Methods of Static Load Balancing for Ray Tracing

Joel Goldfinger*
University of Maryland Baltimore County

Abstract

High quality images using ray tracing takes a long time to produce. There have been many algorithms developed to improve the production time of ray traced images. However, even with current acceleration algorithms, ray tracing takes a long time for a large number of geometric primitives. Another way to shorten the production time is to increase the number of computers processing a single image. This paper presents a couple different methods for static load balancing on parallel computers and compares them to previous methods.

Keywords: ray tracing, parallel computing, load balancing

1 Introduction

There are many methods that decrease the rendering time of a ray traced image. These methods work by decreasing the number of ray intersection tests that need to be done. The various types of methods that have been done in the past will be covered in the previous work section. However, these methods only decrease the rendering time so far and may not be fast enough for certain applications. In addition to those methods, an increase in the number of computers used to process a ray traced image can reduce the rendering time. The problem with this is that as computers are added there is a higher imbalance in rendering time between the machines. Many methods have been developed distribute the pixels of an image in a way to minimize this imbalance. There are even other methods to optimize the way in which to break up the geometric world among each computer when the world is much bigger than the memory of each computer. However, this paper takes the assumption that the geometric world to be rendered is small enough to fit in the memory of each computer.

In this paper I present a couple of different methods to distributing the pixels among the computers before rendering starts in order to minimize the imbalance among each computer. For the first method, pixels are distributed among each processor using a Self-Organizing Map. The Self-Organizing Map is a type of neural network that categorizes data into lower dimensions. The Self-Organizing Map takes in the geometric world to be rendered and outputs a two dimensional array that will be used to assign the computers to the various pixels. The other method distributes pixels by projecting the geometric world onto a plane. An important note about this method is that it does not evenly spread the pixels among the computers like the other methods do, including the Self-Organizing Map. These methods will be compared to two other existing static load balancing methods. The other two methods are called naive tiling and scattered decomposition [Salmon 1988]. Figure 1 shows the three different images that were rendered using the four methods tested in this paper.

The related work section covers some work that is similar to the method proposed in this paper. The implementation section discusses how the algorithm in this paper was implemented. The results section will cover how well the method worked as well as compare it to previous methods developed. The conclusion will summarize the the value of these new methods. The future directions section will discuss possible improvements and extensions that could be tested.

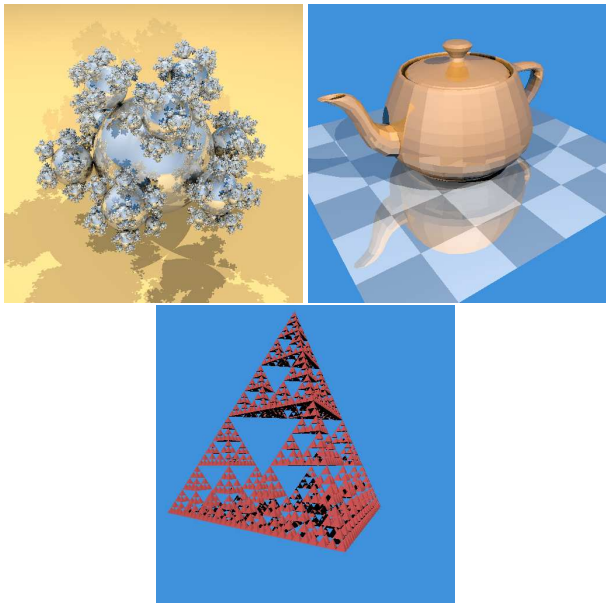

Figure 1: These images were rendered to test the static load balancing methods. A) The upper left picture is an image of many spheres. B) The upper right picture is a teapot. C) The lower picture is an image of many triangles.

2 Related Work

The majority of the time in rendering a ray traced image is ray-object intersection tests. In order to reduce the number of objects that need to be tested, methods have been proposed to create large bounding boxes which encompass multiple objects. There are two common methods to create theses bounding boxes, one is spaced-based [Whang et al. 1995; Van Reeth et al. 1996] and the other is tree-based [Tan et al. 1999]. The spaced-based method breaks the world into voxels. The way the space is broken up is done uniformly or nonuniformly. The tree-based method is where the objects are fitted by a simple bounding volume. Then many of those volumes are bounded by bigger volumes, until all the volumes are bounded by one big volume. The disavantage of space-based is that it wastes a lot of memory where areas of space happen to be empty. Tree-based does not have all the empty volumes, but has to keep braking down the bounding volumes until it hits an object, which takes longer time to compare all those bounding volumes. Another disavantage of both is that they both take time construct their bounding boxes. The paper [Klimaszewski and Sederberg 1997] proposed doing a mix of both space-based and tree-based.

The next step to decreasing the rendering time after minizing intersection tests, is to run the ray tracer over multiple computers. It

*e-mail: joelg1@umbc.edu

should be noted that some existing methods reduce the number of intersections significantly, and thus limit the ability of using multiple computers. There has been much work done on figuring out how to optimize the load among each computer. However, currently load balancing the world to be ray traced over the computers is highly inefficient as the number of computers increase. Two ways exist to balancing a system one being static and the other being dynamic. The static methods decide which pixels are processed by which computer before the actual rendering begins. One such method called naive tiling, which is breaking the image up into tiles where each computer processes a tile. The other method called scattered decomposition [Salmon 1988], which is where the first computer gets a pixel, then the next computer get the next pixel and so on. Dynamic methods change the allocation of pixels to a given computer when imbalances occurs during the rendering of the image. Heirich and Arvo [1998] use a method where each computer asks its neighborhood computers for work when it needs more. Their dynamic method had very little cost because it only communicated with its neighborhood computers, and thus proved to be much better than the existing static methods. They went further to use a hybrid method which did much better than the static and dynamic method by themselves.

When the world's geometric primitives take up more memory than one computer can hold, the the world has to be split up among the computers. In this case not only does load balancing have to be considered, but the placement of geometric primitives have to be distributed in such a way to minimize the transfer of data between computers. This makes minimizing the time a harder issue because not only the load of each computers has to be consider, but the time to pass geometric primitives of the world. The papers [Badouel and Priol 1990; Salmon and Goldsmith 1988] talk about solutions to this type of problem. The paper [Reinhard et al. 1998] goes further to propose calculating the cost up front in order to determine an optimized way to distribute the computation and data among the computers.

3 Implementation

The first method was using a Self-Organizing Map to balance the load among the computers. A Self-Organizing Map is a neural network that is used to lower the dimension of input data into one or two dimensions, while mapping the data into different categories. For this paper a two dimensional Self-Organizing Map was used. The first part of the implementation was to have the Self-Organizing Map to analyze the the geometric primitives of the world. For this paper the only geometric primitives used were spheres and triangles, since they are the most commonly used objects in a ray tracer. The spheres are read into the Self-Organizing Map as a four dimensional vector which are the xyz-coordinates plus the radius. Since the Self-Organizing Map breaks similar data into categories the triangles were encompassed by spheres. The triangles were also encompassed by spheres because a Self-Organizing Map requires that the input vector be the same for all the data.

Self-Organizing Maps use a lattice of nodes whose weights are the same as the input vector size. Figure 3 shows an image of what this lattice looks like. These weights were used to map the pixels onto processors to be ray traced through scattering the pixels among each category. Due to the fact that the Self-Organizing Map takes a longer time the higher number of nodes used, a smaller number than the size of the image to be produced are used. For this implementation a ratio of one node for every sixteen pixels was used. This method is not as accurate as having a node for every pixel, but seems to be accurate enough while not consuming too much time.

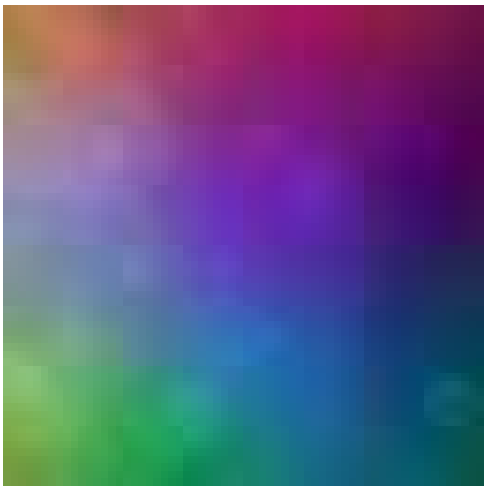

Figure 2: The Self-Organizing Map's visual representation of the spherical geometric world.

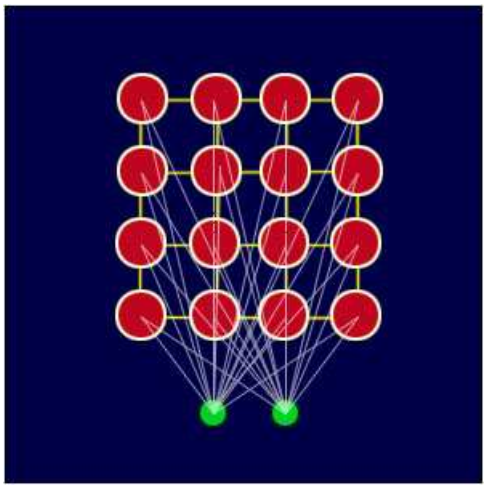

Figure 3: A picture of what a Self-Organizing Map looks like. The red circles are the nodes and the green circles represent the input vector. The lines represent the weights which connect from each green circle to all the nodes.

Figure 2 demonstrates what the categories look like running the method on Figure 1A, using RGB color and a fourth value for the brightness. While the figure shows all the categories the Self-Organizing Map recognizes, this implementation breaks them up into twenty-one categories. These categories were created by breaking the four dimensional weights of the Self-Organizing Map into twenty-one discrete values. The reason for using only twenty-one categories was that with a high number of objects like in figure 1, there are too many small categories which were the size of a few pixels. After the Self-Organizing Map finishes its training and has been broken up into twenty-one categories, the map is used to assign computers to pixels to be ray traced. This mapping was done by using the scattered decomposition method over the twenty-one regions instead of over square tiles. This works because scattering over similar categories means that the pixels assigned to each computer are similar, especially as the number of CPUs are increased.

The other method is projecting the geometric world onto a plane, which would be used to assign pixels to the computers for process-

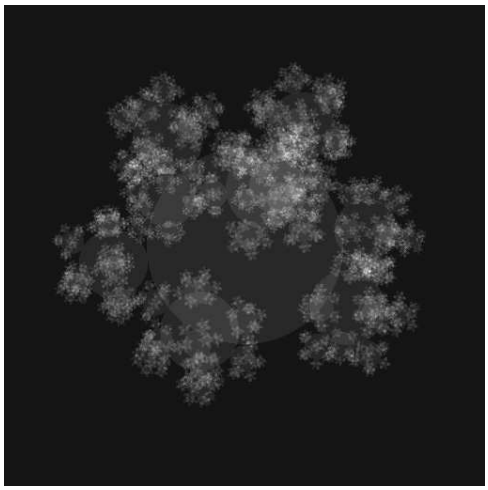

Figure 4: The visualization of the mapping of the spherical geometric world onto a plane.

ing. The idea behind this method is that objects that are larger and closer to the view plane would have more ray intersections. To get the projection data, an intersection test was done for all the objects from the view plane that would be used for the ray traced image. For each intersection of an object, a small positive constant was added to the value at the given pixel location. Adding a small positive constant for each intersection allowed multiple layers of spheres and triangles to be added with in one pixel. Once all the intersection were done there was a two dimensional array which contained the predicted loads of each pixel that needed to be ray traced. Figure 4 shows an example of a two dimensional array created from the image in Figure 1A.

Once the intersection test completed the two dimensional array, the computers were assigned to the pixels in a way that the predicted load was balanced among each computer. This process was done by assigning the computer with the lowest predicted load so far the next pixel. Since pixels were assigned based on their predicted load each computer did not have the same number of pixels assigned. This makes the method different from all the other static methods since they all assigned computers the same number of pixels.

4 Results

The results from table 1 show that both of the new methods proposed in this paper worked much better than the previous ones for the geometric world made up of spheres when the number of CPUs were 256. The Self-Organizing Map even did better than the scattered decomposition method for the lower number of CPUs. For the teapot geometric world the Self-Organizing Map still did better than the scattered decomposition, but only when there were 256 CPUs running. The projection method still performed much better than the naive tiling method, but had slightly worse performance compared to the scattered decomposition method. On the triangle geometric world the Self-Organizing Map and the projection method performed much better than the existing methods for 256 CPUs. However, for the lower number of CPUs both of the methods proposed in this paper did worse performance wise when compared to the scattered decomposition method. Since both of the existing methods had problems as the number of CPUs used increased, both of the new methods showed promise to be able to replace them at the higher level of CPUs.

Model	#CPU	Naive	Scattered	SOM	Projection
Spheres	16	61%	0.7%	0.6%	2.1%
	64	72.8%	4%	3%	5.8%
	256	116%	26%	14.8%	17.2%
Teapot	16	105%	1.4%	3.8%	5.6%
	64	126%	7%	15.5%	11.4%
	256	175%	25.7%	24.3%	29.2%
Triangles	16	43%	1.5%	5.2%	6%
	64	54.4%	2.9%	14.6%	20%
	256	68%	42%	38%	37.3%

Table 1: The numbers represent the percentage error from the optimal load balancing of the ray traced image.

From looking at the results the Self-Organizing Map method worked well on spherical objects, but had performance issues with the triangular objects. The reason for this was most likely do to the fact that the triangles were encompassed by spheres. This led to inaccurate categorizing of the triangles in the Self-Organizing Map, which most likely explains why the Self-Organizing Map did not perform nearly as well as it did with the spherical geometric world. Another issue with the Self-Organizing Map was coming up with the number of categories to use from the map. With a dynamic number of categories it is possible that the Self-Organizing Map may have better performance results.

The projection method only did well for the higher number of CPUs. The reason for this was because it did not scatter the pixels evenly among all the processors, but instead assigned the pixels based on the predicted load. That meant that as the number of computers increased it was better able to spread the load among the computers because it would give computers assigned pixels with higher predicted cost to ray trace less of the total pixels. However, since the pixels were not spread evenly, with a lower number of CPUs the methods predicted load would have a somewhat higher error rate. Another problem with the projection method was that the up front cost for the projection of the geometric world was high. However, there are other methods that can be used to project the geometric world much quicker.

5 Conclusion

The results show that both of the methods proposed in the paper have promise to replace existing static load balancing methods. Furthermore, while the methods did not perform that well with a lower number of CPUs, they did for the higher number which is currently the problem with static load balancing. While both the methods had problems with triangular objects, possible fixes to those problems are addressed in the future directions section. Furthermore, though the projection method had a high cost up front there are other projection methods that could be implemented that would drasticly speed up the projection process. With improvements to either of the two new methods, one of them may be able to be used in conjunction with the dynamic method in Heirich and Arvo [1998] to work better than their existing hybrid method.

6 Future Directions

The methods proposed in this paper had significant performance increasing over the existing static load balancing methods, but with future improves could do much better. One such fix is to improve the way in which the Self-Organizing Map method recognizes the triangles. For the Self-Organizing Map instead of using a sphere to encompass the triangle, a second Self-Organizing Map could be used for the triangles and then merged with the Self-Organizing Map that processed spheres. Another way to include the triangles into the Self-Organizing Map would be to bound both the spheres and triangles by a box oriented with the axis which would only increase the vector size by two. This would make the extra space encompassed by the triangle less than bounding with the spheres, with limited extra space surrounding the spheres in the geometric world. However, using this method would requiring some action being taken to make sure that the Self-Organizing Map was not too slow, since as the number of vectors increase the time increase quite a bit. Another improvement could be to make the number of categories read from the Self-Organizing Map dynamic, which might improve the pixel assignment.

The projection method needs to take into account the angle at which the triangle is relative to the view plane. The problem was that the angle affects the surface area of the triangle, which caused issues for the projection method as shown in the result. A possible solution might be to add some predicted load to the surrounding pixels where the triangle was projected based on the angle at which it was at. Another problem with the projection method was that it checked the intersection of all the objects from the view plane, which makes it a very slow process. However, there are other ways in which to project the geometric worlds onto the view plane much quicker that could be implemented instead of the one used in this paper. Another possible improvement could be to use more than one projection, so that there are multiple slices through the geometric world. This would give a much better prediction of the load of each pixel, since it would include more information about the reflection and refration done by the ray tracer.

An extension for the two static methods in this paper could be implementing them with the dynamic method in Heirich and Arvo [1998]. Since in the results it showed that they worked better than the scattering method for a higher number of CPUs, then the hybrid method the authors created with scattered decomposition should work better with the new methods in this paper.

7 Acknowledgements

I would like to thank Mat Buckland for the use of his Self-Organizing Map code. I would also like to thank Mat Buckland for the use of figure 3, the image of how a Self-Organizing Map is constructed. I would also like to thank Marc Olano for his modification of the rayshade ray tracer program. Lastly, I would like to thank the creators of the Standard Procedural Databases for the use of their geometric worlds to test the load balancing methods in this paper.

References

BADOUEL, D., AND PRIOL, T. 1990. An efficient parallel ray tracing scheme for highly parallel architectures. In *Advances in Computer Graphics Hardware V, Tutorial on Perspectives of Computer Graphics*, R. Grimsdale and A. Kaufman, Eds. Springer-Verlag, New York, 93–106.

HEIRICH, A., AND ARVO, J. 1998. A competitive analysis of load balancing strategies for parallel ray tracing. *The Journal of Supercomputing 12*, 1–2, 57–68.

KLIMASZEWSKI, K., AND SEDERBERG, T. 1997. Faster ray tracing using adaptive grids. *Computer Graphics and Applications, IEEE 17*, 1, 42–51.

REINHARD, E., KOK, A. J. F., AND CHALMERS, A. 1998. Cost distribution prediction for parallel ray tracing. In *Proceedings of the Second Eurographics Workshop on Parallel Graphics and Visualisation*, Eurographics, Rennes, France, 77–90.

SALMON, J., AND GOLDSMITH, J. 1988. A hypercube ray-tracer. In *Proceedings of the third conference on Hypercube concurrent computers and applications*, ACM Press, New York, NY, USA, 1194–1206.

SALMON, J. 1988. A mathematical analysis of the scattered decomposition. In *Proceedings of the third conference on Hypercube concurrent computers and applications*, ACM Press, New York, NY, USA, 239–240.

TAN, T.-S., CHONG, K.-F., AND LOW, K.-L. 1999. Computing bounding volume hierarchies using model simplification. In *I3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics*, ACM Press, New York, NY, USA, 63–69.

VAN REETH, F., MONSIEURS, P., BEKAERT, P., AND FLERACKERS, E. 1996. Ray tracing optimization utilizing projective methods. In *Computer Graphics International, 1996. Proceedings*, 47–53.

WHANG, K.-Y., SONG, J.-W., CHANG, J.-W., KIM, J.-Y., CHO, W.-S., PARK, C.-M., AND SONG, I.-Y. 1995. Octree-r: an adaptive octree for efficient ray tracing. *Visualization and Computer Graphics, IEEE Transactions on 1*, 4, 343–349.