Beta Squad: VR Expedition

Programmer’s Manual

Michael Chiro, Stefan Farian, Westin Fishel, Patrick Seeman

**Table of Contents**

1. Introduction…..………………………………………….. 2
   1. Description…………………………………. 2
   2. Purpose………………………….…………. 2
2. Installation…..…………………………………………… 2
   1. Hardware Requirements…………………. 2
   2. System Requirements……………………. 2
3. Before You Begin…..………..………………………….. 3
   1. Unity Advice……………………………….. 3
   2. Controller Walkthrough…..……………….. 3
      1. Controller Advice…………………… 4
4. Folder Structure and File Map…..………..……………. 5
   1. Scenes……………..……………..………… 5
   2. Prefabs……………………………………… 5
   3. Scripts………………………………………. 5
5. Future Update Recommendations…………………….. 9

**1. Introduction**

1.1 Description:

VR Expedition allows you to take a Virtual Reality (VR) tour with videos the user imports into this 360 video viewer application. With a VR headset and wands, users can effectively customize the tour interface by placing videos in choice locations. Alternatively, keyboard/mouse input also works with the functionality of this application.

The finished project has two modes for the user: administrator mode and tourist mode. Administrator mode allows project managers to build the custom tour with user input, and tourist mode implements user input to follow the tour that was built for them.

1.2 Purpose:

This application is intended for project managers with multiple related omnidirectional (360) videos. A finished tour can serve as an alternative to being present where the video/tour was filmed. Also, this application can be used for marketing a trip or tour to prospective individuals. The motivation comes from an Associate Political Science Professor at John Carroll University who wanted to market his department’s immersion trips, particularly to Honduras.

**2. Installation**

2.1 Hardware Requirements

1. CPU: Intel i5-4590 equivalent or greater
2. GPU: NVIDIA GTX 970 / AMD R9 290 equivalent or greater
3. OS Windows 7 SP1 64 bit or newer
4. RAM: 8GB+
5. USB Ports 3x USB 3.0 ports plus 1x USB 2.0 port
6. Video Output Compatible HDMI 1.3 video output
7. VR Hardware: **​​**Valve Index, or HTC Vive, Oculus Rift, or Windows Mixed Reality

2.2 System Requirements (and download links if applicable)

1. Unity version 2020.3.19f1**;** download link:

<https://unity3d.com/get-unity/download>

1. Steam; download link:

<https://store.steampowered.com/about/>

1. SteamVR; download link:

<https://store.steampowered.com/app/250820/SteamVR/>

1. Folder of videos to be used in the VR Expedition.

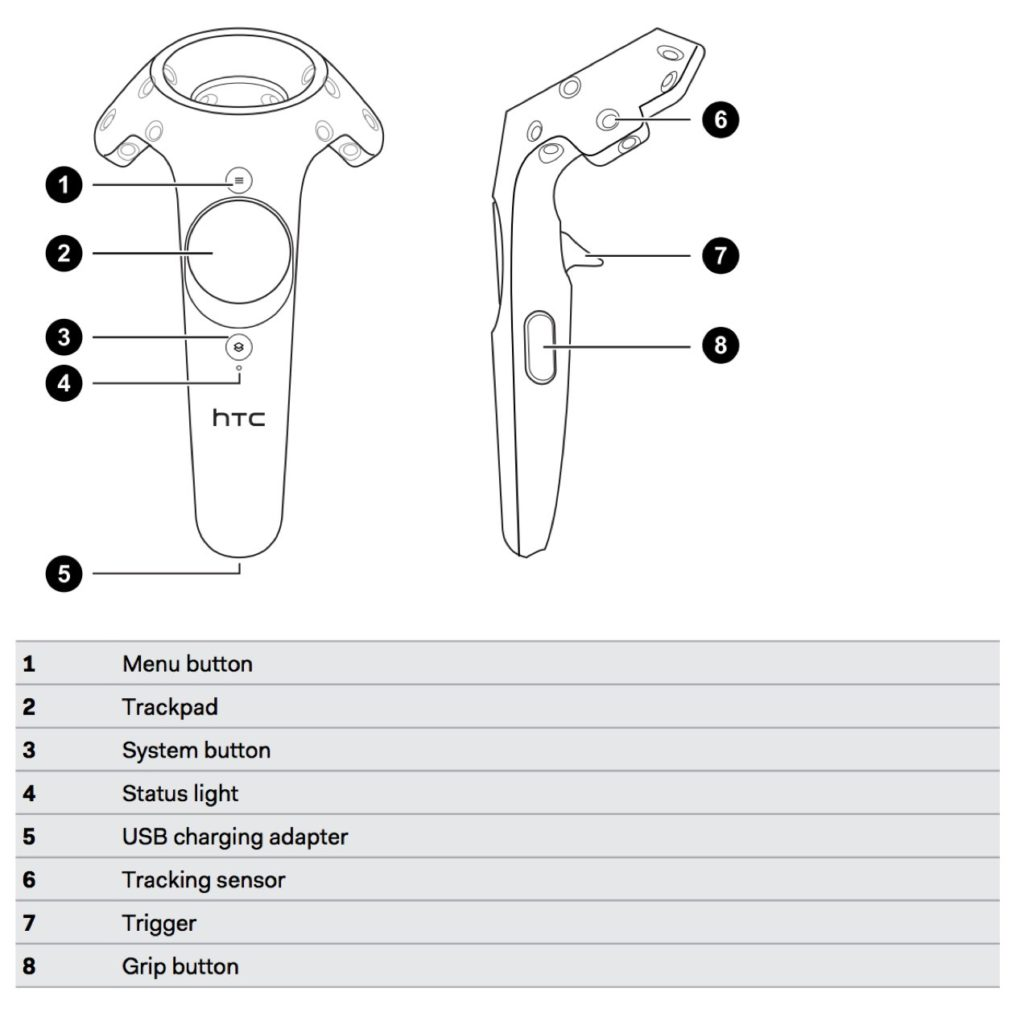
**3. Before You Begin**

3.1 Unity Advice

1. Start your code at the most basic level, then build it up line by line. Troubleshooting code is easier when the difference between functional and nonfunctional code is the line or two you just added.
2. Unity documentation is a great starting spot for troubleshooting. When building scripts, it is easier to find solutions if you can reference the documentation as you code. For example, if you forget what a raycast is, you have a quick reference to the information. Also, if you Google a class built into Unity or a method, it will usually show up as the first result in Google.
   1. <https://docs.unity3d.com/Manual/index.html>
3. Unity forums is another reliable resource for help building what you want. Unity is designed to handle a lot of logic like collision detection, rendering layers, onclick listeners for buttons, and more. Someone out there probably has experienced a similar issue to you.
   1. <https://forum.unity.com/>
4. When you are out of ideas on fixing something, try the opposite of what you thought would work. For example, switching a handful of ‘and’ statements to ‘or’ statements in the code has achieved desired outcomes.
5. Unity is a colossal software with many tools. Get familiar with the basics such as game objects before fully diving into it.
6. Unity scripts are written in C#, so become familiar with it because even though the coding may not be as strenuous as another type of project, there is still plenty of coding to be done
   1. <https://unity.com/how-to/learning-c-sharp-unity-beginners>
7. Make ample use of the folder system in Unity to keep all your assets and code organized - it doesn’t take a lot of work and is well worth the effort.

3.2 Controller Walkthrough

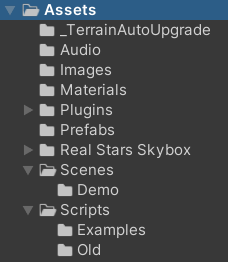
1. Keyboard Inputs (2D functionality):
   1. Arrow keys: rotate the camera to view the video as if you are turning your head with a headset.
   2. A, W, S, D buttons: same functionality as the arrow keys. These buttons serve as an alternate if any arrow keys are broken, or user preference.
   3. R button: resets camera’s position to the default angle.
   4. Escape button: takes the user back to the main menu of the application.
   5. Q button: quits the application entirely.
2. HTC Vive Controllers (3D functionality):
   1. The image below shows the HTC Vive wands as well as the different components of them.



3.2.1 Controller/Headset Advice

1. Make sure your hardware is actually properly plugged in.
2. If your headset requires spatial anchors, make sure they are in their designated spots

**4. Folder Structure and File Map**



Here we discuss the important “Assets” contained within this program: Scenes, Prefabs, and Scripts. They can be accessed in Unity through the file tree shown above.

4.1 Scenes

1. **GameSpace**
   1. This is where the tour takes place. Videos are displayed in the skybox on a loop as the user interacts with the VideochangeInteractables
2. **Menu**
   1. The main menu is used to load videos and navigate to the other scenes
3. **PrefabSetup**
   1. VideoChangeInteractable prefabs can be placed and saved here to create tours

4.2 Prefabs

1. **ColliderButton**
   1. A button with a 3D collider component so that we can detect collisions like a normal gameobject
2. **FileDisplay**
   1. A text prefab for when we display file names and file sizes on the main menu
3. **VideoChangeInteractable**
   1. A sphere collider that runs the Video Change script when clicked. Clicking on these is how you navigate to new videos on a tour

4.3 Scripts

1. **ArrowInput.cs** - Handles keyboard arrow input to rotate camera and is most functional when the headset is not available.
   1. “Update” method
      1. Runs once per frame. Houses loops and statements listening for a button press
2. **CreateInteractable.cs** - This script is used in the Interactable Setup scene and uses mouse/controller click to create a new VideoChangeInteractable prefab.
   1. “Start” method
      1. Make the camera variable
   2. “instantiateAtPoint” method
      1. Makes a video change interactable when you click with the controller
3. **File\_Explorer.cs** - This script opens a file explorer when the button “Click to Load Videos” is clicked. Once you navigate to a folder that has mp4 files, you click load and the files are then displayed on the canvas.
   1. “Browse” method
      1. Browses the file directory
   2. “displayToCanvas” method
      1. Fixes the path if the user selects a file at the end of the path instead of just a folder
      2. Gets the directory data (all files in directory) and the file attributes (name and length), checks for the file being an mp4, and converts the size to megabytes
   3. “DisplayVideos” method
      1. Displays the file names for the first 3 videos on the main menu
   4. “Update” method
      1. Runs the file names in a loop which displays 3 at a time
      2. On button click or “n” key press, it will cycle through the file names from the folder
   5. IEnumerator “ShowLoadDialogCoroutine”
      1. Only shows mp4 files and folders
      2. Shows a load file dialog and waits for a response from user
      3. Loads both file and folder
      4. The initial path is default (Documents) and the initial filename is empty
      5. Title is “Load File” and the submit button text is “Load”
      6. Handles if the file browser has been closed
4. **LaserHandler.cs** - Spawns, loads, and saves interactables, and handles setup for interactables and tours. It also handles controller input collision detection.
   1. “Awake” Method
      1. Call this script's pointer click method after SteamVR\_LaserPointer's pointer click
      2. If in the gamespace, it loads the files at our directory, and then append all those names to a list
      3. Finds the skybox player in the scene, then plays the first video in our list
      4. Then loads the corresponding video's json file which contains the locations of the prefabs
      5. If we're in prefab setup scene, it will load the files at our directory then append all those names to a list
      6. If a json file does not exist for a video, creates an empty json file
      7. Finds the skybox player in the scene, then plays the first video in our list. Then loads the corresponding video's json file which contains the locations of the prefabs
   2. “Update” method
      1. Using either hand controller, if we're in prefab setup, check for controller input
      2. If the controller isn't hitting anything, spawn a prefab and if the controller is hitting an interactable, delete that interactable
      3. If you left click your mouse, it only destroys the object after this frame, which is why saveOnNextFrame is called above this if statement
      4. If the escape key is pressed, it exits to the main menu
   3. “PointerClick” method
      1. Handles button handling
      2. Built-in method that appends to SteamVR\_LaserPointer in StreamVR extras
      3. If the gameObject hit is a video change interactable, activate the video change
      4. Handles the dropdown menu before it is expanded (not shown)
      5. Adds colliders to UI elements so that the laser pointer can hit them
      6. Handles the dropdown menu after it is expanded (shown)
      7. The object structure for json serialization used later on
   4. “instantiateSetupPrefab” method
      1. Instantiates a prefab at a given point
   5. “nextSetupVideo” method
      1. This is used in the prefab setup scene
      2. If there is a next video, it changes the skybox to the next video and load corresponding prefabs
   6. “prevSetupVideo” method
      1. Used in the prefab setup scene
      2. If there is a previous video, it changes the skybox to the previous video and loads the corresponding prefabs
   7. “saveLocations” method
      1. Save json file of locations list
         1. Filename is the path to the json file, which is just the video name concatenated with .json
         2. Parses json of vectors at a given filename
         3. Reads in json string from file
         4. This is the parsed json string that we will turn into an Interactable struct
         5. Tracks if we've parse a json object
         6. Parses objects from json string
         7. Instantiate a new prefab at the points in the list, with each corresponding newClip string
5. **OnClickQuit.cs** - Quits the application
   1. “Start” method
      1. References the current gameObject script is attached to, finds the button component, then adds a listener for a click
   2. “quitGame” method
      1. Quits the application
6. **SceneChanger.cs** - Changes the scene on a button click
   1. “mainMenu” method
      1. Assign these methods to a button which loads the scene specified
7. **SteamVR\_LaserPointer** - Script automatically implemented once the programmer has downloaded SteamVR
8. **VideoChange.cs** - Changes the video
   1. “Awake” method
      1. Finds the skybox player object in the scene
   2. “Activate” method
      1. Changes the video being played to a variable called “newClip” which is assigned via JSON data during the prefab setup
      2. Finds and destroys interactables to create a clean slate for the next video and interactables to play

**5. Future Update Recommendations**

1. The main menu has four modes/options in the form of buttons to use the software: Start Tour, Select Video Folder, Tour Setup, and Quit. Future programmers should consider updating/adding functionality and system requirements from that starting spot.
2. For 2-d functionality, or ArrowInput.cs, make the speed of camera rotation a customizable feature, or make rotation speed a function of time: the longer the button is held down, the faster it goes (to an extent). Make every main function customizable to their button choice: like quitting, going to the main menu, going to the next scene, etc.
3. Since the videos are a fixed length, consider adding an automatic progression after so many loops through the video. This could be nice for users looking to take a backseat and let the tour run itself.