Implementing Immersive Clustering with VR Juggler

A. Bierbaum[1], P. Hartling[1], P. Morillo[2] and C. Cruz-Neira[1]

[1] Virtual Reality Applications Center, Iowa State University. USA
[2] Departamento de Informática, Universidad de Valencia. SPAIN
aronb@vrac.iastate.edu, Pedro.Morillo@uv.es

Abstract. The last advances in commodity hardware have allowed users of immersive visualization to create high-performance systems using a set of interconnected computers. These systems, called cluster computers, allow to employ high-quality graphics cards, high-speed processors and significant amounts of memory for much lower costs than would be possible with high-end, shared memory computers traditionally used for virtual reality purposes.
In this paper we present ClusterJuggler, a system based on the VR Juggler architecture that enables the use of distributed and clustered computers for the display of immersive virtual environments. We provide and overview of the potential ways to synchronize a cluster for immersive visualization. Then, we describe the ClusterJuggler architecture in detail, and we show how ClusterJuggler allows virtual reality application developers to combine various existing clustering techniques to meet the needs of their specific applications. A performance evaluation of our clustered technique on real 3D real-time immersive applications demonstrates the efficiency of ClusterJuggler with respect to both number of nodes in the cluster and the bandwidth of the interconnection network system.

1 Introduction

Traditionally, multi-screen immersive visualization systems have relied upon dedicated, high-end, shared memory graphics computers to generate interactive virtual environments. These systems must not be confused with distributed virtual environment (DVE) systems where many users remotely connected from different computers, typically connected through the Internet, share the same 3D virtual world [17]. Such multi-screen immersive systems typically require one or two video outputs for each projection surface, and they often utilize many input devices simultaneously. In recent years, the nearly exclusive use of high-end computers for these purposes has shifted to commodity hardware as it has become a viable alternative [2, 10, 18]. Continuous, rapid improvements in commodity hardware have allowed users of immersive visualization to employ high-quality graphics hardware, high-speed processors, and significant amounts of memory for much lower costs than would be possible with high-end, shared memory computers. However, to drive a multi-screen immersive visualization system, we need multiple commodity systems working as a single unit, thereby mimicking the behavior of a single, shared memory computer. This transparency of the VR system can be accomplished through the use of a tightly synchronized cluster.

Clustering techniques have been utilized to parallelize complex computations for many years in high-performance computing (HPC) [5, 18]. Despite HPC clusters offer an alternative to expensive supercomputers and can be used to drive multi-screen visualization systems, the existing parallelization techniques used for HPC cannot be applied directly to graphics clusters. In this sense, graphics clusters add some constraints to virtual reality (VR) software. While these constraints are all solved at the hardware of shared memory computers, they must be solved at the software level for a graphics cluster. These constraints are related to:

- **High-performance network:** Interactive graphics require extremely low latency communication networks in order to maintain real-time frame rates. Also certain clustering techniques require high bandwidths because of the substantially large amounts of data they need to transfer each frame.
- **Swap buffer synchronization:** In order to prevent tearing while combining images rendered on multiple cluster nodes we must synchronize the swapping of the front and back frame buffers.
- **Consistent random number generation:** Applications that use random numbers in their calculations require consistent random numbers across the entire cluster to ensure identical results.
- **Frame delta:** Many applications use the elapsed time since the last frame in their physics calculations. Because each node is executing at different speeds this time delta must be shared to ensure consistent results.
- **Start barrier:** Certain clustering techniques require that each node starts the first frame of the application at the same time. In order to accommodate this each node must wait at a barrier before starting the frame loop.
- **Multiple input devices:** Most VR input devices communicate with a computer using a serial port. This causes a limitation for commodity hardware because of the limited number of serial ports. This can be addressed by allowing input devices to transparently reside on multiple nodes.

In this paper, we present ClusterJuggler [4], a system based on the VR Juggler architecture that enables the use of distributed and clustered computers for the display of immersive virtual environments. The main goals of ClusterJuggler are to allow the cluster software to adapt to the particular hardware configuration of the virtual reality system; to provide application portability and scalability from high-end systems to commodity clusters by hiding the clustering from developers; and to allow users to customize the clustering methods being used to best meet their specific needs. It's worth mention that ClusterJuggler works transparently to VR application designers and requires no code changes when inmersive applications are executed in different multi-screen configurations including CAVEs, head mounted displays (HMD) and desktop VR.

The rest of the paper is organized as follows: Section 2 describes the most important approaches to simulate multi-screen immersive visualization systems on a cluster of computers. Section 3 shows the modular architecture based of layers of ClusterJuggler and also how the features of this clustering platform oriented to 3D real-time environments can be extended. Next, Section 4 presents the performance evaluation results of the current version of ClusterJuggler. Finally, Section 5 presents some concluding remarks and future work to be completed.

2 Background

Several software libraries generate immersive environments by utilizing clusters of commodity computers. Each of these solutions attacks the issues listed in the previous section at one of four locations: input data [2, 8, 16], remote shared memory [11], scene graph change lists [15, 16], or graphics primitives [10, 12, 18].

In order for a user to become fully immersed in a virtual environment, they must interact with it using one or more physical input devices such as a position tracker or glove. Since the objective is to provide the user with a sense of immersion these devices obtain all the input needed to determine the changes in application state. Clustering solutions utilizing *input data* sharing start a distinct complete copy of the application on each node in the cluster. All input data is then synchronized across the cluster at the beginning of each frame loop. Thus, application state remains consistent as long as it depends solely on input events. Despite this approach does not require any changes to the application relative to a shared memory architecture (since the application still has access to the same input data and rendering targets), random number generation, consistent frame deltas, and a start barrier are not addressed. Examples of multi-screen immersive visualization systems based on cluster architectures are Net Juggler [2] and Syzygy [16]. While Net Juggler [2] uses message passing via the Message Passing Interface (MPI) [8] in its implementation, frameworks such as VR Juggler [4] and Syzygy [16] use the TCP/IP suite directly.

Remote shared memory approach offers another way to ensure that each node has an identical snapshot of that state for rendering each frame. Implementations of remote shared memory often require that the application programmer take special steps to use it. Special storage areas must be created, and in some cases, access to the shared memory must be controlled so that there may be multiple readers but only one writer. Different designs put more or less of the burden on the application programmer for understanding and managing these details. Implementations as DIVERSE [11] are based on a shared memory architecture where a general inter-process communication programming tool guarantee identical copies among the nodes of the cluster.

Since most graphic applications are based on the manipulation of scene graphs [19], if one node keeps track of all changes made to the scene each frame, that node can send the changes to each of the other nodes to be reapplied to the local memory copy of the scene graph. Therefore, each node always has the information it needs to render an accurate version of the scene. This approach, called *scene graph change lists*, takes advantage of the fact that visual consistency and coherency is the critical aspect of all graphics clusters. Both OpenSG [15] and Syzygy [16] implement this clustering method.

At the lowest level, all immersive applications generate a stream of graphics commands that are delivered to the graphics hardware for rendering. This is accomplished by making calls to a low level graphics application programming interface (API) such as OpenGL. Software libraries such as Chromium [10] and DGL [12] are designed to intercept the *graphics primitives* for the rendering of each frame and distribute them over a network in order to divide the rendering task among multiple nodes. This approach tends to require more bandwidth than any of the previously mentioned methods [18].

3 Conceptual Model and Architecture

Since clustering techniques presented above have their own unique benefits and drawbacks, we present a modular and extensible architecture, called *ClusterJuggler*, that combines the advantages of all of them. The ClusterJuggler design contributes several key features not found in other clustering architectures: a layered architecture, run-time reconfiguration, and an extensible, component-based system.

The architecture of ClusterJuggler separates the aspects of clustering for VR into several layers. Each layer builds upon the functionality of those below to provide additional features. This modular design allows us to implement and test each layer independently, and changes made to one can happen transparently to the layers above. Cluster Juggler uses the same advanced configuration infrastructure that VR Juggler [3, 4]. In this configuration infrastructure, information arrives in the form of config elements. Basically, these elements are XML files and they are the fundamental unit of configuration in VR Juggler [9]. Handlers of config elements are registered with an entity known as the Configuration Manager, and newly received config elements are delivered to the appropriate handlers. New config elements may arrive at any time during the lifetime of an application, thus allowing run-time reconfiguration of the software. Since Cluster Juggler is based on the VR Juggler architecture, it takes advantage of this feature [4]. ClusterJuggler allows nodes, displays, and input devices to be added, removed, or reconfigured as needed at run time. We have followed the traditional component-based approach for developing this architecture [20]. The code that uses the components is then responsible for loading implementations at run time based on some specification. Each component, called plug-in, is a standalone module loaded at run time based on the user-specified cluster configuration. The plug-ins extend the ClusterJuggler core with specialized clustering functionality. Users can choose any of the plug-ins needed for their applications and their specific cluster configuration.

3.1 An Architecture based on layers

As both Figure 1-a and Figure 1-b show, the architecture of ClusterJuggler has been designed as a set of components that are arranged into layers. At the lowest level, the Cluster Network provides a messaging interface for communicating with the entire cluster. The Cluster Plug-ins are built on top of the Cluster Network and provide the application developer with a set of components to construct the best solution for their applications needs. The top layer is the Cluster Manager which acts as a interface to ClusterJuggler. Higher level code utilizes the Cluster Manager to control ClusterJuggler.

Cluster Manager Layer. This is the main layer in ClusterJuggler. This layer is responsible for handling the cluster configuration and for synchronizing the calls to each plug-in. Once all nodes of the cluster load the application code into memory, an entry point function is called to create an instance per plug-in in the Cluster Manager. In this moment, each plug-in (selected by the users) becomes a mechanism which allows and defines the communication among the nodes of the cluster. In this sense, a cluster can incorporate a master/slave or a P2P network protocol depending the selected plug-ins. Since plugins can generate data inconsistency problem in the cluster, the Cluster

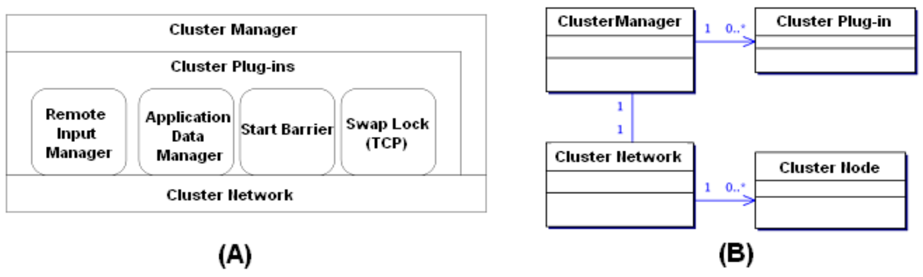

Fig. 1. The architecture of Cluster Juggler: (a) A layered view, b) a simplified UML class diagram

Manager is responsible for making sure that all plugins have their run-time information dependencies satisfied during the simulation. In order to accommodate all possible needs, each plug-in has a well-defined interface and a contract that specifies the invocation timing. In order to guarantee a full compatibility between VR Juggler [4] and ClusterJuggler a micro-kernel architecture is adopted. In this case, each pass through the "kernel loop" guarantees can be made about the state of input, graphics contexts, and the Cluster Manager. The Cluster Manager can in turn invoke the methods of the plugins at well-defined times during the kernel loop.

Cluster Network Layer. This layer maintains an abstract representation of the system of interconnected nodes that comprise the cluster. This abstraction provides ClusterJuggler with a messaging interface for communicating with the entire cluster. Internally, it maintains a list of the nodes in the cluster along with the current network connections used to communicate with them.

Cluster Plug-ins They represent the point of extension for ClusterJuggler. This aspect of the design allows the addition of new plug-ins to address cluster-specific application issues not handled by the standard set of Cluster Plug-ins. By default ClusterJuggler incorporates the next plugins: the Remote Input Manager (*RIM*) plug-in, Application Data Manager (*ADM*), the Swap Lock (*SL*) plug-ins, and the Start Barrier (*SB*) plug-in.

RIM plug-in is responsible for distributing synchronized device data across the cluster. In order to ensure that all nodes in the cluster have a consistent snapshot of all input data, (regardless of the location of the physical hardware) RIM emulates each node of the cluster as a "device server". The device data is shared over the network using the platform-independent protocol provided by the Cluster Network layer. Users of the device server idea can take advantage of this to utilize input devices that might not otherwise be usable due to hardware or software limitations. This device location transparency allows not only to construct a cluster from any combination of the platforms supported by VR Juggler [4], but also to balance the workload generated by a large number of VR devices by connecting them to separate computers.

ADM provides application developers with a method for sharing arbitrary application state across the cluster. This capability extends the fundamental input data sharing and demonstrates that the ClusterJuggler design allows multiple clustering techniques to be utilized in a single application. Sharing of application-specific data structures works by providing the application developers with a base class that they extend with their own type. The base class defines an interface for serializing and de-serializing the data structure. Application developers must implement this interface with serialization code that is specific to their data type. In order to ensure data consistency across the cluster, ADM not only maintains a different GUID (128-bit Globally Unique Identifiers) for each application specific data type [21], but also does not allow distinct nodes have different copies of the same data. All the serialize function calls are performed in the same node of the cluster (which is configured by the user) called "host node".

SL plug-in is used with the RIM Plug-in to ensure that the applications running on the cluster nodes all begin their execution on the same frame. SL plug-in creates a software barrier by sending signals between the cluster nodes. The plug-in uses a master/slave paradigm where each slave sends a signal to the master immediately before swapping the frame buffers. The master is identified through a configuration specific to the plug-in, and the remaining nodes are then identified as slaves. All the slaves then suspend their execution, waiting for the master to send a response signal. The master sends its response immediately before it invokes the frame buffer swap operation. Upon receiving the response from the master, the slaves perform the frame buffer swap. Depending on how the interconnection nodes is configured, ClusterJuggler incorporates three different versions of SL plug-in: TCP swap lock, the parallel port swap lock, and the hybrid TCP/serial port swap lock [1].

Since each node in the cluster runs a distinct and complete copy of the VR Juggler application, Cluster Juggler needs a mechanism to guarantee that all nodes begin their execution on the same frame. This feature is provided by the *SB plug-in* using a master/slave paradigm similar to the SL plug-in. One node in the cluster is identified to be the master with a configuration specific to this plug-in type. The remaining nodes are therefore slaves. When each slave is ready to begin its frame loop, it sends a message to the master and waits for a response. When the master has received all the messages from the slaves, it sends the responses to them. At that point, all nodes may begin their frame loop, thereby guaranteeing the goal of the SB plug-in.

4 Performance Evaluation

In this section, we present the performance evaluation of ClusterJuggler. Instead of analyzing the efficiency of our clustering platform by using simple and well-known 3D models [18], we have performed our experiments on real VR applications. In order to show the performance of ClusterJuggler we have wanted to select a group of representative VR applications that would cover the spectrum of graphics-intensive and computationally-intensive workloads of VR applications. We have chosen four application based on this criteria: "*cubes*", "*agua*", "*hindu*" and "*mpapp*".

"*Cubes*" is an extremely simple application where 1000 cubes are drawn floating in the space. This simplicity allows "*cubes*" application to generate low levels of workload in terms of both graphic and computation requirements. Next application, "*agua*", takes advantage of special hardware techniques, such as vertex shading [7], in order to recreate a real-time travel around a complete deep sea reef. Despite the computational workload generated by "*agua*" is very low, the huge use of the graphical card capabilities allows this application to be considered as highly graphics-intensive. The third application, "*hindu*", is a virtual walkthrough which allows users to explore the Radharaman Temple (Vrindavan, India). This application uses a set of animated virtual characters in order to perform a traditional religious ceremony inside of the temple. Hindu"*hindu*" is not only graphically intensive as it contains large amounts of polygons and textures (for both temple and characters), but also is computational intensive due to time it takes in to generate the final the movements and shadows for all the characters in each frame of the simulation. Finally, on the opposite extreme of the application spectrum, "*mpapp*" performs a real-time simulation of a square piece of cloth which has been modelled as a simple mesh surface. Since this mesh is generated by means of a complex 3D polynomial equation, "*mpapp*" requires a minimal graphical workload but it is highly computational intensive. Figure 2 depicts different snapshots of the four proposed VR applications taken when they are executed in a stand-alone configuration on VR Juggler. All the applications use OpenGL (with any type of graphic optimization or advanced tool to speed-up rendering) as an average programmer would use it.

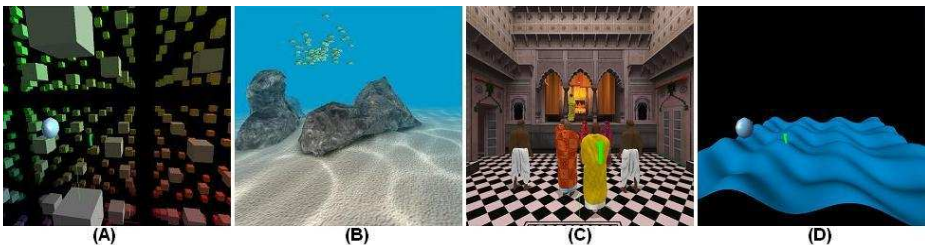

Fig. 2. Snapshots obtained from VR Juggler for: (a) Cubes (b) Agua (c) Hindu and d) Mpapp

Our test-environment is composed by 8 Linux-PCs, each running RedHat 8.0 with a NVIDIA GeForce3 Ti200 (128 MBytes) graphics card, a 2GHz Intel Pentium Processor, 1 GByte of RAM, and 512 Kbyte of cache memory. The machines are connected to a Cisco Catalyst 3750 Gigabit Ethernet switch.

Since we have considered the system throughput as the number of frames per second (denoted as FPS) performed by the graphics cluster [18], we have measured this parameter by varying both the number of nodes and the network bandwidth in the cluster. The results of this variations are shown in Figure 3-a and Figure 3-b, respectively. The Y-axis of both figures show FPS values for the simulations performed with each system configuration. Each point in both plots represents the average value of the FPS obtained after 25 executions of the same application benchmark. The standard deviation for any of the points shown in the plots was not higher than 4 FPS in any case.

Figure 3-a shows the values of FPS reached by ClusterJuggler depending on the number of nodes in the cluster. This figure shows on the X-axis the number of nodes ranging from a C1 to a C8 configuration. While C1 is a classical VR Juggler configuration composed of a single stand-alone node [4], ClusterJuggler achieves to execute the considered benchmark applications by means of eight synchronized computers in C8. It shows that, for all the considered benchmark applications, FPS is almost linearly reduced as more nodes are added to the cluster. Moreover, the linear factor of the throughput reduction decreases with the workload generated by the application in a stand-alone configuration. In this sense, applications as "*hindu*" or "*mpapp*" only have an average reduction of nine and seven FPS when they are ported from a stand-alone to a C8 configuration. The reason of this behavior is related to the linear network overhead that is incurred as new nodes are added to the cluster system. This linear overhead is caused by the master/slave configuration of the SL plug-in described in section 3.1.

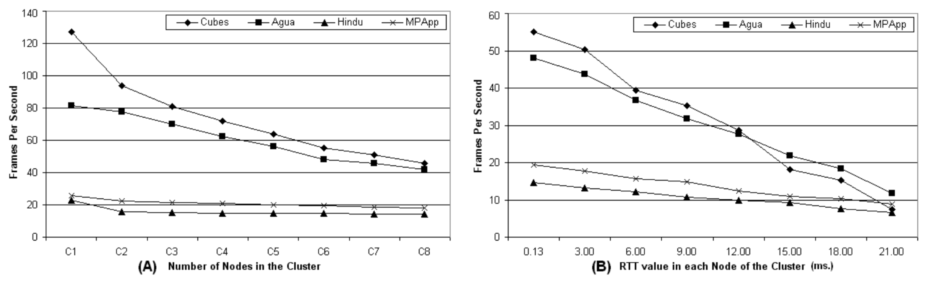

Fig. 3. Values of system throughput (FPS) for different (a) number of nodes and b) network bandwidth in ClusterJuggler

Despite an important topic when analyzing the performance of clusters is how network bandwidth and system throughput are related [14], this concept does not use to be taken into account when virtual reality cluster systems are analyzed [2, 10, 15, 16]. In order to study this behavior in detail, we have taken advantage of a recent tool called *Netem* [13]. Netem (Network Emulator) is a general-purpose tool for emulating band-limited links in real-time in order to study the effects of bandwidth limitations on system performance and user interaction. By operating at the IP level, Netem can emulate the critical end-to-end performance characteristics imposed by various wide area network situations (e.g., congestion loss) or by various underlying subnetwork technologies (e.g., Ethernet, Fast Ethernet, cable modems). Basically, Netem allows each node of the cluster to ensure a non-uniform RTT (Round Trip Time) value for the transmitted TCP packets according to the specifications of the subnetwork technology. Since the correlation between RTT delay and the type of the physical network connection has been widely described in the literature of cluster computing [5, 6], Netem becomes an excellent tool for emulating performance dynamics in our test-environment hardware.

Figure 3-b shows the performance evaluation results obtained by Cluster Juggler when different values of RTT are considered in a C6 configuration. The first value on the X-axis (0.13 ms.) corresponds to the case in which no delay was added to all packets going out of the local Ethernet. This case shows the effective (and minimum) RTT value obtained in this configuration composed of six nodes and based on a Gigabit Ethernet backbone. In order to decrease the network throughput of the system, the above values correspond to the situations where Netem is used. Despite the main goal of our study was to determine the performance of Cluster Juggler in LAN configurations, a large range of delays is considered. This figure shows how FPS linearly decreases as communication link delay increases for all the considered benchmark applications. Unlike the above case, the FPS value tend to converge towards a similar threshold level when high latencies are emulated for all the benchmark applications. In this situations, Cluster Juggler spends most of rendering period while waiting for the synchronization from both SL and SB plug-ins. Besides that situation is typical of WAN environments, Figure 3-b shows that Cluster Juggler provides a very low reduction of performance levels, in terms of FPS, when VR Juggler immersive stand-alone applications are launched on commodity LAN clusters. Since values of RTT in these systems are not higher than a couple of milliseconds [5], these results show Cluster Juggler can be considered as a efficient tool to simulate multi-screen immersive visualization systems on a cluster of computers.

5 Conclusions and Future Work

In this paper, we have described the architecture and the performance evaluation of ClusterJuggler, a evolution of VR Juggler that enables commodity computing and rendering hardware to drive immersive visualization systems. The opened architecture of ClusterJuggler has been specifically designed to allow VR application developers to combine various existing clustering techniques, both at the hardware level and at the software level, to meet their own specific needs. In this sense, ClusterJuggler not only allows users to configure the software to meet the needs of their specific hardware, but also its plug-in framework allows programmers to extend ClusterJuggler with new clustering features. Moreover, the results of the performance evaluation show Cluster Juggler provide application portability and scalability from high-end systems to commodity clusters by hiding the clustering from developers. In our case, ClusterJuggler has allowed us to migrate existing applications (designed initially for high-end shared memory computers) to the newer cluster-based configurations while keeping high levels of frame-rate and without changes required to the application code.

As future work to be done, we plan to add to Cluster Juggler the ability to use additional network protocols such as IP multicast. Since the current version of Cluster Network layer is limited to using point-to-point TCP connections, our intention is to provide further network efficiency beyond those domain-specific optimizations presented in the current implementation. Finally, the addition of a plug-in for monitoring the performance of the cluster at run time and for validating correct synchronization is also planned.

References

1. J. Allard, V. Gouranton, G. Lamarque, E. Melin, and B. Raffin. Softgenlock: Active stereo and genlock for pc cluster. In *Proceedings of the Joint IPT/EGVE'03 Workshop*, Zurich, Switzerland, May 2003.

2. J. Allard, V. Gouranton, E. Melin, and B. Raffin. Parallelizing pre-rendering computations on a net juggler pc cluster. In *IPT (Intl. Workshop on Immersive Projection) 2002 Proceedings*, Orlando, Florida, United States, March 2002.
3. A. Bierbaum and C. Cruz-Neira. Run-time reconfiguration of vr juggler. In *IPT (Intl. Workshop on Immersive Projection) 2000 Proceedings*, Ames, Iowa, United States, June 2000.
4. A. Bierbaum, C. Just, P. Hartling, K. Meinert, A. Baker, and C. Cruz-Neira. VR Juggler: A virtual platform for virtual reality application development. In *Proceedings of IEEE Virtual Reality*, pages 89–96, Yokohama, Japan, March 2001.
5. R. C. Booth. A system area network characterization in a commercial cluster. Master's thesis, Dept. of Electrical and Computer Engineering, University of Minnesota, 1998.
6. M. Carson and D. Santay. Nist net: a linux-based network emulation tool. *ACM SIGCOMM Computer Communication Review*, 33(3):111–126, 2003.
7. W. Engel. *Programming Vertex and Pixel Shaders*. Programming Series. Charles River Media, 2004.
8. Message Passing Interface Forum. MPI: A message-passing interface standard. *International Journal of Supercomputer Applications and High Performance Computing*, 8(3/4):165–414, Fall/Winter 1994.
9. P. Hartling, A. Bierbaum, and C. Cruz-Neira. Tweek: Merging 2d and 3d interaction in immersive environments. In Nagib Callaos, Alexander Pisarchik, and Mitsuyoshi Ueda, editors, *Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics*, volume VI, pages 1–5, Orlando, Florida, United States, July 2002.
10. G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski. Chromium: A stream processing framework for interactive graphics on clusters. In *ACM SIGGRAPH 2002 Sketches and Applications*, Texas, United States, July 2002. ACM Press.
11. J. Kelso, L. Arsenault, S. Satterfield, and R. Kriz. Diverse: A framework for building extensible and reconfigurable device independent virtual environments. In *IEEE Virtual Reality 2002 Proceedings*, pages 183–190, Orlando, Florida, United States, March 2002.
12. K. Li, H. Chen, Y. Chen, D. W. Clark, P. Cook, S. Damianakis, G. Essl, A. Finkelstein, T. Funkhouser, A. Klein, Z. Liu, E. Praun, R. Samanta, B. Shedd, J. P. Singh, G. Tzanetakis, and J. Zheng. Early experiences and challenges in building and using a scalable display wall system. *IEEE Computer Graphics and Applications*, 20(4):671–680, 2000.
13. Netem: Network Emulator Home Page. http://developer.osdl.org/shemminger/netem/.
14. A. Plaat, H. Bal, , and R. Hofman. Sensitivity of parallel applications to large differences in bandwidth and latency in two-layer interconnects. In *Proceedings 5th IEEE HPCA99*, pages 244–253, Orlando, Florida, United States, January 1999.
15. M. Roth, G. Voß, and D. Reiners. Multi-threading and clustering for scene graph systems. *Computers & Graphics*, 28(1):63–66, February 2004.
16. B. Schaeffer and C. Goudeseune. Syzygy: Native pc cluster vr. In *IEEE Virtual Reality 2003 Proceedings*, pages 15–22, Los Angeles, California, United States, March 2003.
17. S. Singhal and M. Zyda. *Networked Virtual Environments: Design and Implementation*. Addison-Wesley, 1999.
18. O. G. Staadt, J. Walker, C. Nuber, and B. Hamann. A survey and performance analysis of software platforms for interactive cluster-based multi-screen rendering. In *Proceedings of the Workshop on Virtual Environments 2003*, pages 261–270, Zurich, Switzerland, 2003. ACM Press.
19. P. Strauss and R. Carey. An object-oriented 3d graphics toolkit. In *Proceedings of 19th ACM SIGGRAPH*, pages 341–349. ACM Press, August 1992.
20. C. Szyperski, D. Gruntz, and S. Murer. *Component Software: Beyond Object Oriented Programming*. Component Software Series. Addison-Wesley Publishing Company, New York, NY, second edition, 2002.
21. The Open Group. *DCE 1.1: Remote Procedure Call*. The Open Group, August 1997.