Overview

This assignment will demonstrate the difference between rotation and position tracking. It will help you appreciate how much depth perception relies on both of these, and teach you how to enable and disable them. It’ll also help you understand the rigid body transformations covered in class.

We have provided you with a .unitypackage containing a scene (‘Assignment 2’). The provided scene contains an OVRCameraRig prefab, which doesn’t allow player movement but still uses DK2’s rotation and position trackers to modify camera position and rotation. The prefab contains scripts (OVRCameraRig & OVRManager) and GameObjects that you will extend in the following ways:

NOTE: For those that are implementing this assignment using Gear VR, position tracking is not supported. Thus there is no need to disable position tracking. However, since there is no keyboard input for the Gear VR, you will have to utilize the haptic controller of the device as triggers for the tasks described below. These are unchartered waters; you will help push the frontiers of this course.

Part 1 VR Mirror

This part of the assignment will familiarize you with manipulating a GameObject’s rotation and position. Tasks that need to be completed in this part include:

- Pressing tab should reset the camera position to (0,0,0).
- Pressing the F key should flip the user 180 degrees so that he is looking behind where he was looking.
- Pressing the M key should make a certain cube either mirror or follow the user’s movements.

Create a script named CameraFlipper that flips the OVRCameraRig 180 degrees when you press the F key, like you were turning around to look behind you. Now you should be facing a transparent window with a floating cube face on the other side. Think of this cube as your head. You should be able to press the F key multiple times to repeat this operation. Also include the reset camera position function in the CameraFlipper script. You should not
change the initial position of the OVRCameraRig. But rather reset the camera position during runtime.

Write a script VRMirror to modify the position and rotation of this cube to match or mirror your own. Pressing the M key should make the cube match your movements. This includes positional and rotational movements. Furthermore, note that this also means that the cube should be looking in the same direction the camera is (and as result, the user should see the back of the cube’s head). For example, if you bring your face closer to the screen, the cube moves further away from the screen.

Pressing the M key again should make the cube mirror your movements (positional and rotational). This means that the cube is facing the camera (and as a result, the user should see the cube’s face). In this case, if you bring your face close to the screen, the cube moves closer to the screen as well. Imagine looking into a mirror to get an intuition of this.

Part 2 Disabling Position and Rotation Tracking

Now you will figure out a way to disable position and rotation tracking. Create a script named ToggleTracking that turns tracking on and off. Tasks that need to be completed in this part include:

- Pressing the R key should toggle rotation tracking on and off.
- Pressing the P key should toggle position tracking on and off.

Temporarily losing rotation and position tracking in this Assignment should help you understand how much they contribute towards presence in VR.

Hint: Think about how to reverse/counteract the effects of position and rotation tracking.

Part 3 Depth Perception and Relative Size

In your scene, there are three spherical gameobjects: Small, Medium, and Large that are children of a the parent gameobject called StimulusManager. Your task is to write GenerateStimuli, a script that arranges these three gameobjects so that they appear to be of equal sizes when viewed from camera position (0,0,0) with rotation tracking and position tracking disabled. Tasks that need to be completed in this part include:

- Pressing the S key should make gameobjects disappear and reappear.
This script should activate when the user presses the S key. On activation, the script should do the following:

1. Make the 3 objects disappear.
2. Randomly assign the 3 stimuli to be in the left, center, or right position (from the perspective of the camera). There should be one stimulus in each position.
3. Determine where to place the stimuli so that they all appear to be of the same size from the camera’s perspective (at position (0,0,0)).
4. Wait 3 seconds and make the left stimulus appear.
5. Wait a further 3 seconds and make the center stimulus appear.
6. Wait a further 3 seconds and make the right stimulus appear.

Pressing the S key again should start this process (1-6) over again.

**Note on Point 3**

Determining these positions is very non-trivial. Thus we’re only going to give half credit if you precompute these positions and bake them into your submission. There are some really interesting ways to compute these positions on the fly (not baked / hard coded). For full credit, the positions must be computed dynamically, meaning the stimuli would appear to be of the same size regardless of the camera’s position and rotation.

In case of ambiguity, here’s what the spheres should look like when they “appear to be of the same size from the camera’s perspective”.

![Diagram of spheres](image.png)
Part 4 Written assignment

Complete the following problems. Only electronic submissions are accepted.

1. In one sentence, explain what the following homogeneous transformation accomplishes when applied to a point \((x, y, z)\), in terms of yaw, pitch, row, and translation.

\[
T_1 = \begin{pmatrix}
\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & -1 \\
\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 2 \\
\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
z \\
1 \\
\end{pmatrix}
\]

2. Write out the 4x4 homogeneous transformation \(T_2\), when applied to a point \((x, y, z)\) in global coordinate frame, translates the point by \((3,0,2)^T\), then followed by a pitch of 45 degrees. Your answer need not be simplified, and may be represented as a single matrix or the product of two or more matrices.

3. We would like to reverse the transformation applied by \(T_2 T_1\), that is, write out \((T_2 T_1)^{-1}\). Your answer need not be simplified, and may be represented as a single matrix or the product of two or more matrices.

4. Write out the Quaternion equivalent to the rotations in \(T_1\) and \(T_2\) as \(q_1\) and \(q_2\). Then calculate the product, that is, \(q_1 \circ q_2\) (Hint: Steve’s book may be a good source of reference).
Rubric

<table>
<thead>
<tr>
<th>Criteria Name</th>
<th>Points</th>
<th>Description for full credit</th>
</tr>
</thead>
<tbody>
<tr>
<td>VR Mirror 1</td>
<td>10%</td>
<td>Pressing tab resets the camera position to (0,0,0). Pressing F flips the camera 180 degrees.</td>
</tr>
<tr>
<td>VR Mirror 2</td>
<td>10%</td>
<td>Cube correctly matches your movements.</td>
</tr>
<tr>
<td>VR Mirror 3</td>
<td>10%</td>
<td>Cube correctly mirrors your movements.</td>
</tr>
<tr>
<td>VR Mirror 4</td>
<td>5%</td>
<td>Pressing the M key toggles the cube between mirroring and matching your movements.</td>
</tr>
<tr>
<td>Disable Tracking 1</td>
<td>20%</td>
<td>Pressing the R key toggles rotation tracking on and off.</td>
</tr>
<tr>
<td>Disable Tracking 2</td>
<td>5%</td>
<td>Pressing the P key toggles position tracking on and off.</td>
</tr>
<tr>
<td>Depth perception 1</td>
<td>10%</td>
<td>Objects appear in the correct order, following the correct timing. The script can be run repeatedly</td>
</tr>
<tr>
<td>Depth perception 2</td>
<td>10%</td>
<td>Objects appear to be of equal size once the script has been run. (not precomputed)</td>
</tr>
<tr>
<td>Written Assignment</td>
<td>20%</td>
<td>All computations are correct; show work.</td>
</tr>
</tbody>
</table>

Submitting

Follow the steps described below for submission.

Additionally, if you’re attempting the Extra Credit task, make sure you explicitly mention that in your README.txt and explain your approach.

**NOTE: Assignments must be completed in groups of 2.**

**Step 1: Create a .unitypackage file**

1) Save your Unity scene in the Assets folder with the title “CS498HW2”

2) Using the editor, find the created scene in the Project menu
3) Right click on the scene and select Export Package...

4) Export the file using default settings (“Include dependencies” should be checked by default)

**Step 2: Create a standalone game build**

1) Go to Edit → Project Settings → Player. Make sure the “Virtual Reality Supported” box under Other Settings is checked.

2) Go to File → Build Settings

3) Click “Add Current”. This will add the current scene to the build. You must have saved the scene to the Assets folder for this to work (you should do that anyways).

4) Hit “Build”. Save the project to C:\Users\student’s netid\project name, rather than your networked folder.

5) This should create an executable (.exe) for running the build, as well as a folder containing your scene data. Make sure this executable runs correctly on the Rift before submitting.

**Step 3: Create a PDF file for the written assignment**

1) Write out the written assignment using LaTeX, word or editor of your preference.

2) Export the file to PDF format. Name the file in the format of “hw2.pdf”

3) Only one written assignment needs to be submitted for each group

**Step 4: Zip the files and submit them through Compass**

1) Create a zip file containing 3 items:
   a) The .unitypackage created in Step 1
   b) The .exe and data folder created in Step 2
   c) The .pdf file created in Step 3
   d) A README.txt file containing any special instructions or notes you think are relevant for evaluating your assignment.

2) Name the file by separating NetIDs with underscores._cs498sl_HW2.zip EXAMPLE: If john1 and carmack2 worked together, the file should be called john1_carmack2_cs498sl_HW2.zip
DO NOT SUBMIT YOUR ENTIRE PROJECT FOLDER