DICELIB: A REAL TIME SYNCHRONIZATION LIBRARY FOR MULTI-PROJECTION VIRTUAL REALITY DISTRIBUTED ENVIRONMENTS

Bruno Barberi Gnecco
Paulo Alexandre Bressan
Roseli de Deus Lopes
Marcelo Knörich Zuffo

Av. Prof. Luciano Gualberto, 158 – Trav. 3 – Butantã
CEP: 05508-900 – São Paulo – SP – Brazil
Laboratory of Integrated Systems – Polytechnic School – USP
Tel: (+55 11) 3818-5254 – Fax: (+55 11) 3818-5665
{brunobg, pbressan, roseli, mkzuffo}@lsi.usp.br

Abstract: The recent availability of PC-clusters offers an alternative solution instead of high-end graphics workstations, which are commonly used to support multi-projection oriented virtual reality applications, such as CAVE. Among the advantages of PC-clusters we can mention the low cost, the scalability and the use of open development platforms such as the Linux operating system. From the programmer’s point of view, the main challenge is the lack of efficient libraries and the difficulty of synchronizing and managing coherence of graphics datasets in real time. We proposed and implemented DICELib, a socket based library matching the requirements of low processing time and high speed. Results presented are from a graphical PC-cluster being developed by our laboratory.

Keywords: PC-Cluster, CAVE, real-time, distributed computing.

1 Introduction

Multi-projection systems are important to many virtual reality applications and can be used in situations that involve collaborative work, education and decision-making. They consist of a set of projected images that visualize one unique database, and each projection may be from a different viewpoint. This kind of system is affected by many variables: computer architecture, software support, application characteristics and image quality.

Lately, CAVEs (CAVE Automatic Virtual Environment) have received considerable attention of industries and universities due to the innumerable applications recently developed for them. All academic areas can take advantage of such systems, but currently medicine, petroleum industry, meteorology and dynamics fluids are particularly interested. Unfortunately, the cost of a CAVE is not limited to the screens and projectors. The computer systems needed to simulate the virtual world are complex and expensive in direct sense of new challenges. Both hardware and software requirements grow with the user’s and developer’s dreams.
Traditionally, supercomputers are designed with the objective of achieving the highest computational and communication performance physically possible. At the other extreme are low-cost computer architectures, where the performance is subordinate to the end-user price; commodity PCs fill this role. Advances in the performance of commodity PCs and the availability of commodity high-speed networks led to the discovery that supercomputing performance could be delivered in some applications with PC clusters. Besides, the price-performance ratio is an order of magnitude smaller than typical supercomputers [1]. However, the cluster usability and programmability depend seriously on the availability of adequate programming environments.

In this paper we present a library that synchronizes a PC-cluster used to render images for a multi-projector system, CAVE-like. The cluster nodes keep the coherence between the several viewpoints (projectors or monitors), allowing virtual reality and collaborative applications. These libraries are being particularly used in CAVerna Digital of Laboratory of Integrated Systems, a CAVE environment with five sides build in Polytechnic School of University of São Paulo. The cluster being used, named Polux, is composed by 6 Dual Pentium III Xeon 1GHz 1GB-RAM nodes connected by Gigabit-Ethernet network adapters.

2 Motivation and Related Work

Several research groups are working in tools for clusters or CAVE; among these tools we can mention WireGL, CAVELib, CAVERNsoft and pvmsync.

WireGL [3] is an active research project at the Stanford University Computer Graphics Lab to explore rendering cluster systems. Implemented as an OpenGL driver the software allows unmodified applications to render images using clusters, and there’s support for both TCP/IP and Myrinet GM protocols up to 32 nodes. The WireGL has a geometry bucketing that only sends geometry to servers responsible by to render primitives. The implementation, however, works only for a single point of view, and each node is responsible for rendering a small part of the image.

CAVELib [5] is an Application Programmers Interface (API) that provides general support for building virtual environments for Spatially Immersive Displays and virtual equipment. The CAVELib configures the display device, synchronizes processes, draws stereoscopic views, communicate with tracker devices, creates a viewer-centered perspective and provides basic networking between remote virtual environments. The CAVELib allows a single program to be available on a wide variety of virtual display devices without rewriting or recompiling. Developed initially for SGI's, a version for clusters running Linux has been recently made available.

CAVERNsoft G2 [6] is a toolkit for supporting high performance computing and data intensive systems that are coupled to collaborative, immersive environments. The G2 tool comprises low-level modules to provide full control of networking at the socket level; middle-level data distribution modules such as remote procedure calls; and high-level modules such as application shells and classes for rendering avatars.
and does not influence on the architecture of the application being developed. When the application is run and DICElib is called, it takes control and runs the internal server. This server spawns the processes among the cluster nodes. This approach has some advantages: the developer can see the cluster as one supercomputer, instead of a number of nodes; and communication is reduced, since nodes are connected only to the server, and the server can filter redundant updates.

Currently DICElib implements synchronization and synchronous data sharing among the nodes (see Figure 1). The functionality is very simple, the server waits until that every node sends a message to it, and then the server sends a message to each node.

Another facility provided by DICElib is synchronous data sharing. Users can declare, on-the-fly, variables that are updated in other nodes only when the DICE_sync function is called. This is extremely handy for CAVEs, because while different nodes will be rendering different views, all have to be done exactly from the position. The developer may create new variable types, ensuring that structures or complex data is easily spread over the cluster. The most common data types (floats, integers, strings, etc) are available by default.

DICElib (DIstributed Cave Engine Library) is an effort to make easy the use of clusters to run multi-projections systems, like CAVE, or any system that needs synchronization of the cluster machines. The major objectives of its development are: high speed, low processing time and easiness to adapt existing programs to use it. It was decided to implement it using TCP/IP, instead of libraries such as MPI or PVM, since the former approach would give more freedom to the user, who could use those libraries himself, and also would meet the speed and processing time requirements.

DICElib uses a server-client architecture internally, but this is transparent to the developer and does not influence on the architecture of the application being developed. When the application is run and DICElib is called, it takes control and runs the internal server. This server spawns the processes among the cluster nodes. This approach has some advantages: the developer can see the cluster as one supercomputer, instead of a number of nodes; and communication is reduced, since nodes are connected only to the server, and the server can filter redundant updates.

Currently DICElib implements synchronization and synchronous data sharing among the nodes (see Figure 1). The functionality is very simple, the server waits until that every node sends a message to it, and then the server sends a message to each node.

Another facility provided by DICElib is synchronous data sharing. Users can declare, on-the-fly, variables that are updated in other nodes only when the DICE_sync function is called. This is extremely handy for CAVEs, because while different nodes will be rendering different views, all have to be done exactly from the position. The developer may create new variable types, ensuring that structures or complex data is easily spread over the cluster. The most common data types (floats, integers, strings, etc) are available by default.

The solutions presented above, however, either constraint the developer to some particular libraries (such as OpenGL), or are low-level and require extra work by the developer to be able to synchronize computers. Description of the library we developed is presented below. After this, we will present the performance evaluation.
synchronization, and is very useful for most graphical applications. These applications could use one of the nodes to process input and physics, and the other nodes would only render the data, all that transparently to the developer.

4 Performance Evaluation

We have benchmarked DICElib to some extent. The hardware used was the Polux graphics cluster, were we considered two of the cluster nodes (Dual Pentium III 933MHz, 1GB RAM, optical Gigabit-Ethernet), setup that already provides a significant test of stability and performance.

The test used to benchmark was the following: given a vector of integers (4 bytes), update the vector and synchronize the applications ten thousand times. We ran this test for several different vector sizes, and the results are stated in the following graphs. Figure 2 shows the wall clock with low machine load. Figure 3 presents the time spent by process on the system.

We notice that the growth is linear with the number of variables, proving that there's no overhead for large packets, and the time depends only of the amount of data to be transferred.
The results are quite good: even transferring 4000 variables (that is, 16kb of data) per frame, we achieved 550.1 updates per second. It is more than enough for video synchrony, and most virtual reality applications are likely to transfer only a few bytes of data, containing information such as position, orientation, etc. The highest update frequency was for 250 variables, with an average of 5988.0 updates per second. The bandwidth taken is small: only 8.8Mb/s. The limitation seems to be imposed by TCP/IP buffering constraints.

![Figure 2. Wall Clock.](image1.png)

![Figure 3. User Time.](image2.png)
It is also to be noted that the user time is small, meaning that DICElib does not take much processor time. Since the bottleneck of real applications is, of course, the main loop (rendering, physical simulation, etc), update frequency will be determined directly by the application frame rate. It should be noted that the application took about 34% of one processor for all the tests.

5 Conclusion and Future Work

In the future, DICElib will support asynchronous shared memory, as well as support for tracker devices. DICElib is still under development, thus the results presented in this paper are likely to improve. Besides that, more features are planned, such as asynchronous variables and support for input systems, such as trackers, data gloves and haptic devices.

The results presented, however, show that DICElib could already be used for development of multi-projection virtual reality applications on clusters, since it achieves rates much higher than the needed for coherent video synchronization. The DICElib will be used to all kinds of applications, such as renderers (raycasting, raytracing, radiosity), teleconference (images), simulators (physics, engineering, medicine) and virtual reality. It has a great potential for multi-projection or others distributed environments, providing a low cost alternative to existing solutions. The platform used to develop DICElib is the Polux cluster, which is one of the first PC graphical clusters controlling a CAVE.

6 Acknowledgments

This project is in partly funded by Fundação de Amparo à Pesquisa do Estado de São Paulo, grant # 99/12693-1, with additional support from Intel Foundation and FINEP (Financiadora de Estudos e Projetos).

7 Bibliography