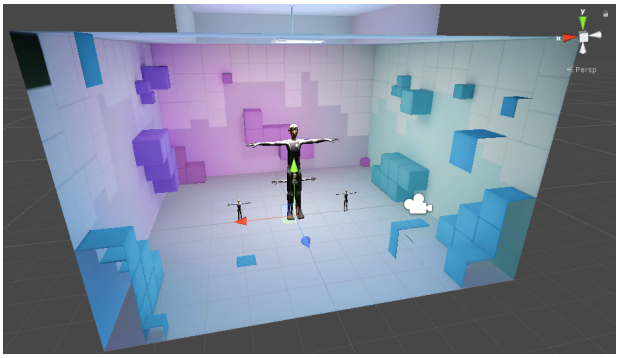

Assignment 4 — Hall of Mirrors

Deadline: 2019-06-27 23:59

In this exercise, you will combine all of the previous exercises to experience a virtual *Hall of Mirrors*, with full head tracking from exercise 2, skeleton tracking using a motion capture suit, and VR rendering using the cardboard and VR renderer from exercise 1. You will incorporate your own avatar for increased immersion and include different humanoid models (monsters, other people, ...) that will move according to your movement. **The lab sessions mentioned at the end are not optional, you have to attend at least during one of the slots and show some progress!**

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. The main scene that contains most of the necessary functionality is `OurVR_Task4_Client.unity`. Feel free to use your own scenes and only use our scene as a template and/or only use parts of our provided scene. The provided server scene can be used to test the program in your own local network. For this task, the networking functionality has been modified compared to Assignment 2, therefore there have been changes in many scripts compared to Assignment 2. Please make sure that you do not overwrite functionality (gaze-interaction, scene elements...) you want to keep from Assignment 2. I recommend to use all files from the provided package and re-integrate your changes after importing. Most of your changes for this task have to be done inside the `ClientReceiver.cs` file.

2 Tracking Interface (Addendum to Assignment 2)

Since there were severe issues with the UDP Multicast setup in the Deskotheque lab, we decided to change the network code to a conventional server-client model. The `ClientReceiver` now connects to the Server PC directly via UDP. The Server-PC in the Deskotheque uses the IP-Address `172.28.0.1`, so you can prepare for that before you come to the lab sessions.

In addition to the revised network functionality, we also now include the necessary callbacks for the skeleton tracking, which is called `OptitrackSkeletonStateCallback`. Similar to the head tracking, we provide pre-recorded files to test the skeleton tracking before the lab sessions, which you can find in `Assets/RecordedPaths/AnimationData`.

3 Skeleton Tracking Retargeting (15 Points)

The tracking interface provides us with raw bone transforms (position, orientation) of all the bones defined by the Optitrack system. Since your scanned avatars from Assignment 3 are all slightly different, we can not just apply the raw bone transforms to your avatars. In order to accommodate for varying bone lengths, we need to perform a process called *retargeting*.

Unity's *Mecanim* animation system already provides retargeting functionality for humanoid avatars. Internally, Unity transforms the muscle and bone transformations of a *source* humanoid avatar to a *destination* humanoid avatar. This requires a Unity `Avatar` as well as the corresponding hierarchy of `Transform` objects.

In order to make use of this, we need a (hidden) source avatar onto which we apply the bone transforms we received over the network. In the `OurVR_Task4_Client` scene, this is the *RetargetingAvatar*. This is the same avatar as used on the server side, therefore the bone transforms can be applied *directly without retargeting*. Once this is done, we can use Unity's retargeting system to apply the animation from this avatar to any other humanoid avatars in the scene, even with different bone lengths.

The following subsections will describe the necessary additions to `ClientReceiver.cs` in order for the skeleton tracking to be fully implemented. They use the same numbering as in `ClientReceiver.cs`.

1.) SetupHumanPoseHandlers() The provided `.unitypackage` contains an empty `SetupHumanPoseHandlers()` function. This function is called once at the startup of `ClientRe-`

ceiver (during Start), and should be used to initialize member variables for HumanPoseHandler objects. You will need a single *source* HumanPoseHandler as well as N destination HumanPoseHandlers, one for each avatar you want to animate. These HumanPoseHandlers correspond to the RetargetingAvatarAnimator (1a) and DestinationAvatarAnimators (1b) which are defined in the ClientReceiver already. The constructor for HumanPoseHandler requires an Avatar and a root Transform, all of which are simply members of RetargetingAvatarAnimator and DestinationAvatarAnimators[...]. These HumanPoseHandlers will allow us later on to retarget an animation from our source avatar (RetargetingAvatar) to all of the destination avatars (DestinationAvatarAnimators).

2.) Applying the Networked Skeleton Data to `RetargetingAvatar` The provided `.unitypackage` contains an empty `OptitrackSkeletonStateCallback(VRVUTypes.OptitrackSkeletonState skeleton_state)` function. This function is already setup such that it is called every time a new skeleton pose is received via the network connection. In this subtask, it is necessary to apply the received bone poses (position, orientation) to the RetargetingAvatar such that we can further use the RetargetingAvatar for retargeting to our final avatars. VRVUTypes.OptitrackSkeletonState skeleton_state is a Dictionary which contains HumanBodyBones as its key and VRVUTypes.OptitrackPose as its value. You can iterate over this dictionary, query the corresponding Unity Transform from the RetargetingAvatar with RetargetingAvatarAnimator.GetBoneTransform(), and assign the position and orientation of all the dictionary entries to the bones of the RetargetingAvatar. Make sure to only do this for non-null return values of RetargetingAvatarAnimator.GetBoneTransform(). Once this is done, you can verify the functionality by 'enabling' (checking the checkbox in the Unity scene editor) the initially disabled RetargetingAvatar. If done correctly, you should see this avatar move, at least if the entry for the RetargetingAvatarAnimator in the ClientReceiver object is set correctly, and you're receiving data from either a prerecorded animation or from networking.

3.) Retargeting from `RetargetingAvatarAnimator` to `DestinationAvatarAnimators` In this step, we require the HumanPoseHandlers initialized in step 1. In order to retarget our animation from RetargetingAvatarAnimator to the DestinationAvatarAnimators, we can use Unity's animation retargeting feature. This is done by calling GetHumanPose(ref human_pose) on our *source* HumanPoseHandler. human_pose then contains all the relevant pose information required to retarget it to our destination avatars. We can then iterate over the DestinationAvatarAnimators, and call SetHumanPose(ref human_pose) for each of them, with the human pose we acquired from our source handler. After doing this, you should already see animations applied to all DestinationAvatarAnimators. In order to prepare for the 'Hall of Mirrors', you will also need to add a way to lock the position of the animated avatars to a certain point, since otherwise they will move around like the RetargetingAvatar. Each AnimatorControl object in DestinationAvatarAnimators

already provides a `bool lock_position` that you can check, and lock the destination avatar position to its base or initial position if checked. Additionally, if you want your scanned avatar to follow the absolute coordinates of the received tracking pose, you will probably need (depending on your Avatar's scale) to implement functionality that follows the `RetargetingAvatar` hip position. Each `AnimatorControl` object also provides an additional `Animator track_position_animator` for this purpose.

4.) Add your Scanned Avatar from Assignment 3 You should be familiar with this process, since you did this in Assignment 3. Add your scanned avatar into the scene as one of the destination avatars, and make sure that the animation works properly.

4 Virtual Hall of Mirrors (20 Points)

Now that the network tracking is working and you got your own scanned avatar inside your VR scene, it is up to you to create an interesting, exciting and immersive virtual hall of mirrors!

Find free humanoid models online and/or on the Unity asset store, arrange the scene to your liking (keep the rough bounds of 6x4 meters of the tracking space in mind and/or have functionality to teleport around!), and put different models and avatars into your scene. You can easily achieve a 'mirror-like' effect by placing the models at a fixed point (use the position lock feature from 3.) and rotating/scaling them accordingly. Another example for an idea you could play around with is Distortion effects by using billboards and Unity's 'GrabPass' shader feature. Everything you implement is fully up to you, try to create something immersive which is fun to move around and look at the mirror avatars in!

5 Submission

Make sure to describe what exactly you implemented in your "Virtual Hall of Mirrors". Submit a `.txt` file **describing your implementation**. Export your full solution in Unity a `.unitypackage`. Additionally, please describe any modifications you did to the framework. 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 interesting parts of your scene, and show that the destination avatars of

[1] https://obsproject.com/de/download

your choice move according to one of the provided animation paths. Finally, please provide your solution as an .apk file as it was used to test in the Deskotheque lab. Upload your report, package, video and .apk as a `.zip` file in the TeachCenter as a submission. If you submission is too large, please upload it to a filesharing service of your choice and/or bring it to us via USB stick.

6 Deskotheque Lab Sessions

These lab sessions are not optional, you have to attend at least once and show some progress! In order to test your application and play around with the motion tracking system in the Deskotheque, there will be 10 days from June 11th to June 25th where we you can develop/test under our supervision and guidance in the Deskotheque.

You can register for a slot at the following link:

`https://doodle.com/poll/897xye4p5ewhrr6x` [2]

Since we're ≈ 18 participants, please only reserve around 3 hours each. There are 2hour slots in the first lab sessions which are mainly meant for experimentation, development and pose recording. The final 1-hour slots are for final tests, and to make sure everything is still working fine with the motion capture and head tracking in the lab. Please only register for a single 1-hour slot, unless everyone else already registered. Ideally, you should try to impress us with your solutions during these final 1-hour slots, as we will collect the most interesting, creative and immersive solutions for the final look after the exam.

For all lab sessions, please prepare your development environment, such that you can deploy onto your phone. Additionally, please bring everything you might need, i.e. your avatar, your Assignment 1/2/3 projects, ... While we can deploy onto your phone using one of the Lab PCs, you will probably enjoy working on your own environment much more.

All the best for the assignment, and have fun!

[2] https://doodle.com/poll/897xye4p5ewhrr6x