

Assignment 2 - Explore Your World

Virtual navigation is a key interaction when the virtual environment is larger than the available space in the real world. This second assignment is split into a theoretical and a practical part. In the theory part you will learn about the advantages and drawbacks of different navigation techniques and deepen your understanding of how they relate to a user's scene graph nodes. In the practical part you apply your conceptual knowledge of scene graph transformations and state handling to implement and evaluate some fundamental navigation techniques.

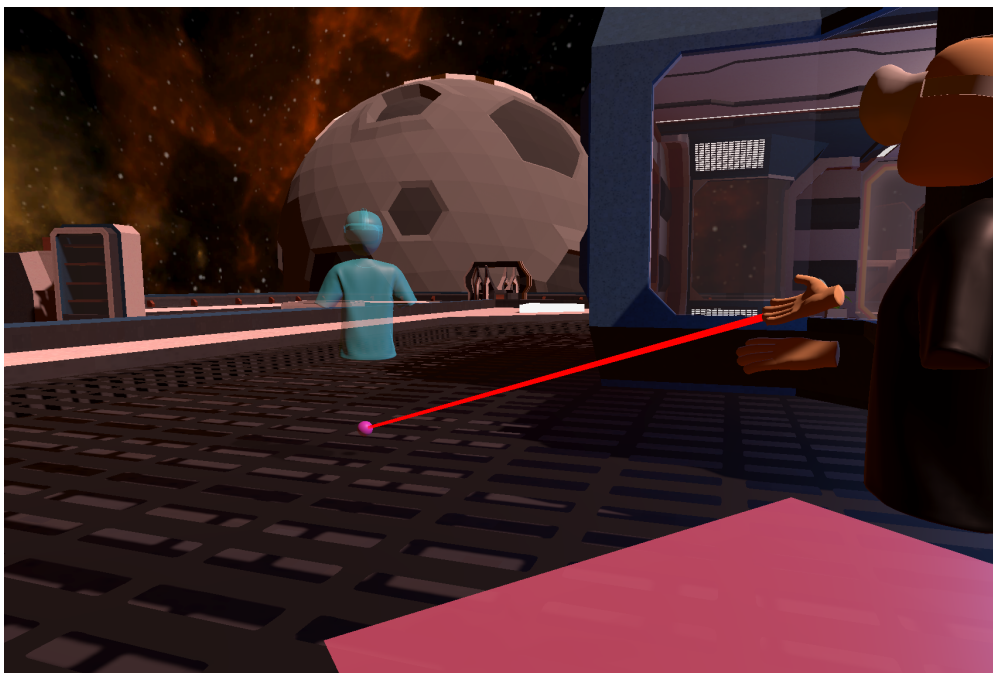




Figure 1: A snapshot from the completed assignment Unity project.

Submission and General Remarks

You are required to submit this assignment by **7th Dec 2023 23:55** on Moodle. Furthermore, you will be asked to present and discuss your results in the lab class on **8th Dec 2023** to one of the teaching assistants. Your submission should consist of one **.zip**-archive named **<your-group-name>.zip**. The archive should contain a clean Unity project with your solutions for the tasks explained in this assignment sheet. Make sure to clean out all temporary files before zipping. Relevant folders are **Assets**, **Packages**, **ProjectSettings**, **UserSettings** (see Figure 2).



Name	Änderungsdatum	Typ	Größe
.vs	13.10.2022 10:44	Dateiordner	
Assets	13.10.2022 10:37	Dateiordner	
Library	13.10.2022 19:15	Dateiordner	
Logs	13.10.2022 10:37	Dateiordner	
obj	13.10.2022 10:44	Dateiordner	
Packages	13.10.2022 10:33	Dateiordner	
ProjectSettings	13.10.2022 10:43	Dateiordner	
UserSettings	13.10.2022 10:37	Dateiordner	
.vsconfig	13.10.2022 10:34	VSCONFIG-Datei	1 KB
Assembly-CSharp.csproj	13.10.2022 10:49	C# Project file	49 KB
vr-lab-class-2022-build-your-world.sln	13.10.2022 10:37	Visual Studio Solution	1 KB



Name	Änderungsdatum	Typ	Größe
Assets	13.10.2022 10:37	Dateiordner	
Packages	13.10.2022 10:33	Dateiordner	
ProjectSettings	13.10.2022 10:43	Dateiordner	
UserSettings	13.10.2022 10:37	Dateiordner	

Figure 2: Top: A Unity project after generating temporary files (not ready to be zipped as submission). Bottom: A clean Unity project before temporary files are generated (ready to be zipped as submission).

Over the course of all assignments you will be guided through a series of tasks that will teach you how to build social VR experiences. Due to the vast scope of challenges involved when creating multi-user VR applications, we will only focus on an essential subset of VR techniques. We want to encourage you to expand your journey into VR and explore beyond the tasks that we provide. At all times, feel free to reach out for our feedback and guidance, so that we may help to expand your skills!

Please note: If not mentioned otherwise, Unity Asset Packages supplied to you as part of the assignments are strictly for course use only and they may not be shared or reused in projects unrelated to the VR Lecture. Using Asset Store packages for any of the coding related tasks is not allowed and will lead to a 0% assessment for the affected tasks.

Assessment

Each task in this assignment shows a percentage in brackets. The sum of task percentages add up to 100 (excluding bonus tasks) within each assignment. Consequently, you may treat task percentages both as a maximum score per task as well as an assignment progress indicator. The contribution of each assignment towards your overall lab class grade depends on its weight, which are as follows:

- ▶ Assignment 1: 20%
- ▶ **Assignment 2: 20%**
- ▶ Assignment 3: 20%
- ▶ Assignment 4: 40%

Getting Started

Download the asset package `vrsys-2023-labpack-assignment-2.unitypackage`. It contains the required assets and scripts that you will use to complete the tasks in this assignment. Alternatively, you can download the project `vrsys-2023-lab-assignment-2.zip` which contains the solutions to assignment 1 and the required assets and scripts for assignment 2. The project was built and tested using Unity Editor 2021.3.26f and 2021.3.32f.

By now you should have received a *Meta Quest HMD* (Quest 2, Quest Pro or Quest 3 - depending on state of inventory) - if not please reach out to us or the student assistants.

We encourage you to use git as a version control system for your lab assignments (see <https://git-scm.com/downloads>). For convenience, we supply a `.gitignore`-file within the archive mentioned above. It is setup to ignore all temporary files generated by Unity, when versioning your progress.

If you work from the VR lab, the aforementioned github project, Unity version and Oculus desktop app will already be available on your account. If you work from home, please install *Unity HUB* (see <https://unity3d.com/get-unity/download>) and add Editor version 2021.3.26f (see <https://unity3d.com/get-unity/download/archive>).

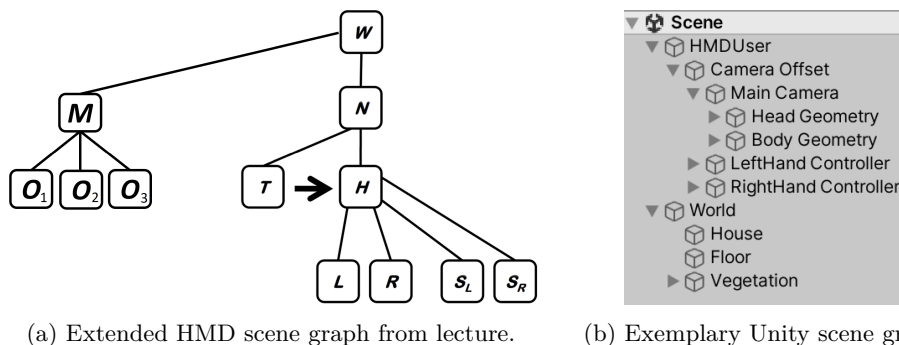
Part I: Theory

Bring your answers and illustrations for the theory part in a suitable format (written notes, pdf, etc.) to the presentation of Assignment 2 (8th Dec 2023).

User Representation in the Scene Graph

Exercise 2.1 (15%)

Scene graphs are data structures that arrange the semantic and often spatial representation of a graphical scene. Slide 38 of *Lecture 3 - Stereoscopic Viewing Setups* shows the scene graph of an HMD user. In Figure 3a we extended the lecture scene graph with a branch for models (M).



(a) Extended HMD scene graph from lecture. (b) Exemplary Unity scene graph.

Figure 3: Images of a theoretical lecture scene graph and a Unity scene graph.

For this exercise investigate and compare the scene graphs shown in Figure 3a and 3b. Answer the following questions:

- What do the nodes **L**, **R**, **S_L**, **S_R** in Figure 3a stand for?
Note: Find help in *Lecture 3 - Stereoscopic Viewing Setups*.
- Which nodes exist in both scene graphs and correspond to each other? Label corresponding nodes that exist in both scene graphs and discuss differences.
Note: Not all nodes exist in both scene graphs.
- Re-draw the left scene graph (Fig. 3a) with the missing nodes from the Unity scene graph (Fig. 3b). Ignore the **T** node in your scene graph illustration.

Exercise 2.2 (10%)

Different scene graph representations show or omit different node types. Unity separates between *GameObjects* and *Components*. Answer the following two questions:

- What is the difference between *GameObjects* and *Components*.
- Inspect the *HMDUserPrefab - Assignment2* prefab in Unity. Which Unity *Component* does the **T** node from the lecture scene graph correspond to?

User Navigation Theory in the Scene Graph

Exercise 2.3 (10%)

The previous exercises showed that a VR user is represented through several nodes in the scene graph. Understanding the relation between these nodes is important for implementing efficient and comprehensive navigation techniques.

Bowman et al. (2001)¹ subdivides navigation into different categories such as *physical movement*, *steering* and *target-based travel*. The different navigation approaches are often also applied to different scene graph nodes. This task aims at sharpening the awareness of node relations and properties. Follow the instructions below to observe and learn about the difference between physical walking and steering/teleportation.

- ▶ Open your last assignment project in Unity and run it as a HMD user.
- ▶ While your application is running, switch to the scene view and observe the user.
- ▶ Move around physically with the HMD and observe the position and rotation of the Transform component of the *HMDUserPrefab* and the *Main Camera* nodes.
- ▶ Manually change the position and rotation of the *HMDUserPrefab* and the *Main Camera* nodes in the editor.

Based on your observations, **answer** the following questions:

- ▶ How does physical movement affect the position or rotation of the *HMDUserPrefab* and the *Main Camera*?
- ▶ What happens if you manually change the position or rotation of the *HMDUserPrefab* in the Unity editor?
- ▶ What happens if you manually change the position or rotation of the *Main Camera*?
- ▶ Which node should be transformed when implementing a Steering/Teleport navigation technique?

Exercise 2.4 (5%)

Draw a scene graph with two HMD users that can move independently of each other and node for further models. Use the scene graph representation of the lecture also shown in Figure 3a.

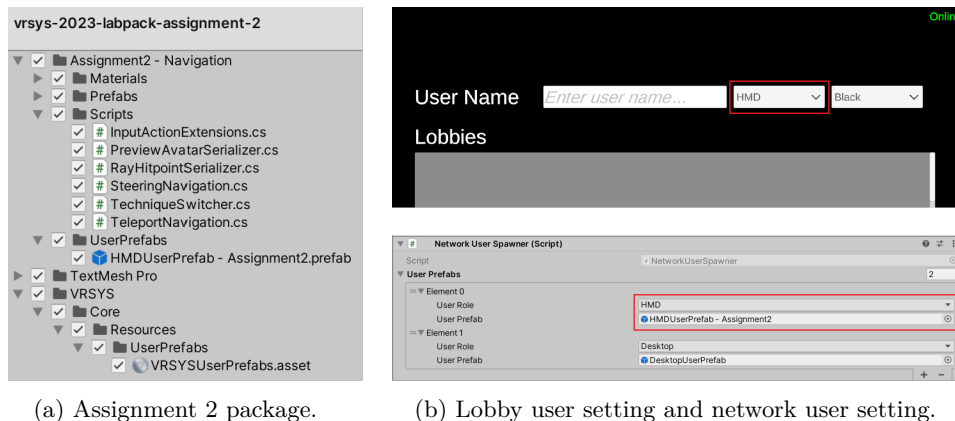
¹D. A. Bowman, E. Kruijff, J. J. LaViola, and I. Poupyrev. An introduction to 3-d user interface design. *Presence*, 10(1):96–108, 2001

Part II: Practice

Upload and submit the practical part of this assignment via *moodle* as described in the section Submission and General Remarks.

Setup

Exercise 2.5 (5%)



(a) Assignment 2 package.

(b) Lobby user setting and network user setting.

Figure 4: Images of a theoretical lecture scene graph and a Unity scene graph.

Import the asset package (`vrsys-2023-labpack-assignment-2.unitypackage`) that comes with this assignment into the last project or download our new project for assignment 2 (`vrsys-2023-lab-assignment-2.zip`). This assignment comes with an adapted `HMDUserPrefab` called `HMDUserPrefab - Assignment2` (see Figure 4a). This setup task explains step-by-step how to change the user prefab that VRSYS instantiates for us.

- Find the GameObject **VRSYS-Networking** in the hierarchy of the assignment scene and look at the *NetworkUserSpawner* component.
- The *NetworkUserSpawner* component is responsible for the instantiation of users in the distributed application. Put the new **HMDUserPrefab - Assignment2** of this assignment in the *User Prefab* field of the HMD user role (as shown in Figure 4b).
- Run the application as a HMD user. The right hand should show a menu with the current navigation technique. Time to implement a steering and teleport navigation in the next exercises. Pressing the thumb stick should toggle between *steering* and *teleport* navigation.

Steering Navigation

Exercise 2.6 (10%)

The **HMDUserPrefab - Assignment2** prefab contains the **SteeringNavigation.cs** script. Your task is to implement a simple steering navigation that moves your *navigation origin* along the forward direction of your hand.

- Open the **SteeringNavigation.cs** and adapt the Update function to change the position of the *navigation origin* according to the input each frame.
- The movement speed depends on the pressed state of the trigger. The continuous input value of trigger should be continuously mapped onto the movement speed. Consequently, slightly pressing the trigger results in a slower movement than when fully pressing the trigger.
- The movement speed needs to be frame rate independent.
- The movement direction depends on the hand forward direction.

Exercise 2.7 (5% optional)

After completing the previous exercise: How can you enable Gaze Directed Steering without adapting the code of the **SteeringNavigation.cs** script.

Teleport Navigation

Exercise 2.8 (30%)

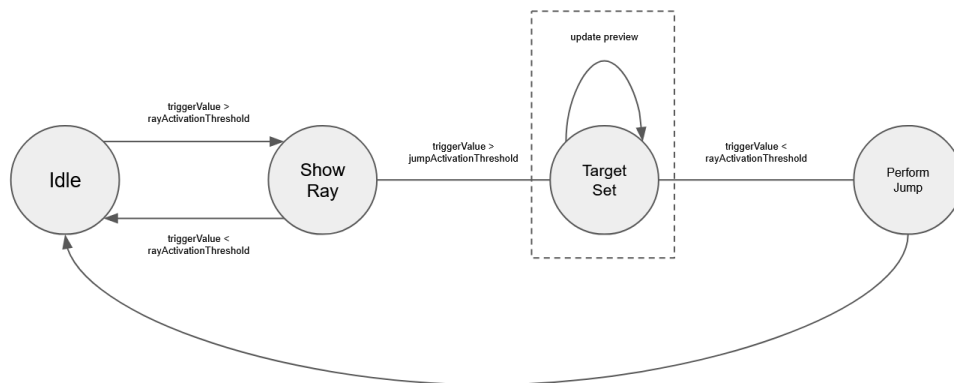


Figure 5: State flow of the teleport navigation technique.

Steering navigation can lead to cyber sickness. A common alternative to steering is the target-based travel also known as "Teleportation". In this task we want you to implement a ray-based teleport technique. Your teleport needs to support a target and

direction selection with a straight ray. It should follow the state diagram shown in Figure 5. We supply a video of the teleport technique that shows the expected result and behaviour on moodle.

When slightly pressing the trigger button (index finger) of your Oculus controller, a straight ray should emanate from your virtual hand. The intersection point of this ray with the scene is computed, visualized and updated every frame. When fully pressing the trigger button, the user sets their jumping target to the current intersection point. A preview avatar of the user's future pose should appear at the target position. While the trigger button is fully pressed the direction of the preview avatar is updated every frame to face the latest intersection point (*Note: only the preview avatar's orientation should change - the position stays the same*).

Releasing the trigger button, instantly teleports the user to the set target position. The user's orientation is changed to face the selected direction. All indicators disappear after the jump. The class *TeleportNavigation.cs* already implements a ray whose position and length is updated every frame and only shows when the trigger button is touched. Extend the this script with the described functionality to complete this task. The following aspects have to be realized for full grading:

- ▶ Calculate and update an intersection point when the trigger is slightly pressed. We already added a hit point geometry that can be updated.
 - ▶ When the trigger is fully pressed, set the current intersection point as your teleport target location. Show a semi-transparent preview avatar of the user at the teleport target location.
 - ▶ While the trigger is fully pressed update the avatars direction to face the current intersection.
 - ▶ The displayed preview avatar should have the same height as the user.
 - ▶ Teleport the user to the indicated position when the trigger button is released.
 - ▶ Account for the offset between user head and user platform when teleporting.
- Attention:** It is not enough to only move the user's origin. Make sure that the user's position is exactly where their preview avatar was shown.
- ▶ Make sure the user faces the set direction after the teleport.

Note: Be aware of the scene graph structure of the user. We want the teleport destination and orientation to be accurate. To test your implementation we recommend to switch your Oculus from *Stationary Mode* to *Roomscale Mode*. Even if your camera is not aligned with the center of your XR Origin you should end up with the selected position and orientation. For your help we added the *DebugNavigationPlatform* GameObject to the prefab origin. Observing the relation of platform and user should help

Teleport Preview Avatar Distribution

Exercise 2.9 (15%)

Distributing the states of users' actions to other users is important for a shared virtual surrounding. Unfortunately, the actions of others are not distributed automatically in Unity. We already added a `RayHitpointSerializer.cs` component for the distribution of the teleport ray to the user. You can find this component attached to the **TeleportPreviewObjects** GameObject in the user prefabs hierarchy. However, the preview avatar is not distributed yet.

- Read about **NetworkVariables** in the Unity documentation.
- Review the `RayHitpointSerializer.cs` script on the **TeleportPreviewObjects** GameObject and explain the if-case with **IsOwner** and **!IsOwner**.
- Implement the serialization of the *PreviewAvatar* into the `PreviewAvatarSerializer.cs` script.
- Ensure that the exact position and orientation of the *PreviewAvatar* is visible for other clients when the HMD has set the target.

Exercise 2.10 (5% optional)

Inspect the `RayHitpointSerializer.cs` and your `PreviewAvatarSerializer.cs` script. Suggest a way how the serialization can be optimized.