PARALLEL RAY TRACING WITH 5D ADAPTIVE SUBDIVISION

G. Simiakakis
Harokopio University
Athens, 176 71
Greece
gsimiak@hua.gr

Th. Theoharis
Department of Informatics
University of Athens
Panepistimiopolis 157 71
Greece
theotheo@di.uoa.gr

A. M. Day
School of Information Systems
University of East Anglia
Norwich NR4 7TJ
England
amd@sys.uea.ac.uk

ABSTRACT

We present strategies for parallelising ray tracing based on 5D adaptive subdivision. Our goals are to obtain good speed-up and to efficiently balance the load between the processors while minimising the required memory per processor inherently large in 5D subdivision. First, loosely coupled strategies are presented, which are ideal for implementation on clusters of workstations, the most commonly used form of parallel processing nowadays. Then we consider a tightly coupled algorithm ideal for multiprocessors with fast interconnection network or shared memory. Finally, results on a cluster of workstations are presented and discussed.

Keywords: ray tracing, directional subdivision, ray classification, adaptive subdivision, parallel processing, distributed processing

1. INTRODUCTION

Ray tracing is a popular rendering method for realistic image synthesis. It is renowned for rendering reflections, refractions, shadows and has many applications. One of the major research areas in ray tracing is the acceleration of the brute-force algorithm which is extremely slow. Many sequential methods have been developed, including bounding volume hierarchies [Rubin80, Kay86], spatial subdivision [Glasn84, Fujim86, Kapla87, Devil89, Woo92], hybrid methods [Klima97] and directional subdivision methods [Haine86, Ohta87, Arvo87, Speer92, Simia94]. Furthermore, several parallel algorithms have been proposed. Some of them were never actually implemented or were only simulated in sequential systems, which makes their comparison difficult. All parallel methods were based on a bounding volume hierarchy, space subdivision method or even the brute-force algorithm, but none was based on directional subdivision.

Problems that were usually addressed are memory restrictions which lead to distributing the scene amongst the processors, and load balancing which can be difficult to achieve because of this distribution. The communication overhead was sometimes ignored. As the area of parallel processing changes, and as distributed processing becomes more widespread, the original significance of these problems constantly changes. Of course, load balancing is always desired. However, memory restrictions become increasingly relaxed while the communication overhead becomes more significant as the performance of processors rockets up. We propose a parallel ray tracing algorithm based on 5D adaptive subdivision (FAS) [Simia94] which is suitable for distributed processing on heterogeneous clusters of workstations.

In the next two subsections we look briefly at previous work in parallel ray tracing and directional methods with emphasis on FAS. In section 2 we present the loosely coupled parallel FAS (PAFAS), and in section 3 the tightly coupled. In section 4 we present and discuss our results. Finally, section 5 is dedicated to the conclusions.

1.1. Parallel ray tracing: previous work

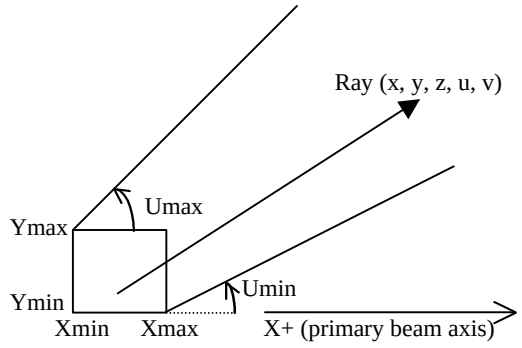
Some of the existing methods duplicate the entire scene to all processors, resulting in non data-flow algorithms [Bouvi85, Narus87], each processor executing essentially a sequential ray tracer. We shall see the advantages and disadvantages of this strategy in section 2. On the other hand, there are plenty of methods which distribute the objects amongst the processors. Of these, some use ray-flow (rays are passed to the processors which hold the primitives they have to be intersected with) and

some object-flow [Green90, Badou90]. The latter use a shared virtual memory model. Objects are sent to a processor when they are needed to perform ray-object intersections and they do not reside in the local memory. The efficiency of this approach largely depends on the percentage of the scene description that can be stored in the processors' cache. A simpler algorithm can derive from the use of a shared memory multiprocessor [Parke99].

The former category, which uses data oriented parallelisation, includes BV hierarchies [Salmo88, Caspa89] and spatial subdivision algorithms [Clear86, Dippe84, Nemot86, Caube88, Priol89, Isler91, Kim96], which are prone to big communication overheads and/or load balancing problems. The algorithm described in section 3 falls into this category.

1.2. Directional methods and FAS

The first directional method comes from Haines and Greenberg [Haine86]. They invented the light buffer, a 2D directional method restricted to speeding up shadow rays for point light sources. They introduced the direction cube which is widely used in directional methods to define the ray direction. Other directional approaches were attempted by Ohta and Maekawa [Ohta87] (for spheres or sphere-bounded primitives), by Arvo and Kirk [Arvo87] and by Speer [Speer92]. The latter does not seem to provide a sufficiently good intersection culling function for general scenes.



Projection of a 5D beam on x-y plane
Figure 1

Five-dimensional adaptive subdivision [Simia94] (FAS) is based on Ray Classification [Arvo97]. FAS improves on the shortcomings of Ray Classification reported by Speer [Speer92]. A brief description of the algorithm follows. The three dimensions of space together with two dimensions for the ray direction constitute a 5D space, which is the domain of all possible rays. A ray is defined by its origin and direction. The origin is given in the form of a 3D point in space, and the direction as two co-ordinates on one of the six faces of the direction cube. With the exception of eye rays, all rays are emitted from the point of intersection of a ray and an object. If there are no unbounded objects in the scene, all these points are inside the scene bounding box, so we can limit the spatial dimensions of the ray domain to that box. Rays emitted outside the scene bounding box are intersected with it. If they hit, their origins are translated to the point of intersection before being classified in 5D. Of course, all possible directions have to be included in the ray domain.

The algorithm adaptively subdivides (using lazy subdivision) the ray domain into 5D hypercubes and associates a candidate set of objects with each hypercube. A 5D hypercube in space is a polygonal beam starting at an axis-aligned box and surrounded by at most 6 infinite planes. The candidate set of a hypercube is the set of all objects whose bounding boxes intersect the beam. To intersect a ray with the scene objects we have to classify it to one of the hypercubes, and then intersect it with the objects of the associated candidate set. Subdivision and candidate set creation are done only on demand. The subdivision criteria are adaptively tuned to the scene during ray tracing. The algorithm incorporates a number of other features such as caching, backface culling, and candidate set truncation which improve its efficiency further. For a more detailed description see [Simia94].

2. LOOSELY COUPLED PAFAS: DISTRIBUTING THE IMAGE PLAIN

A straightforward strategy to parallelise ray tracing is to duplicate the entire scene description on all processors and distribute the image plane between them. A master processor sends pieces of the image plane to the slaves, collects the returned pixels and composes the final image [Chalm96]. The slave process can be easily derived from the sequential version of the ray tracer. If we suppose that the pixels are independent of each other, there is no need for communication between the slaves. Problems might arise if we want to use some kind of adaptive or progressive sampling like in [Notki97]. In this case the solution is to let the master handle the sampling and reconstruction of the image.

The advantages of this strategy are significant:

- minimal communication overhead
- dynamic load balancing is easy (as the master can send pieces of the image plane on demand to the slaves)
- it is suitable for a multiprocessor as well as a cluster of connected multi-user workstations
- deadlocks are not a problem, and even processor crashes can be handled gracefully
- it is easy to implement, which is why it is surely the most widely used strategy

However, the disadvantages should not be ignored:

- the amount of memory needed per processor can be prohibitive in the case of very complex scenes.
- geometrical sorting of all objects (like space subdivision, some kinds of directional subdivision, etc.) needs to be done on all slaves, thus is not parallelisable. This might happen during pre-processing or during the ray tracing phase as is the case with FAS.

The most important restriction of this approach is the memory per processor. If we have enough memory per processor for the scenes that we want to render, this strategy is the best. If not, it is simply not applicable. Of course, the required memory depends on the sequential acceleration method that is going to be used. Apart from the geometric and other properties of the objects we need to store the data structures used by the acceleration method. These can be significantly large in the case of spatial subdivision and much more in the case of directional subdivision.

In the case of spatial subdivision the same grid or octree is created on each slave. The significance of the cost of creating this subdivision compared to the ray tracing time, depends on the scene and the number of processors. For a large number of processors it might be that the pre-processing time becomes comparable with the actual ray tracing time.

The key to efficiency in parallelising FAS is that subdivision is done on demand. Only the parts of the hyper-trees that are needed to trace the rays allocated to a particular slave will be created on that slave. Thus different parts of the trees will be created on different slaves, and consequently and most importantly not all tree paths that were needed in the sequential FAS will be needed on each slave. Obviously the parts of the hyper-trees created on a slave will not be disjoint from the parts created on the other slaves. Minimising the amount of overlap is important in order to improve the speed-up, and this is essentially what we will attempt in the rest of this paper.

According to the principle of ray coherence, rays that are close to each other (have similar origins *and* directions) are likely to classify to the same leaf of the 5D hyper-trees, and more importantly rays that differ significantly in origin *or* direction are very unlikely to classify to the same leaf. The principle can be extended to whole ray trees, so that neighbouring pixels will require similar sets of leaves, whereas faraway pixels require almost disjoint sets. This principle holds for the leaves of the hyper-trees, but gradually weakens for higher levels in the trees, which is why the 5D trees on different slaves cannot be completely disjoint.
It is obvious that the required memory and the workload on a slave are connected properties. The more rays a slave has to trace, the more 5D beams it will need to generate. Of course, the relationship between the two is not constant since some beams might be used for many more rays than others. The tuning of the subdivision parameters is done locally on each slave according to its own statistics, so that it is possible for different slaves to create hyper-trees of different depths. This way the algorithm can adapt the subdivision better to the scene than sequential FAS can. In the strategies that follow we address the workload balance, and monitor how the memory requirements change with an increasing number of processors.

2.1. Scan lines in round robin

The easiest way to subdivide the image plane is to divide the number of scan lines by the number of slaves and send this number of scan lines to each slave. In order to experiment with static load balancing on a cluster of dedicated workstations, we implemented a version where a scan line S is sent to slave n if and only if S mod N = n where N is the total number of slaves. This way the image plane coherence is compromised to achieve static load balancing.

2.2. Using image coherence: bands of scan lines

Using the image distribution in section 2.1. with a spatial subdivision based ray tracer would be good enough, since little attempt to take advantage of image or ray coherence would be made. The only improvements that are necessary are dynamic load balancing and the control of the task granularity depending on the number of processors. However, ignoring the ray coherence in the case of FAS will lead to a significant overlap in the hyper-trees on different slaves. These trees require memory to be stored and take time to be created, so we have to investigate how to minimise the overlap by distributing the image plane differently.

Sending equally-sized bands of adjacent scan lines to each slave will take more advantage of ray coherence, but will cause havoc with the load balancing. So instead we are going to encourage slaves to render adjacent scan lines, unless the load becomes unbalanced, in which case we allow them to render other scan lines as well. The following strategy works well with a number of slaves at least one order of magnitude less that the number of scan lines. Each slave is initially allocated two adjacent scan lines. These initial `small bands' are spaced out evenly over the image, with the first slave getting the two first scan lines and the last slave getting the

last two scan lines. When a slave has finished rendering the first scan line it sends it to the master together with a request for more work. The master keeps track of the bands already sent and assigns to the slave the two scan lines that have not already been allocated and that are *the closest* to the ones already rendered by that slave.

If the scene is completely uniform each slave will end up rendering a band of adjacent scan lines, but usually scan lines around the more complex areas of the scene (quite often around the centre) will be rendered by various slaves. A disadvantage of this approach is that the most complicated areas will be rendered last. This might lead to a long delay for the last two scan lines from one slave while all the others have finished. The advantages of this strategy are the dynamic load balancing and its ability to take advantage of ray coherence in order to reduce the amount of hyper-tree duplication.

2.3. Distributing blocks of pixels

A scan line is a region with a relatively long border (2p+2 for p horizontal pixels) and containing a relatively small area (just p). It is possible that by using rectangular blocks to split up the image we will make better use of image coherence and thus of ray coherence. A rectangular block of r by s pixels has a border of length 2(r+s) and an area of r*s. Of course, rendering a number of adjacent scan lines or blocks on the same slave eliminates their 'internal borders' which reduces the difference between the two methods in the case when the load is quite balanced. However if we manage to keep the regions made up of blocks compact, the gain will be significant when the load is unevenly balanced.

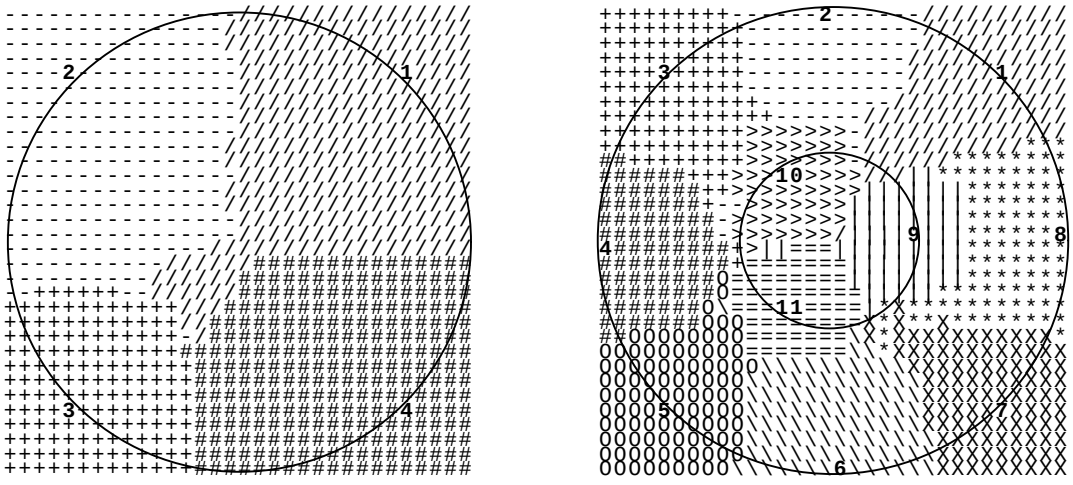
The block distribution is done in the following way.

The master subdivides the image into n^2 blocks and selects a starting block for each slave as we shall describe later. It then sends the starting blocks to the slaves and waits for the results. When a result arrives from a slave, it is sent another non-rendered block which is close to the slave's starting block. This can be done by checking a table of block indicators in a ring-like pattern with the starting block as the centre. In fig. 2 we see the first few rings of the pattern we used. The maximum needed diameter of the pattern has to be $\sqrt{2}$ to include the whole image starting from any block, but in practice a much smaller region will be needed when many slaves are available.

48	44	36	29	35	43	47
40	24	20	13	19	23	39
32	16	8	5	7	15	31
26	10	2	1	3	11	27
30	14	6	4	9	17	33
38	22	18	12	21	25	41
46	42	34	28	37	45	49

The block tiling pattern. The numbers indicate the order in which the blocks are rendered (with the starting block in the center)
Figure 2

To reduce the number of messages, and thus the communication overhead, more than one block can be grouped together in one message. The slave returns the results when it has finished all but the last block, thus reducing its idle time. The grouping can be bigger at the beginning but has to be reduced towards the end-phase of the rendering to make sure that we do not wait for the few slaves that received too much work too late (dynamic control of task granularity).



Tiling of the image with blocks (a) using a circle of diameter n for 4 slaves and (b) using two circles (diameters n/3 and n) to inscribe slave starting blocks for 11 slaves. Each block is shown as a character which indicates the slave that rendered it. The starting blocks are marked by the slave numbers.
Figure 3

With this method the image will be tiled with blocks by the available slaves in a way that resembles a weighted Voronoi diagram of the starting blocks, the weight being the difficulty in rendering the individual blocks. The choice of starting blocks is crucial as the wrong choice could yield much slower results than the last two strategies. We tested three different schemes. In the first, all starting blocks were inscribed in a circle of diameter n/3 centred at the centre of the image. The advantage of this is that the blocks rendered last are the ones around the edges of the image which are usually the easiest. This facilitates an almost simultaneous termination of all slaves. The drawback is that if there is a large number of slaves, their expanding rings are almost concentric and their allocated blocks are mixed randomly.

In the second scheme we increased the diameter of the circle to n/2 to avoid the drawback of the previous scheme. The corners of the image are still left last. Very few 5D nodes are usually needed to ray trace these blocks so any slave can take them on, independent of its starting block.

In the last scheme we used two concentric circles to inscribe the starting blocks if there are more than 8 slaves available, otherwise only the outer circle is used. The inner circle has diameter n/3 and the outer n. One third of the slaves are allocated starting blocks in the inner circle and the rest in the outer one. This scheme changes the wedge-like regions generated by the last scheme to more compact and coherent regions and outperformed the other two schemes in all cases. Examples of the tiling generated by the last scheme can be seen in fig. 3.

3. TIGHTLY COUPLED PAFAS: DISTRIBUTING THE 5D HYPERSPACE

In the loosely coupled methods discussed above as the number of slaves increases the number of hyper-tree nodes per slave decreases, but the percentage of overlap of the trees in different slaves increases. One way to eliminate almost all overlap is to distribute the 5D trees amongst the slaves. Only one slave will be allowed to generate some specific 5D node, and all rays that classify to that node have to be sent to this slave to be traced and their results sent back to the originating slave.

The distribution of the hyper-trees can be done in the following manner. Each processor takes on a number of subtrees starting from level 2 of the hyper-trees. Level n can contain a maximum of $6*(2^5)^n$ nodes, thus there are 6144 possible subtrees to be distributed. In practice only a fraction of these will be generated. A subtree can be uniquely defined by a number from 0 to 6143, which is easily computed during the traversal of the two top levels. For n slaves, a subtree with number s will be assigned to a slave p if s mod n = p. The static load balance achieved by this distribution is satisfactory for a moderate number of slaves, and preferably if this number is odd or, even better, a prime number.

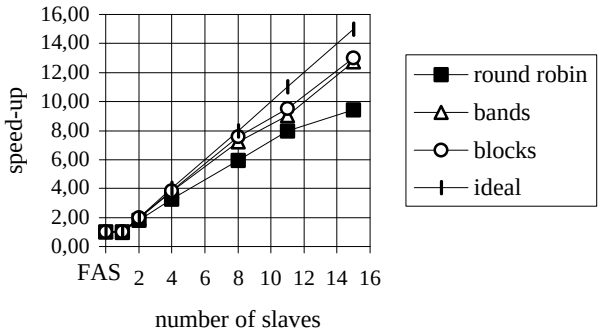
The choice of level 2 for the distribution is not a random one. Level 1 contains a maximum of 192 nodes, which are not enough by any means to guarantee a relatively acceptable load balance. The storage requirements for level 0 (the 6 roots) and level 1 is small, and the object-beam classifications needed to generate the nodes at level 1 are comparatively few, so that levels 0 and 1 can be duplicated on each slave at a small cost. On the other hand, distributing level 3 would increase the needed memory, object-beam classifications, and tree traversal costs without a significant gain in load balance.

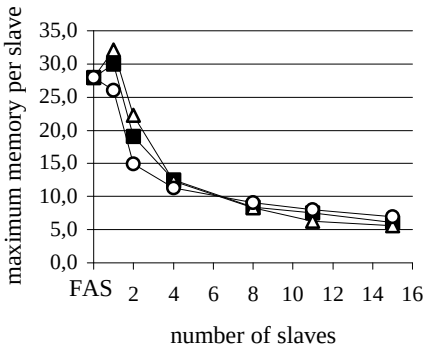
Compared to the loosely coupled strategies the number of messages is increased by at most 2r for r rays. Some rays need not be sent since their candidate sets will happen to reside on their originating slave. However, this chance is reduced for an increasing number of slaves. This number of messages compares favourably with the number of messages in ray-flow algorithms based on spatial subdivision where many messages are usually needed for a single ray. It does, however, assume a full connected network, since any slave can send a ray to any other, whereas in spatial subdivision ray-flow algorithms messages are usually passed only between ‘neighbouring’ processors.

4. RESULTS - DISCUSSION

For our experiments we used a cluster of workstations on a local area network. FAS was added upon the ray tracing kernel of Rayshade, a public domain ray tracer [Kolb92]. The test scenes we used are the well-known scenes proposed by Eric Haines [Haine87] (plus the teapot), with a resolution of 512 by 512 pixels, one sample per pixel. All the tests were repeated 5 to 7 times and the normalised standard deviation of the run times was always small, so we can assume a dedicated use of the cluster.

In parallel processing the performance of an algorithm depends largely on the actual hardware configuration. The timing results of the tightly coupled PAFAS on the cluster proved that the communications were a bottleneck. From the used memory, the tree growth and the intersections done, we could deduce that the static load balancing works well with the number of available slaves. To use this algorithm, however, a multiprocessor with a





		Sequential FAS	PAFAS: number of slaves 1	2	4	8	11	15
speed-up	round robin	1,00	1,00	1,80	3,27	5,95	7,96	9,42
	bands	1,00	0,97	1,96	3,80	7,22	9,03	12,72
	blocks	1,00	1,02	2,01	3,84	7,38	9,24	12,69
max. memory per slave	round robin	28,0 Mb	30,0 Mb	19,1 Mb	12,6 Mb	8,4 Mb	7,5 Mb	6,1 Mb
	bands	28,0 Mb	32,1 Mb	22,2 Mb	12,4 Mb	8,3 Mb	6,3 Mb	5,6 Mb
	blocks	28,0 Mb	26,1 Mb	14,9 Mb	11,3 Mb	9,1 Mb	8,0 Mb	7,0 Mb

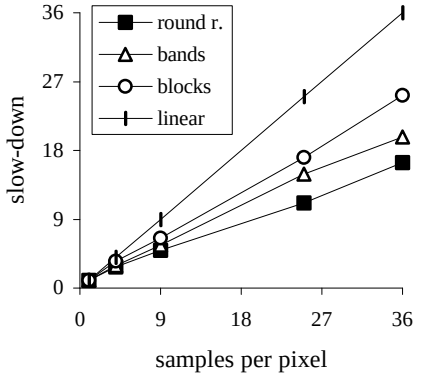
The results for scene rings2
Figure 4

fast interconnection network is needed. In general, the task of intersecting a ray against the objects of a beam (and even more against the objects of a voxel) is quite small, so that ray-flow algorithms have fine grain.

An object-flow FAS would perform well in a shared memory multiprocessor as the principle of ray coherence closely relates to the locality of reference. However, the object-flow algorithm also has too fine a grain for a cluster of distributed workstations.

The only efficient algorithm for such a cluster is the loosely coupled PAFAS. In fig. 4 we see the results for scene rings2, which are representative of most results we obtained. The average (over 7 runs) speed-up compared to the sequential FAS, and the *maximum* memory per slave are shown for the three load balancing strategies. All three strategies exhibit satisfactory reduction in the required memory per processor, thus solving the memory requirement problem of the sequential algorithm. The speed-up is nearly linear, with the blocks strategy approaching closer the ideal speed-up. The ideal speed-up cannot be approached due to the maintenance cost of the 5D data structure which is not distributed in full.

As we can see from the 1 slave column, the 5D subdivision is a highly dynamic algorithm, where even the order in which pixels are calculated can affect its performance. This is due to the decision-making process of the automatic subdivision tuner which bases its decisions on the existing statistics during ray tracing.



samples per pixel		1	4	9	25	36
slow-down	round robin	1,0	2,8	4,9	11,1	16,4
	bands	1,0	2,9	5,6	14,9	19,8
	blocks	1,0	3,5	6,5	17,1	25,2

Supersampling results on 11 slaves
Figure 5

Ray tracing acceleration methods are usually tested on 512 by 512 pixel images with 1 sample per pixel, even though supersampling is used to render demo pictures. This does injustice to 5D adaptive subdivision, which benefits from a larger population of rays and thus greater ray coherence. To demonstrate this property, we rendered the same scene with 11 slaves (a) with 4, 9, 25 and 36 samples per pixel. The results appear in fig. 5. As a general rule the ‘slow-down’ due to the supersampling was less than linear, with the simpler round-robin method benefiting the most. Other acceleration

methods can also benefit from supersampling, but not to the extend that FAS can.

5. CONCLUSIONS

We have presented several attempts to parallelise a directional subdivision algorithm for ray tracing. The tightly coupled PAFAS can only be used on multiprocessors with fast interconnection networks. It can also be adapted to shared memory multiprocessors.

The loosely coupled PAFAS algorithm, on the other hand, is ideal for distributed processing. It can be used to take advantage of the idle time of workstation clusters, which are cheaper and more widely available than multiprocessors.

We have demonstrated the inherent property of FAS to distribute the memory-demanding 5D hyper-tree structure *without any data-flow*. The obtained speed-up comes close to linear. The hyper-tree distribution reduces the memory per processor and makes PAFAS usable with more complicated scenes than FAS. However, the memory requirements still limit it to medium complexity scenes. In any case, with high complexity scenes it is preferable to use spatial subdivision since the cost of the object-beam classifications would exceed the cost of voxel walking [Simia94]. For complex scenes a hybrid hierarchy-FAS method could be used. If the scene becomes so complicated that the memory needed for the object descriptions becomes too big, the objects could be distributed amongst the processors using shared virtual memory at the expense of the page requests and replies communication overhead [Badou90].

ACKNOWLEDGEMENTS

We would like to thank everyone in the graphics group of the University of East Anglia. Many thanks to Shaun McCullagh and Matt Beare for their valuable help. Finally, thanks to Craig Kolb, the author of Rayshade.

BIBLIOGRAPHY

[Arvo87] Arvo,J., Kirk,D.: Fast Ray Tracing by Ray Classification, *Computer Graphics (Proc. SIGGRAPH)*, Vol. 21, No. 4, pp. 55-64, 1987.

[Badou90] Badouel,D., Priol,T.: *Advances in Computer Graphics Hardware (ed. R. Grismdale and A. Kaufman), Springer-Verlag, New York,* An Efficient Parallel Ray Tracing Scheme for Highly Parallel Architectures, pp. 93-106, 1990.

[Bouvi85] Bouville,C., Brusq,R., Dubois,J., Marshal,I.: Generating High Quality Pictures by Ray Tracing, *Computer Graphics Forum*, Vol. 4, No. 2, pp. 87-99, 1985.

[Caspa89] Caspary,E., Scherson,I.: A Self-balanced Parallel Ray-tracing Algorithm, *Parallel Processing for Computer Vision and Display*, pp. 408-419, 1989.

[Caube88] Caubet,R., Duthen,Y., Gaildrat,V.: Voxar: A Tridimensional Architecture for Fast Realistic Image Synthesis, *New Trends in Computer Graphics (Proc. of CGI '88), Springer Verlag, New York*, pp. 135-149, 1988.

[Chalm96] Chalmers,A., Tidmus,J.: Practical Parallel Processing, International Thomson Publishing, 1996.

[Clear86] Cleary,J., Wyvill,G., Birtwistle,G., Vatti,R.: Multiprocessor Ray Tracing, *Computer Graphics Forum*, Vol. 5, No. 1, pp. 3-12, 1986.

[Devil89] Devillers,O.: The Macro Regions: An Efficient Space Subdivision Structure for Ray Tracing, *Proc. Eurographics '89, Elsevier Science Publishers, Amsterdam, North-Holland*, pp. 27-38, 1989.

[Dippe84] Dippe,M., Swensen,J.: An Adaptive Subdivision Algorithm and Parallel Architecture for Realistic Image Synthesis, *Computer Graphics (proc. SIGGRAPH '84)*, Vol. 18, No. 3, pp. 149-158, 1984.

[Fujim86] Fujimoto,A., Tanaka,T., Iwata,K.: ARTS: Accelerated Ray-tracing System, *IEEE CG&A*, Vol. 6, No. 4, pp. 16-26, 1986.

[Glass84] Glassner,A.: Space Subdivision for Fast Ray Tracing, *IEEE CG&A*, Vol. 4, No. 10, pp. 15-22, 1984.

[Green90] Green,S., Paddon,D.: A Highly Flexible Multiprocessor Solution for Ray Tracing, *The Visual Computer*, Vol. 6, No. 2, pp. 62-73, 1990.

[Haine86] Haines,E., Greenberg,D.: The Light Buffer: A Ray Tracer Shadow Testing Accelerator, *IEEE CG&A*, Vol. 6, No. 9, pp. 6-16, 1986.

[Haine87] Haines,E.: A Proposal for Standard Graphics Environments, *IEEE CG&A*, Vol. 7, No. 11, pp. 3-5, 1987.

[Isler91] Isler,V., Aykanat,C., Oaguc,B.: Sub-division of 3d Space Based on the Graph Partitioning of Parallel Ray Tracing, *Second Eurographics Workshop on Rendering, Barcelona, Spain*, pp. 182-191, 1991.

[Kapla87] Kaplan,M.: The Use of Spatial Coherence in Ray Tracing, *Techniques for Computer Graphics, Springer-Verlag*, pp. 173-93, 1987.

[Kay86] Kay,T., Kajiya,J.: Ray Tracing Complex Scenes, *Computer Graphics (SIGGRAPH '86*

Proceedings), Vol. 20, No. 4, pp. 269-278, 1986.

[Kim96] Kim,H.-J., Kyung,C.-M.: A New Parallel Ray-tracing System Based on Object Decomposition, *The Visual Computer*, Vol. 12, No. 5, pp. 244-253, 1996.

[Klima97] Klimaszewski,K., Sederberg,Th.: Faster Ray Tracing Using Adaptive Grids, *IEEE CG&A*, Vol. 17, No. 1, 1997.

[Kolb92] Kolb,C.: Rayshade: A Public Domain Ray Tracer for Unix Systems.

[Narus87] Naruse,T., Yoshida,M., Takahashi,T., Naito,S: Sight – a Dedicated Computer Graphics Machine, *Computer Graphics Forum*, Vol. 6, No. 4, pp. 327-334, 1987.

[Nemot86] Nemoto,K., Omachi,T.: An Adaptive Subdivision Algorithm by Sliding Boundary Surfaces for Fast Ray Tracing, *Proc. of Graphics Interface '86, Canadian Information Processing Society, Toronto, Ontario*, pp. 43-48, 1986.

[Notki97] Notkin,I., Gotsman,C.: Parallel Progressive Ray-tracing, *Computer Graphics Forum*, Vol. 16, No. 1, pp. 43-55, 1997.

[Ohta87] Ohta,M., Maekawa,M.: Ray Coherence Theorem and Constant Time Ray Tracing Algorithm, *Computer Graphics 1987 (Proceedings of CG International '87), Springer-Verlag*, pp. 303-314, 1987.

[Parke99] Parker,S., Parker,M., Livnat,Y., Sloan,P.-P., Hansen,Ch., Shirley,P.: Interactive Ray Tracing for Volume Visualization, *IEEE Transactions on Visualization and Computer Graphics*, Vol. 5, No. 3, pp. 238-250, 1999.

[Priol89] Priol,T., Bouatouch,K.: Static Load Balancing for a Parallel Ray Tracing on a MIMD Hypercube, *The Visual Computer*, Vol. 5, No. 1, pp. 109-119, 1989.

[Rubin80] Rubin,S., Whitted,T.: A 3-dimensional Representation for Fast Rendering of Complex Scenes, *Computer Graphics (Proc. SIGGRAPH '80)*, Vol. 14, No. 3, pp. 110-116, 1980.

[Salmo88] Salmon,J., Goldsmith,J.: A Hypercube Ray-tracer, *Proc. of the Third Conf. on Hypercube Concurrent Computers and Applications, ACM Press*, pp. 1194-1206, 1988.

[Simia94] Simiakakis,G., Day,A.: Five-dimensional Adaptive Subdivision for Ray Tracing, *Computer Graphics Forum*, Vol. 13, No. 2, pp. 133-140, 1994.

[Speer92] Speer,L.: A New Subdivision Method for High-speed, Memory Efficient Ray Shooting, *Third Eurographics Workshop on Rendering, Bristol, UK*, pp. 45-60, 1992.

[Woo92] Woo,A.: Ray Tracing Polygons Using Spatial Subdivision, *Proc. Graphics Interface '92, Canadian Information Processing Society, Toronto, Ontario*, pp. 184-191, 1992.