Distributed 3D Rendering System in a Multi-agent Platform

Risto Rangel-Kuoppa[1,2] Carlos Avilés-Cruz[1] David Mould[2]
[1]Departamento de Electrónica C. B. I.
Universidad Autónoma Metropolitana-Azcapotzalco
Av. San Pablo No. 180, Col. Reynosa, C. P. 02200,
México D. F.
rir785@mail.usask.ca caviles@correo.azc.uam.mx
[2] Department of Computer Science
University of Saskatchewan
57 Campus Drive, Saskatoon
Saskatchewan, Canada
mould@cs.usask.ca

Abstract

In this work, we propose a 3D rendering system that distributes rendering tasks across a multi-agent platform. The new approach is based on a multi-agent platform, where the goal is to create a virtual 3D environment. The main task is the rendering of individual objects. Each 3D object must be rendered in a remote unit; the resulting rendering is sent through the network to a 3D visualization process which generates the visualization of the whole 3D environment. The object movement and remote communication requirements have been implemented using a multi-agent system platform. The distributed system is implemented in Windows O.S., using DirectX graphical libraries and JAVA programming. The multi-agent platform used is JADE. The computer connection is a LAN at 100 MBS in a star topology.

1. Introduction

The work in this paper is motivated by our effort to build a distributed 3D rendering system that uses a set of personal computers to provide real-time rendering performance. Rendering 3D objects on a single PC presents no problem [4,8,9,10,11,12,14,15]; additionally existing protocols can be used to establish a network [16,17]. It remains to show how the computers communicate in order to share the rendered objects and generate a single 3D visualization of the distributed environment.

The general idea, having relaxed the constraints involved, is that distributed computing has proven to be a good approach to solve problems by decreasing the time of execution when executing parallel tasks: several computing resources can be used instead of one single powerful computing resource. In this work, we employ this approach to get a system for rendering virtual environments while avoiding the bottleneck of a centralized rendering process. The resulting system is able to support in terms of visualization performance more complex distributed virtual worlds than would a system with a centralized rendering process. Also, the approach we will describe makes use of a multi-agent architecture[1,2,3,21,23] for the distributed computing requirements.

The remainder of the paper is organized as follows. In section 2 we discuss the problem formulation; in section 3 we describe a possible solution; in section 4 we give details of our approach; we present our evaluation in section 5; and finally, we show results and conclusions in sections 6 and 7 respectively.

2. Problem formulation

Computing systems that are used to make 3D visualizations often have the architecture shown in Fig. 1. These architectures may or may not have distributed computing processes for the behavior of the objects in the virtual environment, especially if it is a dynamic environment, but the 3D visualization is centralized.

Centralizing the 3D processing creates a bottleneck that restricts the complexity of the 3D environment. Since computing power has increased not only in standard processing capabilities but also in graphics processing, using this graphics processing power in a distributed way is a natural consequence of distributed computing. We want to make use of distributed processing units to obtain a 3D visualization as shown in Fig. 2.

Proceedings of the Fourth Mexican International Conference on Computer Science (ENC'03)
0-7695-1915-6/03 $17.00 © 2003 IEEE

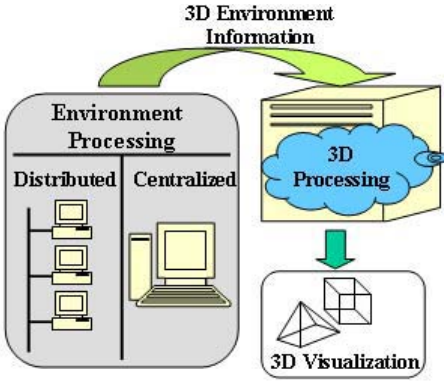

Figure 1 - Typical architecture of a 3D visualization system. The 3D environment information is computed either by a distributed or a centralized *Environment Processing* component and passed to a centralized 3D processing component in order to generate the 3D visualization.

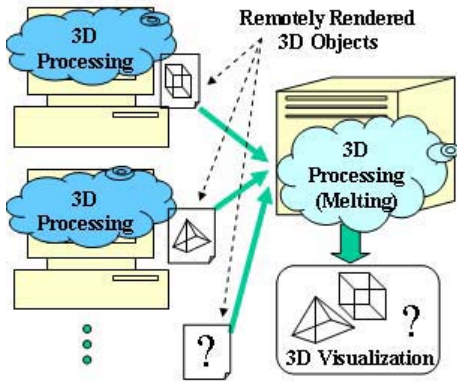

Figure 2 - Schema of the architecture which uses distributed rendering units to create a 3D visualization.

In particular, the problem is to create a virtual 3D environment in which there exist 3D objects associated with remote entities. Each of these 3D objects must be rendered in a remote graphical unit. The rendered objects will be sent through the network to a 3D visualization process which will compile the remote rendered images of all the objects into a visualization of the whole 3D environment. The problem is inspired by the idea of having distributed systems that support much more complex virtual environments by using distributed rendering capabilities than those using centralized rendering. We can therefore obtain the capability of supporting more complex environments with the same infrastructure. Applications of this work would be any architecture for distributed environments with a graphical visualization, such as video games, CAVEs, distributed virtual environments, etc.

The object movement and remote communication requirements have been implemented using a multi-agent system platform. We use a multi-agent platform in order to reduce the development time by using a tested tool. So, the remote entities with which the 3D objects are associated are remote agents in the multi-agent system

3. Overview of technology

This section has been split into two parts: the first describes technical considerations, while the second one describes the logical solutions, i.e., the algorithms used and the reasons why.

3.1 Technical considerations

The following considerations were taken into account in developing the project:

- Operating system - Two main options were available; Windows based or Linux. No other operating system was considered since these two are the most popular. We chose Windows since there are more 3D API's available and these make real use of the graphic acceleration hardware in the computers, unlike Linux API's that often do not use all the hardware acceleration capabilities.
- 3D API – Two main options were considered; DirectX or OpenGL. Since it is not clear which of DirectX[19] or OpenGL[20] has better performance, the decision was based on the authors preferences which is for DirectX.
- Development Language – For purposes of timely development the chosen language is Java, with NetBeans' IDE.
- Multi-Agent platform – The options were: Zeus, ADK and JADE [18], from which JADE was chosen due to the wealth of documentation and on-line resources available.

3.2 Logical solutions

There is no difficulty in rendering 3D objects in different computers; but we have many options in deciding how the computers communicate in order to share the rendered objects and generate a single 3D visualization of the distributed environment.

The first approach that we considered follows a request policy. This means that for generating each frame of the 3D visualization, all the remote agents are requested to send the rendered image of their associated 3D object. To generate the rendered image of the 3D object each

agent must know the *state* of the object. This state (speed, position, orientation) is sent to the agent so it can re-orient the 3D object as needed and render it correctly according to its current state in the environment. This approach is shown in Fig. 3.

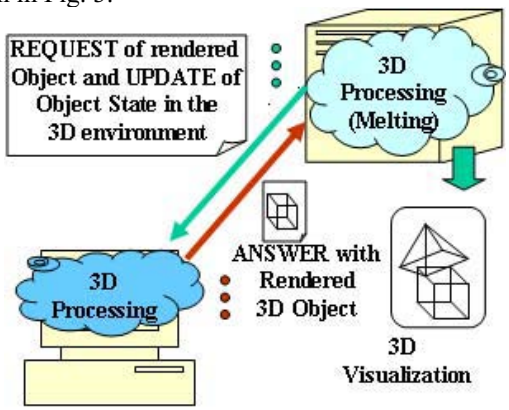

Figure 3 - First approach: to generate each frame of the 3D visualization each agent is requested to send its rendered object, and the request is done by updating the object state in the environment.

It is important to recognize that the multi-agent platform not only includes agents that make the remote rendering, but also one specialized agent, the visualization agent, whose main task is management of the 3D environment (objects' movement, collisions, etc.) and the rendering of the 3D visualization, i.e., the final view.

Preliminary tests suggested that this approach would have worse performance than the one finally implemented. The performance measured in the preliminary tests was such that the frame rate would be less than 10 frames per second (fps) with the server agent and one client agent running in the same computer

4. Description of our approach

Instead of a request-answer policy, our approach uses polling. Each remote agent is associated with a buffer in which the rendered image is stored; the agent that generates the 3D visualization takes the information from the buffer. This approach has better performance than the one explained in section 3.2, but it has the disadvantage that, depending on the speed of transferring the object from the remote agent to the visualization agent, there will be a difference between the logical state of the object in the environment and the visualized state. This approach is shown in Fig. 4.

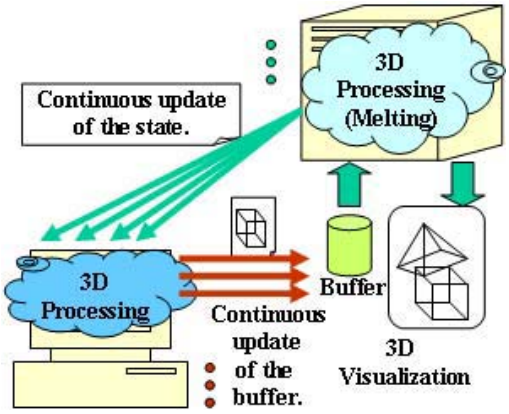

Figure 4 - Implemented approach: updates to the state of the object and update to the buffer are made asynchronously.

It is also important to understand that in this approach, updating the state of the object in the environment is done asynchronously with delivering the rendered objects.

Now that the main functionality of the implemented approach has been exposed, an important feature will be explained in detail. Synchronization in distributed environments is a very important issue because it allows having the state of the environment shared among its distributed components. In this work, synchronization is not only present when communicating the state of the object in the environment, but also when the graphical information is ready to be communicated. In Fig. 5 we can see the problems that may arise when synchronization is not available at the moment of sharing graphical information.

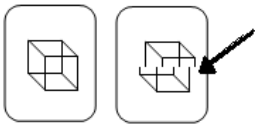

Figure 5 – Synchronization problem: If the image is sent when the frame has not been completely rendered, then visual distortion, due to object's movements in the environment, may occur.

Another problem of visual synchronization is called *flickering,* where the image at the display seems to flicker. The usual source of flicker is that the information displayed is being erased and rewritten several times per second. This problem is usually solved by using the *back buffer rendering* technique[22] and since this is automatically implemented in Java3D there was no related implementation in the project. In Fig. 6 we can see the points at which synchronization has been implemented in this paper.

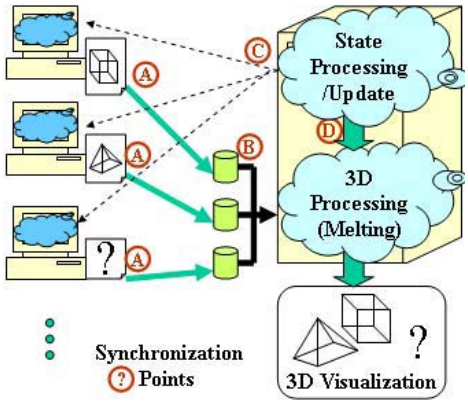

Figure 6 - Synchronization points of the project; A-Remote Rendering, B-Buffers Access, C-Remote state update, D-Environment state update.

The synchronization points are as follows:

A. Remote rendering synchronization. A rendered object will be sent to the visualization agent only when the object is fully rendered.

B. Buffer access synchronizations. Access/read to the buffers from the visualization agent is done prior and only once per frame of the 3D visualization and only when the buffers have a full rendered object on them. I.e., the visualization agent reads from the buffers once per frame, and is permitted to read only when the buffer contains a completely rendered object. The remote agents can write to the buffer only when the visualization agent is finished reading.

C & D. State update synchronizations. Remote state updates are done once per step; updates are sent to all remote agents. A step is an advance in virtual time by a small increment and the calculation of the new states of all objects in the environment in accordance to their positions, orientations, speed, and interactions with other objects – such as collisions.

To generate a frame of the 3D visualization, information from the buffers is used as well as information related to the virtual environment, such as background, ground geometry, position of objects, etc. The synchronization at this point is implemented by granting access to this information only once per frame, after each step calculation.

5. Evaluation of our approach

Why this approach? As mentioned in section 3.2, preliminary tests suggested that the first approach would have a lower performance than the one implemented. The explanation is as follows.

Sending and receiving messages in the multi-agent platform (JADE) is done through behaviors defined in each agent. The communication between the visualization agent and the remote agent is done through the respective behaviors. However, since the behaviors are executed in a round robin schedule and messages are delivered through a queue, the usage of behaviors to send messages *and* wait for an answer will slow the performance of the system due to the behaviors that can not be executed until the waiting behavior receives its answer. Actually, the documentation of the JADE platform strongly suggests designing behaviors so that they don't take too much time to execute.

As can be inferred from section 2, this result is a step into the research of distributed rendering systems, whose main goal is to avoid the bottleneck associated with a centralized rendering system. Also, as suggested in section 4, there remains a bottleneck associated with the rendering of the 3D visualization when it has to read the buffers of the remote rendered objects. Even if these buffers are used to generate the 3D visualization, however, this is faster than generating the 3D visualization by rendering each 3D object in a single computer. This is because rendering an object involves complex calculations for determining visible surfaces, light, color, shadow, texture management, etc., rather than the use of a buffer from which an image is taken and used as a sprite in the 3D visualization. To include a sprite in the rendering of 3D visualization the only computing process needed is the overwriting of memory locations, which is far simpler and faster than rendering a 3D object.

6. Results

Table 1 - Developed tests

	Software running in PC ...				
	Server	Client			
Test	1	1	2	3	4
I.1	A	A	-	-	-
I.2	A	A	A	-	-
I.3	A	A	A	A	-
I.4	A	A	A	A	A
II.1	A	B	-	-	-
II.2	A	A	B	-	-
III.1	A	B	C	-	-
III.2	A	A	B	C	-

Table 1 can be interpreted in the following manner: as an example, test I.3 is done by executing in computer A the server software, and also three clients in computer A.

Test III.1 is done by running the server in computer A, and 1 client in either computer B and C.

The Personal Computers used have the features shown in table 2 (processor type, RAM memory, graphic card and network card).

Table 2 - Hardware configuration for each personal computer used in tests. "A" computer is used as a server and "B" and "C" as a clients.

	Cmp A	Cmp B	Cmp C
Processor	Pent. IV 2 GHz	Pent. IV 1.6 GHz	Pent. IV 1.6 GHz
Mem	512 MB	512 MB	512 MB
O.S.	Win 2K	Win 2K	Win 2K
Vid. Card	32MB ATI Rage 128 Ultra	16MB ATI Rage 128 Ultra	16MB ATI Rage 128 Ultra
Net Card	CNet PRO200 WL PCI Fast Ethernet	Cnet PRO200 WL PCI Fast Ethernet	Cnet PRO200WL PCI Fast Ethernet

All computers were connected to a LAN with 100 MB bandwidth in a star topology.

Two main features were measured in the results stage: the frames per second (fps) and the time. For the first results we were interested in comparing performance when running the server and the clients in the same computer to performance when running the server and clients in different machines. Fig. 7 shows the results.

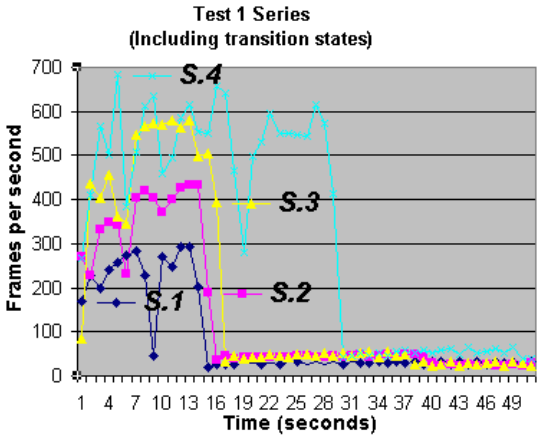

Figure 7 - Results from Test 1 series (.1, .2, .3, .4)

In Fig. 7 we can see how having the server and several clients running in the same computer affects the overall performance. The more clients, the less the frames per second. Further tests give more detailed numbers of this relation and also show the impact of having objects rendered locally and remotely.

Another interesting point to note from Fig. 7 is the *transition states*, which are the time intervals in which Java3D loads all the objects to use in the 3D environment. Also during this time, the JADE platform makes all of its connections and its agents start sending/receiving messages. During this time interval, the rendering process at the DirectX level does not have a lot of objects to render because these objects must be added from the Java3D layer, which is starting, and therefore high frame rates are achieved during the transition states.

Knowing this, the interpretation of Fig. 7 is as follows. The tests in which few clients are running take less time to give all the objects to the DirectX layer and therefore their transition states remain less than those of tests with more clients. Thus, the amount of time the server will be in its transition state directly depends on the number of objects that will be loaded in the rendering process.

We calculated the average behavior for the 100 samples after the transition period ended. The results are shown in Table 3.

Table 3: Average (without transition stages) behavior in each test 1 series without transition states.

Test:	I.1	I.2	I.3	I.4
Average	30.43 fps	28.04 fps	27.32 fps	27.03 fps

Table 3 shows the trend that with more clients running in the same computer with the server, the performance is reduced.

In Figs. 8 and 9 we can see how the frame rate is reduced in both A and B once the client logs into the server. Computer B has a higher frame rate than A because the rendering of a remote object is simpler than the rendering of the whole environment that is done in A.

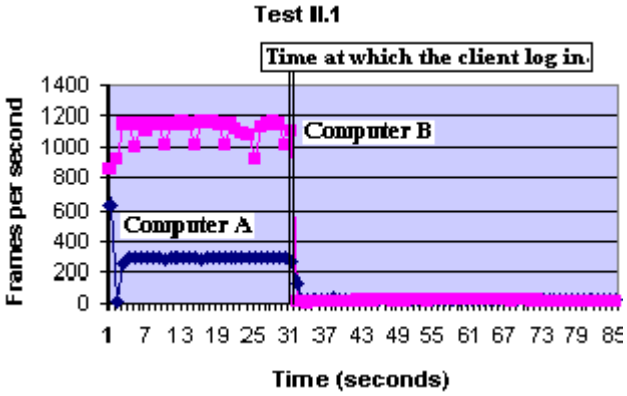

Figure 8 – Test II.1, the frame rates of the client and the server are reduced once the client logs in.

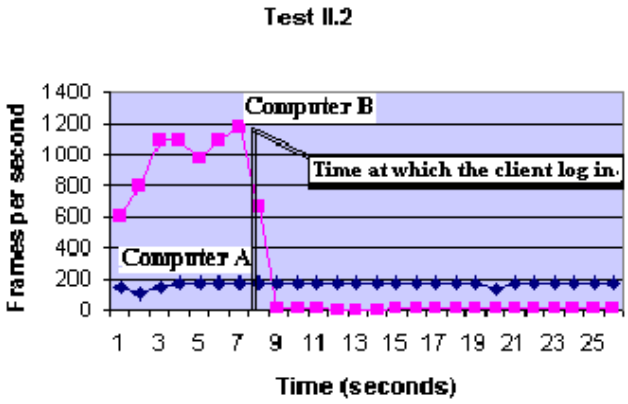

Figure 9 - Test II.2, the frame rates of client and server are reduced once the client logs in. Even if the client rate is reduced, the server rate is almost unaffected.

Without taking into account the transition states, the average frame rates previous and posterior to the client logging into the server are shown in table 4.

Table 4 - Average frame rates previous and posterior to the client logging into the server.

Test	PC	Previous	Posterior
II.1	A	282.433	36.301
	B	1029.931	17.666
II.2	A	170.400	43.513
	B	1032.625	8.904

Also notice that the frame rate of computer A, previous to the client log, is higher in test II.1 than in test II.2. This is because in test II.2, computer A supports the server and a client. Test II has been repeated in test III by adding a third computer. Figs. 10 and 11 show the results.

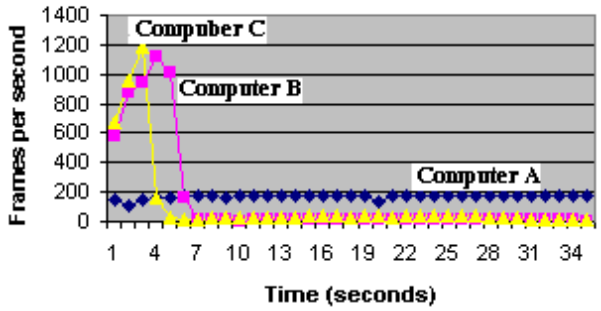

Figure 10 - Test III.1 the client in computer C logs first in the server running in computer A.

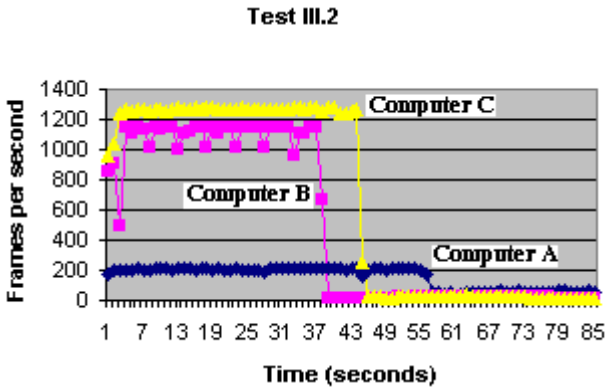

Figure 11 - Test III.2 the client in computer B logs first in the server in computer A that runs a client that logs the last.

Table 5 - Average frame rates for test III, in the states S1=Previous to any login, S2=After client in B log, S3= After client in C log, S4= After client in A logs (only in III.2).

Test	PC	S1	S2	S3	S4
III.1	A	170.4	53.7	52.8	-
	B	782.1	11.3	13.7	-
	C	742.5	766.6	14.7	-
III.2	A	203.8	54.7	54.2	18.7
	B	1072.7	13.1	17.7	16.3
	C	1253.1	1257.7	27.5	26.4

7. Conclusions and future work

From the values in table 4 and 5 and related figures, we can make the following key conclusion: even if the frame rate in the client is reduced when it logs in the server, the frame rate of the server is almost unaffected. This shows that using distributed rendering, adding objects to the environment has almost no effect on the performance in the rendering process. Therefore, systems with distributed rendering may support more complex environments than those with a centralized rendering process.

For the .2 subtest of test II and III we can see the effect of having a client running in the same computer as the server. In either case the execution/logging of a client in the same computer as the server provokes a bigger slowdown of the 3D visualization than the one provoked by a remote client. Also, from the right-most column in table 5 we can see that the logging of the client running in the same computer than the server has a major effect in computer A, but minimal in the other computers. This demonstrates the independence of the remote clients.

Further research must be done since new designs inspired by this project may allow more complex environments running in distributed environments than those supported in the same system but without distributed capacities to support rendering. Even so, the development of a generic platform that provides the services of distributed rendering, in a transparent way is an appealing prospect.

By using platforms for distributed computing (JADE) and visualization (Java3D) the development was fast, but at the cost of performance and delays due to the time required to learn the platforms and implement the additional functionality required for the project. The cost in performance is due to the number of layers involved in systems where several platforms are involved. In Fig. 12 we show the layers involved in this project.

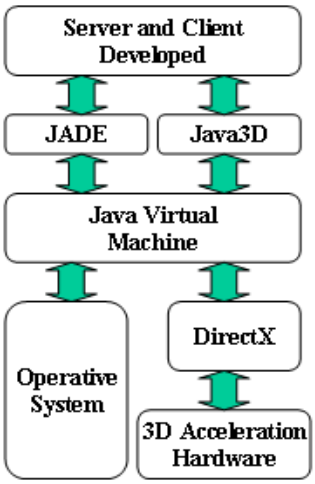

Figure 12 - The layers involved in this project.

Therefore, if a generic platform were to be developed then it is strongly recommended that it follow a design with fewer layers, and making use of another technology that does not use either Java3D or Java, because these are slow and may cause complex behaviors of the 3D objects in the virtual environment to be very slow.

Some questions rise in this project. When would it be better to have distributed rendering than a centralized rendering? Which are the costs and benefits? From the results obtained in this project we infer the relation shown in Fig. 13.

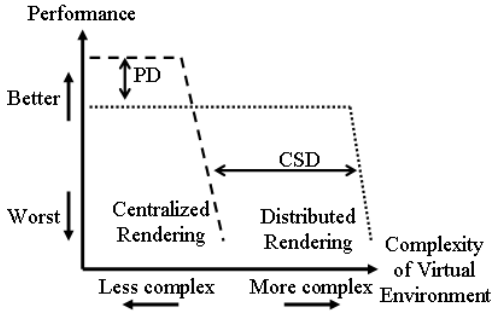

Figure 13 – Performance estimation between local and distributed rendering.

The Performance Difference (PD) between systems with distributed rendering and those with local rendering depends on the speed of the communication that allows the transfer of the remotely rendered objects.

The Complexity Support Difference (CSD) exists because in systems with distributed rendering, the bottleneck associated with a centralized rendering process is avoided and therefore this kind of system may support more complex environments than those supported by systems with centralized rendering.

Finally, it is important to understand that when using or creating a communication platform it is very important that the managing of messages is done asynchronously (parallel programming) to improve performance, rather than in a simulated asynchronously way.

8. Acknowledgements

The authors thank all the persons and organizations involved in the development of this project, in particular the Departamento de Electronica-C.B.I. of the Universidad Autonoma Metropolitana-Azcapotzalco and the CONACyT (National Council for Science and Technology - Mexico) who are sponsoring the PhD studies of Risto Rangel-Kuoppa. Also special thanks to the ARIES Lab at the Department of Computer Science of the University of Saskatchewan for the infrastructure provided for the tests.

9. References

[1] J. P. Singh, A. Gupta and M. Levoy, “Parallel Visualization Algorithms: Performance and Architectural Implications”, *IEEE Computer Journal*, July 1994, pp. 45- 55.

[2] W. H. Scullin, L.F. Tavera and C. L. Elford, “Virtual Reality and Parallel Systems Performance Analysis”, *IEEE Computer Journal*, November 1995, pp. 57- 67.

[3] E. Shaffer, D. A. Reed, S. Whitmore and B. Schaeffer, “Virtue: Performance Visualization of Parallel and Distributed

Applications", *IEEE Computer Journal*, December 1999, pp. 44-51.

[4] H. J. Noordmans, Hans T.M. van der Voort, A. W.M. Smeulders, "Perception-Based Fast Rendering and Antialiasing of Walkthrough Sequences", *IEEE Transaction on Visualization and Computer Graphics*, October 2000, pp. 196- 207.

[5] H. Hauser, L. Mroz, G. I. Bischi and M. E. Gröller " Two-Level Volume Rendering**",** *IEEE Transaction on Visualization and Computer Graphics*, July 2001, pp. 242- 252.

[6] D. Gordon "The Floating Column Algorithm for Shaded, Parallel Display of Function Surfaces without Patches" *IEEE Transaction on Visualization and Computer Graphics*, January 2002, pp. 76 - 91.

[7] K. Mueller, N. Shareef, J. Huang and R. Crawfis "High-Quality Splatting on Rectilinear Grids with Efficient Culling of Occluded Voxels" " *IEEE Transaction on Visualization and Computer Graphics*, April 1999, pp. 116 – 134.

[8] E. B. Lum, K. Ma, J. Clyne "A Hardware-Assisted Scalable Solution for Interactive Volume Rendering of Time-Varying Data", *IEEE Computer Journal*, July 2002, pp. 286-301.

[9] G. Kindlmann, D. Weinstein and D. Hart "Strategies for Direct Volume Rendering of Diffusion Tensor Fields", *IEEE Computer Journal*, April 2000, pp. 124-138.

[10] A. Raviv, G. Elber "Interactive Direct Rendering of Trivariate B-Spline Scalar Functions", *IEEE Computer Journal*, April 2001, pp. 109-119.

[11] G. J. Grevera, J. K. Udupa, . Odhner, "An Order of Magnitude Faster Isosurface Rendering in Software on a PC than Using Dedicated, General Purpose Rendering Hardware", *IEEE Computer Journal*, October 2000, pp. 335-345.

[12] F. Neyret, "Modeling, Animating and Rendering Complex Scenes Using Volumetric Textures", *IEEE Computer Journal*, January 1998, pp. 55-70.

[13] K. Kim, C. M. Wittenbrink, A. Pang, "Extended Specifications and Test Data Sets for Data Level Comparisons of Direct Volume Rendering Algorithms" *IEEE Computer Journal*, October 2001, pp. 299-317.

[14] J. C. Xia, J. El-Sana, A.Varshney, "Adaptive Real-Time Level-of-Detail-Based Rendering for Polygonal Models", *IEEE Computer Journal*, April 1997, pp. 171-183.

[15] Y. Sato, C. Westin, A. Bhalerao, S. Nakajima, N. Shiraga, S. Tamura, R. Bikinis, "Tissue Classification Based on 3D Local Intensity Structures for Volume Rendering", *IEEE Computer Journal*, April 2000, pp. 160-180.

[16] J.M. Frahm, J.F. Evers-senne amd R. CockY., "Network Protocol for Interactions and Scalable Distributed Visualization", *IEEE Proceeding of the first international Symposium on 3D data processing and visualization*, April 2002, pp. 18-26.

[17] L. Moll, A. Heirich and M. Shand, "Sepia: scalable 3D composition using PCI Pamette", *Proceeding of the 2do IEEE Symposium on computer and communications*, September 1997, pp. 230-240.

Internet sites

[18]http://sharon.cselt.it/projects/jade/-Project JADE homepage.

[19]DirectXhomepage
http://www.microsoft.com/windows/directx/default.asp

[20]http://www.j3d.org/implementation/java3d-OpenGLvsDirectX.html

[21]http://www.tommtisystems.de/go.html?http://www.tommti-systems.com/main-Dateien/reviews/opengldirectx/openglvsdirectx.html

[22]http://java.sun.com/products/java-media/3D/

[23]http://www.fraps.com