Uni-CAVE Plugin version 1.0

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

3/4/2017

Change Log

3/4/2017 – version 1.0: Updated for IEEE VR 2017

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

**Contents**

The Uni-CAVE 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.

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 appropriate package that you downloaded from the LEL.

Upon importing the package, several assets will appear in your project, including a LELUnityPlugin, a Plugin directory, and a Prefab directory. Drag the <Your Lab>\_Holder.prefab file into the scene.

Various global settings for the scene can be set on the **MasterTrackingData.cs** script. These values include eye offsets, whether multiple displays need to be activated (set this to true if you need more than one viewport for rendering to your immersive system), tracking system offsets, default background clearing for all of the cameras, near and far clip values, etc. To sync any changes here with all of the left/right eye cameras, you can drag the script onto the “CameraHolder” object, then right click on the Inspector entry “Master Tracking Data (script)” and choose “Sync Camera Settings”.

Next you need to setup some tracking system related information. Click the “planeHolder” object within the prefab you imported and find the “Tracker Host Settings” component. Under **hostname** – enter the **name of the PC connected to your tracking system**. For **type** – keep as **standard** for VRPN usage – there is also a “Vicon” option but this is currently un-tested.

Next click the “Camera Holder” object. For the **Cave Tracker Settings,** under **Object Name** – enter the **head tracker address** for connecting to your tracking system and the **channel** you wish to track. I.e. for our CAVE this is “Isense900” and channel 0.

Perform the same as in the preceeding paragraph for the “TrackerRotation”, “Wand” and “WandControls” objects. (Note you may have a different address for the Wand – and different channels, i.e. ours is “Wand0” and channel 1.

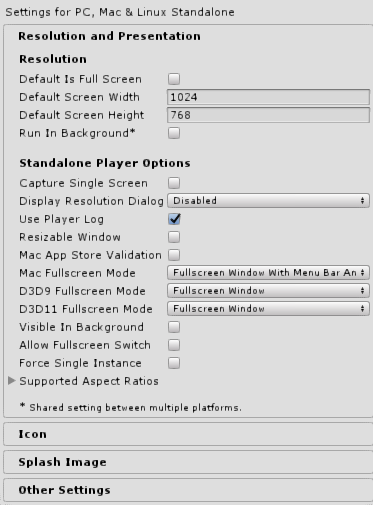
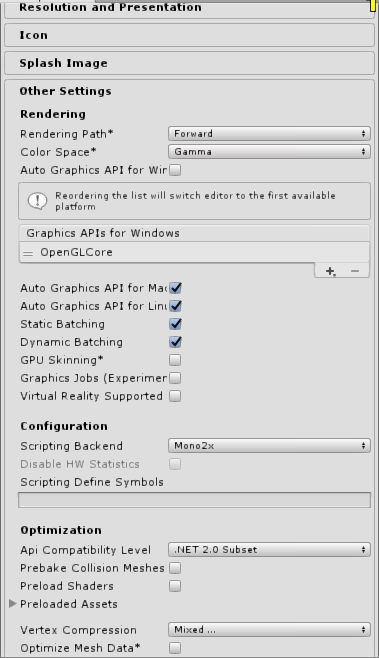
Next click the children objects within “Camera Holder”. Each has a **ProjectionPlane.cs** script. For each object, find this script and for the **Machine Name** entry, enter the **name of the machine** for which your projector / TV / etc. is attached to. (For us we have one machine per wall – this may be different depending on your system setup – perhaps you have the same machine for all walls, go ahead and enter that same machine name for each object).

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.

Finally, there is a **networkingSync.cs** script attached to the Camera Holder object (if not, try adding). Insert values on this script where hostname matches the IP address of your head node, port is a free port on the local network, and Head Node is the machine name of the machine your tracking system is connected to.

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.