Omegalib: a Multi-View Application Framework for Hybrid Reality Display Environments

Alessandro Febretti, Arthur Nishimoto, Victor Mateevitsi, Luc Renambot, Andrew Johnson, Jason Leigh

Electronic Visualization Laboratory, University of Illinois at Chicago

ABSTRACT

In the domain of large-scale visualization instruments, hybrid reality environments (HREs) are a recent innovation that combines the best-in-class capabilities of immersive environments, with the best-in-class capabilities of ultra-high-resolution display walls. HREs create a seamless 2D/3D environment that supports both information-rich analysis as well as virtual reality simulation exploration at a resolution matching human visual acuity. Co-located research groups in HREs tend to work on a variety of tasks during a research session (sometimes in parallel), and these tasks require 2D data views, 3D views, linking between them and the ability to bring in (or hide) data quickly as needed.

In this paper we present Omegalib, a software framework that facilitates application development on HREs. Omegalib is designed to support dynamic reconfigurability of the display environment, so that areas of the display can be interactively allocated to 2D or 3D workspaces as needed. Compared to existing frameworks and toolkits, Omegalib makes it possible to have multiple immersive applications running on a cluster-controlled display system, have different input sources dynamically routed to applications, and have rendering results optionally redirected to a distributed compositing manager. Omegalib supports pluggable front-ends, to simplify the integration of third-party libraries like OpenGL, OpenSceneGraph, and the Visualization Toolkit (VTK).

We present examples of applications developed with Omegalib for the 74-megapixel, 72-tile CAVE2™ system, and show how a Hybrid Reality Environment proved effective in supporting work for a co-located research group in the environmental sciences.

Keywords: Multi-view, Tiled Displays, Cluster, Immersive Environments, Middleware

Index Terms: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism – Virtual Reality; I.3.2 [Computer Graphics]: Graphics Systems – Distributed graphics; K.4.3 [Computers and Society]: Organizational Impacts – Computer-supported collaborative work

1 INTRODUCTION

Today, most research involves the analysis of scientific phenomena of ever-increasing scale and complexity, and requires the concentrated effort of interdisciplinary teams: scientists from different backgrounds whose work involves large, heterogeneous data sources. As the scale and complexity of data continue to grow, large scale visualization instruments like display walls and immersive environments become increasingly essential to researchers, letting them transform raw data into discovery. In particular, immersive systems are an attractive option for exploring 3D spatial data such as molecules, astrophysical phenomena, and geoscience datasets [1]. On the other hand, display walls with their high resolution help interpret large datasets, offering both overview and resolution, or can be used to lay out a variety of correlated visual data and documents for collaborative analysis [2].

Current technology trends lead to the production of larger, affordable thin-bezel LCD monitors with built-in support for stereoscopic 3D [3]. Such recent advancements made it conceivable to merge the best-in-class capabilities of immersive Virtual Reality systems, such as CAVEs, with the best-in-class capabilities of Ultra-high-resolution Display Environments, such as OptIPortals [4] thus creating conceptually new immersive environments which we refer to as Hybrid Reality Environments (HREs), such as the CAVE2 system [5], [6] (Fig. 2)

Naturally, hardware is only half of the equation. To fully take advantage of these novel technological affordances we need a matching software infrastructure that facilitates scientific work within HREs. This infrastructure should be tailored to the real-world needs of research teams, taking into account the way users interact with the instrument, with their applications and with co-located or remote members of their team. So far, software development for display walls and immersive systems has followed independent paths. On display walls, the focus has been on supporting scalable ultra-high resolution visualization, managing multiple views of heterogeneous datasets and co-located collaboration. On immersive environments, the effort is to provide low latency user-centered stereo, naturalistic interaction and remote collaboration.

What is now needed is a convergence in software design for display walls and immersive environments: we envision the integration of display wall software, an immersive environment framework and additional components into an "operating system" for Hybrid Reality Environments (Figure 1). This operating system aims to solve two major challenges with alternative approaches:

1) *Static spatial allocation of 3D and 2D views:* although an HRE is capable of displaying 2D and 3D content at the same time, software relies on static configurations describing how the physical display space should be split into 2D and 3D views. Multiple predefined configurations give the end users some flexibility, but require restarting and resetting all running applications
2) *Lack of unified interaction*: the 2D and 3D portions on an HRE often rely on inconsistent interaction schemes: for instance pointing semantics may use absolute movement on the 3D half and relative movement on the 2D half. Physically separate interaction devices may be required. Or, when a single device is used, interaction may lead to conflicting results (i.e. navigating a 3D view moves 2D views).

The HRE operating system also needs to satisfy the requirements of two distinct, but often overlapping, categories of users: scientific application users (i.e. research teams) and scientific application developers.

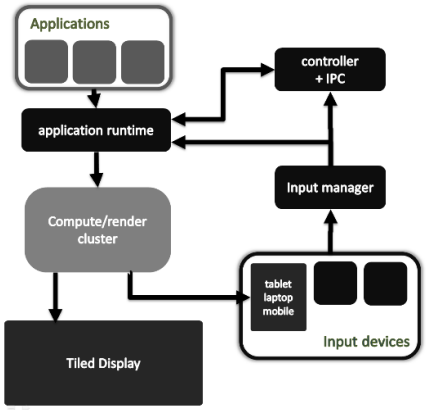

Fig. 1. The high level model for a multi-view operating system for Hybrid Reality Environments. The core components of the operating system are: the distributed application runtime; a controller and inter-process communication manager that handles application lifetime and communication between application instances; a distributed input manager capable of handling heterogeneous devices.

Fig. 2. An overview of the CAVE2 Hybrid Reality Environment. CAVE2 is based on a cylindrical setup composed by 18 columns of 4 displays each. This arrangement provides a panoramic view of 320 degrees. Each display pair is driven by a separate computer, for a total of a 36 computer cluster, plus a head node. CAVE2 uses an optical motion tracking system arranged in a circular configuration above the displays.

1.1. End-User Requirements

Users need a system that lets them easily manage display space and content. We can for instance consider a structural engineering task: engineers wanting to compare different designs of a building load a digital model and explore it in full-scale in the Hybrid Reality Environment. They decide to compare two variants of the design side-by-side, splitting the available screen space in two. The group then chooses to look at pictures of the target site: they hide one of the 3D visualizations, and use the now available screen space to share photographs. The team splits: one group discusses the site while another user navigates the building model. The user notices a flaw in the design and interrupts the rest of the team. He generates a section of the building that gets displayed on a separate 2D view. The team now brings up the alternate design again, observes the 3D visualizations and 2D sections, until they agree on one of the design variants. Before closing the meeting, they display this variant on the full display again, and mark a few points for future revision. This kind of work pattern can be observed in research groups in a variety of disciplines: co-located collaboration often entails multiple phases that alternate full group work with individuals or sub-groups working in parallel: moreover the work may focus on a single issue, on different views into that single issue, or on independent issues [7]–[9]: the ability to easily re-configure the display space is fundamental to support such heterogeneous tasks [10].

1.2. Developer Requirements

Considering the needs of HRE users is not enough: an HRE operating systems needs to provide an easy, yet powerful application programming interface (API) that developers can leverage to implement custom software. Immersive applications are often created to serve interdisciplinary research teams whose members may have limited programming experience. Applications also have a great variance in their complexity: some only need basic visualization and interaction with 3D models. Others require the implementation of complex and custom visualization techniques, need low-level access to the graphics API or need to use specialized visualization libraries. A way to satisfy these requirements is to provide a layered API, with multiple levels of access that offer a tradeoff between complexity and control.

2 Related Work

The benefits of co-located collaborative work have been investigated in a variety of contexts like design [10], software development [7], and engineering [8]. Advantages of co-location include reduced communication friction and distributed cognition: teammates exploit features of the social and physical world as resources for accomplishing a task [11].

The requirements for effective visualization in large scale environments have also been investigated in the past. In terms of data size and format, some scientists need to display data so massive that it exceeds the screen resolution of even a large display [12]. Others find a large display ideal for comparing different data sets of the same phenomenon. Lastly, large display surfaces can help aggregate heterogeneous data about a specific phenomenon [13]. This third scenario (heterogeneous multiple views) is the most common [9], [14], [15]. In [14], the authors note how large display surfaces let researchers organize large sets of information and provide a physical space to organize group work. Part of the work is spent with each user individually working on a separate part of the display. Another part is dedicated to collaborative data consolidation and analysis, so everybody in the group can aggregate and discuss partial findings. During this kind of work, researchers often need to bring in new data and documents from their laptops or from the web: thus, a single pre-packaged application running on a large display rarely covers the full needs of a research group.

Important factors for an efficient co-located collaborative space are, among others, the physical design of the space (space should support both the work of the entire team, and separate sub-teams), and its reconfigurability (both physical and at the software level). Bowman et al. have conducted extensive research on the advantages, challenges and best practices on hybrid 2D/3D visualization, proposing a taxonomy for Information-Rich Virtual Environments (IRVEs). In [16] the authors categorize visualizations depending on the physical and logical positioning of 2D and 3D displays, and in [17] they present a hybrid molecular visualization application for the CAVE system [18] that statically assigns 2D and 3D content to CAVE walls.

2.1. Current Software Approaches

Most research on co-located use of immersive environments concentrated on supporting multi-user stereo vision in a single immersive view. Little work has been done on supporting multiple independent views, or multiple simultaneous applications, within an immersive environment. VR software toolkits like FreeVR [19], VRJuggler [20], CAVELib [21] and CalVR [22] support a single running application at a time. On the other hand, wall display software like CGLX [23], DisplayCluster [13] and the Scalable Adaptive Graphics Environment (SAGE) [24] are designed to allow sharing the available display space between multiple applications. CGLX works by distributing application execution, while DisplayCluster and SAGE work on distributing pixels. An advantage of pixel distribution is flexibility. Since rendering and display resources are decoupled, applications can run on a single node, on a local cluster or on a remote location. The wall display software acts as a *distributed compositing manager* that routes pixels between applications and displays. Moreover, SAGE offers multi-user interaction support and allows input redirection from the wall to applications. Display wall software solutions are not full-fledged application development frameworks: the application developer is still in charge of managing rendering, distribution, interaction, etc.

The Equalizer framework [25] is an application development framework that focuses on scalable parallel rendering. It offers fine-grained, transparent control over rendering resource allocation, although this control is mostly static, specified through a rather complex configuration file.

An observation emerging from the review of current software is that display wall software and scalable rendering software both have features that are desirable in a multi-view HRE software framework. We therefore choose to play on these strengths, and envision a framework based on the integration of successful designs in both fields. The remainder of this paper will present this framework, called Omegalib, detailing our integration approach, and the addition of several novel features (like dynamic reconfigurability and input filtering) that make this framework effective in supporting applications in Hybrid Reality Environments. Moreover, we will present examples of practical use of these features in a co-located collaborative setting.

3 Framework Design

Omegalib acts as an abstraction layer between the HRE hardware and applications. At the back end, Omegalib uses Equalizer to drive the display system. In section 2.1, we observed how Equalizer uses an advanced and verbose syntax to support its flexibility, making it complex and error prone to write or maintain configuration files. Omegalib addresses this issue by describing display system geometry, cluster configuration and input capabilities through a compact system description file. When applications launch, the runtime generates and forwards the extended configuration to Equalizer, which then proceeds to initialize and setup the display system. At the front end, Omegalib integrates several APIs and visualization tools through an abstract scene graph and pluggable render passes. The abstract scene graph makes it possible to decouple object interaction techniques from concrete object representations. A concrete object can be an OpenSceneGraph node, a Visualization Toolkit (VTK) Actor, or some other entity provided by one of the Omegalib front-ends. Objects can be attached to nodes of the scene graph: object transformations, visibility, hierarchy, etc. can therefore be controlled through a unified API that is independent from the library used to draw the objects.

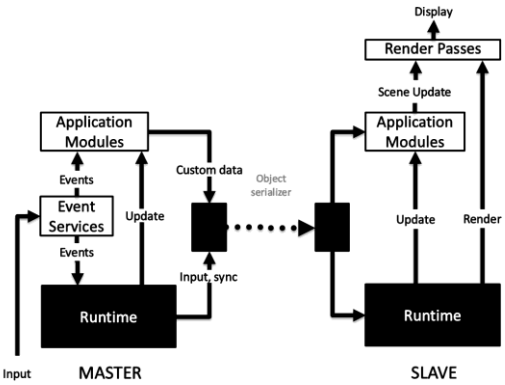

Fig. 3. An overview of the communication flow of a distributed Omegalib application. In this example, the Master node runs in headless mode, and only takes care of application and input event processing.

3.1. Application Model

The default execution and rendering mode for Omegalib is replicated execution: each node in a display cluster runs an instance of the target application, while a master instance synchronizes the updates and buffer swaps across the nodes. It is also possible to control execution on a node-by-node basis (the API offers functions to check on what node the application is executing), and synchronize custom data objects between master and slave instances. Omegalib extensions and applications are implemented as three component types that communicate with the Omegalib runtime and with each other, as represented in Fig. 3: Event Services, Application Modules and Render Passes.

Event Services implement input processing. A service may implement support for a physical input device (like a keyboard or mouse controller), or it can aggregate and reprocess events generated by other services. Event services can also receive input data from a remote server, making it possible to have a separate machine as the hub for all input devices in an HRE installation. Event services run only on the master instance of a distributed application. The Omegalib runtime takes care of serializing and distributing events to slave instances.

Application Modules contain the core of an Omegalib application logic. Modules can receive input events, update the abstract scene graph (or other custom scene representations), and communicate with other modules. Modules run on both master and slave instances of Omegalib regardless of whether they display content or not.

Render Passes implement all the functionality needed to render scenes to cluster tiles or to secondary image streams. Render passes are typically implemented by integration front-ends: application developers don't need to access them, unless they need to perform custom low-level drawing operations. Render Passes expose a GLUT-like callback interface that can be used to execute OpenGL operations. Render passes can be prioritized and can perform 2D or 3D drawing. For instance, a high priority 2D rendering pass can be used to overlay 2D content on top of an interleaved-stereo 3D scene. Render Passes run only on nodes in charge of drawing: a headless master configuration will not run render passes on the master application instance. The rendering system can also be configured to render to additional outputs: applications can generate secondary views whose pixels can be streamed to compositing managers such as SAGE, displaying an Omegalib secondary view as a 2D window on a portion of the tiled display.

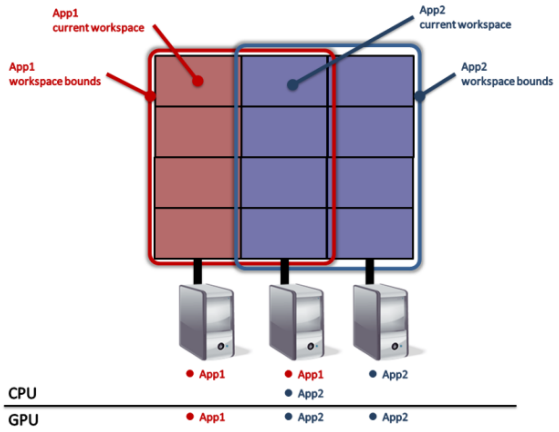

Fig. 4. An example of dynamic workspace configuration for two running applications. Each machine controls one display column. Applications are launched with overlapping workspaces on the central column. In this example, the runtime configuration allocates the central column to Application 2.

3.2. Dynamic Configuration

As mentioned in the introduction, one of the objectives of Omegalib is the creation of user-defined and reconfigurable workspaces: areas of the display that are dedicated to a 2D or 3D application, that can be re-defined while the application is running and can be independently controlled. To support this feature, we let users specify an optional 2D display region as a startup argument. The runtime uses this information to find the subset of rendering nodes that drive displays in the selected region: it then adjusts the system configuration to launch the application only on the identified nodes. Multiple applications can be launched on the system in this fashion. The runtime also manages networking setup, guaranteeing that each application instance uses an independent set of ports for master/slave communication.

To increase the flexibility of workspaces, we also let users expand or shrink the visible area of each workspace within the runtime bounds. When a user shrinks a workspace, the runtime disables rendering on the nodes whose tiles are not covered by the workspace active area (Fig. 4). While GPU resources are de-allocated, the application remains available on the machine: if the user later decides to expand the workspace again, rendering on the inactive tiles can be reset almost instantaneously.

It is therefore possible to run applications on overlapping regions of the display system: the display space shared by multiple regions can be associated to any of the overlapping applications at runtime. This double level of dynamic configuration (launch-time and runtime) provides a good level of control over cluster resource usage versus workspace flexibility. On one end, applications can be launched on non-overlapping regions, optimizing cluster resource allocation (each node is dedicated to a single application) but giving up dynamic display allocation. On the other end, applications can be launched on fully overlapping regions covering the entire display space: in this case the workspaces have the full runtime flexibility, at the cost of sub-optimal cluster resource allocation (each node needs to keep one active instance of each application).

3.3. Input Filtering

When the HRE is running multiple applications, it is necessary to let users interact with any of them without requiring switching to a different physical device or other complex interactions. It is also desirable to let different users control different workspaces and let them easily switch between them. In Omegalib, this feature is implemented through ray-based event filtering. We assume that the main interaction device in the environment offers 6DOF tracking (as is the case for most large scale immersive environments). We use the tracking data to generate a 3D ray starting at the device, and compute whether the ray intersects one of the display tiles. Only the application that is controlling the tile (based on its runtime configuration) will process the event stream generated by the devices. This scales to multiple devices controlling independent applications. It is also possible to choose another 6DOF tracking source as the input to the event filter: for instance, a pair of tracked glasses can be used to filter input events based on gaze direction. Each application has control over its own event filtering policy. An application can also decide to disable filtering if it needs to let users interact with it regardless of its display state.

3.4. Panoptic Stereo

Since Omegalib is aimed at supporting co-located collaborative groups, it is fundamental to provide stereo-viewing capabilities to multiple users in the system. One issue with standard user-centered stereo is eye reversal when the tracked user looks 180 degrees away from a given screen. This is acceptable for a single user, since he or she is no longer looking at the screen, but it makes the stereo view unusable for other team members. To solve this issue, Omegalib supports *panoptic stereo*. Techniques similar to Panoptic stereo have been implemented in the past to support multi-user stereo without generating multiple views [26]. When Panoptic stereo is enabled, the system tracks the main viewer's position, but generates head orientation information based on each display plane normal. The stereo frustum is projected outward to each display panel, preserving stereo separation for each user in the system. Another benefit of Panoptic stereo is its independence from frame latency: When users move their head around, stereo will look correct regardless of the application frame rate, leading to a more comfortable viewing experience. This comes at the price of a slightly incorrect overall projection, particularly when users rotate their head sideways. Panoptic stereo can be turned on and off at runtime.

3.5. Scripting

The Omegalib API is exposed through a C++ and python interface: the Omegalib runtime embeds a script interpreter that can be used to launch standalone script applications, or can be used to control running applications through a runtime console. Scripting support facilitates development access to non-technical users, and has the added advantage of simplifying application portability. A script application can run on a personal computer or on a HRE without requiring recompilation or reconfiguration. Researchers can work on a visualization script on their own computer, save it to a flash drive, plug the drive into the HRE system and re-launch the visualization during a collaborative research session.

3.6. Application Control and Message Passing

We have so far discussed two of the software components of the HRE operating system model presented in Fig. 1: the application runtime and input manager. The third and final component is the controller and Inter-Process Communication (IPC) manager. The purpose of the controller is to manage the execution of applications, providing users with an interface to start and stop application instances and manage their workspace areas. The IPC (whose Omegalib implementation is called MissionControl) runs as a server to which applications connect once started. Connected applications can exchange messages with each other (typically

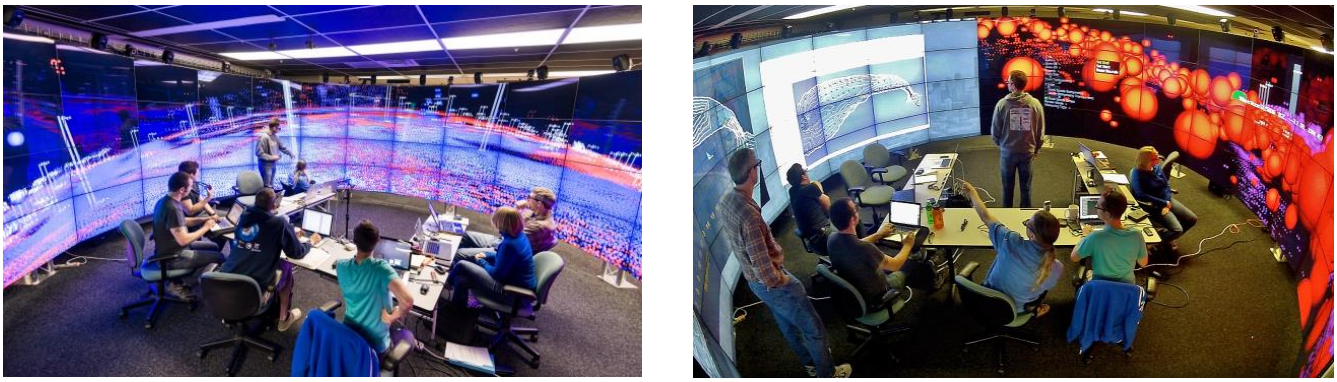

Fig. 5. Two photographs taken during a co-located collaborative meeting in CAVE2. On the left, an Omegalib immersive visualization is running on the full display. On the right, CAVE2 is split into two workspaces to display additional 2D views. Switching between the two modes can be done at runtime, without resetting running applications.

script commands), or receive messages from third party software through an external interface. MissionControl allows multiple views in the HRE to communicate with each other, for instance to coordinate the information they display. Since views run as separate processes, their frame rates are independent: this is a desirable feature when one of the views is computationally or graphically intensive, while others require real-time interactivity.

4 Applications

To evaluate the effectiveness of Omegalib in supporting co-located collaborative work, we used it as the development platform for a geo-science application for the visualization and analysis of sonar data. This application was first used during a two-day meeting of the multidisciplinary team working on the NASA ENDURANCE project.

The Environmentally Non-Disturbing Under-ice Robotic Antarctic Explorer (ENDURANCE) is an autonomous underwater vehicle (AUV) designed to explore the extreme environment of the perennially ice-covered lakes of the McMurdo dry Valleys, Antarctica. ENDURANCE operated during two Antarctic summer seasons (2008 and 2009). The AUV operated depending on 3 distinct science objectives: Water chemistry profiling, Bathymetry scanning, and glacier exploration [27].

Over the course of the full two-day ENDURANCE meeting, the research group had to complete multiple tasks: discuss new vehicle designs for a future mission, analyse mission logs and cross-reference them to water chemistry readings, and generate a new 3D map of the lake based on the collected sonar data. As shown in Fig. 5**,** the team used the display in different configurations during the meeting. During the initial evaluation of sonar data, the entire display was dedicated to a 1-to-1 scale, immersive visualization of the sonar point cloud. This visualization allowed the team to identify issues in the data, compare depth measurements from different data sources and iterate through data collected for each mission. Later in the meeting, the 3D workspace was shrunk to make space for additional 2D views representing satellite imagery from Lake Bonney and different versions of the lake bathymetry represented as a contour map. One of the 2D views was controlled by an Omegalib script running a VTK pipeline and was linked to the 3D view. As researchers picked points in the immersive environment (effectively making 'virtual measurements' of the lake depth at points they deemed relevant), the 2D view would update, regenerating the contour information to take the new points into account. A researcher could use the hand-held interaction device (a tracked game controller) to navigate the 3D view, pick depth points or rearrange and resize the 2D views by pointing in the desired direction on the screen. Other users could control the view arrangement and content from their laptops.

4.1. Other Applications

Another advantage for multiple workspace support is specifically targeted at application developers. As many other large scale display environments, HREs like CAVE2 are expensive and offer limited availability: they are often a highly contended resource, with multiple application developers scheduling access to the system to make sure their work does not conflict with others'. Emulators and smaller system replicas help, but are not a perfect substitute. For instance, estimating the performance of an application in an emulated environment is complex, due to the difference in hardware and display geometry between the two environments. Thanks to runtime workspace configuration, multiple developers can use an HRE system concurrently. We observed this usage pattern multiple times in the CAVE2 system. Developers join a work session and negotiate display allocation with others, so that each developer has a section of the display exclusively available to him/her. Developers then use a command line switch to start their application on their workspace. Developers occasionally ask others to control the full display for a brief time, usually to test a specific feature. Omegalib has also been used to create 'classic' immersive applications, and has been used as a development platform for smaller hybrid environments like the OmegaDesk [28], a hybrid 2D/3D work desk with a multitouch surface. The Omegalib source code is available online[1] and the framework runs on Windows, Linux and OSX. Extensive documentation is available on a wiki site, and there is a small but growing community of Omegalib users, evaluating the framework on new Hybrid Reality Environments, display walls, and classic CAVE systems. Omegalib is also being used for a graduate-level Visual Analytics course, as the development platform for class projects.

5 Directions for Future Work

One of the research directions for Omegalib involves improving the integration with SAGE. In its current version, Omegalib is able to output 2D views to a SAGE workspace, while 3D views need to run on a dedicated workspace. The latest version of SAGE can receive and visualize stereo pixel streams. A challenge to stereo SAGE views is that the physical properties of the output viewport

[1]https://github.com/uic-evl/omegalib

determine the shape of the off-axis frustum for stereo projection. Since users in SAGE can freely move and re-size windows, the Omegalib runtime needs to be notified of such changes and re-compute the projection transformations accordingly. Another challenge is related to rendering resource management for SAGE views: on a cluster-based HRE, multiple applications are running on (possibly overlapping) node subsets, and each application may want to send one or more secondary views to the SAGE workspace. We want to identify what nodes are responsible for rendering each secondary view for each application. An optimal allocation policy needs to take into account view size, node loads and possibly priority based on user interaction. We plan to extend the Omegalib application controller to support research in these directions.

REFERENCES

[1] C. Cruz-Neira, J. Leigh, M. Papka, C. Barnes, S. M. Cohen, S. Das, R. Engelmann, R. Hudson, T. Roy, L. Siegel, C. Vasilakis, T. A. DeFanti, and D. J. Sandin, "Scientists in wonderland: A report on visualization applications in the CAVE virtual reality environment," in Proceedings of 1993 IEEE Research Properties in Virtual Reality Symposium, 1993, pp. 59–66.

[2] C. Andrews, A. Endert, and C. North, "Space to think: large high-resolution displays for sensemaking," Proceedings of the 28th international conference on Human factors in computing systems, 2010.

[3] T. a. DeFanti, D. Acevedo, R. a. Ainsworth, M. D. Brown, S. Cutchin, G. Dawe, K.-U. Doerr, A. Johnson, C. Knox, R. Kooima, F. Kuester, J. Leigh, L. Long, P. Otto, V. Petrovic, K. Ponto, A. Prudhomme, R. Rao, L. Renambot, D. J. Sandin, J. P. Schulze, L. Smarr, M. Srinivasan, P. Weber, and G. Wickham, "The future of the CAVE," Central European Journal of Engineering, vol. 1, no. 1, pp. 16–37, Nov. 2010.

[4] T. A. DeFanti, J. Leigh, L. Renambot, B. Jeong, A. Verlo, L. Long, M. Brown, D. J. Sandin, V. Vishwanath, Q. Liu, M. J. Katz, P. Papadopoulos, J. P. Keefe, G. R. Hidley, G. L. Dawe, I. Kaufman, B. Glogowski, K.-U. Doerr, R. Singh, J. Girado, J. P. Schulze, F. Kuester, and L. Smarr, "The OptIPortal, a scalable visualization, storage, and computing interface device for the OptiPuter," Future Generation Computer Systems, vol. 25, no. 2, pp. 114–123, Feb. 2009.

[5] A. Febretti, A. Nishimoto, T. Thigpen, J. Talandis, L. Long, J. D. Pirtle, T. Peterka, A. Verlo, M. D. Brown, D. Plepys, D. Sandin, L. Renambot, A. Johnson, and J. Leigh, "CAVE2 : A Hybrid Reality Environment for Immersive Simulation and Information Analysis," 1992.

[6] K. Reda, A. Febretti, A. Knoll, J. Aurisano, J. Leigh, A. Johnson, M. E. Papka, and M. Hereld, "Visualizing Large, Heterogeneous Data in Hybrid-Reality Environments," IEEE Computer Graphics and Applications, vol. 33, no. 4, pp. 38–48, Jul. 2013.

[7] S. Teasley, L. Covi, M. Krishnan, and J. Olson, "How does radical collocation help a team succeed?," Proceedings of the 2000 ACM conference on Computer supported cooperative work, 2000.

[8] D. Angelo, G. Wesche, M. Foursa, M. Bogen, and D. d'Angelo, "The Benefits of Co-located Collaboration and Immersion on Assembly Modeling in Virtual Environments," Advances in Visual Computing, pp. 478–487, 2008.

[9] R. Jagodic, "Collaborative Interaction And Display Space Organization In Large High-Resolution Environments," Ph.D. Dissertation, 2012.

[10] M. M. P. Nieminen, M. Tyllinen, and M. Runonen, "Digital War Room for Design," Lecture Notes in Computer Science, pp. 352–361, 2013.

[11] E. Hutchins and L. Palen, "Constructing meaning from space, gesture, and speech," NATO ASI Series F Computer and Systems Sciences, 1997.

[12] R. Wilhelmson, P. Baker, R. Stein, and R. Heiland, "Large Tiled Display Walls and Applications in Metereology, Oceanography and Hydrology," IEEE Computer Graphics And Applications, pp. 12–14.

[13] G. P. Johnson, G. D. Abram, B. Westing, P. Navr'til, and K. Gaither, "DisplayCluster: An Interactive Visualization Environment for Tiled Displays," 2012 IEEE International Conference on Cluster Computing, no. Figure 1, pp. 239–247, Sep. 2012.

[14] M. Beaudouin-Lafon, "Lessons learned from the wild room, a multisurface interactive environment," 23rd French Speaking Conference on Human-Computer Interaction, 2011.

[15] G. S. Schmidt, O. G. Staadt, M. a. Livingston, R. Ball, and R. May, "A Survey of Large High-Resolution Display Technologies, Techniques, and Applications," IEEE Virtual Reality Conference (VR 2006), pp. 223–236, 2006.

[16] N. F. Polys and D. A. Bowman, "Design and display of enhancing information in desktop information-rich virtual environments: challenges and techniques," Virtual Reality, vol. 8, no. 1, pp. 41–54, Jun. 2004.

[17] N. Polys and C. North, "Snap2Diverse: coordinating information visualizations and virtual environments," SPIE 5295, Visualization and Data Analysis 2004, 2004.

[18] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. V. Kenyon, and J. C. Hart, "The CAVE: audio visual experience automatic virtual environment," Communications of the ACM, vol. 35, no. 6, pp. 64–72, Jun. 1992.

[19] W. Sherman, "FreeVR," 2005.

[20] A. Bierbaum and C. Just, "VR Juggler: A virtual platform for virtual reality application development," Virtual Reality, 2001. Proceedings. IEEE, 2001.

[21] D. Pape, "A hardware-independent virtual reality development system," IEEE Computer Graphics and Applications, vol. 16, no. 4, pp. 44–47, Jul. 1996.

[22] J. Schulze, A. Prudhomme, P. Weber, and T. A. Defanti, "CalVR: an advanced open source virtual reality software framework," SPIE 8649 The Engineering Reality of Virtual Reality, 2013.

[23] K.-U. Doerr and F. Kuester, "CGLX: a scalable, high-performance visualization framework for networked display environments.," IEEE transactions on visualization and computer graphics, vol. 17, no. 3, pp. 320–32, Mar. 2011.

[24] L. Renambot, A. Rao, and R. Singh, "SAGE: the scalable adaptive graphics environment," Proceedings of WACE, vol. 9, no. 23, 2004.

[25] S. Eilemann, "Equalizer: A scalable parallel rendering framework," IEEE transactions on visualization and computer graphics, vol. 15, 2009.

[26] T. Holtkämper, S. Scholz, A. Dressler, M. Bogen, and A. Manfred, "Co-located collaborative use of virtual environments," Proceedings AAPG Annual Convention and Exhibition., pp. 1–6, 2007.

[27] K. Richmond, A. Febretti, S. Gulati, C. Flesher, B. P. Hogan, A. Murarka, G. Kuhlman, M. Sridharan, A. Johnson, W. C. Stone, J. Priscu, P. Doran, C. Lane, D. Valle, C. Science, S. M. St, L. J. Hall, and W. T. S. Chicago, "Sub-Ice Exploration of an antarctic lake: results from the Endurance Project," in 17th International Symposium on Unmanned Untethered Submersible Technology (UUST11), 2011.

[28] A. Febretti, V. Mateevitsi, D. Chau, A. Nishimoto, B. McGinnis, J. Misterka, A. Johnson, and J. Leigh, "The OmegaDesk: towards a hybrid 2D and 3D work desk," Advances in Visual Computing, pp. 13–23, 2011.