Uni-CAVE: A Unity3D Plugin for Non-head Mounted VR Display Systems

Ross Tredinnick* Brady Boettcher† Simon Smith‡ Sam Solovy§ Kevin Ponto¶

Living Environments Laboratory, Wisconsin Institute for Discovery, University of Wisconsin - Madison USA

Abstract

Unity3D has become a popular, freely available 3D game engine for design and construction of virtual environments. Unfortunately, the few options that currently exist for adapting Unity3D to distributed immersive tiled or projection-based VR display systems rely on closed commercial products. Uni-CAVE aims to solve this problem by creating a freely-available and easy to use Unity3D extension package for cluster-based VR display systems. This extension provides support for head and device tracking, stereo rendering and display synchronization. Furthermore, Uni-CAVE enables configuration within the Unity environment enabling researchers to get quickly up and running.

Index Terms: I.3.7 [Three-Dimensional Graphics and Realism]: Virtual Reality—Unity3D; I.3.2 [Graphics Systems]: Distributed/Network Graphics—Unity3D

1 Introduction

Unity3D has become a popular, freely available 3D game engine for design and construction of virtual environments. Much of this can be attributed to a friendly user interface for designing content when compared to some other 3D design applications, together with a robust tool set for working with common elements of a 3D environment such as terrains, physics, particle effects, sounds, models, and animated characters. Unfortunately, when it comes to using Unity3D on immersive projection-based virtual reality display systems, some support exists but at an additional monetary cost. While previous efforts have been made to adapt Unity3D to tiled displays [5] and to immersive VR display systems [2], these solutions have utilized closed commercial products. The presented work, entitled Uni-CAVE, adapts Unity3D to general non-head mounted immersive display systems, including CAVEs and tiled display systems. The Uni-CAVE plugin allows for different stereo techniques, such as OpenGL quad buffered stereo, passive stereo, and side-by-side stereo. The intention of this work is to provide a free to use solution that can be adapted to any immersive VR projection system.

2 Method

Uni-CAVE is a Unity3D package for importing into an existing Unity3D scene and contains several pre-defined immersive projection-based VR display system configurations in the form of Unity3D prefabs. Prefabs created with the plugin consists of a series of Unity C# scripts and resources.

Display Configuration: To make the configuration more user-friendly, projection surfaces are created directly inside of the Unity3D editor. To accomplish this, quad objects which define projection surfaces, are paired with camera objects. The UniCAVE plugin subsequently calculates the correct camera projections based on the technique described in [1] and presents a visualization of the viewing frustums in the Unity3D editor. Figure 1 shows an example configuration for a six-sided CAVE style immersive VR display system.

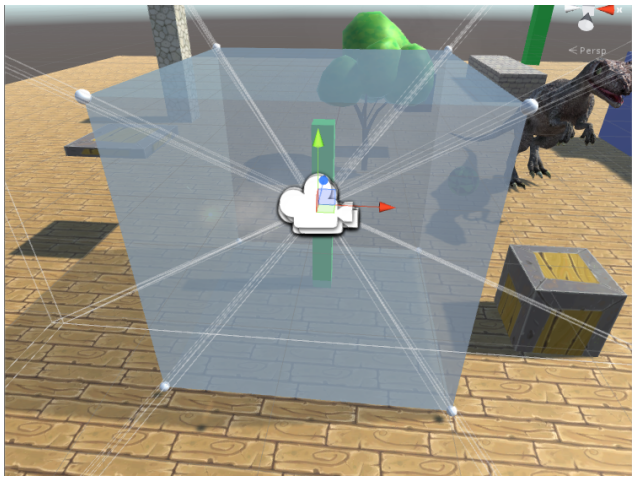
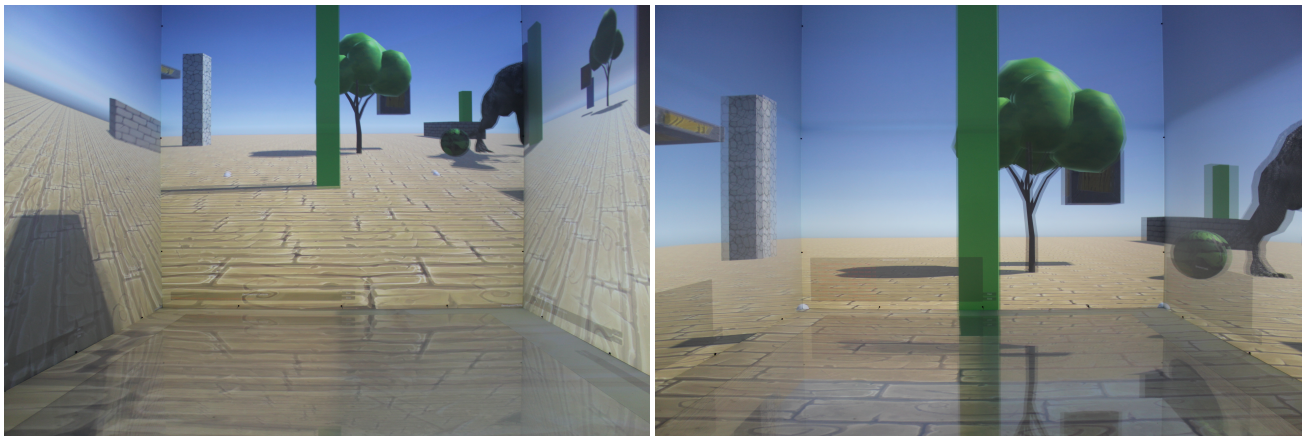


Figure 1: A camera configuration within the Unity3D editor consisting of twelve Unity3D camera objects and six quad objects (transparent grey) representing the walls of a CAVE.

Stereo: The Uni-CAVE plugin provides support for different stereo techniques. Techniques such as quad-buffered stereo, side by side stereo, or split-screen stereo are configured by way of setting up the correct camera and viewport configurations in the Unity3D editor, together with pairing these cameras to the Unity3D quads that represent the physical display setup. For quad-buffered stereo, which often requires setting an underlying operating system stereo flag for the window, different options exist in the plugin depending on the version of Unity3D. Prior to version 5.1, Unity3D provided no support for quad-buffered active stereo. To support OpenGL quad buffered stereo in this case, the Uni-CAVE plugin provides a stereo injection technique by way of GLIntercept [7]. The plugin works by counting how many glClear calls are made on a bound frame buffer that returns true for a glGetBooleanv(GL_STEREO) call. The GL_STEREO check ensures that the counter only increments for the main rendering window, instead of frame buffer objects for off-screen rendering, as they return false for this check.

The release of version 5.1 of Unity3D added a player settings option titled “virtual reality supported”. When checking this box, a list of VR supported devices can be chosen. This was largely for support of upcoming commercially available HMDs such as the Oculus Rift and HTC Vive. One option, titled “Stereo Display (non-head mounted)”, provides support for quad-buffered stereo. As Unity3D has no knowledge of the physical layout of the projection surface, a standard camera projection is used. When used in a CAVE system, this can cause viewing artifacts between display surfaces as shown in Figure 2a. Version 5.4.2 introduced a function Camera.SetStereoProjectionMatrix within Unity’s camera class that allows a user to over-ride the default stereo projection matrix that Unity3D configures internally. With this ability to over-

*e-mail: rdtredinnick@wisc.edu
†e-mail:boettcher2@wisc.edu
‡e-mail:spsmith5@wisc.edu
§e-mail:solovy@cs.wisc.edu
¶e-mail:kbponto@wisc.edu



(a) Unity3D default stereo projection matrix

(b) Unity3D with override of default stereo projection matrix

Figure 2: Photographs of six-sided CAVE system of the scene from Figure 1. a) the scene as displayed using Unity' VR supported stereo (non head-mounted) setting. b) Seamless projection that our plugin provides by overriding Unity3D's default stereo projection matrix.

ride the default projection matrix, seamless visuals between display surfaces are achieved as shown in Figure 2b.

Tracking: The Uni-CAVE plugin makes use of an existing Unity VRPN plugin to interface with common 3D tracking systems [3]. The Unity VRPN plugin interfaces with a compiled version of VRPN via C# scripting to allow for typical VRPN functionality such as six degree of freedom head and wand tracking [6]. In the case of adapting the plugin to a clustered system, and as VRPN does not automatically synchronize data between nodes, a master node is responsible for gathering the data from the VRPN server and disseminating this information to the slave nodes via the methods described below.

Synchronization: The Uni-CAVE plugin uses software-based synchronization via networking to create frame-lock for clustered displays. This is setup by including Unity3D NetworkView components on camera objects and setting the observed property of the NetworkView component to the camera's transform. This ensures that the transformations of all cameras on slave nodes match the corresponding camera transform on the master node within a cluster. This synchronization method handles stereo synchronization; however, several additional subsystems need to be synchronized to maintain a seamless display across PCs - a concept known as "data-lock" [4]. Separate subsystems within game engines such as physics, animation and particle systems create challenges to ensure a seamless display across physical PC display boundaries.

Synchronizing the overall timing of the game engine across nodes is a crucial step needed to maintain a seamless display. While the time of the game engine itself cannot be adjusted via Unity3D's scripting engine, Unity3D does expose a variable to adjust the time scale of the engine, typically used for effects such as slow motion. The Uni-CAVE plugin uses this exposed feature to synchronize nodes within a cluster. When the game time of a slave node is ahead of the master node, the plugin slows down the slave node slightly for a few frames, and likewise if the slave node is behind the master node, the plugin speeds up the time scale of the slave node. This concept is handled by way of the following equation:

$$timeScale = \frac{((M_t - S_t) + \Delta_{sync})}{\Delta_{sync}}$$

Δ_{sync} is a tunable parameter which controls the rate at which timeScale is adjusted. M_t is the time since start of application on the master node, while S_t is the time-scaled game time of a slave node. Setting Δ_{sync} at a rate too frequent or infrequent can cause irregularities in performance. In practice, the authors found that syncing the time every tenth of a second maintains smooth synchronization between nodes. Synchronizing time across a cluster via this method is crucial for maintaining synchronization for animations and particle systems. Since Unity3D's physics engine runs in a separate thread from the rendering system, objects with dynamic rigid body physics must be accounted for in a different manner. To handle such objects, the Uni-CAVE plugin adds Unity3D NetworkView objects that observe the rigid body component of the game object. Particle system synchronization happens by way of matching the time between systems by way of adjusting the time scale as described above, combined with matching random number generation seeds between systems within a cluster.

3 Conclusion

Uni-CAVE is a plugin that is meant to ease the adaptation of the popular game engine Unity3D to immersive 3D VR display systems. The plugin has currently been tested on a six-sided CAVE environment that uses active quad-buffered stereo, on a twenty screen curved tiled display system that uses side-by-side stereo, and also on a two projector power wall using quad buffered active stereo and dual-pipe active stereo. Configurations such as these can be saved as prefabs and dragged and dropped into new Unity3D environments. Additional systems are currently under-going testing with the plugin and going forward we hope to accumulate a collection of immersive VR projection system setups from several partners to improve collaboration opportunities and ease the ability to share 3D environments across different projection-based immersive VR systems.

References

[1] R. Kooima. Generalized perspective projection. *J. Sch. Electron. Eng. Comput. Sci*, 2009.

[2] S. Kuntz. Middlevr a generic vr toolbox. In *2015 IEEE Virtual Reality (VR)*, pages 391–392, March 2015.

[3] Laremere. Simple vrpn wrapper for unity, 2014.

[4] B. Raffin, L. Soares, T. Ni, R. Ball, G. S. Schmidt, M. A. Livingston, O. G. Staadt, and R. May. Pc clusters for virtual reality. In *IEEE Virtual Reality Conference (VR 2006)*, pages 215–222. IEEE, 2006.

[5] A. Sigitov, D. Scherfgen, A. Hinkenjann, and O. Staadt. Adopting a game engine for large, high-resolution displays. *Procedia Computer Science*, 75:257 – 266, 2015.

[6] R. M. Taylor II, T. C. Hudson, A. Seeger, H. Weber, J. Juliano, and A. T. Helser. Vrpn: a device-independent, network-transparent vr peripheral system. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 55–61. ACM, 2001.

[7] D. Trebilco. Glintercept, 2006.