Developing a Cluster Based Parallel Ray Tracer

Chris Lovell

May 16, 2007

Abstract

Parallel ray tracing is an important concept in computer graphics, as it allows for high quality ray traced images to be produced at a faster rate than is possible in single processor ray tracing. Presented in this paper is the conversion of a previously created single processor ray tracer to a parallel ray tracer that is capable of running on a cluster of computers. This paper presents the technical aspects of designing a parallel ray tracer by looking at the theory of transforming ray tracing into a parallel process and by comparing the Condor and Linux based cluster computing technologies. This paper then highlights how the methods available for turning ray tracing into a parallel process can be dependent on the cluster computing technology going to be used.

Keywords: *Parallel Ray Tracing, Cluster computing*

1 Introduction

Parallel ray tracing is important in computer graphics because it is the only way to significantly speed up the rendering time of ray tracing. Parallel ray tracing generally takes place using cluster computing, where the power and number of processors being used increase the speed of the ray tracing. The key area of research in parallel ray tracing is to develop parallel ray tracing algorithms that run more efficiently as a parallel process, in order to speed up the offline rendering time of ray tracing, but also to try and make real time ray tracing possible.

This paper looks at the ray tracing problem in Section 2 along with its conversion to a parallel process and the reasons why ray tracing is still an important research issue. The possible techniques for parallel ray tracing are investigated further in Section 3. The paper ends with a discussion about the practical conversion of a previously created single processor ray tracer [12], to a parallel ray tracer in Section 4. This final discussion also compares two different methods for distributed computing, the Condor system and Linux based cluster computing.

Overall, this paper describes several methods for parallel ray tracing as well as investigating the practical considerations. This paper looks at why ray tracing needs to become a parallel process in order for it to be more usable, how cluster computing aids parallel ray tracing and the problems in using cluster computing.

2 Ray Tracing

Ray tracing is a computationally expensive rendering algorithm that produces high quality, physically accurate images. First described in 1968 [1] through the idea of ray casting, where an image is produced by calculating the rays of light passing through the image into the scene to see if they hit an object or not, ray tracing was developed further to allow full lighting calculations to be determined [29]. Ray tracing is an advancement over other commonly used rendering techniques such as rasterisation, as ray tracing accurately produces physical effects such as shadowing and reflections. The disadvantage of ray tracing is the amount of time that it takes to render an image.

Ray tracing works by tracing "rays of light from the eye back through the image plane into the scene" [15]. This means that ray traced images are built in a similar way to how the human eye will build images, where the retina is where the final image is built and the lens is equivalent to the image plane. To achieve this in ray tracing, a camera position is defined for each image and the viewing plane is positioned a short distance away from this camera along the viewing direction, which is where each ray passes. The rays that go directly from the camera position into the scene are called the primary rays. When a primary ray hits an object, secondary rays will be generated to allow lighting and reflections to be calculated.

In its implementation, a ray tracer is generally given a scene as input, that contains many objects and lights. The objects will have a physical shape and position, described in some manner for example by a mathematical equation, or by a collection of triangles. The objects will also have some material and possibly a texture, which will determine the basic colour of the object. Lights also have a position within the scene, along with properties such as colour and brightness.

To perform the ray tracing, a single primary ray will be generated for each pixel in the output image. The position of the camera determines where each ray will start from and the camera's look at vector is used to calculate the rays direction vector. This primary ray is then cast into the scene to try and find any objects that may intersect the ray, where the closest of any intersecting object will be selected as the object seen from that pixel. The next stage is to generate secondary rays to calculate lighting, and reflections if the object is reflective. Any lighting rays generated start at the intersection point between the object and the primary ray and have a direction vector pointing to a light source. To ensure accurate lighting, there will be a secondary ray generated for every light source in the scene. The Phong illumination model [16] is used to determine how the objects are lit in terms of ambient, diffuse and specular lighting components of each light source. To generate shadows in the scene, the lighting rays are cast into the scene to see if there is a clear path from the object to the light source. If a lighting ray has a clear path from the object to the light source then there is no shadow cast upon the object. If however the path is not clear, then the obstructing object must be casting a shadow over the point on the object being tested and so the lighting calculation will only use the ambient lighting component from the light being tested. The reflection rays are calculated by reflecting the incoming ray against the intersection position to determine a new direction vector. Reflective rays will return their calculated colour value back to the parent ray that generated them. The values returned by reflective rays are combined with the parent rays colour to determine the final colour of the parent ray.

2.1 Parallel Ray Tracing

Parallel ray tracing is the process of rendering an image using multiple ray tracers at the same time, with each processor rendering one part of the image. Parallel ray tracing most commonly takes place across a distributed computer network, however some research into real time ray tracing relies less on distributed computer networks. Parallel ray tracing should not be confused with the term *distributed ray tracing*, which is a method of ray tracing where by each pixel is the average of several rays running close to each other and is used to create effects such as soft shadowing, blurring and depth of field [6]. To further confuse the naming conventions, the term *distributed interactive ray tracing* is used to describe ray tracing across a distributed computing system, in real time [27, 7].

Parallel ray tracing has been a key research area in computer graphics in order to try and speed up the process of producing ray traced images. Initially this was to allow offline rendering to take place quicker, but now real time ray tracing has become an area of increased research interest. The focus of research in parallel ray tracing is now moving towards real time ray tracing, however due to the speed of current hardware not being fast enough, there is still an interest in optimising offline ray tracing.

2.2 Offline Rendering

Offline rendering is the process of generating images that are not required to be shown immediately and is mostly used in video production, where many millions of images are produced to form a final film. An example of such a system is Pixar's RenderMan [17], where the scene files can be distributed to many machines, called a render farm, and the scene is then built up by using the multiple machines on the render farm [18]. The work carried out and described in Section 4 concentrates on offline ray tracing, sped up by using multiple distributed processors.

2.3 Real Time Rendering

Real time rendering requires images to be produced and then displayed at a rate such that there is no noticeable lag in the frame rate. Frame rates around 25 frames per second (FPS) can be used to achieve a real time video frame rate, however higher frame rates in the region of 50-60 FPS are normally preferred in computer animation to allow for more fluid movement.

The problem in ray tracing is to achieve such a high frame rate. On a standalone machine running a ray tracer it is very difficult to achieve frame rates above 10 FPS [28, 2], even for a very basic scene. The problem is of course scalability, and to achieve even standard definition television quality images of 640x480 pixels, requires 307200 separate primary rays to be calculated and then any additional secondary rays, where lighting rays alone will at least double this value.

2.3.1 Alternatives to Cluster Computing

Real time ray tracing has become of interest in the last few years as the power of processors has increased and multiple processor computing devices are becoming more readily available. One key development in this area is the Cell Broadband Engine Architecture [11] and research is being carried out into how the Cell processor can be used to speed up ray tracing [2]. The Cell Broadband Engine architecture is a combination of several processors, where there is a central processor that controls eight other sub processors and determines the jobs that each sub processor will carry out [10]. A processor using an architecture similar to that of the Cell, will be in a better position to carry out real time ray tracing in a commercial market than more traditional cluster computing methods, because the Cell architecture allows for parallel processing to take place in a single, cheaper machine. However in the work by Benthin et al. [2] we see that the current Cell processor is not in viable a position to be used in real time ray tracing and that at least double the computing power is still required before it will begin to be.

Another alternative to using cluster computing in real time ray tracing, is through the development of dedicated ray tracing hardware [23, 24, 30]. The dedicated ray tracing hardware approach is to produce hardware that is similar to the dedicated graphics cards that the majority of computers use for speeding up vector graphics, but instead is optimised for the calculations required in ray tracing. The argument made by Woop et al. [30] for using dedicated ray tracing hardware is that having a cluster of processors is expensive and that ray tracing calculations do not require the full complexity of calculations possible on a CPU. Therefore, Woop et al. argue that developing and using smaller, cheaper processors that meet the requirements of ray tracing is a more promising approach than using computer clusters, in trying to develop real time ray tracing systems for widespread use.

A key use for real time ray tracing would be in the development of interactive simulations and interactive entertainment systems. Recently there has been an interest and success in producing interactive entertainment systems that use ray tracing as the rendering technique [22, 8]. The work carried out by Schmittler et al. [22] found that a cluster of computers with a combined processor speed of 30 GHz was required to achieve a resolution of 640x480 pixels running at a speed of 5-20 FPS.

Although there are areas of research into finding alternative methods to cluster computing for use in parallel ray tracing, the current hardware available is not able to provide a real alternative. Research into turning ray tracing into a parallel process is largely unaffected by the intended target hardware architecture, due to computer clusters and microprocessors like the Cell with multiple cores, having a very similar basic architecture.

3 Techniques

In parallel ray tracing there are two principal methods of implementation as described by Reinhard [19]. Firstly the demand driven approach, where the rendering task is split up so that each processor is given the full scene and a set of rays to render. Secondly a data parallel approach, where the rendering task is split so that each processor handles a subset of the scene and only traces the rays that pass through that processors part of the scene. Both of these methods have benefits and drawbacks, which leads to research trying to find a hybrid of these methods.

3.1 Demand Driven Ray Tracing

The most obvious way of applying a ray tracer to a distributed computer network, is to split the required image up into smaller sections and to run each section as a separate job on different processors [14, 9, 19]. The scene is duplicated on each processor, where each processor is assigned a set of pixels to be calculated and the results returned. This method of parallel ray tracing is called the demand driven approach.

One way of performing demand driven ray tracing is to create a number of jobs that each contain a different subset of pixels to be rendered, each processor then takes a job and calculates the required pixel values independently. Processors then return the results of the pixels that have been calculated to the required location and then take another job if there are any remaining jobs, thus ensuring that all processors are kept busy. While this approach does manage to solve the problem of turning ray tracing into a parallel process, it is essentially just a brute force approach of adding more computers to standard ray tracing.

The benefits of using the demand driven approach are the ease of: implementation of the ray tracing algorithm, the distribution of jobs, and controlling the parallel computation. Using the demand driven approach does not require the general ray tracing algorithm to change, as each processor is just running a set of rays through a scene file, acting as it would if it were a single processor system. The distribution of jobs is also simple, due to splitting the problem up into smaller, equal sized problems, usually based on the number of rays each processor will trace. Finally the control of the parallel computation is kept simple due to all rays being traced independently of each other and so there being no dependencies between jobs.

The main draw back to using the demand driven approach is the problem that all processors must have sufficient memory to be able to store all of the scene file. In cases where the scene is very large this may not be possible and so the use of the demand driven approach may be limited.

3.2 Data Parallel Ray Tracing

Data parallel ray tracing is an alternative approach, which splits the scene up so that each processor in the cluster owns an area of the scene and will process the rays that go through that part of the scene [20, 19, 4, 25, 7].

Before the rendering takes place, the scene is split up into several sections, with each section having a set of neighbouring objects. The sections are then distributed to the processors in the cluster to be used, so that each processor owns one section of the scene. When the ray tracing takes place, the processors first determine whether a ray passes through its section of the scene. If the ray passes through the section of the scene, the ray is traced to see if it collides with any of the objects in the scene and if a collision occurs, secondary rays are calculated as appropriate and the value of the ray is returned to the master controller. If the ray does not pass through the section of the scene owned by the processor, the ray is not traced.

The way that a scene is divided up will determine how efficiently a data parallel ray tracer will run. A scene that is split up with the amount of objects stored in each section being highly imbalanced, will most likely lead to some processors having to trace far more rays than other processors. The hardest problem to overcome is trying to determine the number of rays, especially secondary rays, that will pass through a section of the scene in order to determine how many calculations must be performed for each section and so to estimate the sections that will require the most processing. One approach for splitting a scene into sections is to divide the scene using a cost function. Reinhard [19] describes a cost criteria presented by Salmon and Goldsmith [21], which states that each section should make up roughly the same computational load, each section should have similar memory requirements, and that the communication cost of having to pass rays through multiple processors should be minimised. Fulfilling this requirement is a complex problem, which still does not fully address the problem of load imbalance and is the drawback of using the data parallel method of ray tracing.

3.3 Hybrid Methods

Hybrid methods for achieving parallel ray tracing have been developed to try and avoid the problems of the demand driven and data parallel techniques. The demand driven technique has the problem that the processors act independently of each other and so this leads to a more brute force approach to ray tracing, whilst data parallel techniques have problems with load imbalance of jobs, where one section of the scene may have far more rays passing through it than other sections of the scene. The ideal outcome for the hybrid approach is to have a data parallel system, so as to use the idea of object coherence, but to ensure all processors are kept equally busy at all times, as achieved in the demand driven approach.

To develop hybrid methods, firstly improvements are needed to be made to the demand driven approach of ray tracing, in order to be able to join it to data parallel techniques. Methods have been developed for making demand driven ray tracing more efficient and a common approach is to group and trace coherent rays together to reduce the number of calculations required [20]. In certain circumstances, the rays used in a ray tracer are inherently coherent, meaning that groups of rays will most likely pass through the same objects and have the same intersections. The first set of rays that are coherent are the primary rays that are projected from the camera and are easy to group. Further coherent rays can be the set of rays coming out of a light source, but these rays are more complex to calculate as ray tracing traces rays into light sources and not out of light sources.

In Reinhard and Jansen's work [20], the primary rays are bundled into groups and then enclosed into a bounding cone. The cone for each bundle of rays is then used to find intersecting objects in the scene. The rays in the bundle are then individually traced and compared to only the objects that the cone intersected. The use of an intersecting cone in a complex scene will significantly improve the tracing time as the number of intersection tests required for each ray will be reduced to only the objects the ray is likely to pass through. This method of ray tracing lends itself to demand driven ray tracing, as the scene is initially preprocessed to produce the bundles of rays and their intersecting objects using the cone intersection. These bundles of rays can then be passed to the individual processors for them to render in a demand driven manner.

The coherent rays approach is able to be added to the data parallel idea of splitting the scene across several processors, as the coherent rays approach described above only works for tracing bundled rays. Once the bundled rays have been traced and found to intersect an object, secondary rays will be produced that will also need to be traced. However, the processor handling the tracing of the bundled rays will mostly likely be unable to fully trace the secondary rays, due to the processor only having a subset of the scene file. Reinhard and Jansen [20] use the coherent ray method to first trace the primary rays and then pass the intersection points of each of the rays to the processor that owns the object the rays intersect. The secondary rays are then calculated by the intersecting objects owner and are traced using the data parallel technique. By using this approach, Reinhard and Jansen showed that the hybrid approach performs better than the data parallel approach. In terms of achieving the goal of keeping all processors equally busy, Reinhard and Jansen state that this hybrid approach works well while there are still coherent rays that can be processed using the updated demand driven approach. However, once the demand driven jobs are complete, the system reverts back to an entirely data parallel system and so has the same load imbalance problems found in a purely data parallel system.

4 Implementation

To experiment with the techniques researched, a ten second video sequence to be used to publicise the Open Middleware Infrastructure Institute (OMII) [13] was produced using a distributed computing version of the ray tracer developed previously [12].

The initial stage was to convert the previously developed standalone ray tracer into a form that can be used across a distributed system. To do this the ray tracer was developed using the demand driven ray tracer technique talked about in Section 3.1. The demand driven ray tracer technique was chosen as the scene files being produced were known to be small enough so that they could be run on each processor. This meant that the ray tracer now had to accept as input, the set of pixels it should calculate, along with the scene information and had to output the RGB values of the pixels generated. A separate application would then collect all of the RGB values from all of the processors rendering the same image and convert them into a PNG image file.

4.1 Distributed Technologies

Two ways of handling the distributed computing were looked at, firstly the Windows version of the Condor [5] distributed computing management system and secondly looking at having multiple Linux machines running in parallel. The modified ray tracer works across both platforms, the implementations are discussed below.

4.1.1 Condor

Condor is a High-Throughput Computing (HTC) environment and is designed to run across a pool of networked computers that allow themselves to run submitted jobs when they have been idle for an amount of time [26]. To use a Condor system, jobs are submitted that contain the executable required to run, the input data and the output data requirements, along with the number of tasks to split the job into. The Condor server then holds this data and distributes it to the computers in the pool that are currently open to running Condor jobs.

In implementing the parallel ray tracer, the input data was the entire scene file followed by the set of pixels to be rendered by each job. The output from each job was a list containing all of the pixel locations and the RGB values calculated for each pixel.

The Condor system has some drawbacks, the main one being that the computers in the pool are generally workstations that could be used at any time, which can cause a job to be suspended or in some cases terminated with the results lost. This means that the jobs running on a Condor system have to be designed to be fail safe and one job must not rely on other jobs to complete its execution. All jobs should be small enough so that any job that is aborted half way through is not overly costly in terms of any data lost and the processing time already carried out. In ray tracing this is not a problem if the demand driven approach is used, as the jobs are entirely independent of one another as well as being processor independent. Using a data parallel approach would not be viable in a Condor system as there would be no guarantees that the processor for an area of the scene would be currently running. Although the Condor system allows for idle computers to be utilised, that very technique limits the ways that a parallel ray tracer can operate on the cluster.

Another draw back is that the jobs running on a Condor system will only use a portion of the available processor. When tested using the ray tracer, Condor was found to use around one third of the available processor of the machine running a job on. This means that the jobs being executed will not be running as fast as they could be, so again meaning that only offline parallel ray tracing, where a quick response rate is not required would be suited to a Condor cluster.

A further significant draw back of the Condor system is that a required time frame for execution can not be set when submitting a job and so jobs will complete when they are able to be completed. Again this means that real time rendering would not be possible, but also has a disadvantage for offline rendering, as only estimates of rendering times could be calculated and not precise times.

Overall, the Condor system would work well only for offline rendering as the reliability of the cluster is not sufficient for real time rendering. The speed at which the Condor client runs the ray tracer is also an issue and so even for offline rendering the number cluster of computers would have to be large in order to achieve a significant performance increase over a single machine running the entire process.

4.1.2 Linux Based Cluster

Linux based clusters are the most common form of clusters, with a well known Linux based cluster design being the Beowulf cluster [3]. The advantage of using a Linux based cluster is that a better level of control over how the jobs run can be achieved and a High-Performance Computing (HPC) environment can be produced. The process of using a Linux based cluster is firstly to have a compiled application that is able to accept arguments that will dictate the input the application receives. The application, along with the required inputs can then be transferred to a central file store. The cluster of computers is then told to execute the application with a specified set of inputs. In implementing the parallel ray tracer, the jobs were again split up into sets of pixels that each processor would calculate.

The main advantage of using a Linux based cluster over the Condor system, is that the processors involved will be dedicated to the job being run. In such cluster computing, a time frame along with a number of processors can be booked and processing will take place and during this time on the submitted job only.

A Linux cluster would be able to perform both offline and real time ray tracing due to the hardware being more dedicated and so meaning that the jobs will execute quicker, more reliably and can allow for hybrid parallel ray tracing techniques to be used.

4.2 Findings

The images produced to form the frames of the video sequence were created by using 16 Linux based computers that dedicated the majority of their processing capability to the ray tracer application. Each image was split up so that each of the 16 computers had the same number of pixels per image to render. The video sequence was then made up of 500 frames running at 50 FPS to produce 10 seconds of video footage.

The image shown in Figure 1 is a frame from the video produced, and took 115 computing seconds to render. By splitting the job across 16 Linux machines running in parallel, the total rendering time was only 7.2 seconds. In terms of the entire video, the rendering time was reduced from 16 hours to 1 hour rendering time by using 16 different machines. This has shown that a Linux based cluster using the demand driven approach for parallel ray tracing has been able to reduce the rendering time proportionally to the number of extra processors used in the single ray tracer implementation.

4.3 Increased File Size Requirement

By making the ray tracer work on a distributed computing system, the rendering time was significantly reduced, however the amount of file space required was significantly increased. In the standalone version of the ray tracer, the only file space required was for the scene file and for the output images. In the conversion to the distributed computing version, there were now more and larger files required to do the same job. Firstly each processor requires its own copy of the scene file,

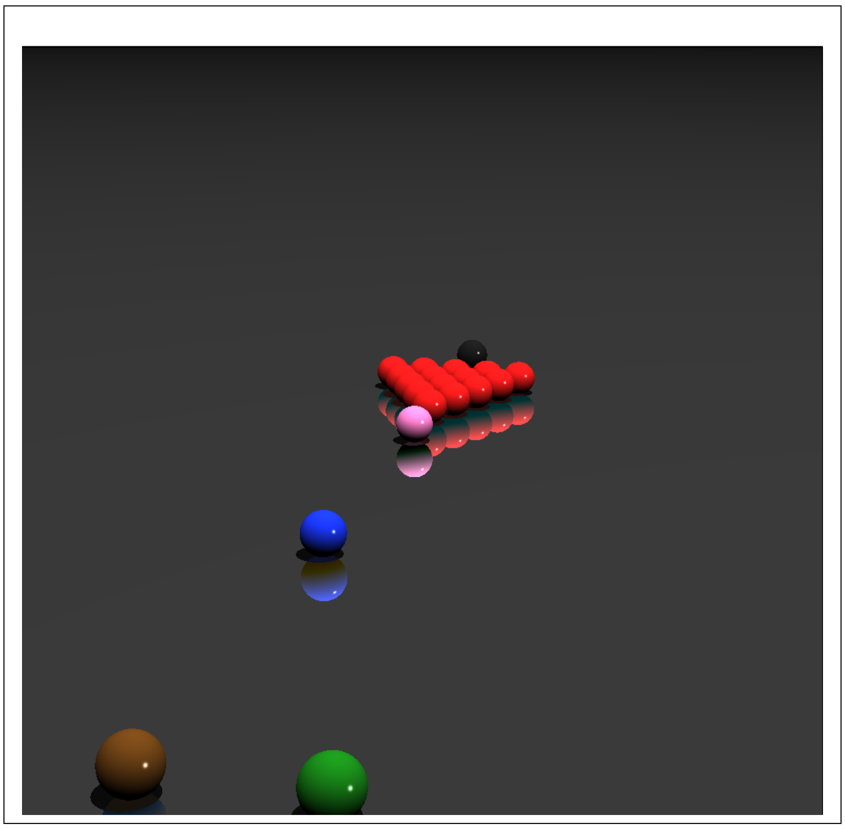

FIGURE 1: A scaled 1000x1000 pixel frame from the produced video of coloured balls sitting on a reflective surface. The rendering time for this image was 7.2 seconds across 16 parallel processors.

however this requirement is still only a very minimal amount of disk space. The main disk space requirement now comes from having to specify the pixels to render and the RGB output of each pixel in a file. Previously the single processor ray tracer would render all of the pixels in an image, so there was no need to specify each pixel to render, instead a loop could be used in the code, whilst the output was built up and handled entirely in memory. In the distributed ray tracer, these additional file requirements meant that for each second of video produced, on average just under 2 GB of data was required for input and output in total.

5 Discussion

This paper has looked at parallel ray tracing, the algorithms that are used and the possible hardware solutions. Whilst the majority of research into parallel ray tracing uses cluster computing, there is research into finding ways for parallel ray tracing to take place on a single machine. Finding a way to make parallel ray tracing work on a single machine would lead to ability for real time ray tracing to be developed, which could then take over from rasterisation rendering as the primary rendering method used in computer graphics.

The algorithms for parallel ray tracing can be defined as being demand driven, data parallel or a hybrid of demand driven and data parallel. As both demand driven and data parallel approaches have significant drawbacks, hybrid approaches are being developed to try and merge the concepts and to minimise the disadvantages of the demand driven and data parallel approaches.

This paper has also looked at the ways in which parallel ray tracing can be implemented using a cluster of computers. Having looked at two cluster computing technologies, this paper has found that whilst a HTC cluster will utilise computers not being used in a dedicated pool, so removing the requirement for building a dedicated cluster, HTC clusters will only work with parallel ray tracing algorithms using the demand driven approach. Where as HPC clusters provide a much greater level of freedom in algorithm design and in controlling the processors, but has the disadvantage that a dedicated cluster must be built.

Overall this paper has looked at a range of parallel ray tracing techniques, issues and research areas. With the development of faster, smaller and cheaper processors, the advancement of parallel ray tracing will most likely now focus on the development of real time ray tracing systems, that do not require large clusters of computers, but instead can run in parallel across a multi-processor computer.

6 Conclusion

The work carried out for this paper has successfully converted a single processor capable ray tracer into a parallel ray tracer. In doing so, a demand driven approach for handling the parallel processing was taken so that the ray tracer worked on both the Condor and Linux based cluster technologies. This paper has also identified that the ways in which the rendering jobs are split up and processed in parallel is where the performance of parallel ray tracing can be further improved, and that developing hybrids of data parallel and demand driven ray tracing is the key area of research for finding this performance improvement.

Remarks

Videos and images discussed in this paper can be viewed at www.ecs.soton.ac.uk/∼cjl203/comp6009. Source code and executable files for running the ray tracer on the Windows Condor system and Linux based cluster can be obtained on request by emailing cjl203@ecs.soton.ac.uk.

Acknowledgements

Thanks to Professor Peter Henderson for supervising this project.

References

[1] A. Appel. Some Techniques for Shading Machine Renderings of Solids. In *SJCC*, volume 32, pages 27–45, 1968.

[2] C. Benthin, I. Wald, M. Scherbaum, and H. Friedrich. Ray Tracing on the CELL Processor. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, pages 15–23, 2006.

[3] Beowulf.org: The Beowulf Cluster Site. `http://www.beowulf.org/`.

[4] A. Chalmers and E. Reinhard. Parallel and Distributed Photo-Realistic Rendering. In *ACM SIGGRAPH '98 Course Notes - Course*, 1998.

[5] Condor Project Homepage. `http://www.cs.wisc.edu/condor/`.

[6] R. L. Cook, T. Porter, and L. Carpenter. Distributed ray tracing. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 137–145, New York, NY, USA, 1984. ACM Press.

[7] D. Demarle, S. Parker, M. Hartner, C. Gribble, and C. Hansen. Distributed Interactive Ray Tracing for Large Volume Visualization. In *Proceedings of the Symposium on Parallel and Large-Data Visualization and Graphics*, pages 87–94. ACM Press, 2003.

[8] H. Friedrich, J. Günther, A. Dietrich, M. Scherbaum, H.-P. Seidel, and P. Slusallek. Exploring the use of Ray Tracing for Future Games. In *sandbox '06: Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*, pages 41–50, New York, NY, USA, 2006. ACM Press.

[9] S. A. Green and D. J. Paddon. Exploiting Coherence for Multiprocessor Ray Tracing. *IEEE Comput. Graph. Appl.*, 9(6):12–26, 1989.

[10] M. Gschwind. Chip Multiprocessing and the Cell Broadband Engine. Technical report, IBM Corporation, February 2006.

[11] H. P. Hofstee. Introduction to the Cell Broadband Engine. Technical report, IBM Corporation, 2005.

[12] C. Lovell. Construction of a Ray Tracing Renderer that Implements Displacement Mapping. Technical Report ELEC6025-2006/07, School of Electronics and Computer Science, University of Southampton, 2007.

[13] OMII: Open Middleware Infrastructure Institute UK. `http://www.omii.ac.uk/`.

[14] D. E. Orcutt. Implementation of Ray tracing on the hypercube. In *Proceedings of the third conference on Hypercube concurrent computers and applications*, pages 1207–1210, New York, NY, USA, 1988. ACM Press.

[15] G. S. Owen. Ray tracing. ACM SIGGRAPH, `http://www.siggraph.org/education/materials/HyperGraph/raytrace/rtrace0.htm`, July 1999.

[16] B. T. Phong. Illumination for Computer Generated Pictures. *Commun. ACM*, 18(6):311–317, 1975.

[17] Pixar's RenderMan, Pixar. `https://renderman.pixar.com/`, last accessed April 2007.

[18] S. Raghavachary. A brief introduction to renderman. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*, page 2, New York, NY, USA, 2006. ACM Press.

[19] E. Reinhard. *Scheduling and Data Management for Parallel Ray Tracing.* PhD thesis, Department of Computer Science, University of Bristol, October 1999.

[20] E. Reinhard and F. W. Jansen. Rendering Large Scenes Using Parallel Ray Tracing. *Parallel Comput.*, 23(7):873–885, 1997.

[21] J. Salmon and J. Goldsmith. A Hypercube Ray-tracer. In *Proceedings of the third conference on Hypercube concurrent computers and applications*, pages 1194–1206, New York, NY, USA, 1988. ACM Press.

[22] J. Schmittler, D. Pohl, T. Dahmen, C. Vogelgesang, and P. Slusallek. Realtime ray Tracing for Current and Future Games. In *Proceedings of 34. Jahrestagung der Gesellschaft für Informatik*, 2004.

[23] J. Schmittler, I. Wald, and P. Slusallek. SaarCOR: A Hardware Architecture for Ray Tracing. In *HWWS '02: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 27–36, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[24] J. Schmittler, S. Woop, D. Wagner, W. J. Paul, and P. Slusallek. Realtime Ray Tracing of Dynamic Scenes on an FPGA Chip. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 95–106, New York, NY, USA, 2004. ACM Press.

[25] G. Simiakakis, T. Theoharis, and A. M. Day. Parallel Ray Tracing with 5D Adaptive Subdivision. In V. Skala, editor, *WSCG 2001 Conference Proceedings*, 2001.

[26] University of Wisconsin-Madison. *Condor Version 6.8.4 Manual*, February 2007.

[27] I. Wald, C. Benthin, and P. Slusallek. Distributed Interactive Ray Tracing of Dynamic Scenes. In *PVG '03: Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, page 11, Washington, DC, USA, 2003. IEEE Computer Society.

[28] I. Wald, C. Benthin, M. Wagner, and P. Slusallek. Interactive Rendering with Coherent Ray Tracing. In A. Chalmers and T.-M. Rhyne, editors, *Computer Graphics Forum Proceedings of EUROGRAPHICS 2001*, volume 20(3), pages 153–164. Blackwell Publishers, Oxford, 2001.

[29] T. Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23(6):343–349, 1980.

[30] S. Woop, J. Schmittler, and P. Slusallek. Rpu: a programmable ray processing unit for realtime ray tracing. *ACM Trans. Graph.*, 24(3):434–444, 2005.