note: we didn't sign an NDA?
    - Unity
- Remember: we want this to be platform non-specific, so that it can work with any type of VR system
- To Do:
    - Find documentation for new system
    - Reviewing existing code
    - SteamVR - how does this come in? (Ask on Friday)
    - Rough draft proposal

1/24/20

Opens

- Bram - Are we going to need access to the github site of your team, or are we going to be storing everything on the google drive already shared with us? Up to us

  NDA? No

- Todd - Will the headset repair the misalignment of the eye or just show us where it is? To repair, how it works exactly is up in the air

-

Technical Opens

-

AR's

- Next Week have us take the reigns and share our screen to go over out schedule, who's doing what and RD that we have done.

- Light hitting the eye to thinking about what you see is ~120ms
- Currently, the xbox controller moves the eye or changes the plane, we want to replace this with the touch controllers
- Parts that we are working on:(High level process) Take the gaze inference and hooking it into the eye model
- Limitations include field of view when reaching the ends of the headset (becomes blurry)
- Opthamologist is one who works with strabismus
- Inference - The brain knows what it is looking out, determines foreground, background and motion etc. This is using (not training) our built in tools

- Downloading visio and other content
  https://azureforeducation.microsoft.com/devtools
  Pdx email sign in (Follow steps for Microsoft Projects) from
  http://web.cecs.pdx.edu/~faustm/ece411/syllabus.pdf
  But search Visio instead or any other software that may be available

- https://www.youtube.com/watch?v=wDRAKcKz9rk great video for starting

1/29/20

Opens--

Bram:

Karla:

Wenhao:

Xuening:

Todd:

AR's

Take photos what we are working with

- Fove headset, steam computer, controller
- What currently we have
- Maybe code?

Create a collaboration site:

- Trello - AR Karla
- Bit bucket - AR Karla

Projected Plan - Team

- Who is doing what, when, when will it be done
- What software

R&D SteamVR Platform

R&D Unity (For VR)

R&D HTC Vive Pro Eye API

Convert FOV C# code to Python code with HTC Vive Pro Eye API (without xbox controller but with the touch controller)

Vive Pro I Source Code:
https://github.com/ViveSoftware/ViveInputUtility-Unity
https://github.com/AnshulSood11/Position-Tracker
UNITY
kbarra2@pdx.edu
USER: CapstoneTeam11
PASS: Odin2@PSU

2/7/20

Opens

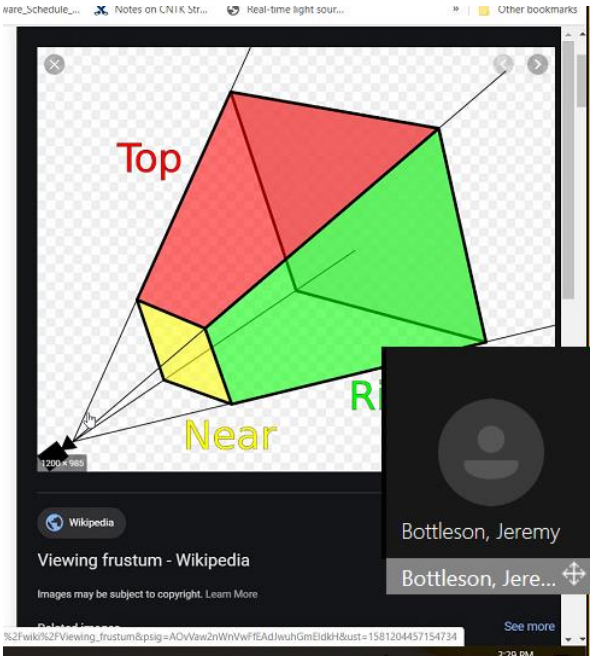
- Headset & funding:
  - Prospectus to be send out by EOD, and ask for hardware and funding needed (Brad)
- Update on existing hardware:

Notes about VR and such:

- The application sits on top of an API
- Physical VR headset
  - Compositor component - runs as a service on the service
    - Runs on the system connected to the VR headset
    - This is what sends the frames to the headset
    - Makes use of graphics driver
    - Custom display drivers
    - It's job is to take incoming frames and push them over to the headset, depending on how fancy the compositor can simply push frames or do "time warp"

- Time warp is when compositor gets frame that is rendered and it was rendered with an assumption as to where the headset was pointing
    - Usually not accurate because of delay
        - 16 ms time slice, the game that is rendering at the start unity polls the compositor and gets the left/right eye frame, then it renders and push it out
        - Compositor wants to push frames to headset at last possible moment so that it's most reminiscent to where the headset is looking on the screen
            - Poll sensors again see where head is looking, calculating delta and will warp the rendered frames its been given to make them look like the area where the actually head is looking
        - Makes frame that you render larger than the field of view
    - Simple translations you can correct for using time warp
- This is what STEAM VR does for you
- What calls the compositor service?
    - The application (ex. unity) would connect to your compositor service
        - Unity provides frames for application and unity and rendering right eye and left eye and pushing these to the compositor service
        - Provides objects and physics for left/right eye view
- The compositor is what talks to the headset and tells the headset what it needs
- There are hooks both in unity and in the compositor service
    - It talks to the HMD to get where the headset is looking at
        - With headsets that don't track eye movement, there is one value that is recorded

- **Frustrum** - when you have a camera or an eye, effectively there is a cone that projects out from the camera or eye and everything within that frustrom is ultimately what gets projected back to the eye



- With VR, you have 2 frustrums that you are rendering, a left eye view and a right eye view
    - Vive: has an IPD setting → the separation distance between the two virtual cameras for the left and right eye

- Two frustums would be rendered but with respect to where each eye is looking
    - The compositor service would take care of looking at those two images and merging them
        - The compositor may be able to give left and right eye frustums
        - The compositor knows about IPD
        - We need to intercept between unity and compositor service to get at these frustums to tweak based on the gaze vectors and how they're representing
            - **Steam VR is our compositor service!!**
                - This may not be something that we can crack open
            - Google search: SteamVr API FOV
                - There is a VR wiki that exposes a lot of this to find where you can modify this value
    - We still want the two parallel frustums, so we should render the two parallel frustums, and maybe we warp the rendered image so it recenters to the location on the screen where the eye is pointing
        - Think Timewarp
        - We could perform our own warp on the rendered image
        - There is a spot where we could adjust the frustum on a per headset basis

- We could have some file with our headset definition ← handled by compositor
    - Each version of the headset has very specific optics and this determines what filters need to be applied
- We need to convince SteamVR to treat us as a Vive Pro Eye
- Suggestion: adding in another layer “custom compositor layer” that goes in between the normal compositor and unity and this layer would handle talking to the compositor & getting the location of the HMD etc and tellng unity what to render
    - What needs to happen is: each frame is rendered with a wider field of view
        - Apply a rotation warp on those returning frames based on where the eye is looking and crop the appropriate field of view and then send that to the compositor
- The pixels are on an x-y coordinate system
    - The frustum itself is in x-y-z coordinates

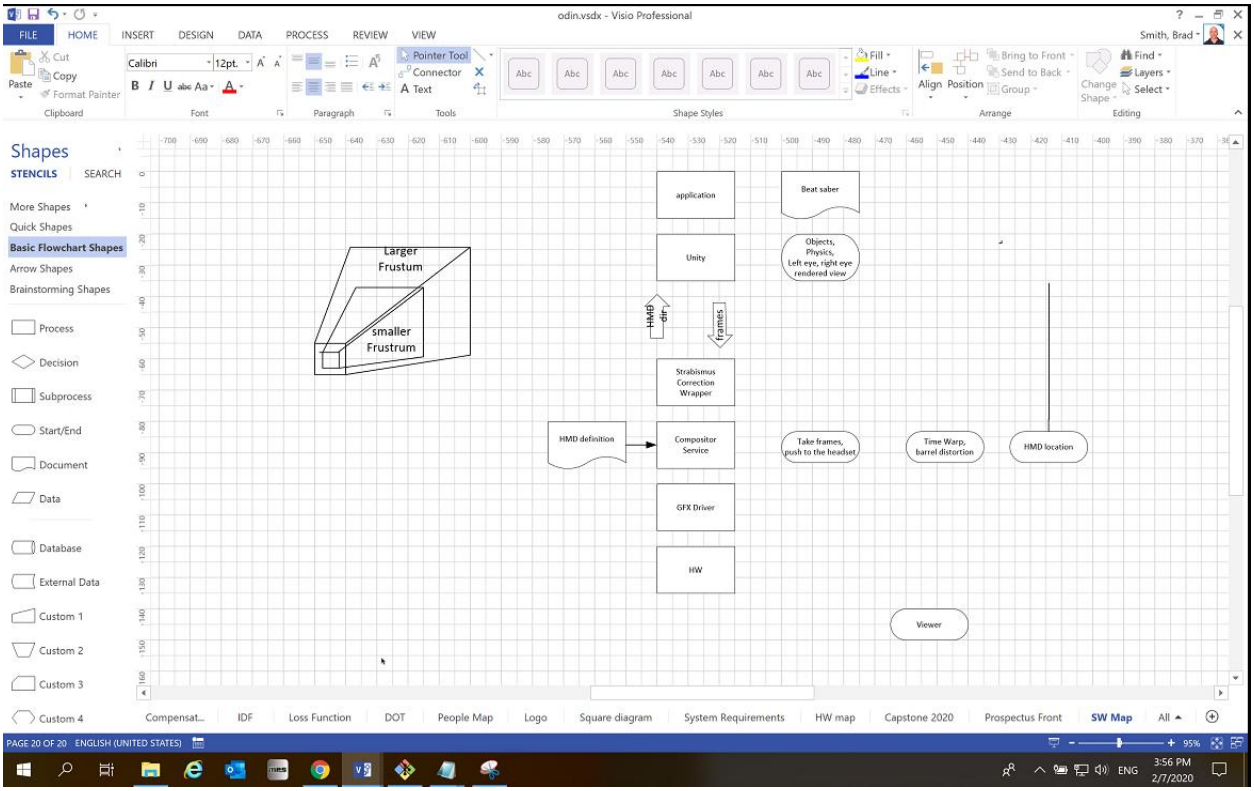
- **Foveated Rendering** - the idea that our eyes only really see detail in the very middle of our retina which only maps to a small % of our FOV
    - If you are aware of that, when you render you don’t have to try to render everything at super high resolution
        - Foveal area = high res
        - Blend region = mid res
        - Peripheral region = low res

How do unity and steamVR community?

- Unity works as a unity plugin to the compositor service
    - Write a standalone module that talks to the compositor, and then unity talks to the module
    - Suspecting that the application will create a link to unity most likely,
    - Or maybe the application makes a link to the strabismus correction wrapper
- Google search: SteamVR Unity Plugin
- Based on eye information, perform some change behavior -- add extra post processing step into the rendering
    - Study foveated rendering to see how these frames are being rendered

Research:

- Look into how foveated rendering is done
- SteamVR API -- look through documentation and what are the upward facing API calls that are exposed
- Looking at the Unity plugin for SteamVR to see what are the upward and downard calls that it does (check the API)



2/14/20

- Develop on the machine we have
- Headset due this coming Friday and next Friday we have the box

Code:

- There's a class called SteamVr Frustrum under Valve.Vr

Options for deliverables:

- Rotate field of view of one eye with touch controller
- Virtual prism lens?
    - So make a VR thing that simulates what it's like to wear a strobismus lens

IP:

- Project is designed as a non-NDA thing
-

2/19/20

Meeting notes:

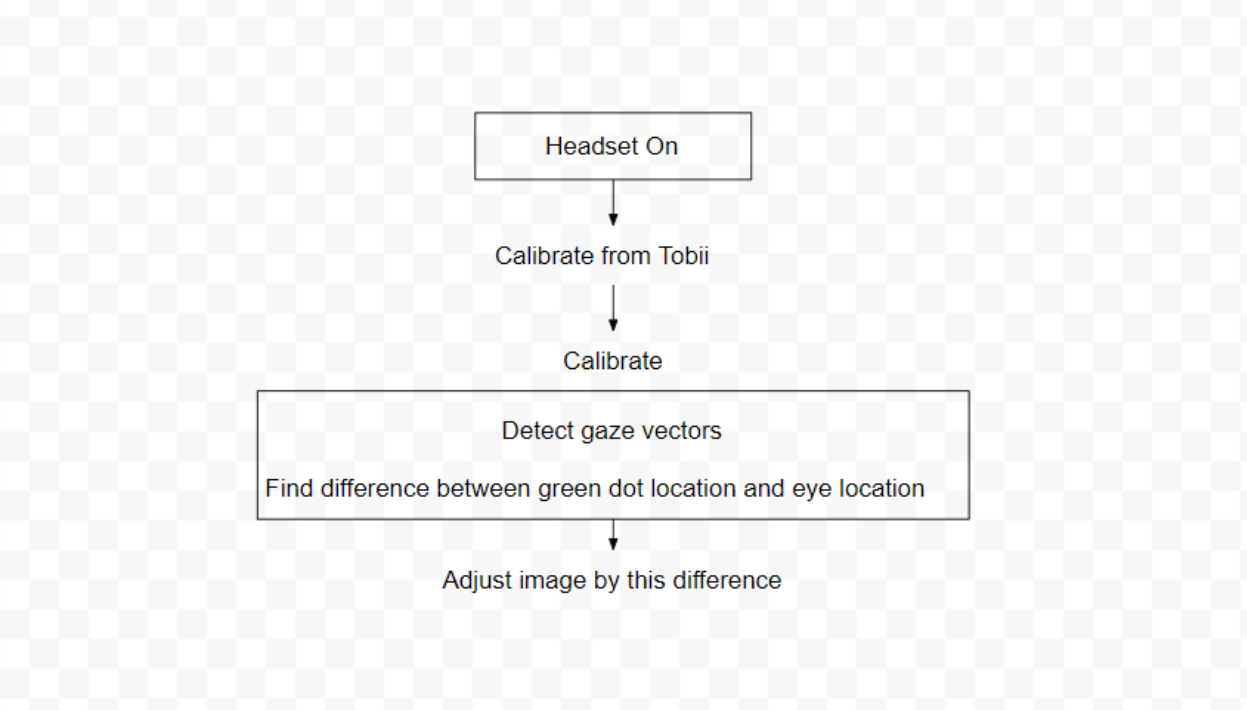
- Bram - got calculation to determine difference between gaze vectors
- Karla - a little bit behind
- Trying to figure things out

Progress:

- Bram - trying to import headset into Unity, create a "child" of it

    - When calibrating FOV create a green dot that goes into the center of each frustrum
        - Match where the eye looks at while calibrating to where the green dots are at, and find difference
    - Has been focusing more on unity, not necessarily api
    - Note: Tobii Eye tracking software
- Wenhao got FOVE working on his computer

3/28/20



RESEARCH SOURCES

Source: SteamVR Platform
URL: https://www.google.com/search?q=steam+vr+platform&rlz=1C1GCEA_enUS850US850&oq=steam+VR+platform&aqs=chrome.0.0.4626j0j7&sourceid=chrome&ie=UTF-8

Source: VIVE Developers
URL: https://developer.vive.com/eu/

Source: Github - VIVE HMD source code
URL: https://github.com/ViveSoftware/VIVE-Wave-SDK-Unreal

Source: Github

URL: https://github.com/AnshulSood11/Position-Tracker

Source: Tobii
URL: https://www.tobii.com/

Source: HTC VIVE eye tracking
URL: https://www.youtube.com/watch?v=wDRAKcKz9rk

Source: Github - VIVE Software
URL: https://github.com/ViveSoftware/ViveInputUtility-Unity

Source: Stack Overflow - SteamVR
URL: https://stackoverflow.com/questions/56786269/unity3d-steamvr-tutorial-vive-pro-source-code

Source: HTC VIVE Pro Eye
URL: https://vr.tobii.com/sdk/products/htc-vive-pro-eye/

Source: Reddit - Foveated Rendering
URL:
https://www.reddit.com/r/oculus/comments/66nfap/made_a_pic_that_shows_how_foveated_rendering/

Source: SteamVR Unity Plugin
URL: https://valvesoftware.github.io/steamvr_unity_plugin/api/index.html

Source: SteamVR Frustum
URL: https://valvesoftware.github.io/steamvr_unity_plugin/api/index.html

Source: Tobii - SDK
URL: https://vr.tobii.com/sdk/develop/unity/documentation/usage-examples/

Source: Tobii - Coordinates
URL: https://developer.tobii.com/community/forums/topic/how-to-get-eye-coordinates-from-htc-vive-pro-eye/

Source: Tobii - API
URL: https://vr.tobii.com/sdk/develop/unity/documentation/api-reference/

Source: Youtube - Unity
URL:
https://www.youtube.com/watch?v=D4I0I3QJAvc&list=PLPV2KyIb3jR5QFsefuO2RlAgWEz6EvVi6&index=7

Source: Github - SteamVR
URL: https://github.com/ValveSoftware/steamvr_unity_plugin/tree/master/Assets/SteamVR

Source: Github - Unity Plugin
URL:
https://github.com/ValveSoftware/steamvr_unity_plugin/blob/master/Assets/SteamVR/Plugins/openvr_api.cs

Source: Google - Eye tracking examples
URL:
https://www.google.com/search?q=how+to+output+gaze+vectors+from+unity+to+steamvr+from+vive+pro+eye&rlz=1C1GCEA_enUS850US850&oq=how+to+output+gaze+vectors+from+unity+to+steamvr+from+vive+pro+eye&aqs=chrome..69i57.46054j0j7&sourceid=chrome&ie=UTF-8

Source: Stack Overflow - Floating point
URL: https://stackoverflow.com/questions/277772/avoid-trailing-zeroes-in-printf

Source: Fresh2refresh - Rounding function
URL: https://fresh2refresh.com/c-programming/c-arithmetic-functions/c-round-function/

Source: VIVE Pro Eye - Command line output examples
URL: https://forum.unity.com/threads/vive-pro-eye-unity-retina-tracking-examples.707000/

Source: Google - Raw gaze vector values
URL:
https://www.google.com/search?q=gaze+vector+raw+values+showing+0.0+unity&rlz=1CDGOYI_enUS863US864&oq=gaze+vector+raw+values+showing+0.0+unity&aqs=chrome..69i57.25404j1j7&hl=en-US&sourceid=chrome-mobile&ie=UTF-8

Source: getfove.com - finding different methods to produce gaze vectors
URL: https://support.getfove.com/hc/en-us/community/posts/360029033073-Unity-how-to-get-raw-x-y-position-of-gaze-as-a-function-of-the-screen-

Source: Tobii
URL: https://developer.tobii.com/pc-gaming/unity-sdk/scripting-api/

UPDATES WITH ADVISOR

February 7th, 2020

Hey Dr. Li,

Here is a brief review of what has gone on this week. Today we presented our proposal to Mark and Andrew. We haven't split off into separate sections, so this will cover what we all have done.

We have completed a PDS that is current to our level of knowledge of the project. We finally got a license for Unity, and plan on getting the old code working soon. We also finished our proposal presentation, and received approval from Brad. We have been looking more into the API of the code for the Vive. We will be receiving the Vive headset February 15th (at least that was Brad's self proclaimed deadline). The headset that we are receiving screen doesn't work, but that's okay because we can see everything we need on the computer.

Once we get the hardware we need a location to store all of our gear, so we don't have to set it up every time we want to use it. Brad mentioned previously that a room in the robotics lab was used. Can you help us set this up?

Thanks,
Capstone Project Odin 2.0

February 26th, 2020

Hey Fu Li,

We missed a weekly update, but there hasn't been too much progress in the development of our project. We have been doing a lot of research with the software we are going to be using and the API, but we haven't received the hardware yet. This Friday (2/28/20) we will be going to Intel to get and demo our new hardware. We have the old hardware working in Unity on our personal machines too. We have a good understanding of what needs to be done to make a significant amount of progress once we have our new hardware. The only potential roadblocks we may encounter are getting the correct drivers and other necessary software to make the hardware work on our work station from Intel.

We have been doing weekly meetings together and weekly meetings with Brad from Intel.

Thanks,
(A)Bram Fouts & Odin II

March 6th, 2020

Hey Fu Li,

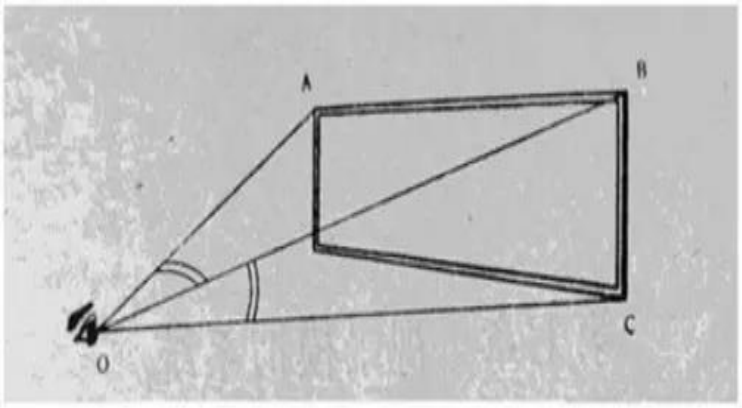
Here's another weekly update on the ODIN 2.0 project. We finally have the new VR gear from intel along with a computer to run it. We are now located in the Special Projects Lab within the Robotics Lab, and we plan on having our team meetings in there from now on (unless another team is in there).

We are starting to separate tasks, but currently we setting up all of the necessary software on our system. We were finally given internet access this afternoon, and we plan on having everything running Monday. We have the VR environment set up inside of our room, and now it's time to get into the nitty-gritty work and build a virtual environment that coincides with our VIVE. We are looking at parenting different objects within unity to track movements of the headset, so we can create our own eye tracking calibration. There are still some unknowns that exist. Whether or not the default Vive tracking takes in consideration that the user may have a lazy eye.

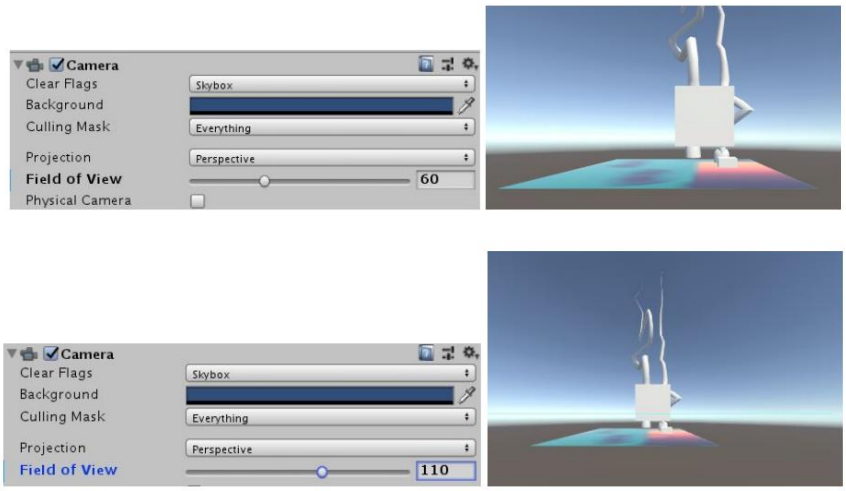
Thanks,
Abram Fouts & ODIN 2.0

March 16th, 2020

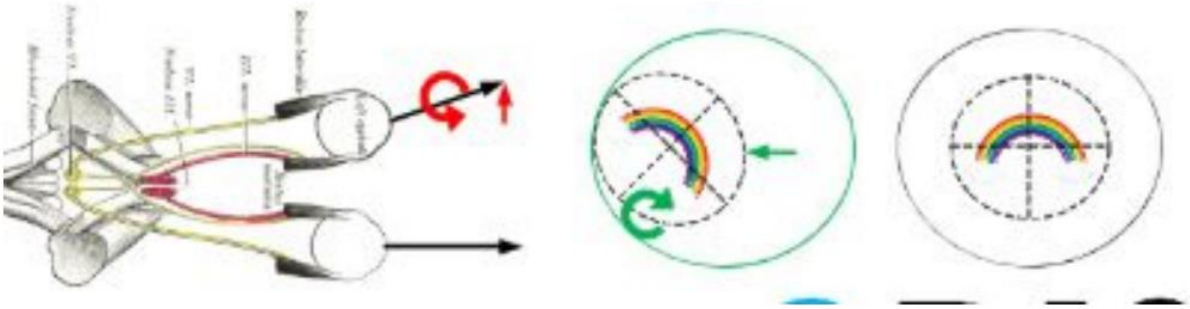
We got the VR gear from Intel along with a computer to run it. We tried to set up VR, and performed a simple operation. It worked pretty well. Moreover, we are going to have our weekly meeting in the robotic lab (70-07). This week, We are starting to separate tasks. Here are the details: Field Of View(FOV): We studied what FOV is. In VR devices, the field of view refers to the field of view in the horizontal direction.



For example, in the figure, the angle AOB is the horizontal field of view. In the real world, a person's normal field of view angle is 110 degrees. When it exceeds 110 degrees, people usually turn their heads to see. We also studied the function of FOV. The size of the field of view directly affects the clarity and immersion of the field of view. Immersive refers to the fact that we are in the VR headset. FOV is embodied in the process of the scene entering the eye through the lens. For instance, if the eye looks at an object through the VR lens, if the design is proper, you can see the entire object, and the size of the field of view corresponds to the height of the character. The highest and lowest rays we can see are refracted into the eye through the lens, and the angle between the two refractive lines is FOV.



When we set the object, we place the object under the camera and set its relative coordinates to 0. This can be used as a reference coordinate. For the amblyopic group, the gaze vector tracked by their eyes is not correct. At this time, we need to analyze the correct gaze vector based on the data(Normal person gaze vector). When we get the correct fixed vector, we need to use the controller to move it to the plane we can render. In this way, the function of correcting amblyopia can be achieved. Now that we understand the basics, we are now looking at Unity API for programming. We are now studying how to connect the controller and the gaze vector.



Bram and Karla are working on the touch controllers and getting them implemented into Unity. We are learning how to have each button perform a specific action, and we are learning how to extract the position and rotation value from the HTC VIVE hardware. Todd is working on extracting the gaze vectors from the TOBII eye tracking software/hardware inside of the VR headset. Each task that the team members are working are our three main deliverables to Intel.

April 24th, 2020

Hi Fu Li,

I realise it's been a while since we updated you on how our project is going, but with everything moving to remote it stinted our development until very recently. Previously we discussed how we tasked everyone with parts of the project. This was near the dead week of Winter quarter. We have only been able to research since then, but we could not have any hands on access to our hardware. Unfortunately we needed hands on access. Up until this week we haven't had any hardware access. Our hardware has been shipped to Brad Smith's house and we are remote accessing it, and he is assisting us in the testing. The deliverables have changed given the state of COVID-19. Our must deliverables have almost been achieved which includes an eye tracking vector hook, and a touch controller hook. We plan to have these complete in the next few days. The part that changed was manipulating the field of view of each eye. Wenhao and Xuening have made good progress, and we are going to assist them in the development of this hook once we finish our assigned tasks. Again, sorry for the long wait in replies, we felt there wasn't enough development happening to prompt an update,

Regards,
Team ODIN II

May 8th, 2020

Hi Fu Li,

Here's a brief update of what's going on in the ODIN II project. We have fully working touch controller input hooks ready for Intel. Todd has been working vigorously on finishing up the gaze vector hooks (retrieving said vectors became a larger challenge than expected), and we believe it's down to debugging our reference code. Since this is an ongoing developing area there are API changes happening FREQUENTLY. From when we first started the project now SteamVR has gone from 2.0 to 2.2 and professional developers are still learning the changed API. A lot of references are to SteamVR 2.0 which limits where we can get assistance/documentation. From what we can tell, we are some of the few (maybe only) attempting to use the gaze vectors detected in the head mounted display.

Karla is debugging some extra touch controller input, and Xeuning and Wenhao are in the process of manipulating the frustums so we can change the FOV per eye in the HMD. This is also a challenge because there is very little documentation on this, and it's very possible it's been outdated with the coming of SteamVR 2.2. On the plus side, our industry advisor Brad seems very pleased with the development we've created given the circumstances of everyone's quarantined lives.
Have a good weekend,
Bram Fouts & ODIN II

May 31st, 2030

Hey Fu Li,

This appears to be the final project update before we present everything to you. I'm going to run down what we accomplished and what we fell short of. We were able to complete the gaze vector and touch controller parts of the project, but we were not able to fully manipulate the

FOV. We were able to adjust rotation of each eye but not the frustums. This was expected and accounted for in scheduling because of the remote access of COVID. We wanted to get live updates on the controller 3D position and rotation, but we failed this too. This wasn't required, but we wanted to attempt to receive all controller data.

We receive each gaze vector from each eye and we are able to create a custom IO function for each controller. We are currently finishing up all of our documentation, slide shows, and log book for our project. We have a lot of useful content for whoever we pass this project onto.

Regards,
Bram Fouts & ODIN II