A concurrent dataflow algorithm for ray tracing

Jean-Christophe Nebel
E-mail: jc@dcs.gla.ac.uk

17 Lilybank Gardens Computer Science Department
University of Glasgow
Glasgow, G12 8QQ, UK

Abstract

Ray tracing has become established as one of the most important and popular rendering techniques for synthesizing photo-realistic images. However, the high quality images require long computation times and memory-consuming scene description. Parallel architectures with distributed memories are increasingly being used for rendering and provide more memory and CPU power. This paper describes a new way to employ such computers efficiently for ray tracing. Here each pixel is processed independently, so a natural method of parallelization is to distribute pixels over the machine nodes. If the entire scene can be duplicated in the memory of each processor (in the absence of global memory), a scheme without dataflow is used, otherwise, objects composing the scene have to be distributed over processor nodes. In this last case two strategies are applicable to the computation: object dataflow and ray dataflow.

The fastest of these algorithms is the one without dataflow. If we want to deal with realistic pictures described by large scenes, we have to distribute objects among processors. Accordingly a choice has to be made between object and ray dataflow. This choice is not easy because of variability in computers, communication networks and scenes, also algorithms might not be based on the same sequential model. Here, we propose a method which chooses the type of flow to use dynamically. This arose from the development of a parallel ray tracing algorithm that included the two modes of dataflow, i.e. a concurrent dataflow algorithm.

As our scheme allows the use of both algorithms, processors need to exchange objects and rays. The processors' load and some local parameters are used to choose between ray or object dataflow at any instant. Global information is transmitted by a non-centralized strategy, which assures scalability and the production of relevant messages. Finally, our algorithm offers a dynamic management of concurrent dataflows, which intrinsically assures dynamic load balancing.

Using concurrent dataflow our algorithm gives very encouraging results, as the computation time for rendering pictures and the number of exchanged messages are reduced by significant factors in relation to classical dataflow algorithms.

This concurrent algorithm is scalable as performances on a CRAY T3E with 64 processors shows. Finally the reduction of the magnitude of the flow of data communication reduces the risk of network saturation, which might otherwise compromise the algorithm.

Key-words

Computer graphics, rendering, ray tracing, parallelism, DMPC, MPP, load balancing.

1. Introduction

Ray tracing has become established as one of the most important and popular rendering techniques for synthesizing photo-realistic images. Originally introduced in a practical form by Whitted [19], it is one of the most complete and efficient rendering methods. The basic idea is simple: when an observer sees a point on a surface, he actually looks at the result of interactions between this point's rays and other rays coming from the scene. These interacting rays may come from light sources, reflections from other surfaces or refraction through transparent objects. Although of high quality, the images require long computation times which limits the practical use of the method, despite numerous acceleration schemes (e.g. Arvo and Kirk [1]). Moreover, the quest for realistic rendering requires memory-consuming scene description details. To deal with these problems parallel architectures with distributed memories are increasingly being used for rendering and provide both more memory and CPU power.

We start this paper by studying different kinds of parallelization used in ray tracing. Next we describe a new way to employ such computers efficiently for ray tracing, that is the use of object and ray dataflow concurrently. Finally, we present results and a discussion of these results.

2. Ray tracing on DMPC

Our paper deals with the parallelization of the ray tracing algorithm on computers with distributed memory. We start with a review of parallelization strategies used on parallel machine. This may provide both more CPU power and more memory, although the increasing of memory size is not always useful.

If data can be duplicated in the memory of every processor, a strategy without dataflow is the most efficient. Otherwise, if data have to be distributed over local memories of the parallel computer processors, communications between processors cannot be avoided. For such inter-processor communication two principal strategies have been proposed: object dataflow and ray dataflow.

2.1. Algorithms without dataflow

When the whole scene is duplicated on the local memory of every processor, parallelization is achieved by distributing pixels on processors, each processor computing independently a part of the picture. The efficiency of the algorithm is dependent on efficient load balancing. In order to achieve good load balancing, an efficient strategy for picture division has to be defined. There are two main kinds of distribution: static and dynamic.

When coherence between pixels is not used, static distribution gives good results. A comparison is made in Murakami and al. [12]. If, however, we want to use pixel coherence for accelerating computation, the use of some kind of dynamic distribution model seems to be essential. Dynamic load balancing schemes based on the master-slave model [16, 18] are efficient but not scalable. So for massively parallel computers, others strategies without any master [2], or with a master hierarchy [6], have to be used.

Parallelization without dataflow provides the highest efficiency, but the size of processor local memory limits its practical use.

2.2. Algorithms with object dataflow

As in the previous algorithms, each processor has to compute the entire subset of pixels allocated to it. When a computation cannot be performed because of a missing object, this piece of data is fetched and then copied in the local memory of the processor in order to complete its calculations. There are several ways to fetch the missing object: use of a virtual shared memory [4], a master-slave model [6] or a tree based architecture [7]. Again the way of distributing pixels on the nodes of the parallel machine is the key to good load balance. Most strategies presented for parallelization without dataflow (section δ2.1) can be used, but dynamic load balancing associated with a static scheme seems to be indispensable in getting the best results.

With these strategies, it is quite easy to get good load balancing on the computer nodes. Nevertheless the acceleration provided by these algorithms is strongly linked to the size of the local memory of each processor.

2.3. Algorithms with ray dataflow

In this kind of algorithm, each processor is responsible for a part of the scene's 3-D space. Accordingly they have in their local memory the objects which belong to their region of concern, then processors are only able to compute rays which cross their part of space.

When a ray leaves a region to enter another one the processor which originated the computation sends the ray to the processor owning data of the crossed area. This last processor continues the ray computations until the ray crosses another region of the space.

Load balancing of these algorithms is quite difficult, because it is the object partition which determines the processor load. Many different scene partition schemes have been proposed, most of them are based on uniform space subdivision [10], but some are based on pre-sampling [3, 9].

An efficient algorithm with ray dataflow should also use dynamic load balancing strategies. Load balancing is often performed using several processes on each processor [11, 17]. Some processes are involved in ray-object intersection computation, while others execute tasks which are not dependent on objects such as the determination of the region of space crossed by rays. With this strategy, processors which do not have any intersections to compute can be kept busy executing this second kind of process.

Load balancing for algorithms with ray dataflow is very scene dependent, so balancing schemes need to be very efficient regardless of scene complexity.

2.4. Conclusion

The fastest algorithms are those without dataflow. If a dataflow algorithm has to be used because data do not fit into local memory, a choice between ray and object dataflow algorithms must be done. The comparison between these two kinds of algorithms is not easy because of variability in computers, communication networks and scenes, also algorithms might not be based on the same sequential model. The literature has nothing to say about this choice.

3. A concurrent dataflow algorithm

Using results obtained previously [13], we define an alternative kind of dataflow algorithm. We give the main principles of the algorithm we implemented: a concurrent dataflow algorithm. Finally we show our experimental results.

3.1. An alternative algorithm

At first, we wish to deal with complex scenes. In this case we have no choice but to distribute the work of computing the scene which means we have to use an algorithm with dataflow. The use of massively parallel computers (MPP) such as the CRAY T3E leads to a decentralized algorithm.

In previous work [13], we showed that classical dynamic load balancing algorithms are often only used at the end of computation in order to correct load balance. This kind of strategy does not change the way in which dataflow algorithms are executed, so only limited improvements can be expected. Moreover it appears that the communication costs are still important, despite asynchronous messages. So more efficient algorithms have to address the problem of network load due to message handling.

Finally as there is no evidence of the superiority of one kind of dataflow algorithm over an other, it seems better not to choose arbitrarily between these algorithms.

The new algorithm we propose is not a modification of a classical algorithm, but an algorithm which radically changes the way pixels are computed. Our algorithm gives every processor the ability to choose the most efficient type of dataflow dynamically. This arose from the development of a parallel ray tracing algorithm that included the two modes of dataflow, i.e. a concurrent dataflow algorithm.

3.2. Principles

With this algorithm, every processor can exchange both objects and rays at any moment. The key is the way to choose between these two kinds of dataflow for optimal efficiency. This choice has to be guided by two main goals:

1. Keeping optimal load balancing between nodes of the parallel machine.
2. Reducing the amount of communication.

In order to achieve these goals, every processor needs several parameters for obtaining optimal choice. Some are local, but others are not. Every processor needs to know:

1. The load of every processor.
2. The number of rays which cannot be computed in the absence of communication.
3. The list of all the remote objects needed for the computation of the next intersection of each ray.

Most of these parameters can be computed locally, but the load on other processors cannot. This new algorithm needs a scheme which allows every processor to know the load of the others. Furthermore, considering that we wish to work on a MPP, we have to avoid “master-slave

schemes" because it leads to communication bottlenecks. Global information should be transmitted by a non-centralized strategy, which assures scalability and does not produce extra messages. Thus each time a message is sent because objects are missing, it includes a header, which contains the load of the sender i.e. the number of rays still to be computed. After a few messages, each processor have the knowledge of the load of every other processor in the parallel system.

With this scheme, every processor gets sufficiently precise information without any communication cost. Using this information and its own parameters a node has the capability to choose between object and ray dataflow at any instant.

The following is an outline of the algorithm:

```
IF my load is superior to a minimal load (I can send rays)
THEN
    FOR every processors
        IF my load is superior to the processor load
            AND the number of rays to send is sufficient
        THEN
            Send rays to the processor
        ENDIF
    ENDFOR
ENDIF
FOR every missing object (each object is supposed to be own by a single processor)
    IF the number of rays needing the object is sufficient
     THEN
        Ask for the object
    ENDIF
ENDFOR
```

For example, if a processor is busy, it will choose to send rays to processors which have a lower load. If a processor is idle, it will choose to ask for objects in order to perform ray computation locally.

Finally, our algorithm offers a dynamic management of concurrent dataflow, which intrinsically assures dynamic load balancing.

3.3. Experimental results

Now that the principles of our algorithm have been explained, we present our experimental results (the entire results can be found in [14]).

Our strategy has been implemented using the sequential ray tracer OORT [20]. The results presented here have been simulated on a CRAY T3E with 128 processors interconnected by a 3-D torus topology. We used PVM as a message-passing library. The size of each picture is 512x512 pixels and the maximal ray depth is 5. Scenes, i.e. *Mountain5*, *Ring1* and *Tetrahedron6*, come from the SPD database [8] and contain between 341 and 8688 objects.

We compare the results given by our concurrent dataflow algorithm with those of a classical object dataflow algorithm, which appears to be more efficient than the classical ray dataflow algorithm for the studied configurations [14]. The comparison between these two

algorithms is expressed in term of the *time profit* of the concurrent dataflow algorithm, defined here as the processing time needed by the object dataflow algorithm divided by the processing time needed by the concurrent dataflow algorithm.

The next figure shows the time profit obtained by the concurrent dataflow algorithm for various scenes using 64 processors of the CRAY T3E:

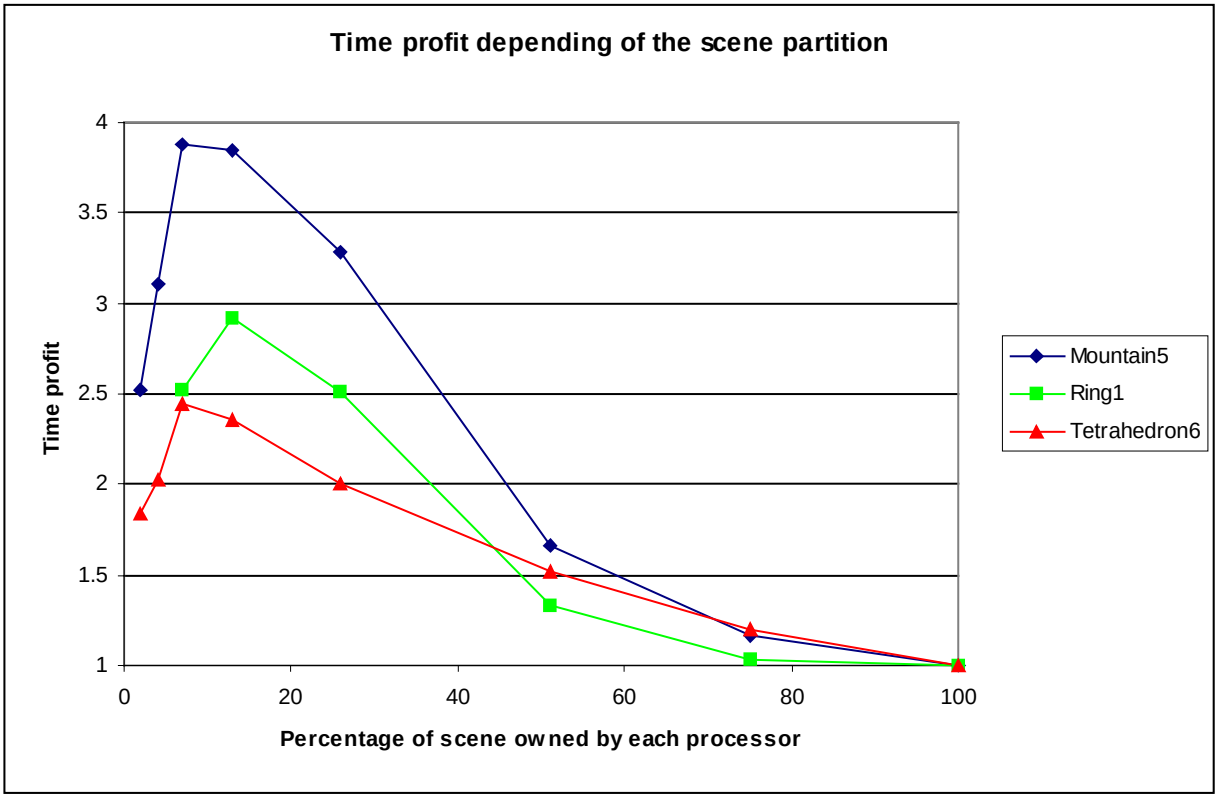

Figure 1: Time profit depending of the scene partition.

At first it appears that whatever scene is rendered, the time profits provided by our new algorithm are always quite high: the processing time of the *Mountain5* scene has been divided by almost 4! Moreover charts have very similar shapes, which suggests that the general behavior of the algorithm is scene independent. The best results are obtained when each processor owns a low percentage of the scene. When processors own less than the half of the object database, profits vary between 3.9 and 1.5, otherwise they are between 1.7 and 1.

This fact can be easily explained by the way in which global information is transmitted to processors. This information is distributed by messages generated for missing objects. When each processor owns most of the scene objects, very few messages are sent, so refresh rates for this information is quite low. Accordingly its precision may not be good enough to permit the best choice in every case. One solution to resolve this lack of precision would be to generate extra messages providing load information when the refresh rate is too low.

Our algorithm needs a sufficient flow of communications to reach its highest efficiency. When there is too much communication - when less than 10 % of the scene is owned by each processor -, some messages cause wait states. Then time profits provided by our algorithm decrease. (Nevertheless they continue to be better than profit of 1.8).

One goal of our concurrent algorithm was to reduce computation time by reducing the number of messages which are sent. In the next figure, we show the *message profits* induced by our new algorithm, defined here as the number of messages sent by the object dataflow algorithm divided by the number of messages sent by the concurrent dataflow algorithm..

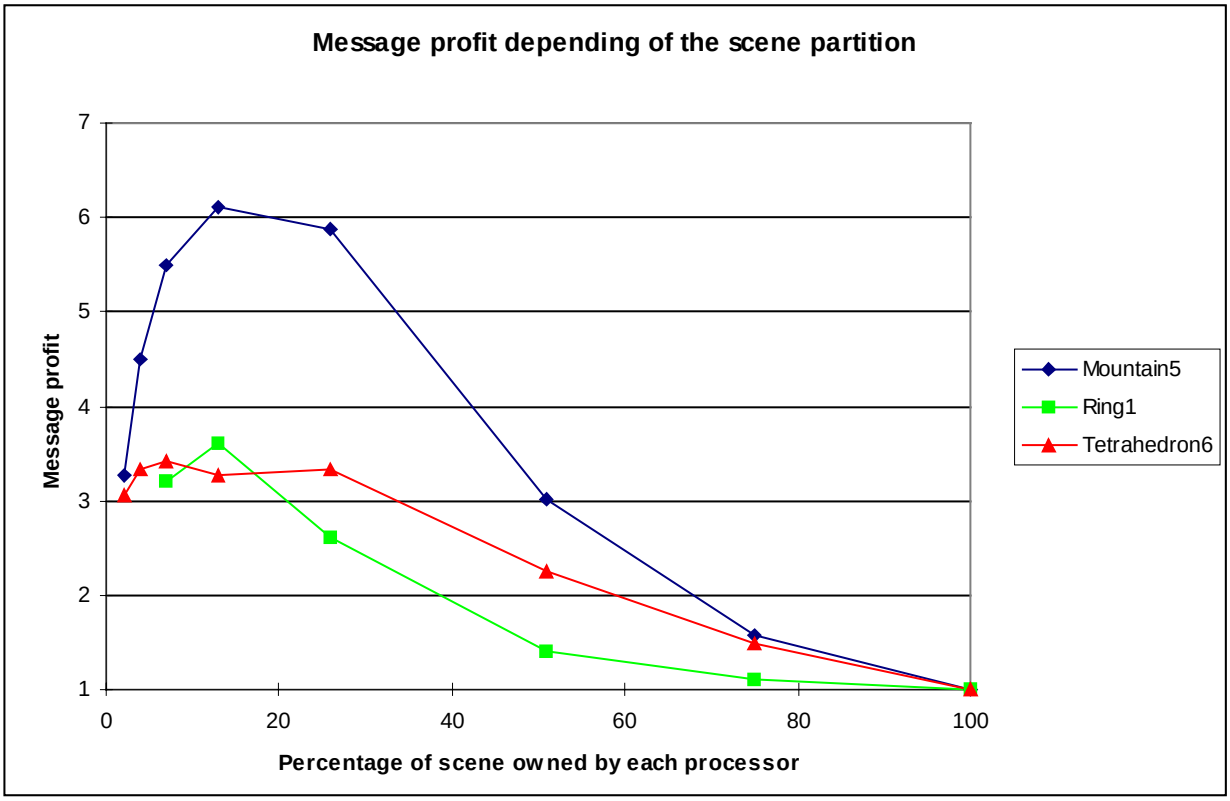

Figure 2: Message profit depending of the scene partition.

The charts on figure 2 are very similar to these of the figure 1, which clearly indicates that our concurrent algorithm improves computation times due to the reduction of message numbers. Message profits are between 6.1 and 1.7 when processors have loaded less than half of the database. Again we see that waiting for messages decreases profits.

Our algorithm using concurrent dataflow gives very encouraging results, as the computation time for rendering pictures and the number of exchanged messages are reduced by significant factors in relation to a classical dataflow algorithm.

This concurrent algorithm is scalable, as performances on a CRAY T3E show. Finally the reduction of the magnitude of the flow of data communication reduces the risk of network saturation.

5. Conclusion and future work

In this paper, we have presented a new kind of dataflow algorithm for parallel ray tracing which fits on MPP. The use of a concurrent dataflow gives encouraging results, seeing that the computation time for a picture has been reduce by 3.9 in relation to the performance of a classical object dataflow algorithm. Moreover communication flows are substantially reduced.
With this new type of dataflow, saturation of networks can be delayed, results on massively parallel machines are improved, and computation time reduced by a significant factor.

We are currently studying behavior of this scheme on networks of workstations and exploring the use of this algorithm for the parallelization of other algorithms (e.g. progressive radiosity [5,15]).

Acknowledgements

We thanks Eric Haines for supplying the SPD package [8] and Dr. John Patterson for helpful comments on the paper. This work was supported by the CEA (Commisariat à l'Energie Atomique) and the EMSE (Ecole des Mines de Saint-Etienne).

References

[1] J. Arvo and D. Kirk. *An introduction to ray tracing*, chapter 5. A survey of ray tracing acceleration techniques, pages 201-262. Academic press, 1989.

[2] D. Badouel. *Schémas d'exécution pour les machines parallèles à mémoire distribuée. Une étude de cas: le lancer de rayon*. PhD thesis, Université de Rennes I - IFSIC, Rennes, October 1990.

[3] D. Badouel, K. Bouatouch and T. Priol. Distributed data and control for ray tracing in parallel. *IEEE computer graphics and applications*, 14(4), pages 69-76, July 1994.

[4] D. Badouel and T. Priol. An efficient parallel ray tracing scheme for highly parallel architectures. *Advances in computer hardware v. rendering, ray tracing audiovisualisation systems. Lausanne, CH, 2-3 September 1990*, pages 93-106, September 1990.

[5] C. M. Goral, K. E. Torrance, D. P. Greenberg and B. Battaile. Modeling the interaction of light between diffuse surfaces. *SIGGRAPH'84*, 18(3), pages 213-222, July 1984.

[6] S.A. Green and D.J. Paddon. A highly flexible multiprocessor solution for ray tracing. *The visual computer*, 6(2), pages 62-73, March 1990.

[7] S.A. Green, D.J. Paddon and E. Lewis. A parallel algorithm and tree-based computer architecture for ray traced computer graphics. *Parallel processing for computer vision and display. Leeds, UK, 1988*, January 1988.

[8] E. Haines. A proposal for standard graphics environments. *IEEE computer graphics and applications*, 7(11), pages 3-5, November 1987.

[9] V. Isler, C. Aykanat and B. Ozguc. Subdivision of 3-D space based on the graph partitioning for parallel ray tracing. *Proc. second eurographics workshop on rendering, univ. of Catalonia, Barcelona*, 1991.

[10] H. Kobayashi, S. Nishimura, H. Kubota, T. Nakamura and Y. Shigei. Load balancing strategies for a parallel ray-tracing system based on constant subdivision. *The visual computer*, 4(4), pages 197-209, October 1988.

[11] W. Lefer. An efficient parallel ray tracing scheme for distributed memory parallel computers. *Proc. of parallel rendering symposium (1993), San Jose, California, October 25-26,* pages 77-80, October 1993.
[12] K. Murakami, K. Hirota and M. Ishii. Fast ray tracing. *Fujitsu scientific and technical journal*, 24(2), pages 150-159, June 1988.
[13] J.-C. Nebel. A mixed dataflow algorithm for ray tracing on the CRAY T3E. *Third European SGI/Cray MPP Workshop (1997), Paris, September 11-12*, September 1997.
[14] J.-C. Nebel. *Développement de techniques de lancer de rayon dans des géométries 3-D adaptées aux machines massivement parallèles.* PhD thesis, Université de Saint-Etienne, Saint-Etienne, December 1997.
[15] T. Nishita and E. Nakamae. Continuous tone representation of three-dimensional objects taking account of shadows and interreflection. *SIGGRAPH'85*, 19(3), pages 23-30, July 1985.
[16] I. Pandzic, N. Magnetat and M. Roethlisberger. Parallel ray tracing on the IBM SP2 and CRAY T3D. EPFL - *Supercomputing review*, 7, November 1995.
[17] T. Priol. *Lancer de rayon sur des architectures parallèles : étude et mise en oeuvre*. PhD thesis, IFSIC, Rennes, June 1989.
[18] F.V. Reeth, W. Lamotte and E. Flerackers. Ray tracing speed-up techniques using MIMD architectures. *Programming and computer software*, 18(4), pages 173-181, July 1992.
[19] T. Whitted. An improved illumination model for shaded display. *Communication of the ACM*, 23, pages 343-349, 1980.
[20] N. Wilt. *Object-oriented ray tracing.* Wiley, 1994.