ELSEVIER

Computer Networks and ISDN Systems 29 (1997) 1635–1644

COMPUTER NETWORKS and ISDN SYSTEMS

A World-Wide Web based distributed animation environment

Chungnan Lee *, Tong-yee Lee [1], Tain-chi Lu [1], Yao-tsung Chen

Institute of Computer and Information Engineering, National Sun Yat-Sen University, Kaohsiung, Taiwan, ROC

Abstract

In this paper we describe the design of a distributed animation system built using the Java language, a Parallel Virtual Machine platform, and the World-Wide Web. We focus on two aspects. One is the design of a platform to support distributed 3D animation, the other is the improvement of the efficiency of the parallel computing. Due to the collaborative and distributed nature of the Web, the Web browser is integrated with the distributed computing system like a Parallel Virtual Machine. The model emphasizes the separation of interface and function. It provides a very friendly and portable interface to manipulate the PVM console and the 3D animation system. To improve the efficiency of the parallel computing, we propose a new load balancing strategy, called global distributed control to balance the load in the network processors. The algorithm not only has the ability to dynamically adjust to the load imbalance, but also has the fault tolerance ability. It performs the best when it is compared with three traditional load balancing schemes. © 1997 Elsevier Science B.V.

Keywords: World-Wide Web; Distributed animation; Java; Ray tracing; Parallel virtual machine

1. Introduction

3D animation has many important applications in areas such as film industry, engineering simulation, scientific visualization, and fluid dynamics. However, it suffers from the need of huge computational power and the lack of a collaborative environment. The distributed computing can provide large computation power for scientific visualization, and the Web can provide a highly portable display interface for users to access network resources. As the access of WWW network resources and use of WWW browser to obtain information has grown tremendously in the recent years, it is natural to incorporate the Web with 3D graphics and the distributed computing system into a distributed 3D animation system. The system allows users at different places to have interaction in the design of 3D animation through the computer network.

Hence, in this paper, we aim to develop an interactive, distributed 3D animation system across the Internet using the Web browser, Java and Javascript [1–4] and using a Parallel Virtual Machine (PVM) as the distributed computing platform [5,6]. The two main tasks are: the design of a friendly, interactive, collaborative network environment and the design of an efficient load balancing algorithm for ray tracing that is of paramount importance to 3D animation.

Currently most of Web documents are written in HTML and use the Common Gateway Interface (CGI) to run the external application at the server

* Corresponding author. Email: cnlee@mail.nsysu.edu.tw.

[1] Current address: Department of Computer Science and Information Engineering, National Cheng-Kung University, Tainan, Taiwan, ROC.

0169-7552/97/$17.00 © 1997 Elsevier Science B.V. All rights reserved.
PII S0169-7552(97)00078-0

site. The output of the external program is sent back to the client viewer. Instead of using CGI, we use HTML, Java and the Javascript language to simulate the PVM console in the WWW browser. Hence, users can use a WWW browser, like Netscape Navigator and Microsoft Internet Explorer, to connect a PVM host machine from any network workstation or a personal computer, and then be able to dynamically set the parameters, such as viewpoint change, background color, resolution, etc., and display the rendering result. Consequently, users can discuss, activate, and display the animation through the network. Java and Javascript are object-oriented, multi-thread and portable languages. Therefore, it is easy to build and add function models to the distributed animation system. The main purpose is to depart from the traditional single-user manipulation system to distributed rendering images.

Ray tracing [7–12] has produced the most impressive photorealistic images in 3D computer graphics and is the key component to the animation, which is widely applied to many areas. But tracing a realistic image requires numerous light contributions to a scene, and a large number of ray–object intersections needs to be calculated, it makes ray tracing very computational intense. When the objects are complex, it is difficult to maintain the sequential ray tracing computation efficient. Therefore, researchers have made great efforts to parallelize the graphics rendering on parallel architectures or on a cluster of workstations [9,11,12].

A PVM that can be a cluster of heterogeneous workstations to be viewed as a single parallel system, is used as the parallel computing platform. However, users have to use command-driven PVM console to add or delete computers in the parallel virtual machine. It is neither efficient nor user friendly. Alternatively, users can use a XPVM built on the X-windows system to manipulate the PVM system. However, it is limited to some machine architectures, like SUN Sparc, HP, and DEC workstations, the running of XPVM and the visualization of the rendering. As a result, it makes difficult to build a portable distributed environment. This motivated us to built a distributed 3D animation system that is not only efficient to render the 3D graphic image in parallel, but also to provide a convenient and interactive graphics user interface.

To speed up the rendering efficiency for the 3D animation, the careful design of a load balancing scheme is very important and has been a major issue in network computing for ray tracing and other applications. Hence, in this paper we also present a new load balancing strategy, called global distributed control. We show the results of comparison with a couple of traditional load balance schemes, such as the master–slave and the interleaved schemes [13–18].

The rest of the paper is organized as follows. Section 2 describes the system architecture. In Section 3 we present a global distributed control algorithm, performance evaluation, and results. Section 4 gives an overview of the system. The paper concludes in Section 5 with suggestions for further developments.

2. System architecture

Our distributed animation system is a client–server architecture. In the client site, we use HTML incorporated with Java applet to implement a front-end Graphical User Interface (GUI). With this arrangement, the WWW browser accepts a user's service request and then executes a Java applet. This Java applet will send the environment's parameters to the web server for further processing. For example, we implement a GUI for the PVM software in the WWW environment. The users use icon-driven GUI provided by the Java applet to dynamically control the PVM environment in the remote site. Then the Java applet will send the web server environment's parameters to reconfigure PVM architecture. Note that in our implementation, there is a socket-based connection between the web server and PVM daemon. This connection is in charge of passing PVM parameters to the PVM daemon. With this design philosophy, a user can easily obtain our service from any kind of machine as long as he/she has an access to the WWW browser. Our client–server system architecture is shown in Fig. 1.

In Fig. 1, the user requests our service from WWW browser. Then a Java byte code will be shipped to the client site. The end-user can configure the PVM heterogeneous computing environment by adding or deleting computing hosts from our front-

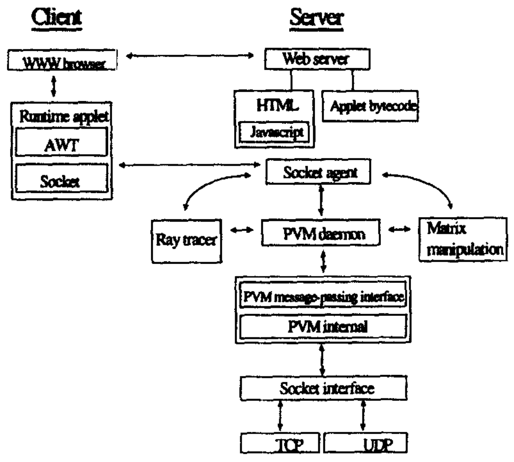


Fig. 1. The high level block architecture of the distributed 3D animation system.

end GUI. The Java applet establishes connection with a remote PVM daemon through the socket agent as inter-mediator and sends the system configuration to the PVM daemon. This PVM daemon later will execute the pvm_addhosts routine to form a virtual parallel machine that consists of the cluster of hosts being added.

Similarly, we send the graphics rendering information to our parallel ray tracer or send back the rendering results to the WWW browser through the same socket agent. Once the virtual parallel machine is available, the rendering information, including the parameters of the image data, is broadcast to each processor by the ray tracer through the PVM daemon.

3. Algorithm

In addition to developing new acceleration techniques for ray tracing on sequential machines, researchers have been devoted to parallelizing ray tracing on many experimental and commercial multiprocessor machines [11,12,14,18–21]. However, only a few work have been reported on the parallelization of the ray tracing problem on a network of workstations. In this section, we first present the global distributed control algorithm [14] to parallelize ray tracing on networked clusters of workstations. Then, we use an experimental protocol to evaluate the

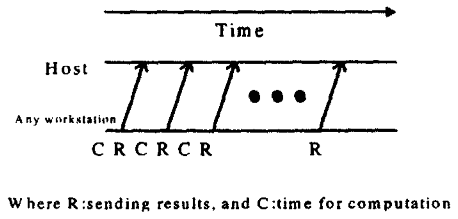


Fig. 2. The communication mechanism for the first phase of the GDC algorithm.

performance of the GDC algorithm and other methods [15]. Finally, we present the results in the last subsection.

3.1. Global distributed control

There are two phases in the GDC algorithm. In the first phase, the equal partition to assign scanlines I among N workstations in an interleaved manner (i.e., I mod N) is sued. This phase consists of two kinds of process, a master and a slave. The master is responsible to the collection of the results which are sent from slaves and to display the image. The communication diagram for the GDC algorithm in the first phase is shown in Fig. 2.

In this communication diagram, when a slave finishes the current scanline, it sends the rendered results back to the master without waiting for the arrival of the new job from the master. Because the computing power of workstations may be different and the load may be imbalanced, the slaves in the high computing power or low load workstation will finish their jobs faster than those in the low computing power or heavy load workstation. Under the circumstances, the GDC algorithm will dynamically adjust the imbalanced workload by the second phase procedure. The second phase of GDC is described as follows.

We assume N processors are logically connected by using a ring structure as shown in Fig. 3.

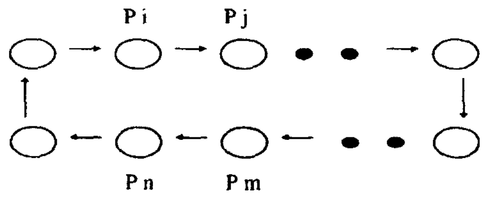


Fig. 3. The ring structure of N processors

During rendering, when a processor P_i becomes idle, it sends a message to its next processor P_j to request unfinished scanlines. However, there are several cases to be considered.

Case 1: If P_i is not idle, P_j will send an extra task to P_i.

Case 2: If the processor P_j has finished its pre-assigned tasks (i.e., in the first phase), then P_i will use the P_{i-1}, P_{i-2}, P_{i-3}, ..., P_k sequence to search an unvisited processor that is not idle. If P_i finds an unvisited process P_m that is not idle, P_i process will keep its integer task identifier. If P_i wants to request a new job, P_i will skip the processes between P_i and P_m and directly ask a new job from P_m.

Case 3: If P_i's next processor P_j was idle, but P_j has found a processor P_m that is not idle. Hence, the processor P_i sends a message to the processor P_j to ask a new job, P_j will acknowledge P_i to take a new job from P_m instead. Thus, the processor P_i does not have to waste time to search an unvisited processor which is not idle.

If P_i has visited itself, then the processor P_i will terminate.

GDC method is a kind of decentralized parallel computation environment. In the traditional workload method "master–slave", the master process is the bottleneck and each slave must wait for the arrival of new job from the master. However, the master in the GDC algorithm does not dynamically adjust load balancing. Each slave must perform real computation and control its own load balancing. The communication mechanism is shown in Fig. 4.

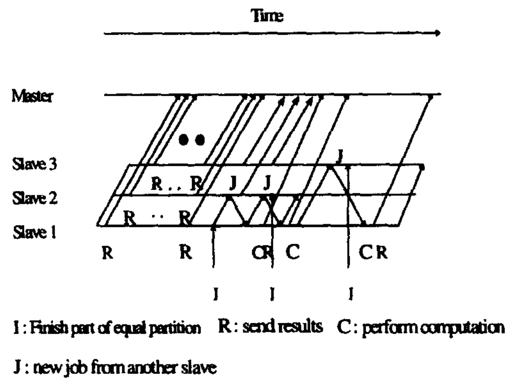


Fig. 4. The communication mechanism of the GDC algorithm.

3.2. Experiments

In order to evaluate the performance and understand the behavior of many applications on network computing environment systems, it is necessary to establish a common experimental protocol. The protocol should describe what is the system configuration, what application is run, what kinds of data types and volume are used, what parameters to be used, how to measure the performance, how change in parameters affect the performance of the application, and how to make improvements. The purpose of this section is to design and apply the experimental protocol to evaluate the performance of different algorithms on the network computing environments. The parameters considered in the protocol include environment configuration, data type, problem size, algorithm selection, number of workstation, and number of slave. For each experiment fifty data measurements are taken and the mean and variance is calculated. The parameters used in the experiments are briefly discussed in the following sections. The details are given in [15]

3.2.1. Environment configuration

Workstations used for the experiments include twelve HP 715/33, one Sun Sparc 2, and one Sun Sparc 20. The computing power for each type of workstation is measured by the data to be used in the experiment. The results are listed in Table 1.

3.2.2. Data type and problem size

To perform the experiments, a set of standard scenes from Eric Haines's database [13], called SPD, is used. The geometric characteristics of three test scenes — gears, balls, and tetrahedral pyramid — are shown in Fig. 5a. The complexity of these test scene data is shown in Table 2. To produce a biased object distribution, we adjust the view point of the test scene as shown in Fig. 5b. This change will result in different variances of pixel computations over this test scene. The large variance in scene tends to create difficulty to achieve a good load balancing. Therefore, to evaluate the soundness of a load balancing scheme, we control the view points to allow different variances of pixel computations.

Table 1
The relative power of the different workstations

	HP_UX	SPARC 2	SPARC 10	SPARC 20
Relative power	1	0.442	0.929	1.347

Table 2
The complexities of three test scene images

	Gears	Balls	Tetragonal
Time complexity	11.15	1	0.455
Size of structure	1152 rectangles	91 spheres	16 triangles

3.2.3. The number of slaves

Since the UNIX based workstation uses time sharing for it tasks. The more slaves are running in the system, the larger portion of CPU resources is allocated to the slaves. We would to like to know how the number of slaves affects the speed-up in one workstation.

3.2.4. Number of workstations

Currently fourteen workstations are used in the experiments. As indicated in [15], the order in adding hosts workstation will slightly affect the speed-up. Letting *a* represent for the HP 715/33 workstation, *b* represent for the Sun sparc 2, and *c* represent for the Sun sparc20. Then the sequence to add the workstation is an ordered set (*a, a, a, a, a, a, a, a, a, a, a, b, b, c*).

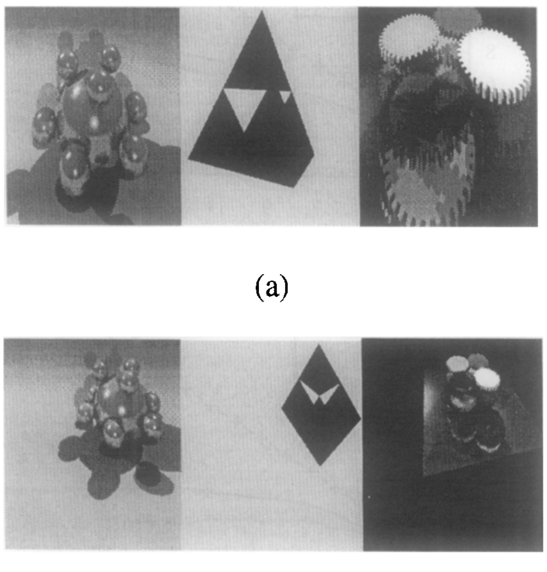


Fig. 5. (a) The balance-rendered image data. (b) The imbalance-rendered image data.

3.2.5. The algorithm selection

In addition to the GDC algorithm, we use three other algorithms to compare the performance of the load balancing. The first one is a static load balancing scheme called "interleaved" assignment. The idea is to assign scanline *I* to workstation *I* mod *N*, where *N* is the total number of workstations. Since neighboring scanlines should have similar computational complexity for ray tracing, the computational load can be more or less scattered evenly among all the workstations. The second, a dynamic load balancing scheme called "master–slave" approach, which is a popular strategy in network computing

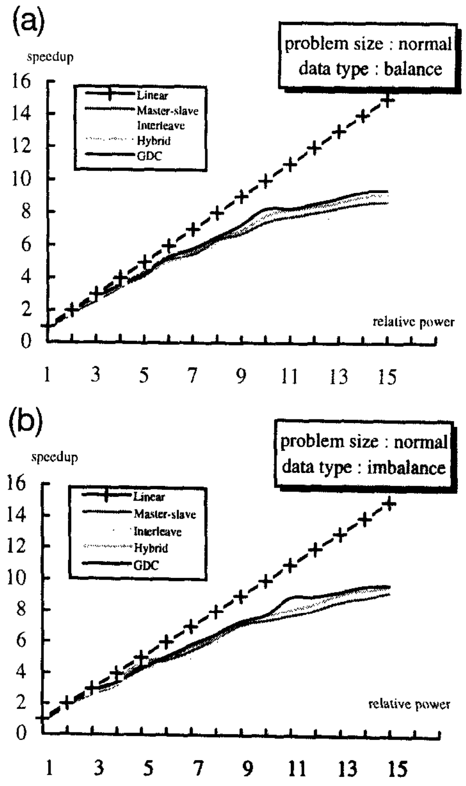
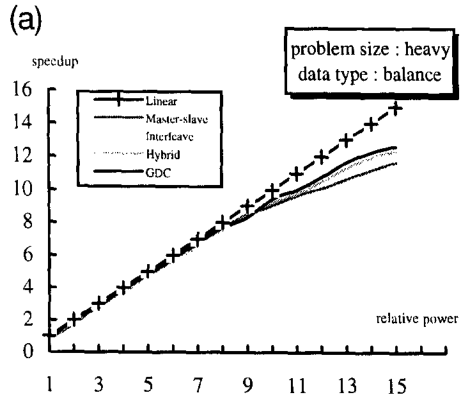


Fig. 6. The experimental results for the ball image data under balance and imbalance.



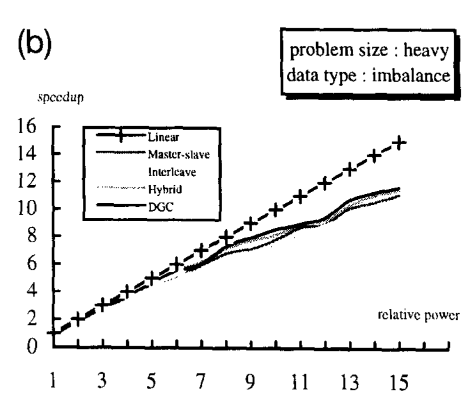


Fig. 7. The experimental results for the gear image data under balance and imbalance.

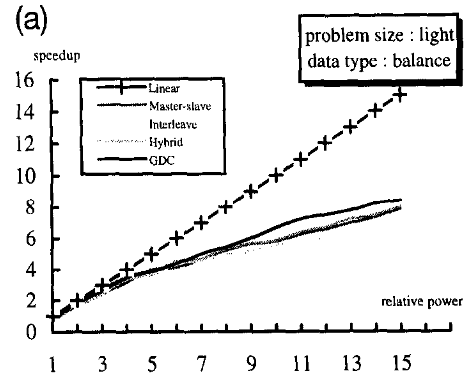
[16], uses a single master for task scheduling, results collections and image display, and uses multiple slaves to perform real computation. The third one is called a hybrid scheme as proposed in [15], takes advantage of both "interleaved" and "master–slave" schemes. There are two phases in the hybrid scheme where the first phase attempts to reduce the waiting time (W) in the master–slave scheme and the second phase attempts to dynamically adjust load imbalances incurred in simple "interleaved" scheme.

3.3. Results

Some results are shown in Figs. 6–8.

From the experimental protocol, the following observations can be made:

1. The GDC algorithm gives the best performance for all types of data.
2. The hybrid scheme takes the advantage of both interleaved and master–slave schemes, it gives the second best performance.
3. The interleaved algorithm gives the worst performance for all types of data. The static load scheme cannot adapt to the unequal computation power of workstations that causes the worst performance of the interleaved algorithm.
4. The higher the complexity of the image data, the better the performance in speed-up.
5. The performance deteriorates under imbalanced image data. As expected, the performance for the imbalance image is worse than for the balance image. They tend to saturate earlier.
6. The lower the complexity of image data, the earlier the saturation occurs.
7. The better linear speed-up is possible, if two slaves are spawned. When the number of slaves is increased, the execution time is decreased. But it is saturated very soon. That means that spawning



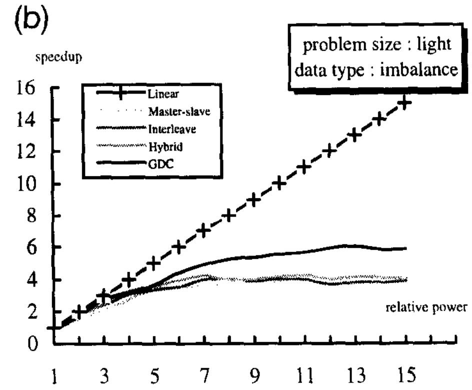


Fig. 8. The experimental results for the tetrahedral pyramid image data under balance and imbalance.

a couple of slaves in each workstation will speed up the execution of the task.

4. System overview

Our system integrates computer graphics, parallel computing and WWW technologies to provide a distributed animation environment. With the PVM software support, our computationally intensive graphics application can be executed on a set of machines in the networked environment. The WWW technology allows our system to be world-widely accessed via the Internet from different platforms. Our system consists of three parts: namely, the PVM console, the Rendering console and the Display console. We briefly describe each part to some detail as follows.

4.1. PVM console

The PVM console as illustrated in Fig. 9 provides a GUI front-end for users to add or delete a set of hosts in an interactive manner. Before adding hosts to the virtual parallel machine, the user can click the "Initial" button to list all available computing resources in our environment. This information is shown in the "available machines" subwindow. Then the user can interactively configure his/her

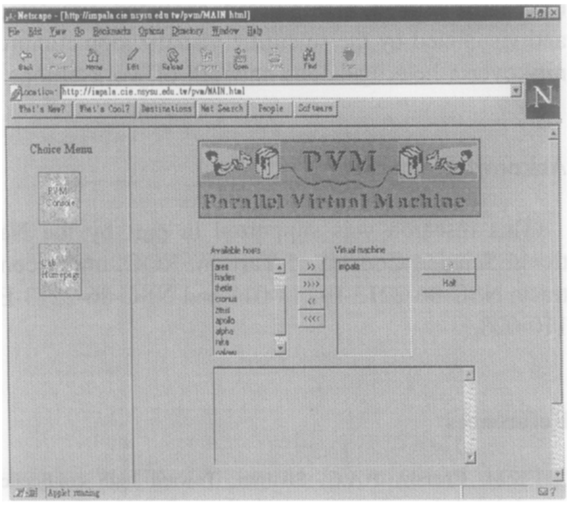


Fig. 9. The user interface of the PVM console.

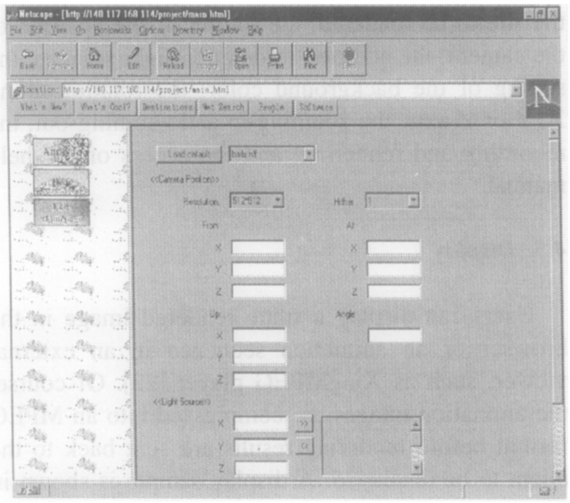


Fig. 10. The typical screen of the Rendering console.

virtual machine according to their computing requirement. The user adds any host to the virtual machine by moving the mouse to the host name listed in the "available machines" and then press the arrow button (> >) to confirm this selection. After this confirmation, the selected host name will be shown in the "virtual machine" subwindow. Similarly, the user can dynamically delete any host in the reverse direction (i.e., (<)). To provide more convenience, the user presses the (> > >) button to add all hosts into or the (<) button to delete all hosts from the virtual machine. In addition, we provide an extra subwindow to show the execution status of the PVM system. After successfully configuring their virtual parallel machine, the users must click the "Continue" button to perform parallel computation for ray tracing and other applications. Otherwise they can press the "Halt" button to shut down the entire PVM system. This command kills all PVM tasks, deletes all hosts from the virtual machine and stops the PVM system.

4.2. Rendering

After configuring their parallel virtual machine, the user can proceed to the "Rendering" console as shown in Fig. 10 for ray tracing images. The "Rendering" console provides options to dynamically control the process of ray tracing. The functions of the Rendering console include the specification of the file to be rendered, the setting of the position of the camera, the addition or deletion light sources, the setting of the background color, the rotation along X, Y or Z axis, the zooming-in and zooming-out, the recording and rendering, and the access of the help manual.

4.3. Display

Users can display a static rendered image in the browser or an animation sequence in an external viewer, such as XingMPEG player [22]. Of course, the animation images are compressed into an MPEG format before rendering results are sent back to the client to be displayed. A display sample is shown in Fig. 11.

4.4. Processor management

The processor management module allows users to allocate processors to execute applications without interference. A global processor pool is established by a PVM console for the user at the server site. Users can allocate a number of processors to form user's processor pool from the global processor pool. However, the user's processor pools are mutually inclusive. This allows the load to be uniformly distributed over the cluster of networked workstations, without degrading the performance of the usage by the foreground users. The policy is that of first come, first served. To avoid the waste of resources in a global processor pool, the processor will be released from the users' processor pool after it is idle for a certain period of time.

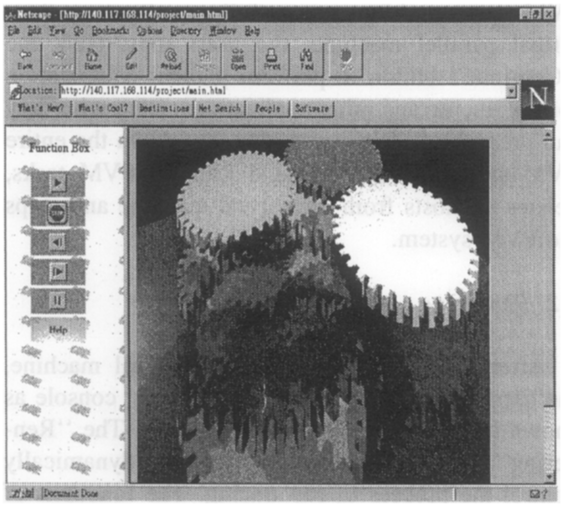

Netscape - [http://140.117.168.114/project/main.html]
Location: http://140.117.168.114/project/main.html
Function Box
Help

Fig. 11. The display screen.

5. Conclusions and future work

In this paper, a WWW-based interface is incorporated into a distributed animation system using Java and Java socket classes. This approach integrates a collection of function-specific tools into a distributed and extensible environment. It provides the fundamental components such as the WWW browser, the PVM console, the Rendering console, and 3D animation. To improve the efficiency of the parallel computing, we have proposed a new load balancing strategy, called global distributed control to balance the load in the network processors. In order to characterize the performance of different algorithms, we have used an experimental protocol based on the parameters: problem size, data type, number of workstation, and environment configuration. Results show that the global distributed control algorithm gives the best performance for all types of data compared with the other traditional algorithms. Currently, the functions for the collaborative work are primitive. In our future work, we would like to build multi-user (collaborative) design and analysis of environments such that a 3D animation system that can be created, shared, manipulated, analyzed, simulated, and visualized by the scientific visualization community over a heterogeneous network of workstations.

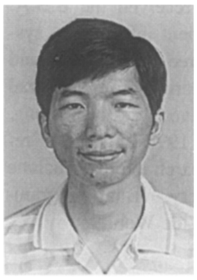
Acknowledgements

This research was supported in part by the National Science Council of Taiwan, ROC, under contracts NSC-86-2213-E-218-011 and NSC-86-2213-E-110-028.

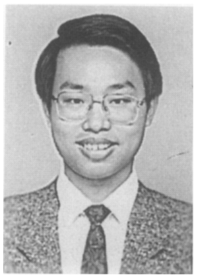
References

[1] K.M. Chandy, P.A.G. Sivilotti, Toward high confidence distributed programming with Java: Reliable Thread Libraries, in: Proc. Int. Conf. on Software Engineering, 1996.

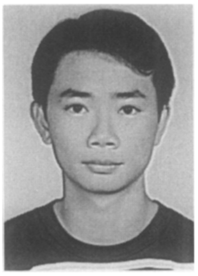
[2] J. Gosling, F. Yelin, Java Team, The Java Application Programming Interface, Addison-Wesley, Reading, MA, 1996.
[3] A. van Hoff, S. Shaio, O. Starbuck, Hooked on Java™ Sun Microsystems, Mountain View, CA, 1995.
[4] B. Ibrahim, World-wide algorithm animation, http://cuiwww.unige.ch/www/Bertrand.Html.
[5] V.S. Sunderam, PVM: A framework for parallel distributed computing, Concurrency: Practice Experience 2 (1990) 315–339.
[6] C.H. Cap, V. Strumpen, Efficient parallel computing in distributed workstation environments, Parallel Comput. 19 (1993) 1221–1234.
[7] J. Arvo, D. Kirk, A survey of ray tracing acceleration techniques, in: A.S. Glassner (Ed.), An Introduction to Ray Tracing, Kluwer, London, 1989, pp. 201–262.
[8] D. Badouel, K. Bouatouch, T. Priol, Distributed data and control for ray tracing in parallel, IEEE Comput. Graphics Appl. 14 (1994) 69–77.
[9] C. Giertsen, J. Petersen, Parallel volume rendering on a network of workstations, IEEE Comput. Graphics Appl. 13 (1993) 16–23.
[10] T.L. Kay, J.T. Kajiya, Ray tracing complexity scenes, ACM SIGRAPH 20 (1986) 269–278.
[11] W. Lefer, An efficient parallel ray tracing scheme for distributed memory parallel computers, in: Proc. Parallel Rendering Symp., San Jose, CA, 1993, pp. 77–80.
[12] S. Whitman, A task adaptive parallel graphics renderer, in: Proc. Parallel Rendering Symp., San Jose, CA, 1993, pp. 27–34.
[13] E. Haines, A proposal for standard graphics environment, IEEE Comput. Graphics Appl. 7 (1987) 3–5.
[14] T.Y. Lee, C.S. Raghavendra, J.B. Nicholas, Experimental evaluation of load balancing strategies for ray tracing on parallel processor, Integrated Comput.-aided Eng. J. 4 (1997) (in press).
[15] C.N. Lee, T.Y. Lee, S.F. Hsiao, T.C. Lu, Performance evaluation for parallel computing on network environments, J. High Performance Comput. (1996).
[16] D. May, Toward general purpose parallel computers, Mass. Inst. Technol., Cambridge, MA, 1989.
[17] B.K. Schmidt, V.S. Sunderam, Empirical analysis of overheads in cluster environments, http://www.netlib.org/pvm3.
[18] S. Whitman, Dynamic load balancing for parallel polygon rendering, IEEE Comput. Graphics Appl. 14 (1994) 41–48.
[19] M.B. Carter, K.A. Teague, The hypercube ray tracer, in: Proc. 5th Distributed Memory Computing Conf., 1990.
[20] H. Kobayashi, S. Nishimura, H. Kubota, T. Nakamura, Y. Shigei, Load balancing strategies for a parallel ray tracing system based on constant subdivision, Visual Comput. 4 (1990) 197–209.
[21] J. Packer, Exploiting concurrency: a ray tracing example, in: The Transputer Application Notebook, Inmos Ltd., Bristol, 1989.
[22] XingMPEG Player, Xing is a registered trademark of Xing Technology Corp., San Luis Obispo, CA.



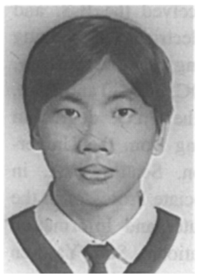
Chungnan Lee received the B.S. and M.S. degree in electrical engineering from National Cheng Kung University, Tainan, Taiwan, ROC, in 1980 and 1982, respectively, and the Ph.D. degree in electrical engineering from the University of Washington, Seattle, WA, in 1992.He is an Associate Professor in the Institute of Computer and Information Engineering at National Sun Yat-Sen University, Kaohsiung, Taiwan, since 1992. Prior to joining the faculty he was a system manger of Intelligent System Laboratory, a teaching assistant, and a research associate, while pursuing his graduate studies at the University of Washington. His current research interests include computer vision, character recognition, computer graphics, Web and Java computing, Web-based knowledge discovery, and parallel computing.



Tong-Yee Lee was born in Tainan county, Taiwan, ROC, in 1966. He received his B.S. in computer engineering from Tatung Institute of Technology in Taipei, Taiwan, in 1988, his M.S. in computer engineering from National Taiwan University in 1990, and his Ph.D in computer engineering from Washington State University, Pullman, in May 1995. He is an Assistant Professor in the Department of Computer Science and Information Engineering, National Cheng-Kung University Tainan, Taiwan. Prior to joining the Ph.D. program at WSU in 1992, he worked for Hitron Technology company in 1990 and Tatung company in 1991, in Taipei, Taiwan. He has collaborated with institute of computer and information engineering at National Sun Yat-Sen University in Koushung, Taiwan. His research interests include parallel rendering design, computer graphics, visualization, virtual reality, parallel processing, virtual surgical/medical system, distributed system, networking, heterogeneous computing, and collaborative computing framework design.



Tain-chi Lu was born in Pingtung, Taiwan, ROC, in 1973. He received the B.S. in computer science from Soochow University, Taiwan, in 1995. Currently he is working toward his Ph.D. degree in computer and information engineering at National Sun Yat-Sen University, Taiwan. His research interests are in the areas of computer graphics and parallel computing.



Yao-Tsung Chen received the B.B.A. degree in management information system and M.S. degree in computer and information engineering from National Sun Yat-sen University, Kaohsiung, Taiwan, ROC, in 1995 and 1997, respectively. His research interests include knowledge discovery, visual programming and World-Wide Web.