Net Juggler : Running VR Juggler with Multiple Displays on a Commodity Component Cluster

Jérémie Allard Valérie Gouranton Loïck Lecointre Emmanuel Melin
Bruno Raffin
Université d'Orléans
Laboratoire d'Informatique Fondamentale d'Orléans (LIFO)
F-45067 Orleans Cedex 2, France
gouranton@lifo.univ-orleans.fr, melin@lifo.univ-orleans.fr, raffin@lifo.univ-orleans.fr

Abstract

Net Juggler is an open source library that turns a commodity component cluster running the VR Juggler platform on each node into a single VR Juggler image cluster. Application parallelization is transparent to the user and leads to high performance executions even with limited bandwidth networks.

1 Introduction

Today's virtual reality environments are mainly driven by high-end proprietary computers, like SGI Onyx systems, but high performance commodity components are also available to build clusters that could provide the necessary computation power. Nevertheless, to use a PC cluster with loosely coupled processors and graphics cards faces several limitations. Data and computations must be carefully distributed on the different nodes to ensure communications stay below the capacity of the network. Computations must be synchronized to ensure the different projectors display complementary parts of a large image. These tasks should not be the responsibility of the programmer. Parallel programming is time consuming and diverts the programmer from its main goal, which is to develop innovative interactive graphics applications. Thus, using PC clusters for virtual reality requires programming environments that hide the complexity of the underlying architecture and provide automatic and efficient parallelization schemes.

One approach proposes to parallelize the application at the level of graphics primitives [3, 4]. The application is executed on one or several nodes. Graphics primitives are intercepted and broadcasted to each node controlling a display. These nodes render the scene according to their local viewport. The amount of data to send over the network can be important, limiting the performance. The WireGL protocol [4] integrates caching and compression techniques to reduce the network bottleneck. But these techniques have limiting effect for real-time applications with frequent updates of the scene.

An other approach consists in duplicating the application on each node [2, 3]. Every frame, time and input device data are broadcasted to each copy to guaranty data coherency. Compared to the previous approach the required network bandwidth is limited due to the small amount of data communicated, ensuring high performance even for real-time applications making frequent updates of the scene.

In this paper, we extend this approach to turn a cluster running a VR system on each node into a single VR system image cluster. Our work is based on the open source VR Juggler library [1] that defines a execution platform for virtual reality applications. VR Juggler provides an abstraction of the underlying system, while giving direct access to various graphics API for maximum control over applications. The application is independent of the displays, the input and output devices. System components are configured with a set of files when launching the application. Currently VR Juggler supports several architectures like PCs, workstations and Onyx systems, but it does not support cluster configurations. We developed Net Juggler, a C++ library that turns a cluster running VR Juggler on each node into a single VR Juggler image cluster. Net Juggler does not modify the VR Juggler API. Running a VR Juggler application on a Net Juggler cluster only requires to adapt the VR Juggler configuration files to the cluster configuration.

2 Net Juggler

VR Juggler [1] is organized around a kernel and different components called managers. Each manager handles a set of specific system details, while the kernel controls the run-time system and brokers communications between the different managers. Every input device is controlled by the input manager. When an application requests access to a device, it contacts a proxy. The proxy hides the actual device and tracks the most recent data received from the device. The draw manager gives direct access to the graphics API. The display manager takes care of the windows and displays. The configuration manager handles a database with configuration information, like window properties, proxy names and associated devices. The environment manager is the user's entry point to exchange data with VR Juggler. With the graphics utility called VjControl, the user can reconfigure the application at run-time or collect performance data.

Adding cluster support to VR Juggler requires new functionalities. Following VR Juggler micro-kernel organization we implemented new managers. A Net Juggler kernel derives from the VR Juggler kernel to handle them.

Net Juggler has to collect data from each device and broadcast these data to each node of the cluster. VR Juggler uses proxies to hides the actual devices. Net Juggler replaces these proxies by client and server proxies. The node where the device is connected runs a server proxy while the nodes requiring the data run a client proxy. Because proxies provide an abstraction of input devices, their number is limited and should not increase significantly in the future.

VR Juggler instantiates the application according to the execution environment with a set of configuration files or at run-time from requests sent by the user through VjControl. Configuration data are organized in chunks, each chunk having information about a part of the system. The same functionalities are available with Net Juggler. For a Net Juggler cluster, chunks have to be modified to include a host parameter. The host indicates the node the chunk is related to. We also defined a new type of chunk for client/server proxy couples. In this case, the host parameter has a different semantics: it points out the node that runs the server proxy, all the other nodes having a client proxy. On each node, a cluster configuration manager stores the chunks in a database. The whole cluster configuration is then easily available from any node of the cluster. Each node selects the Net Juggler chunks it is concerned with and translates them into VR Juggler chunks that the VR Juggler configuration manager stores locally.

The classical stream paradigm is used and extended to provide an abstraction of the actual data communications. There is one stream by server proxy and by cluster environment server. A stream is associated to a specific node source and can have several destination nodes. Each stream is identified by a unique id and can be created, deleted or modified at run-time. Actual data communications take place only once per frame. The nodes also execute a synchronization barrier just before swapping their frames buffers to ensure the consistency of the images displayed. Communications are implemented with the MPI standard.

Tests show Net Juggler does no introduce a significant overhead (less than 4% of the overall computation time for 4 dual Pentium III 800 MHz nodes equipped with GeForce 2 graphics cards, 4 displays and a 100 Mbits/s Fast Ethernet network). The overhead introduced by communications depends on the number of nodes, the number and type of input devices and their repartition on the different nodes, but not on the complexity of the scene. Thus, generally the amount of data communicated is small even for fast changing complex scenes.

3 Conclusion

In this paper, we presented Net Juggler that provides cluster support for VR Juggler. Application parallelization is transparent to the user and ensures high performance even for real-time applications making frequent updates of the scene. Net Juggler instantiates the application according to the execution environment through a set of configuration files or at run-time from requests sent by the user through VjControl.

The parallelization scheme adopted is based on running a copy of the application on each node. Data are duplicated and some computations are redundant. For some applications this can be a memory and performance bottleneck that we will address in future works.

References

[1] 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 *IEEE VR 2001*, Yokohama, Japan, March 2001.

[2] M. Bues, R. Blach, S. Stegmaier, U. Häfner, H. Hoffmann, and F. Haselberger. Towards a Scalable High Performance Application Platform for Immersive Virtual Environements. In J. D. B. Fröhlich and H.-J. Bullinger, editors, *Immersive Projection Technology and Virtual Environements 2001*, pages 165–174, Stuttgart, Germany, May 2001. Springer.

[3] H. Chen, Y. Chen, A. Finkelstein, T. Funkhouser, K. Li, Z. Liu, R. Samanta, and G. Wallace. Data Distribution Strategies for High-Resolution Displays. *Computers & Graphics, Special Issue on Mixed Realities*, 2001.

[4] G. Humphreys, M. Eldridge, I. Buck, G. Stoll, M. Everett, and P. Hanrahan. WireGL: A Scalable Graphics System for Clusters. In *Proceedings of SIGGRAPH 2001*, 2001.