TODO – test out removal of UpdateProjectionPosition script…

Elaborate on the Multiple Displays flag.

Turn MasterTrackingData into referenced object instead of singleton.

Does TrackerHostSettings need to be on planeHolder? Would it make more sense to have it on the root prefab object?

UniCAVE Plugin version 1.0

Ross Tredinnick, Brady Boettcher, Sam Solovy, Kevin Ponto, Simon Smith

3/4/2017

**Contents**

1. Introduction
2. Installation
3. Adapting to an existing Unity Scene
4. Creating a new Immersive Projection Setup
5. Examples
6. Best Practices
7. Change Log
8. **Introduction**

The UniCAVE Plugin is a solution for running Unity within immersive projection VR display systems. The following document will guide on setup of the Plugin within Unity. The plugin has been tested on Unity version 5.4.03f and Unity version 5.5.03f.

The UniCAVE plugin is an open source project meant to be built upon to increase support for Unity across different forms of immersive projection display setups and to increase support for Unity functionality for clusters of PCs.

1. **Installation**

Pull or download a zip of the current repository to a desired location.

The plugin can be added to a Unity project by clicking the “Assets” menu within the Unity editor, followed by clicking the “Import Package” sub-menu and choosing Custom Package…

Select the CavePlugin.unitypackage in the PluginFiles directory of the repository or extracted zip.

Upon importing the package, several assets will appear in Plugins, Prefabs, Scripts, and ExampleScenes directories in your project.

The Plugins directory has plugin support for interfacing with 3D tracking systems via a VRPN Unity plugin.

The Scripts directory contains the necessary scripts for tracking and for asymmetric projection matrix calculation.

The Prefabs directory contains several example immersive projection setups. These prefabs can be dragged into your scene to enable support of the associated immersive projection setup. Each prefab consists of a hierarchy of Unity game objects which make up the immersive projection setup. The objects within the hierarchy consist of quads which represent projection surfaces, cameras, and general game objects that contain scripts for performing correct projection. The ordering of objects within the hierarchy matters (particularly camera order where left cameras appear as children before right eye cameras).

The ExampleScenes directory contains example Unity scenes that already have one of the prefabs within them enabled for use within the scene. These scenes can be useful for debugging your particular physical display setup.

1. **Adapting to an Existing Unity Scene**

To adapt the plugin to an existing scene follow these directions:

1. Drag and drop a prefab that matches your physical display setup into your scene.
2. Click on the root object of the prefab that you just dragged in. There should be a script called **Networking Sync** on the object. If you are running the UniCAVE plugin across a cluster of machines, keep this script on the object, if not, you can remove the script. For a cluster setup, 3 fields exist in this script that should be edited, depending on your immersive projection setup. - - “Head Node IP” should match the head node’s IP address of your cluster.

- “Port” should be set to an open port for opening a network connection between the head and slave nodes.

- “Num Slave Nodes” should be set the number of slave nodes in the cluster.

An optional setting that can be edited is the “Network Updates per Second” which will alter how fast Unity’s networking system updates, which can have an influence on distributed rendering in a cluster.

1. Another script called **Master Tracking Data** contains several global settings for the immersive projection display setup.

* “Head Node Machine Name” is a string that should be set to the name of the head node in a cluster setup, or the name of the machine running the plugin if no cluster is present.
* “Tracking System Offset” is a vector that can be added onto the tracking system’s per-frame position.
* “Multiple Displays” will activate additional displays within Unity when working with a multiple monitor or projector setup.
* Left stereo first (default = true) determines which eye Unity will render first, this should be set to match whichever eye is displayed first via the settings on your projectors or 3D monitors.
* The rest of the settings (Eye Offsets, clear flags, near/far clip, background color) all serve as shortcuts for editing properties across all cameras that are within the hierarchy of the immersive display projection prefab. To sync edits made to Master Tracking Data to all of the cameras, right click on the title of the script and choose “Sync Camera Settings”

1. The next script to edit is called **Tracker Host Settings**.

* “Hostname” should be edited to have the VRPN address name that is on the suffix of your VRPN device naming. This is often the computer name that your tracking system is physically connected to. For example, Isense900@C6\_V1\_HEAD – C6\_V1\_HEAD should be “Hostname” in this case.
* “Type” defaults to “Standard” and standard assumes a VRPN interface to your tracking system. There is also an option for Vicon but this has been untested.

1. Several **Cave Tracker Settings** scripts exist on different objects in the hierarchy. This script connects a particular tracked channel to an object in Unity. For example, by default the object “Tracker Rotation” obtains the tracked rotation (check box “Track Rotation” is on) returned by VRPN for channel 0.

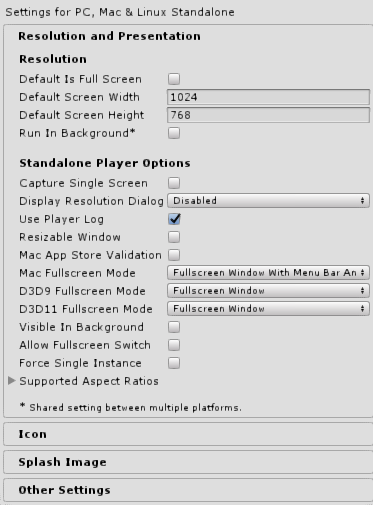
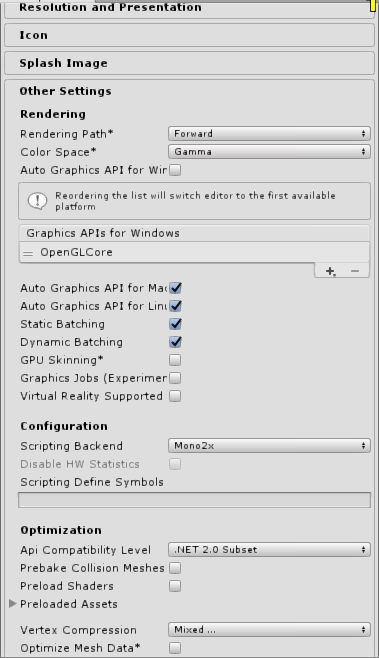
* “Tracker Prefix” should be set to the prefix used on your tracker address name as used by VRPN. For example, for an Intersene900 tracking system, “Isense900” is used for 6-dof trackers, and “Wand#” is used for button and analog tracking.
* Most of the prefabs are setup to use a CAVE wand-style input device for basic navigation.
* “Holder” should have a reference back to the root object in the prefab hierarchy, this allows the basic navigation – holder only needs to be set for the “WandControls” object.

1. The final script to edit is **Projection Plane** which exists on each child of the “Camera Holder” object within the hierarchy. In many of the prefabs, each child of “Camera Holder” defines the link between the plane geometry used to setup the physical projection geometry, and the cameras being used to render to that surface. In each script, “Machine Name” should be set to the machine name of the PC that drives the associated display as defined by the geometry of the “Plane” object that this scripts references. For example, if one machine in your setup drives a front wall, The **Projection Plane** script on the “Frontcam” object’s “Machine Name” variable should be set to that machine’s name. If another machine drives a floor wall, the **Projection Plane** script on the “FloorCam” object should be set to the floor machine. In situations where a single machine drives several walls, just set each “Machine Name” variable to that same machine name.
2. Special considerations need to be adopted for situations where a single machine drives several displays.
3. Next some adjustments need to be made within the **VRPN.cs** script inside of the LELUnityPlugins\Scripts\VRPN Scripts folder. The two functions **vrpnTrackerPos** and **vrpnTrackerRot** should be adjusted to reflect the transformation from your tracking system to Unity’s world space (z in/out, y up/down, x left/right). Any sort of tracking system offset from your CAVE can be done in the **vrpnTrackerPos** function while any rotation can be done in **vrpnTrackerRot**. Our system’s default values may work, but probably not.
4. **Making your own Immersive Projection Setup**

---------- Old Documentation Below ----------

Once these adjustments have been made, go ahead and build your project. Choose x86\_64 and a Windows build (only tested on Windows as of 10/18 – will be tested on Linux soon).

Here is a screen shots of the necessary Player Settings. The plugin is meant to be run with OpenGLCore as the only Graphics API.

Now, wherever you’ve built your exe, take the packaged “GLQBStereo.dll” and “OpenGL32.dll” files and place them in the same directory as your exe. In addition, open the “config.ini” file and change the “NumViewports” option to match the number of stereo viewports you have in your scene **per machine**. I.e. if you run 4 stereo walls on a single machine, set this value to **8.** Finally in the packaged “gliConfig.ini” file, scroll all the way to the bottom to the “PluginData” section. Change the BaseDir to the directory of your exe. Copy the ini files plus the “GLFunctions” directory to the location of the exe as well.

Run the exe using whichever cluster-based launcher tool you use to launch your VR applications! When running the exe, supply a –popupWindow command line argument.

**Todo List**

Currently input buttons from a tracked wand device are weakly supported – this will be improving in coming weeks, in addition to support for a game pad.

**7. Change Log**

3/4/2017 – version 1.0: Updated for handling dual pipe active stereo, updated documentation on VR Supported Flag - Stereo (Non-Head Mounted) functionality.

12/15/2016 – version 0.3: Added functionality to support Unity’s native Stereoscopic (non-head mounted) VR flag. Got rid of the need for the UpdateProjectionPosition script on all cameras. Corrections to MasterTrackingData script. Testing and verification on Untiy 5.5.

11/2/2016 – version 0.2: Attempt at correcting some bugs with multiple viewports on a single display source, as well as VRPN sources on non-head node machines.

10/15/2016 – version 0.1: Initial release