# CalVR: An Advanced Open Source Virtual Reality Software Framework

Jürgen P. Schulze, Andrew Prudhomme, Philip Weber, Thomas A. DeFanti
University of California San Diego, Calit2, La Jolla, CA

## ABSTRACT

We developed CalVR because none of the existing virtual reality software frameworks offered everything we needed, such as cluster-awareness, multi-GPU capability, Linux compatibility, multi-user support, collaborative session support, or custom menu widgets. CalVR combines features from multiple existing VR frameworks into an open-source system, which we use in our laboratory on a daily basis, and for which dozens of VR applications have already been written at UCSD but also other research laboratories world-wide. In this paper, we describe the philosophy behind CalVR, its standard and unique features and functions, its programming interface, and its inner workings.

**Keywords:** VR framework, middleware, PC clusters, multi-user, Linux

## 1. INTRODUCTION

CalVR is a new virtual reality middleware system, which we developed from the ground up. CalVR implements the core functionality of commonly used existing virtual reality middleware, such as CAVElib, VRUI, FreeVR, VR Juggler or COVISE. It adds to those that it supports several non-standard VR system configurations, multiple users and input devices, sound effects, and high level programming interfaces for interactive applications. CalVR consists of an object oriented class hierarchy which is written in C++. It was developed for CentOS Linux, but can run in most other Linux systems, and even Apple Mac OS. CalVR uses the OpenSceneGraph (OSG) library for its graphics output, and through OSG it supports native OpenGL code as well. Functionality and entire VR programs can be added through a simple plug-in system which allows compiling new modules separately from the main code. CalVR has several built-in navigation methods, an extensible 3D menu system, support for a variety of 3D display and tracking systems, as well as support for collaborative work at different network-connected sites. Over the past two years, we have used CalVR in a large variety of research and development projects.

In the following chapters, we give a thorough overview of CalVR's software structure, its capabilities, and we will discuss what type of applications CalVR was developed for, and what its limitations are. We will also give typical examples for applications built on top of CalVR.

## 2. RELATED WORK

Commercial virtual reality authoring systems, such as WorldViz,[1] EON Studio,[2] COVISE[3] or AVANGO[4] allow users to build custom VR applications, but they are not open source and only allow the programmer to implement things they anticipate. If they do not anticipate that multiple users with multiple input devices will use a system, they will not support that. If they do not anticipate a certain display technology, such as auto-stereo to be used, they will not allow the user to add that functionality.

Another class of VR frameworks are designed to be rather limited in scope, such as Mechdyne's CAVElib[5] and Carolina Cruz-Neira's VR Juggler[6] which mostly abstract from PC clusters to drive a multi-screen display device, and which support a variety of tracking systems, but they leave higher level tasks to the programmer, such as navigation or menu widgets.

Some existing open source VR systems are just as extensible as CalVR and can for many tasks be used interchangeably with it. Examples are Oliver Kreylos' VRUI,[7] Bill Sherman's FreeVR[8] or Fraunhofer IAO's Lightning.[9]

CalVR integrates OpenSceneGraph[10] (OSG) for its graphical output. OpenSceneGraph was modeled after SGI Performer,[11] and is syntactically very similar, but has evolved into its own programming library over the many years of its existance. OpenSceneGraph alone will not drive VR systems. It lacks cluster-ability, tracking support, a way to describe multi-display layouts, as well as a menu widget and interaction system.

## 3. OVERVIEW AND STANDARD FEATURES

Many of CalVR's features are standard for modern VR frameworks. Among these are:

- CalVR is being developed for CentOS Linux, but also compiles under Apple's Mac OS. Most of it even compiles and runs under Microsoft Windows but without cluster support.
- Implemented in C++: the entire framework has been written in modular, object-oriented C++. For example, there are classes for screen configuration, user interaction, navigation, socket communication, tracking, configuration file handling, etc.
- CalVR uses the CMake[12] build system to compile its source code.
- Support for clusters of networked rendering PCs. Each cluster node runs a copy of CalVR via remote ssh from the head node, and by default all nodes load the same data. Frame buffer swaps are synchronized over TCP. Views are synchronized by distributing user input events to all rendering nodes. While CalVR can run with slower networks, 1 Gbps or more is preferred. A multi-cast channel is supported for fast data broadcast from the head node to the rendering nodes, for instance to playback live video.
- Built on top of a scene graph library: CalVR uses OpenSceneGraph as its graphics library. OSG not only provides a layer of abstraction on top of OpenGL, but also import routines for graphics file types, intersection testing, and a variety of node kits for additional functionality.
- Through OSG, CalVR has support for most stereo modes (active, passive, interlaced, etc). CalVR can be configured to run with Twinview, Xinerama, as well as separate displays.
- A plugin system allows the creation of applications using CalVR, without having to re-compile the entire CalVR system. Plugins are loaded dynamically at run-time by libDL.
- All display configuration, tracking, framework and plugin parameters are specified in an XML configuration file, which is parsed with MiniXMl.[13] CalVR's coordinate system has X pointing to the right, Y forward and Z up.
- CalVR comes with a customizable menu system, allowing straightforward integration of user-adjustable parameters into plugins. Figure 1 shows two examples of this menu. We call this default menu style the "Board Menu".

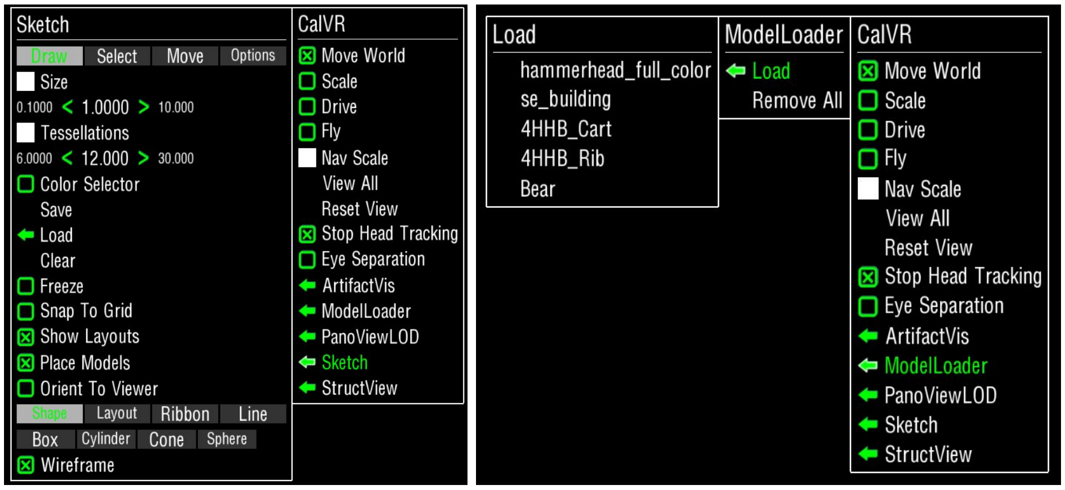

Figure 1. Left image: CalVR's Board menu style main menu on the right, two levels of sub-menus are appended towards the left of it. Right image: Board menu with all menu item types.

- Support for standard tracking libraries, such as VRPN and TrackD. A custom VRPN server supports USB devices, such as 2D mice or presentation devices, accessing them through libUSB, so that mouse input will not accidentally trigger events on the computer the mouse is connected to.
- Default navigation modes for “fly”, “drive”, and “scale” are available for use by all plugins. Additional navigation modes have to be implemented by a plugin.
- Multiple sites running CalVR can be networked to share view points and work collaboratively with data.
- We created an OpenAL-based audio server for sound effects, which runs under Linux and can, for instance, run on the head node of a clustered VR system. A C++ class encapsulates the audio server’s functionality.

Figure 2 breaks out the different parts of CalVR. CalVR itself is built on top of OSG, which in turn builds on OpenGL. The Menu API currently supports two menu widget libraries: the Board menu and the Bubble menu. CalVR uses a set of device drivers, for instance for a Kinect or a Ring Mouse, and it allows custom plugins to run.

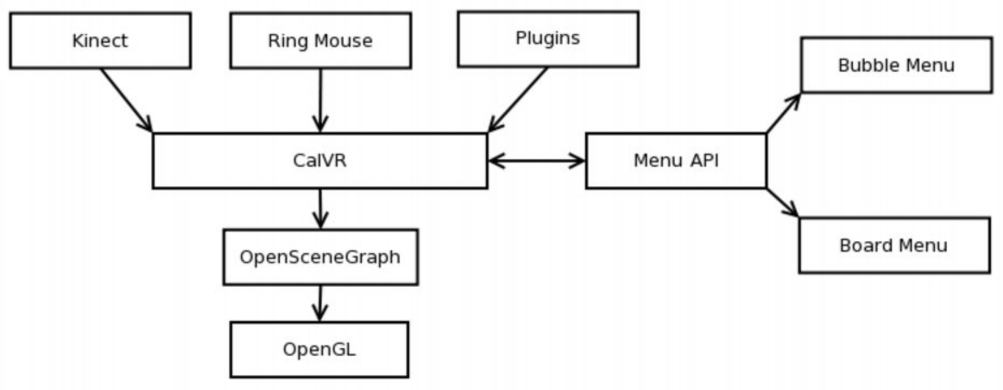

Figure 2. CalVR’s software design.

## 4. UNIQUE FEATURES OF CALVR

One of the reasons to start implementing an entirely new VR middleware system was that we wanted to add functionality to existing VR frameworks but found that this was sometimes hard to do without completely changing some of the framework’s inner workings, which would have been either impossible because of licensing restrictions, or prohibitive given that existing users of the system would have had to modify all their custom code. In the following sections we will introduce CalVR’s unique features.

### 4.1 Multi-User and Device Support

CalVR supports multiple head-tracked users, all of whom potentially with multiple input devices. We even developed a novel “viewer democratization” approach,[14] which renders from a camera position equally good or bad for all users. The number of users and input devices is specified in CalVR’s configuration file.

### 4.2 Additional Rendering Modes

CalVR supports additional rendering modes, on top of what OSG comes with. Among those are: support for autostereoscopic multi-view displays by means of a library by Robert Kooima.[15] An anti-aliased mode for interlaced passive stereo displays (it renders above and below images anti-aliased with multi-sampling, then combines them with a GPU shader into an interlaced pattern. An omnistereo mode in which each screen is rendered with the head orientation set so that the head looks straight at the screen. And finally, a mode in which a dedicated, custom head position can be specified for each screen of a multi-display system - this allows the creation of convex tiled display surfaces, which are viewed from the outside by many users without head tracking.

## 4.3 Scene Objects

CalVR's API contains additions to improve interactive applications with multiple, independent geometrical objects. Our "SceneObject" class manages navigation, interactions and context menu related to one particular object. The context menu pops up with a click of a dedicated button on the wand (similar to right-clicks in Windows), see Figure 3. Its default functionality offers control over movability and navigation. Additionally, the programmer can add custom object-specific functionality to this menu. SceneObjects can be hierarchically nested, which gives the programmer control over how interactions are interpreted. To speed up interaction with many SceneObjects, the intersection test for the wand cursor first checks for bounding sphere intersections with the SceneObjects, and only if those are found does it do a more accurate bounding box check.

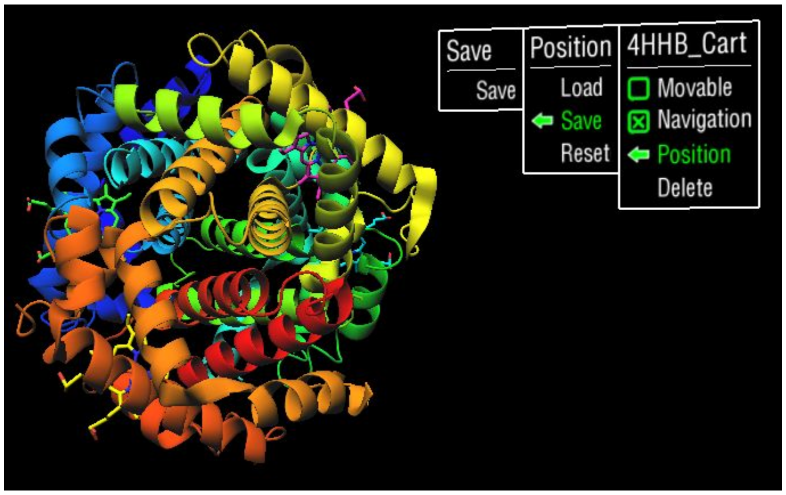

Figure 3. Protein structure with context menu, as well as two levels of sub-menus thereof.

## 4.4 Support for Tiled Display Walls

While initially developed for 3D VR systems, CalVR has special extensions for tiled display walls with monoscopic displays, such as the one in Figure 4. An example is the "TiledWallSceneObject", which is derived from the "SceneObject" class and always stays in the plane of the wall (only moves horizontally and vertically and does not rotate), regardless of user input. Another extension for 2D walls is the pointer class interactor, which we designed to improve the interaction with a tiled display wall using a 2D mouse: the mouse cursor remains in the plane of the displays and is constrained to the wall's bounds. CalVR's multi-user capability allows support for multiple such 2D cursors, which could be controlled by different people with separate mice.

## 4.5 Layered Menu System

CalVR's menu system consists of two layers, separating hierarchy and functionality of a menu from its GUI widget implementation. This allows the rapid creation of new GUI widget libraries, which will even be compatible with existing plugins and can be used interchangeably. We recently implemented a new, sphere-based menu system, which we call the Bubble Menu. Once it was implemented, we were able to run unmodified existing plugins with a Bubble Menu. Figure 5 shows a straight on view of a bubble menu, as well as a picture of it running in a tiled display system.

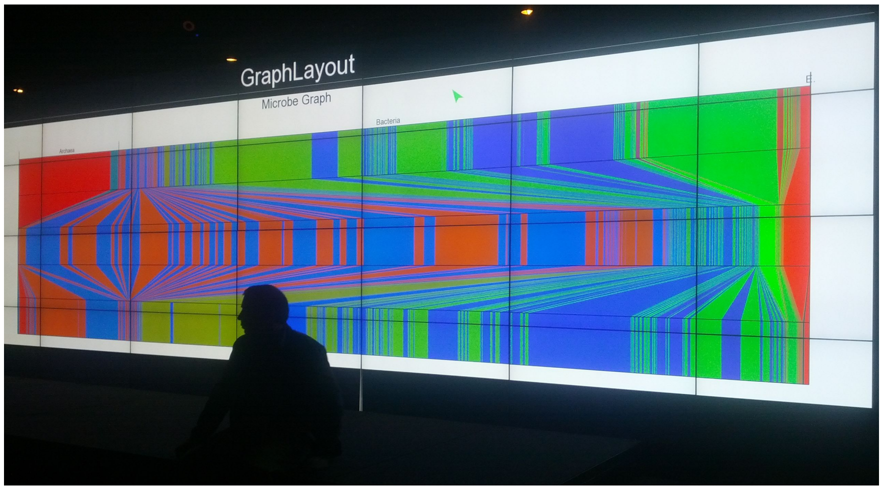

Figure 4. CalVR showing 2D graphs on a tiled display wall. Notice the green 2D cursor in the top center.

## 4.6 Asynchronous Tracking

Many VR systems poll tracker information at the same rate as they do frame updates. This means that when an application runs at a low frame rate, input events will get lost when they happen in-between two frame updates. CalVR's tracking sub-system runs in a separate thread at a constant pull rate of 60Hz. This reduces the likelyhood of lost tracker events, and allows support of tracking events happening on a short time scale, such as double clicks or gestures.

## 4.7 Custom Cull Visitor

CalVR includes a custom OSG cull visitor to help avoid lengthy loading times when running on graphics clusters. Special OSG nodemask bits are used to detemine if a node was shown before, if not display lists are forced to load. This feature can be disabled by blanking a different OSG nodemask bit. This functionality addresses a problem with loading a large data set in a cluster-based VR system: each node will by default only load the data set once part of it intersects the view frustum. Because at startup the data set might only be visible on a few screens, there will be loading delays whenever camera or data set move and the data is shown on a new screen. The custom cull visitor forces loading of the data set at startup, regardless of data set visibility.

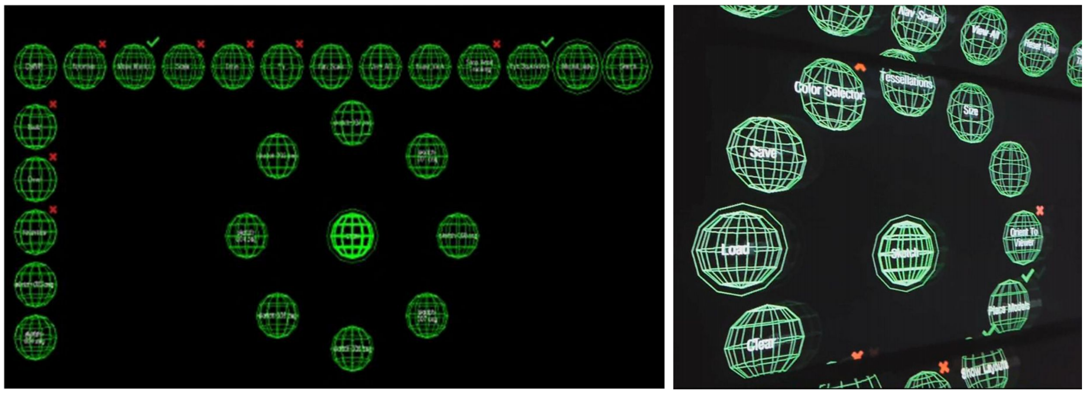

Figure 5. Left image: CalVR's bubble menu: the top row is the main menu, the left column is shortcut bar, which does not have an equivalent in Board menus. The bubbles arranged in a circle are sub-menu items. Right image: Bubble menu in the NexCAVE, an LCD-based VR system.

# 5. PLUGINS

CalVR's plugin system was mentioned before, but it deserves a closer look, which we are going to give it in this chapter.

Generally, all new applications for CalVR are added in the form of plugins. These plugin modules implement a virtual base class in the CalVR core and compile into a separate shared library. At runtime, this library is loaded and the plugin receives callbacks from the CalVR core through the implemented virtual functions.

## 5.1 Plugin Base Class

Every plugin will need to have a class that inherits from the CalVR plugin base class. This is the CVRPlugin class, defined in `kernel/CVRPlugin.h`. The main plugin class may implement any needed virtual functions from `CVRPlugin.h` to receive calls from the CalVR core during runtime. The most frequently used "callback" functions are:

- `bool init()`: Called once right after the plugin is created. A return of true means the init went okay and false means there was some failure and the plugin should be unloaded. This is a good place to do one-time setup operations like creating menus or a base scene graph structure.
- `void preFrame()`: Called once each frame right before the scene is rendered. This is where things like scene graph changes could take place and affect the current frame.
- `void postFrame()`: Called once each frame after rendering and buffer swap (for double buffered rendering).
- `bool processEvent(InteractionEvent* event)`: This function allows the plugin to use events generated by the CalVR core. These events are things like button presses, keyboard presses, valuators, etc. If true is returned the event is consumed, otherwise it continues through the event pipeline.
- `void message(int type, char* &data, bool collaborative=false)`: Called when the plugin is sent a message from another plugin or a collaborative session.
- `int getPriority()`: Returns a priority value for this plugin. This function is called once when the plugin is loaded. It is used to order all loaded plugins. A plugin with a higher priority gets function calls and events before lower priority plugins. The default is 50. Plugins with equal priority are ordered alphabetically by name. This is designed to allow enforcing a relative order of plugin execution or event handling.

All plugins must also include a macro in the main .cpp file: `CVRPLUGIN(PluginName)`. `PluginName` is the name of the class implementing the `CVRPlugin` class. This creates a small function that is used as an interface into the plugin shared library.

# 6. EVENTS

CalVR uses an event handling system with an event queue. All user input is turned into events. Events are generated by a tracking system driver, running on the head node. There is a variety of event types, implemented as sub-classes of the `InteractionEvent` class, for instance `HandInteractionEvent`, `Mouse InteractionEvent`, `ValuatorInteractionEvent`, or `KeyboardInteractionEvent`.

Events are synchronized across the cluster, then sent through each cluster node's event pipeline. The event pipeline is processed in the following order. Once the event is consumed, it is removed from the pipeline. The menu system has first access to the event pipeline, then scene objects, then plugins (sorted by priority), finally navigation.

## 7. MENUS

Our abstract description of the menu hierarchy supports the following menu item types: action buttons (trigger an event when clicked, don't have a state), check boxes (have on and off states), image (to put an image in the menu), list (provides a number of items to choose one from), range (allows selecting a value), text (just a label), and sub menus (open another menu). Once a menu item is created, a callback function can be connected to it, in order to process events triggered by the menu item.

## 8. MESSAGE PASSING

CalVR can pass messages from one plugin to another on the same node, or from head node to rendering nodes and vice versa. When passing messages locally on a node, they can be passed within the same plugin, or to another plugin running on the same machine, specified by the plugin name. Messages consist of an integer message type value and an optional data pointer.

Alternatively, plugins can send and receive data across the cluster. Send and receive commands always need to be coupled, to avoid deadlocks. These commands can either be sent from the head node to all rendering nodes, or from the rendering nodes to the head node. In the latter case, the head node receives an array of messages, one from each rendering node.

## 9. COLLABORATIVE MODE

CalVR comes with a special plugin, the `CollaborativePlugin`, which allows connecting multiple CalVR sessions over the network. For the communication to work, the standalone command-line tool `CollabServer`, which is also part of CalVR, needs to be run on one of the participating computers, or on another machine on the network. `CollabServer` shows the names of all connected clients, as well as all their tracking data. Plugins can use the `CollabServer` to send messages to all connected clients. Synchronization of tracking data is done by distributing these to all clients.

## 10. TERRAIN RENDERING

Many applications benefit from or require the ability to show data in its geographical context. Google Earth was designed for this purpose, but it cannot easily be integrated into an OSG-based application. We chose osgEarth, which is an OSG-based rendering engine, to be our terrain engine for geo-science applications. Figure 6 shows osgEarth running in our NexCAVE. A special adaptation we had to make for osgEarth is that the earth covers a huge range of depth because it is so big, which is in stark contrast to the menus or objects near the user. In order to render geometry correctly and without z fighting both near and far, we employ a two-step rendering approach: first we set OpenGL's near and far planes to encompass the earth and call osgEarth's rendering routine, and render all geometry attached to the earth's scene graph node. Then we set near and far to CalVR's default or user selected values and render the rest (near part) of the scene.

## 11. CONCLUSION AND FUTURE WORK

We presented our new VR framework, CalVR. We described its unique features, and gave an overview of what it can do and how it works.

In the future, we will further develop CalVR into an even more flexible, multi-user VR system. We have already started adding support for smart phones and tablets, which we use to control plugin behavior, or even to replace our 3D wand with them. We also already have support for the Microsoft Kinect, but we have yet to come up with more robust ways for user interaction before we can replace the wand with a gestural interface.

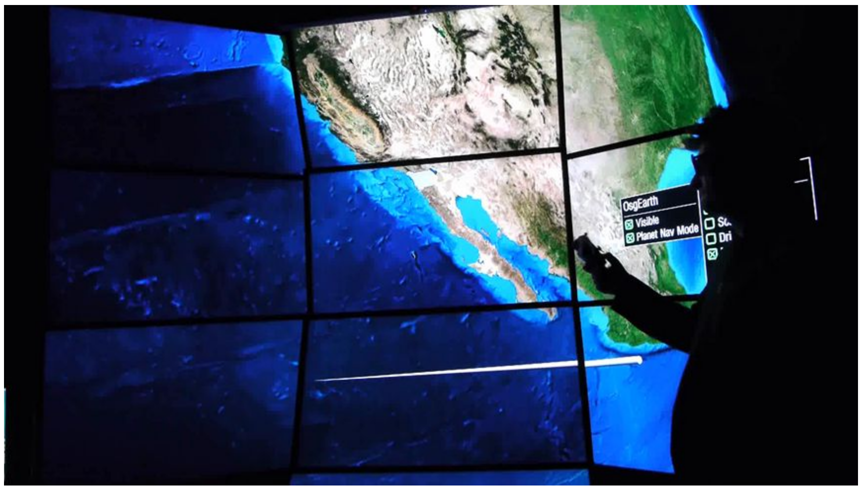

Figure 6. CalVR with osgEarth plugin in 3DTV-based NexCAVE.

## REFERENCES

[1] WorldViz, "Worldviz home page, *http : //www.worldviz.com/*," (2012).

[2] EON, "Eonstudio home page, *http : //www.eonreality.com/products*$_s$*tudio.html*," *EON* (2012).

[3] Rantzau, D., Frank, K., Lang, U., Rainer, D., and Woessner, U., "Covise in the cube: An environment for analyzing large and complex simulation data," *Proceedings of the IPTW* (1998).

[4] Tramberend, H., "Avocado: A distributed virtual environment framework," *PhD Dissertation, University of Bielefeld* (2003).

[5] Mechdyne, "Cavelib home page, *http : //www.mechdyne.com/cavelib.aspx*," (2013).

[6] Bierbaum, A., Just, C., Hartling, P., Meinert, K., Baker, A., and Cruz-Neira, C., "Vr juggler: a virtual platform for virtual reality application development," in [*Virtual Reality, 2001. Proceedings. IEEE*], 89 –96 (march 2001).

[7] Kreylos, O., "Environment-independent vr development," in [*Advances in Visual Computing*], Bebis, G., Boyle, R., Parvin, B., Koracin, D., Remagnino, P., Porikli, F., Peters, J., Klosowski, J., Arns, L., Chun, Y., Rhyne, T.-M., and Monroe, L., eds., *Lecture Notes in Computer Science* **5358**, 901–912, Springer Berlin Heidelberg (2008).

[8] Sherman, B., "Freevr: Virtual reality integration library, *http : //www.freevr.org/*," (2008).

[9] Blach, R., Landauer, J., L, J., Rsch, A., and Simon, A., "A highly flexible virtual reality system," *Future Generation Computer Systems, 14(3-4)* , 167–178 (1998).

[10] OpenSceneGraph, "Openscenegraph home page, *http : //www.openscenegraph.org/*," (2013).

[11] Rohlf, J. and Helman, J., "Iris performer: a high performance multiprocessing toolkit for real-time 3d graphics," in [*Proceedings of the 21st annual conference on Computer graphics and interactive techniques*], *SIGGRAPH '94*, 381–394, ACM, New York, NY, USA (1994).

[12] CMake, "Cmake: Cross-platform make, *http : //www.cmake.org/*," (2013).

[13] MiniXML, "Minixml: Lightweight xml library, *http : //www.minixml.org/*," (2013).

[14] Schulze, J., Acevedo, D., Mangan, J., Prudhomme, A., Nguyen, P., and Weber, P., "Democratizing rendering for multiple viewers in surround vr systems," in [*3D User Interfaces (3DUI), 2012 IEEE Symposium on*], 77 –80 (march 2012).

[15] Kooima, R., Peterka, T., Girado, J., Ge, J., Sandin, D. J., and DeFanti, T. A., "A gpu sub-pixel algorithm for autostereoscopic virtual reality," in [*IEEE Virtual Reality Conference, VR 2007, 10-14 March 2007, Charlotte, NC, USA, Proceedings*], Sherman, W. R., Lin, M. C., and Steed, A., eds., 131–137, IEEE Computer Society (2007).