Combining VR and Motion Capture with Unity®

Abstract

Hello everyone, in this tutorial we will have a look how to combine optical motion tracking and VR within the Unity® engine. The Leap Motion Sensor provides a simple and affordable solution for markerless optical hand tracking. At first we will have a look at the setup and the basic configuration options. After this, we will check out the standard assets and the Leap API. In the next step, we will integrate the Leap sensor in an experimental setup. Finally, we will have a look at the combination of the Oculus and the Leap. The structure of this document is as follows. First, the setup of Unity® and the Leap Sensor is described. Second, there is a short overview of the sensor, its assets and drawbacks. After this, I describe the target scenario discussed in the tutorial. At the end of the document, you will find some general tutorial suggestions for scripting in Unity®, if you are interested in.

Contents

1 Installing Unity and the Leap Runtime

In this tutorial, we will rely on the free version of Unity®, features of the pro version are not covered. Installers are available for Mac OS X (10.8. and above) and Windows (at least Windows 7 SP1). A detailed description of the system requirements can be found here. Installing Unity on Linux systems is possible, however, it requires a Windows runtime environment, like Wine. A description for the setup can found in the Unity Community Wiki. By now, there is also an experimental build, but it relies on a - comparatively - dated version of Unity® with limited VR support.

During the installation, you have the option to install Visual Studio Community 2015 as the default editor. If you have no other C# editor installed, than I would recommend you to install it. However, this can take pretty long. In the end you can edit the scripts with any editor you like, Unity® will compile the code.

Leap motion released a new version of their SDK (a new version of the sensor will follow soon), which I would highly recommend. Installers are available for Mac OS and Windows, install instructions for Linux can be found here, however, the SDK available for Linux systems is somewhat dated. You can obtain the Orion SDK here. The recommended system requirements can be found here (if your machine can handle the Oculus, the Leap will run fine). After installing, you should see the Leap server in your task bar. The Unity® assets can be obtained from here. If you would like to try them out, then start the Unity® editor, hit the import button, select the `unityasset` file from the Leap install director and open the scene you are interested in.

2 The Leap Motion Sensor

The Leap sensor is an optical hand tracking system. Like the Microsoft Kinect, it does not rely on markers (like for instance Vicon setups), but uses image processing algorithms to infer the 3D position of the tracked objects (i.e. the hands, or tools). While the Kinect produces a depth map based on stereo images, the Leap sensor computes depth information by distortions of an infrared grid constructed via three LEDs, tracked by two cameras. The raw data is filtered to remove background objects, after this a hand model is fitted to the raw data. The processed data is then broadcasted using a TCP/IP protocol. The framerate is quite high, depending on the processor load it can be around 200 Hz. Some details regarding the inner workings can be found here.

In experimental setups and everyday usage, markerless motion capture systems have some advantages. Most notably, the setup is very fast: You do not need to put markers on your participants and the tracking works out of the box, with-

out prior calibration. However, the tracking is less accurate and since it relies on - partially heuristic - image processing algorithms, the error range is larger compared to OptiTrack or Vicon systems. Further, the Leap sensor is not a dedicated scientific device and literature regarding the overall accuracy is scarce. One paper can be found here, but it relies on a dated version of the SDK. If your application requires high accuracy, the Leap motion sensor is probably not the device of choice. Besides these issues, it is unclear how virtual object interaction by means of the Leap sensor compares to natural object interaction. That is, there is no literature available which compares the movement trajectories, speed, velocity and aperture profiles obtained with the Leap sensor and natural objects. Hence some caution is advised when using this device. Further, the Leap sensor suffers from the same problems as all optical tracking devices: Occlusion. Especially, when you want to investigate fine motorics and subtle finger movements, an inertial tracking system, relying on gloves, or a optical tracking system with multiple cameras, will provide more reliable results.

Based on our experience and data, the Leap sensor is very well suited to create a sense of spatial presence, since participants see there hands moving in a predictable and natural way. Especially, when combined with object interaction, the sense of presence seems high. As the API allows full control how the data is applied in VR setups, the Leap sensor seems to be a viable tool to investigate coarse scale object interaction and body image manipulations.

3 Target Scenario

So far, we used the Leap sensor in experiments on working memory, visuo-motor control, approach-avoidance scenarios and body image manipulations. The setup in this tutorial will feature a combination of object interaction and body image manipulation. We will use the Leap sensor to administer a bimanual task during which we will dissociate the actual and felt hand positions.

3.1 Leap Sensor Setup

In most cases the sensor works out of the box. We will have a look at the default visualization and check the tracking range. Since the Leap relies on infrared tracking, it is affected by external light sources. The compensation has improved largely with the introduction of the Orion SDK, but some issues remain.

- The Visualizer utility
- Tool tracking
- Lighting conditions

3.2 The Leap Unity Assets

The older - pre Orion release - Leap Asset bundle came with a lot of examples, showing the capabilities of the Leap Sensor together with Unity. We will have a look at some of the samples and how they are implemented. We will also have a closer look at the default grasping mechanism provided in the Leap Assets.

- Leap data streaming
- Visual and physical hand models
- Grasping with the Leap sensor
- Raw data serialization

3.3 Position Checks and Data Acquisition

If using the Leap sensor in an experimental setup, it is necessary to assure constant initial hand positions at the beginning of a trial and to keep track of the hand constantly. We will have look how to implement position checks and how to obtain positional data from the raw data stream.

- Position checks
- Data acquisition and serialization

3.4 Combining the Leap Sensor and the Oculus HMD

Of course you can place the Leap sensor in front of the participants, but there is a nice way to combine locomotion (Oculus tracking) and the Leap sensor. In such a setup, participants can take their virtual hands with them while moving through a scene. We will have look how to do this and check the robustness of the setup.

3.5 Sample Experiment

Now we have got an overview of the Leap sensors capabilities and how to integrate it into a VR setup. We will have a look at a sample experiment, where participants have to pick petals from a flower. During this bimanual interaction we will shift the visual hand model, inducing multisensory conflict between visual and felt hand position. Such a setup might be used to study visual dominance, or changes in the perception of peripersonal space, which can be tested afterwards by pseudo-neglect effects (e.g. Longo & Lourenco 2007).

- Implement a bimanual task
- Implement a shift in position under certain circumstances
- Control task execution and hand drift

This is the general outline, I hope the points you are interested in are covered. If you miss a certain point than just write me before the tutorial and I will try to incorporate it.

4 Further Tutorial Suggestions

As mentioned above, Unity® supports different scripting languages, I would suggest to use C#. There is a dedicated section on the tutorial page, which is concerned with scripting. Most tutorials feature a video guide and code samples.

If you are interested in the basics, I would recommend the following tutorials:

- Scripts as Behaviour Components: This tutorial introduces the interface between the editor and scripts that is how to assign scripts to objects via the editor.
- Awake and Start: All scripts that are assigned to objects are derived from the MonoBehaviour class. This class defines various events, like Awake and Start which can be used to initialize the respective scripts.
- Update and FixedUpdate: Two other important events are Update and FixedUpdate, they are called once per update cycle of the graphics (Update) and the physics engine (FixedUpdate).
- Instantiate: In object oriented languages (like C#) you are usually using so called constructors to create objects. In Unity® you will usually rely on the Instantiate method.
- Activating GameObjects: You can enable and disable objects at runtime, this is quite useful to avoid heavy CPU load due to complex operations that not need to be carried out all the time (for instance our fixation check should only run at certain times). Activating / Deactivating is one possible solution in such cases.

C# in general is a versatile programming language, some of its features are covered in the intermediate section, I would recommend the following tutorials:

- Lists and Dictionaries: These data types are more flexible than simple arrays, especially if you need to create some kinds of mappings, dictionaries are the data structure of choice.

- Coroutines: Sometimes it is necessary to handle tasks in parallel, or in terms of background tasks. Due to its implementation, it is not advisable to use the common C# threading system in Unity®. If you want to parallelize tasks, Coroutines are the method of choice.

- Delegates: Sometimes you want to respond to events with the invocation of a certain function. For instance, you might want to inform the main script if the participant walks into an object. Delegates are an elegant way to realize such behaviors.

All these tutorials are detailed and provide example code. However, they focus quite exclusively on the respective concept. If you would like to see these concepts mixed and applied in a broader context, I would recommend the following tutorials:

- A 3D Fractal: This tutorial features dynamic object generation at runtime and shows how to manipulate object properties on the fly.

- Swirly-Pipe: A small racing game that covers a lot of the engines features.

In contrast to the tutorials on the Unity® page, the catlikecoding tutorials feature the complete setup of a project.