An Efficient Load Balancing Method for Parallel Ray Tracing on Heterogeneous Workstation Networks*

Pascual González, José L. Sánchez
Departamento de Informática
Univ. de Castilla-La Mancha
02071 Albacete, Spain

F. José Alfaro
Departamento de I.T. de Computadores
Universidad de Murcia
30071 Murcia, Spain

Abstract *In recent years, research on ray tracing has mostly concentrated on speeding up the algorithm by means of parallelisation. These approaches are usually based on the use of multicomputer and all the analysis of load balancing have been carried out by these architectures. But now the advances in high performance network allow us to think of another type of parallel platform. These architectures present new problems that hinder load balancing. In a network, the workstations connected to it usually have different computational power. This situation causes that some previous proposals of load balancing are not valid now. This paper examines the effectiveness of load balancing strategies for ray tracing on heterogeneous workstations networks. After this analysis we present an efficient load balancing strategy for this type of architectures that bears in mind the computational power of the connected stations to the network, and it reduces the communication needs.*

Keywords: Workstation network, load balancing, parallel ray tracing

1 Introduction

From the beginning, personal computer and workstations are small systems, designed to provide fast and predictable interactive performance on modest size jobs. Servers and mainframes are more expensive, oriented to more demanding applications and a large numbers of users. Supercomputers aim to achieve the ultimate in performance at any cost. But now, the computational power and the price/performance ratios of the new personal computers and workstations are improving more than supercomputers. Although the computational power of the personal computers and workstations are increasing, this increment is not enough to process some applications of intensive calculation yet.

So, how can this transformation of the technology base toward small computers be exploited?. Now we can formulate our answer to that question leaning on the ongoing technological convergence of local area networks and massively parallel processor interconnections [1].

In this case algorithms such as ray tracing, that were implemented by means of massively parallel computer ([2, 3, 4, 5], etc.), now would be analysed from others perspectives. The calculations can be divided among several computers connected to a fast network. In this new perspective we can use all the solutions contributed by the MIMD computers, but there are other problems that must be solved. Now we can have heterogeneous processors and it is very possible that these processors can be carrying out some other task. So the load balancing task must be different.

In this paper we present a new implementation of the ray-tracing algorithm in a personal computer network. In this implementation we have used the almost standard parallel library MPI [6, 7] that offers us a platform to pro-

*This work was supported in part by Spanish CICYT under Grant TIC97-0897-C04-02

gram this type of systems. Before explaining our implementation we will centre our exposition in the analysis of parallel ray tracing and the problems of load balancing. After that, we will present our algorithm, analysing basically the fundamental task of our proposal of load balancing. Finally, before the conclusion, we will analyse the behaviour of this algorithm and compare it with other proposals of static and dynamic load balancing in image space subdivision.

2 Load Balancing in Ray Tracing

Generating a realistic image of a complex scene by means of ray tracing requires to evaluate a lot of ray contributions, and therefore a large number of ray-object intersections. In recent years, attempts to reduce this rendering time have been based on the parallelisation of this algorithm. Ray tracing is intrinsically parallel because each pixel can be evaluated independently.

Like in the other alternatives for computer graphics rendering, in the parallelisation of the ray tracing there are two main approaches: *image space subdivision*, and *object space subdivision*. The first one, distributes the pixels to the processors, and the second one distributes the objects of the scene through the processors. This last distribution is mainly adopted for the parallelisation of ray tracing on distributed-memory message-passing architectures.

In the first scheme, image space subdivision, the overall pixel domain of the image space is decomposed into subdomains. Then, each subdomain is assigned to, and computed by a different processor. However, each processor should have access to all the information about the objects in the scene in order to trace the rays associated with the pixels assigned to it. The simplest way to facility this access, is that the scene can be duplicated in the local memory of each processor. But now there is an approached alternative that uses virtual shared-memory parallel computer ([5, 8], etc). This approach can ensure efficient data distribution and access by all nodes to the scene.

In the second alternative, object space subdivision, the object space is subdivided and stored in the local memories of the processors. The subdivision of the object space requires interprocessor communication, because a processor can not determine the colour of a ray for itself. There are two main alternatives for the interprocessor communication: based on movement of object or based on movement of rays

There are some authors that analyse this alternative in different MIMD architectures [9, 10, 4]. But according to D. Badouel, K. Bouatouch and T. Priol [5], this approach has important drawbacks. In the first place, this approach based on message passing is more difficult to design and its efficiency decreases with increased processor. In the second place, several regions may share the same object, implying repeated intersections with this object when the ray moves from region to region. And finally, and probably the most important, load balancing is difficult and expensive because the scene decomposition is not obvious when the aim is to balance the workload.

But anyway, to exploit any of these types of parallelism, we must ensure that the load is balanced, so all processors must have finished their works at the same time so that none of them are left idle while others are busy.

There are two ways of carrying out the load balancing: *static* or *dynamic*. If you use a static approach you must define the workload to each processor before starting the rendering phase. In another case, if a dynamic balancing is used, a load is assigned initially to each processor and when it has finished its process it requests more work to the host process.

In the static load balancing, you should predict the load of each part of the scene. For it you can use a preprocessing stage ([2], etc.) that allows you to see which is the distribution of the time of processing in each part of the scene. With this purpose, the image is subsampled in order to analyse the cost of tracing a set of rays by processing only one ray. This approach is mainly used in the object space

distribution and although it can be used to the image space distribution, in this case there are other solutions that do not need this preprocessing stage.

These other proposals try to obtain a static balancing by means of a proportional distribution of the different parts of the image. This distribution could be done by a *static contiguous method* or a *static interleave method*. If you select the static contiguous method you assign a block of contiguous pixels to each processor. In this case you must suppose that the distribution of the workload is uniform in the image and so you can assign a contiguous block. But usually the scenes have not a uniform distribution and the objects are strongly concentrated in some places of them. In these situations the static contiguous method does not carry out a correct load balancing, since some parts of the image are easier to render than others and so these processors will complete their jobs earlier than other ones.

According to Isler and Ozguc [3], the former problem is solved by applying the static interleave method. This method involves partitioning the image among P processors in such fashion that each processor i would process all scan lines i, $i+P$, $i+2P$, $i+3P$, etc. This distribution allows us to assure that each processor processes areas of the image with a similar load. The main disadvantage of this approach is the loss of coherence that hinders their integration with other techniques of acceleration, but if you suppose that the number of processors is low [11] and the computational power of all the processors is the same, an excellent load balancing can be obtained.

The last alternative to obtain a good load balancing is making a distribution dynamically. Although there are some works in the dynamic load balancing for object space subdivision, this alternative is more frequently used in image space subdivision [9, 5]. In this case, when a processor finishes its work it asks for new rays to the host. This solution ensures a load balancing but it usually needs a great number of remote access. This interprocessor communication increases the final rendering time.

In the next sections we are going to present a new proposal of load balancing for parallel ray tracing in heterogeneous workstation networks. This new proposal bears in mind the computational power of the connected stations to the network.

3 Our parallel algorithm for load balancing

Our proposal of parallelisation is based on the use of image space subdivision, so that the whole scene is accessible to all processors. In this case each processor can trace, in a completely independent way, the rays that are assigned to it. The election of this approach is based on the efficiency of this alternative with respect to the object space subdivision [5].

To implement the parallel version we have based our study on the library developed by N. Wilt [12] (Object Oriented Ray Tracing - OORT). This library implements in C++ all the basic classes needed to render a scene. OORT implements some algorithm [13, 14] to accelerate the ray tracing. This acceleration technique is based on the use of a bounding volume hierarchy. A user interface has been integrated to this library in order to facilitate the scene definition and the image representation.

In the same way, the development of our algorithm has been carried out using the standard MPI in its implementation LAM version 6.0 [15]. The goal of the Message Passing Interface is to offer a widely used standard for writing message-passing programs. MPI provides a wide variety of point-to-point and collective communications routines [6, 16].

In the implementation of our algorithm not all the processors connected to the network trace the scene. There is a process, denominated *host*, that carries out only the distribution of the scene among the rest of processors and finally it is responsible of picking up and visualising the obtained image. All the other processors, denominated *nodes*, are responsible

of tracing the assigned rays. With this task distribution, the program has been divided into two parts. One of them has the user interface and carries out the tasks assigned to the host. In the other hand, the nodes trace the area of the scene that has been assigned to them.

Our algorithm adapts other previous proposals to this framework. This approach keeps in mind the heterogeneity from the connected machines to the network. The algorithm has two basic tasks: homogenise the system and calculate the global computational power; as well as carry out the distribution of the rays to trace. Next we will analyse each one in detail.

The objective of the first one is to carry out a weighing of the system features that allows us to calculate the number of rays we have to assign to each process. To do this, firstly we should make an evaluation of the computational power of each different workstation of the system. To evaluate the computational power of any new machine, we use as benchmark the sequential program itself processing a standard scene. This evaluation is only carried out by new machines or those that change their characteristics. This task is performed by the nodes and each of them send to the host the computational power.

When the host receives these data, it determines which is the quickest machine and in function of this information, it calculates the relative speed of the resting machines. With this data, the host weighs the computational power of the different machines. After that, it can calculate the global computational power of the systems and so it can obtain the number of rays that each workstation must process. The host carries out the distribution of rays assigning to each process a certain number of lines, L_j, of the image (equation 1). This assignment allows us to take advantage of the ray coherence that the shadow cache method [12] uses for the acceleration of shadow rays.

$$L_j = \frac{CP_j \cdot VR}{\sum CP_j} \qquad (1)$$

where CP_j indicates the computational power of the processor j, and VR indicates the vertical resolution.

At this moment, our algorithm is at the same point that those other ones that analyse the load balancing in image space subdivision algorithms, because we know the number of scan lines that every process must trace. So, we can think of using any of the algorithms previously proposed. But these algorithms suppose that all the processors must trace the same number of rays, and in our case this number can be different, so the assignment and the distribution of rays must be distinct. If we analyse one method like the standard static interleave that obtains excellent results when we process the algorithm in a homogeneous network, however, we can see that its behaviour cannot be so good with a network of heterogeneous workstations.

Let us suppose that we assign the pixel in blocks of one scan line, and the number of lines assigned to the processes is sufficiently different, in such a way that a process P_1 should trace only 7% of the whole image. As we can see in figure 1, if the vertical resolution is 300 and the number of processes is 10, the last scan line assigned to this process is approximately 210. So, if the scene has a strong concentration of objects in the bottom of the image, these last scan lines should need more time than the other ones previously traced. Therefore, the process P_1 that has not any scan line in this area will finish its work earlier, provoking a load imbalance.

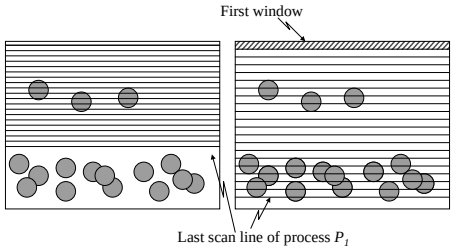


Figure 1: Different alternatives of scan lines distribution

So, to solve this problem, our algorithm intends to carry out the distribution of the rays to trace in a different way. In our case, the assignment is carried out on a piece of the image denominated window (figure 1), instead of considering the whole image. The size of the window is calculated in such a way that, the lines assigned to each process are distributed all around the image in the most homogeneous way. To do this we should keep in mind the process P_i to which a smaller number of rays has been assigned.

If we want to assign to this process scan lines in a homogeneous way, we should ensure that in each window we assign at least one line. So, as you can see in equation 2, the number of windows, NW, should be the same as the minimum number of scan lines assigned to this process P_i, and so, we can calculate the size of the windows, WW, dividing the vertical resolution of the image for this value (equation 3). In this way, as depicted in figure 1, being P_1 the process with smaller computational power, we will have 21 windows. So, this process traces one line in each one of them. The assignment of lines inside each window is made in an interleaved way, in function of its computational power regarding the size of the window and minimising the accumulated errors that take place due to the rounding during this process.

$$NW = minL_j \tag{2}$$

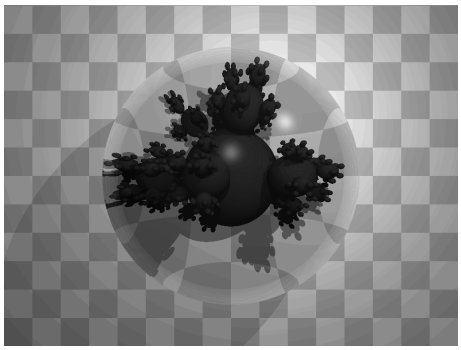
$$WW = \frac{VR}{NW} \tag{3}$$

It is interesting to stand out that our algorithm behaves the same as the static interleaved method, in case of a homogeneous system.

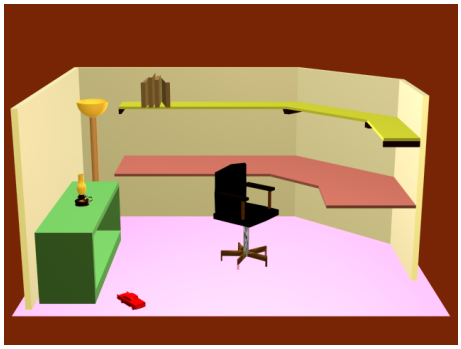
In the next section we are going to analyse the behaviour of our algorithm regarding other load balancing alternatives.

4 Analysis of the behaviour of the algorithm

To analyse its behaviour, our algorithm is compared with other load balancing alternatives for image space subdivision. Our proposal is compared to the static interleaved method and a dynamic alternative. The static interleaved method shows excellent results when the processors are homogeneous and they are not processing any other load. On the other hand, when the system is composed of heterogeneous machines, the dynamic method produces an excellent load balancing but it has the added cost of communications.



(a)



(b)

Figure 2: Test scenes used for evaluation

In order to test these algorithms we have used two different systems. The first of them is a network of 11 workstations (one host and ten nodes) PC's Pentium to 166 MHz with 16 Mb

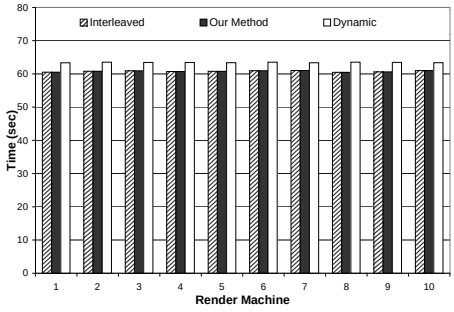
RAM. In the second case we use a network with 11 heterogeneous workstations. This is composed of the following computers: 2 PC's Pentium to 100 MHz with 16 Mb RAM (PC100), 1 PC's Pentium to 133 MHz with 32 Mb RAM (PC133), 8 PC's Pentium to 166 MHz with 16 Mb RAM (PC166). In both systems the host is always the same computer and all of them have the Linux operating system with Red Hat 5.0 distribution and with nucleus version 2.0.32.

Among the different scenes used to prove our algorithm, we have selected two of them. The first one is very simple and allows us to appreciate the behaviour of the different analysed alternatives in a better way (figure 2a). It has a transparent sphere that contains a sphereflake [17]. In this scene, most of the objects are located in the centre of the scene, so the scan lines associated to the centre of the image need more processing time than the other ones associated either to the beginning or to the end. The second one (figure 2b), is a more realistic scene that has a great concentration of polygons at the bottom of it. The scene contains approximately 50000 polygons, and about 20000 belong to the car located on the floor of the room. In all the tests we obtain an image with a resolution of 800x600.

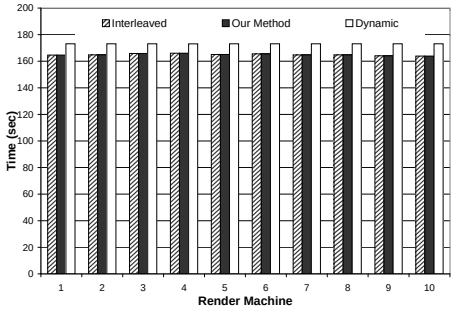
If we observe the results of all the methods analysed (figure 3), we can see that in all alternatives the processes take practically the same time tracing the piece that has been assigned to them, and so, they have an excellent load balancing. With these results we can show that our algorithm has the same behaviour as the interleaved method in these kind of systems. The differences among the dynamic method and the other ones are due to the added cost of communication.

Up to now, the machines used are supposed to have the same characteristics and all their computational power at our disposal. This hypothesis is not very realistic and clearly erroneous when we work with a network of workstations. Therefore, to analyse the behaviour of the different algorithms in this type of situations, we have used a system composed of heterogeneous workstations, which are initially unloaded. In figure 4 we can see the situation and the main characteristics of these workstations.

Now we analyse the behaviour of all the methods in these new situations. If we observe the results of the figure 4, we can see that the interleaved method obtains unexpected results. In these cases, as we have explained, the number of scan lines assigned to each process is different and it depends on the computational power of each ones. Now the assigned lines are not uniformly distributed in the scene, and so, we obtain an incorrect load balancing. Therefore, this method, that it is so good in homogeneous unload systems, has serious problems balancing the load in real systems in which the network is composed of workstations with different characteristics.



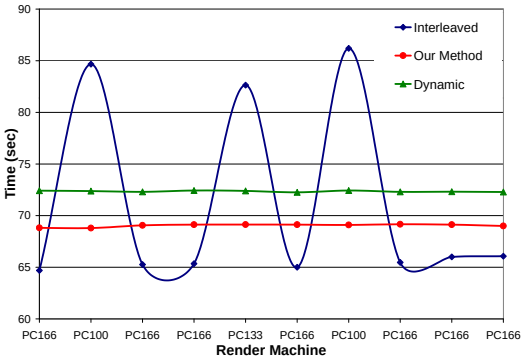
(a)



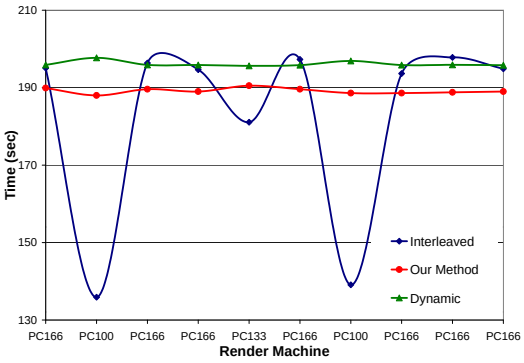
(b)

Figure 3: Behaviour of the different methods in a homogeneous unload network, a) for the first scene and b) for the second one

On the other hand, if we analyse the behaviour of the others algorithms, we can see in each case that the final rendering time of each process is very similar, and therefore they produce an excellent load balancing. But, if we observe the rendering time of the dynamic approach, we can see that this time es bigger than the one obtained by our algorithm. This bigger time is due to the fact that in the dynamic algorithm each process needs to ask for new scan lines when it finishes its previous load, and the final time is increased. So, our algorithm obtains results as good as the dynamic approach but with lower communications time.



(a)



(b)

Figure 4: Behaviour of the static interleaved, dynamic and our method in a heterogeneous network, a) for the first scene and b) for the second one

5 Conclusions

In this paper we have presented a new approach to balance the load in a distributed system for ray tracing. We have compared the behaviour of our algorithm with different approaches (static and dynamic) previously proposed. But, as we have been able to see, the static algorithms, that have an excellent performance in homogeneous unload systems, have bad results when are applied to heterogeneous systems. This is because these algorithms are based on the fact that the computational power of all the processors is always the same one.

In this environment we should keep in mind other factors when carrying out the load distribution among the different workstations connected to the network.

In that line we have presented an algorithm that takes into account the characteristics of the workstations. These data are used to calculate the global computational power of the system and in function of this, determine the lines to be assigned to each process. Finally the lines assigned to each process are distributed all around the image in the most homogeneous way. To do this, we define a window that is moved along the image. The distribution of the scan lines is made according to this window, obtaining, in that way, a uniform distribution regarding the whole image.

This new approach obtains as good results as the dynamic one, but its final rendering time is lower than in the dynamic alternative. These better results are due to the fact that the communication cost of our algorithm is lower.

References

[1] T.E. Anderson, D.E. Culler, and D.A. Patterson. A case for NOW (networks of workstations). *IEEE Micro*, pages 54–64, February 1995.

[2] T. Priol and K. Bouatouch. Static load balancing for a parallel ray tracing on a

MIMD hypercube. *The Visual Computer*, 4(12):109–119, March 1989.

[3] V. Isler and B. Ozugc. Fast ray tracing 3D models. *Computer & Graphics*, 15(2):205–216, 1991.

[4] C. Aykanat, V. Isler, and B. Ozugc. Efficient parallel sp subdivision algorithm for object-based parallel ray tracing. *Computer-Aided Design*, 26(12):883–890, December 1994.

[5] D. Badouel, K. Bouatouch, and T. Priol. Distributing data and control for ray tracing in parallel. *IEEE Computer Graphics and Applications*, pages 69–77, July 1994.

[6] Message Passing Interface Forum. MPI: A message-passing interface standard. *International Journal of Supercomputer Applications and High Performance Computing*, 8(3/4), 1994. Available at ftp://www.netlib.org/mpi/mpi-report.ps.

[7] Message Passing Interface Forum. MPI: A message-passing interface standard. Technical Report CS-94-230, Computer Science Dept, University of Tennessee, 1994.

[8] M.J. Keates and R.J. Hubbold. Accelerated ray tracing on the KSR1 virtual shared-memory parallel computer. Technical Report UMCS-94-2-2, Univ. Of Manchester, 1994.

[9] H. Kobayashi, S. Nishimura, H. Kubota, T. Nakamura, and Y. Shigei. Load balancing parallel ray-tracing system based on constant subdivision. *The Visual Computer*, 4:197–209, 1988.

[10] S.A. Green and D.J. Paddon. Exploiting coherence for multiprocessor ray tracing. *IEEE Computer Graphics and Applications*, 4(10):12–26, 1989.

[11] A. Heirich and J. Arvo. A competitive analysis of load balancing strategies for parallel ray tracing. *The Journal of Supercomputing*, 12(1/2):57–68, January 1998.

[12] N. Wilt. *Object-Oriented Ray Tracing in C++*. John Wiley & Sons Inc, 1994.

[13] T. Kay and J. Kajiya. Raytracing complex scenes. *Computer Graphics*, 20(4):269–278, 1986.

[14] J. Goldsmith and J. Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications*, 7(5):14–20, 1987.

[15] Ohio Supercomputer Center & The Ohio State University. *Local Area Multicomputer (LAM)*. LAM/MPI Parallel Computing. In http://www.mpi.nd.edu/lam.

[16] J.J. Dongarra, S.W. Otto, N. Snir, S. Huss-Lederman, and D. Walker. Message passing standard for MPP and workstations. *Communications of the ACM*, 39(7):84–90, July 1996.

[17] E. Haines. A proposal for standard graphics environment. *IEEE Computer Graphics and Applications*, pages 3–5, November 1987.