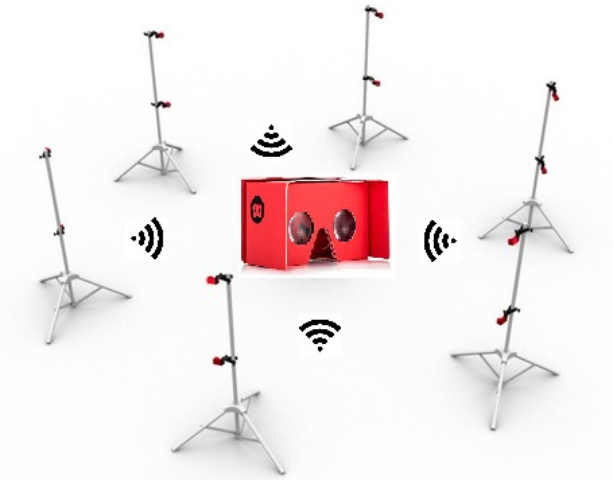

Assignment 2 — Head Tracking for VR

Deadline: 2019-05-02 23:59

In this exercise, you will extend the VR renderer that you built in exercise 1 with advanced outside-in headtracking functionality. In our lab, the *"Deskotheque"*, we have an *Optitrack* motion capture system setup, with 20 cameras to provide a large, tracked space to walk around in. For this exercise, we will only use the tracking to track a single object (the HMD, which will be augmented with tracking markers). For Assignment 4, we will also use our motion capture suit for full skeleton tracking. We provide all necessary utility classes for you to interface with the network stream of our tracking server. In addition to integrating these classes into your project, you will also implement gaze-based user interaction, which will allow you to interact with your virtual world.

1 Using the Assignment Package

We have uploaded a `.unitypackage` file for the assignment to the TeachCenter. Download it and import it into Unity. This is done by creating a new Unity project, choosing `Assets | Import Package | Custom Package...` and selecting the downloaded file. Import all of its contents. This package directly builds upon assignment 1, therefore it only contains additional files and should not overwrite anything from Assignment 1. All of the additional scripts reside inside the `Assets/Scripts/Networking` directory.

2 Tracking Interface

For this assignment, we provide the tracking data interface `TrackingDataProcessor.cs`, which is able to parse incoming network traffic from our tracking server and issue callbacks depending on the received message type. Additionally, we also provide an almost complete `ClientReceiver.cs`, which sets up a `UDPClient.cs`, in order to listen to the UDP multicast events that are sent by the server. You will find that the callback for tracking the HMD pose is already set up in `ClientReceiver.cs`, and the only thing left to do is to apply to received pose to your HMD transform.

For offline testing outside of the Deskotheque, the `ClientReceiver` class also provides functionality to play back previously recorded pose paths. The provided `.unitypackage` contains several paths that reside inside `Assets/RecordedPaths/HMDPaths`, and can be used to test the tracking offline. By default, these recorded paths will loop infinitely. The paths were recorded for the `OurVR_Task2_Client` scene.

The `OurVR_Task2_Client` scene comes preconfigured with the `Prerecorded Data` field of the `ClientReceiver` set to `smooth_slow_path`. If you want to use the networking functionality, you have to *delete or empty the* `Prerecorded Data` *field, otherwise it will run from the prerecorded file*.

Finally, you can also test the networking in your own local network by using the `OurVR_Task2_Server` scene. This scene already contains a setup that tracks your movement (using the same control script as in Assignment 1) and sends it via multicast to all clients. If you run the server on your desktop PC, and the client on your phone, you should see your movement on the PC being reflected on your phone.

Our networking interface was tested on Android 6.0, so it should work for more modern versions of Android as well. However, UDP multicast is known to cause issues on some versions of Android and/or different phones, so let us know if you run into issues with your local network setup.

3 Network Message Format

While this is already handled by our provided classes, it might still be of interest to discuss the network message format that we use between the tracking server and the client. All messages start with a `1-byte message id`, describing the message type. This is followed by a `4-byte float timestamp`, that simply measures the time relative to the server-start time. Afterwards, raw binary data is appended that corresponds to the serialized structures in `SerializableTypes.cs`. Note that the serialized data is not tightly packed and contains

additional information to make the de-serialization easier, so do not be surprised if the size of these serialized structs is larger than what you would expect.

4 Head Tracking (5 Points)

For this exercise, you have to correctly include the `ClientReceiver` class into your project, and set it up such that your head is correctly tracked according to the received network messages. We recommend to simply add a new `Empty GameObject` (Right Click in the Object Hierarchy inside the Unity Editor, `Create Empty`), to which you can attach a new instance of the `ClientReceiver` script. The `MultiCastIPAddress` and `MultiCastPort` attributes can be left as is, and the `PrerecordedData` attribute can be assigned to use one of the prerecorded paths. You can test this by using the pre-recorded paths that are found in `Assets/RecordedPaths/HMDPaths`, which you can simply drag into the public `PrerecordedData` attribute of the `ClientReceiver` script instance.

Note: Depending on how you apply the received pose to your HMD GameObject, you might have to disable or remove the `HeadMover.cs` script, as they might simultaneously modify position and rotation of the affected GameObject. Keep this in mind, otherwise your position or rotation might not update correctly.

5 Gaze Interactive World (10 Points)

Implement a way to interact with your scene, based on your gaze. The sample project in assignment 1 already provided a reticle that sits in the center of your field of view. For this part, your task is to use this reticle (and therefore your gaze) to interact with the scene in some way. Be creative, and try to make your virtual world as interactive and fun as possible! Some examples of gaze-based interaction could be:

- Teleportation (e.g. after N seconds, teleport to gaze target)
- Animating Objects (e.g. opening doors, moving objects, activating switches)
- Changing Materials (e.g. material changes its parameters depending on how you look at it)
- Gaze-Based UI (e.g. buttons/sliders that can be controlled just via gaze)

For testing or recording the video, you can either use the `smooth_slow_path.bytes` pre-recorded pathinside the `OurVR` scene, which includes several ‘stationary’ gazes, or you

can also move around in the desktop application using the `HeadMover.cs` script that was already included in assignment 1. Don't feel limited by the prerecorded paths - you can change the scene to your liking and test the interactivity without the tracking (using just the Gyroscope and game controller) as well. When you're able to try it out in the Deskotheque, you will be able to test your interaction with full head tracking.

6 Submission

Make sure to describe what exactly you implemented in your version of a "Gaze-Interactive World". Submit a `.txt` file **reporting each way of interaction**. Export your full solution in Unity a `.unitypackage`. You should also submit a **30s-60s video of your application running in Unity** (`.mp4`). We recommend using `OBS`, a free, open source window and screen capture solution, to record your video (`https://obsproject.com/de/download`) [1]. When recording the video, make sure to show how your application interacts with gazes. To make sure that your head tracking is set up correctly, also include at least a short section (5s-10s) that shows one of the prerecorded paths being played back correctly.

Upload your report, package and video as a `.zip` file in the TeachCenter as a submission.

7 Trying it out in the Deskotheque

You can register for a slot to try out your solution in our deskotheque lab at the following link:

`https://doodle.com/poll/erdq99r6evq7gaiw` [2]

While this is optional, we encourage you to take this chance in order to get familiar with our lab setup. Since the time slots are only 30 minutes each, please make sure that you have your application ready to deploy. You will at the very least need a built client on your phone that uses the same multicast IP that was configured in the package you received. Otherwise, you will need to tell us your modified network settings (IP, port), so we can change the server accordingly. Additionally, please also bring your Unity project as well as cables for your phones, in case we need to change the network configuration and/or rebuild on-the-fly.

[1] https://obsproject.com/de/download
[2] https://doodle.com/poll/erdq99r6evq7gaiw