I Want My Virtual Friends to be Life Size! Adapting Second Life to Multi-Screen Projected Environments

Kip Haynes* Jacquelyn Morie Eric Chance
USC's Institute for Creative Technologies

Figure 1: Screenshot of Second Life in a synchronized three projector display

Overview

Second Life (SL) is a popular 3D online virtual world designed for human interaction (also known as a MUVE, or multi-user virtual environment). It typically supports 60-70 thousand concurrent users. The assets and physical environments within SL are easy to create and use, and the environments themselves are very much part of the human interaction experience. However, the typical means of accessing SL is through a single computer screen, which lessens the immersion that is inherent in such a rich 3D world. Because of this, the SL virtual world is a good candidate for adaptation to large scale immersive displays such as a CAVE™ or other multi projector systems.

We wanted to provide a solution that would utilize direct communication between SL viewers and have minimal dependence on 3rd party libraries or interfere with the OpenGL rendering pipeline While there are several ongoing efforts to adapt SL to a stereoscopic display, there are no freely available native adaptations to a CAVE™ like or other multi-screen environment at the time of this writing. There are several methods to distribute OpenGL based applications onto multiple computers and displays such as Chromium, [Humphreys et al. 2002] or VRizer [Berger et al. 2004]. However, most of these streaming applications either require changes to the rendering code or are no longer available or supported.

Approach

Our implementation uses multiple concurrent SL logins across an unlimited number of machines. A client-server relationship exists between the SL viewer that the user interacts with (leader) and all other viewers (followers). Inside the open source SL viewer, each viewer synchronizes whenever updateCamera() is called. Simultaneously the leader will broadcast the position, rotation and focus of the camera.

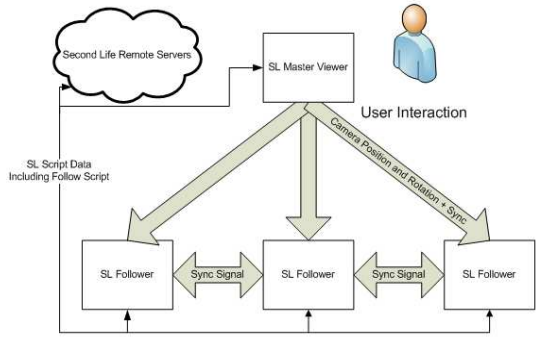

Figure 2: Diagram of interaction between SL viewers

Data and Display Synchronization

The follower viewers receive the position of the leader camera and synchronization signal. Each follower reads its assigned rotation from a local config file and performs the proper translation and rotation. A customized 2D rotation was written to handle the translation and rotation of the camera about the X axis. Some simple real time configuration functions have been added to assist in achieving the proper orientation and alignment. Additionally, each node reads its own FOV (field of view) info from a local config, because the Second Life FOV adjustment is difficult to set with any accuracy (slider bar).

The above described approach provides a well synchronized multi-viewer environment using direct communication between the clients. However, Second Life consists of multiple regions or "sims," which are sections of land that are often controlled by different servers. Initial tests revealed that when the leader crosses a sim boundary, the followers remain mapped to the original sim coordinates and show the camera positions in the old sim. This is because the local coordinates of each region or sim in Second Life only form a square grid whose coordinates number from 0 – 255 meters. This required a fix on the native scripting side, enabling the follower avatars to subtly follow the leader around the environment and into the next sim. Crossing sim boundaries is an elusive and undocumented process, and essentially means that a user is switching from one simulation server to another. To solve this problem, we created a vehicle object that the follower avatars could sit on and ride in order to follow the leader's camera around. In order to achieve a proper sim crossing, the movement of the leader is always cached and used as a trajectory for the follower vehicle to be able to cross the sim. In most cases this works well, but if the leader zigzags back and forth across a boundary, the follower can miscalculate the trajectory.

Requirements

Our implementation uses the Message Passing Interface (MPI) to handle all data and frame synchronization, remote launching and session management. It is multi-platform and is considered a low latency interconnect for high performance distributed systems.

Limitations

Currently the system is specifically designed for a large, three screen VR theater and only supports vertical rotation of the follower cameras. However, it would be very simple to add additional rotations for additional display environments. Also due to the rapid prototype of this project, the user currently has limited control of some of the native Second Life camera functions. This will be made available in a future release.

References

HUMPHREYS, G. HOUSTON, M. NG, R. FRANK, R. AHERN, S. KIRCHNER, J. KLOSOWSKI, P. Chromium: A Stream Processing Framework for interactive Rendering on Clusters. In *ACM Transaction on Graphics* 2002

BERGER, F. LINDINGER, C. ZIEGLER, W. VRizer -- Using Arbitrary OpenGL Software in the CAVE or other Virtual -- Environments. In *IEEE Virtual Reality* 2004

*e-mail: haynes@ict.usc.edu

SIGGRAPH 2010, Los Angeles, California, July 25 – 29, 2010.
ISBN 978-1-4503-0210-4/10/0007