Stingray: Cone Tracing using a software DSM for SCI clusters

Alexandre Meyer, Emmanuel Cecchet

Stingray: Cone tracing using a software DSM for SCI clusters

Alexandre Meyer[1], Emmanuel Cecchet[2]
iMAGIS[1] GRAVIR/IMAG – SIRAC[2]
INRIA Rhône-Alpes – France
{alexandre.meyer, emmanuel.cecchet}@inrialpes.fr

Abstract

In this paper we consider the use of a supercomputer with a hardware shared memory versus a cluster of workstations using a software Distributed Shared Memory (DSM). We focus on ray tracing applications to compare both architectures. We have ported Stingray, a parallel cone tracer developed on a SGI Origin 2000 supercomputer, on a cluster using a Scalable Coherent Interface (SCI) network and a software DSM called SciFS.

We present concepts of cone tracing with Stingray, concepts of SCI cluster with a DSM and the implementation issues. We compare the results obtained with the two architectures and we discuss the trade-off – price/performance/programming ease – of both architectures. We show with Stingray that a modest 12 nodes SCI cluster with an efficient software DSM is 5 times cheaper and can perform up to 2.3 times better than a SGI Origin 2000 with 6 processors. We think that a software DSM is well suited for this kind of applications and provides both ease of programming and scalable performance.

1. Introduction

The work presented in this paper is the results of the collaboration of two research groups. The iMAGIS laboratory develops research projects in image synthesis, mostly using supercomputers like SGI Origin 2000. The Sirac laboratory has set up a cluster of PCs interconnected by a Scalable Coherent Interface (SCI) network. We have developed SciFS, a Distributed Shared Memory (DSM) tightly integrated with the operating system that tries to benefit from the high performances and the remote addressing capabilities of SCI. Our goal is to compare the use of a supercomputer with hardware shared memory with a SCI cluster with a software DSM. To evaluate both architectures, we have ported Stingray, a parallel cone tracer developed for SGI Origin 2000, on the SCI cluster.

The Stingray prototype implements a distributed cone tracer algorithm. The main goal of this algorithm is to compute realistic images like the one illustrated by Figure 1. Because of its nature, this algorithm can't use the functionality of the 3D graphic boards, which usually implement a Z-buffer algorithm. This algorithm only needs processing power, especially arithmetic co-processors.

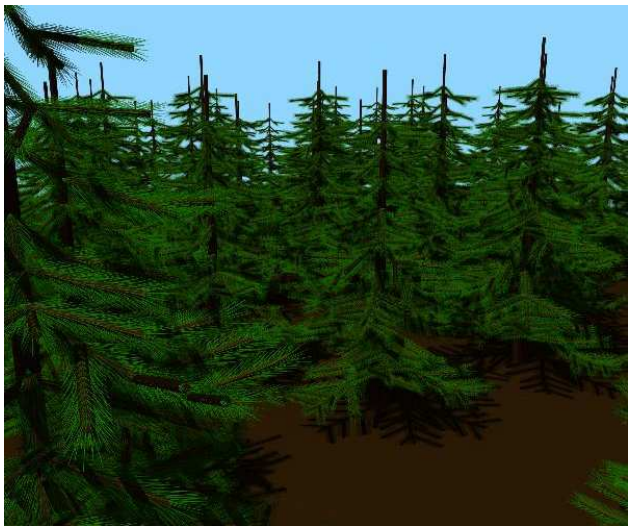

Figure 1. Computed image example

The Stingray prototype is used to research new representations of repetitive objects in domains of level of detail and shaders, two important topics in computer graphics. To illustrate our research in this domain we have to generate images and often animations, which can take several hours of computation.

[1] IMAGIS is joint project of Institut National Polytechnique de Grenoble, Université Joseph Fourier, Centre National de Recherche Scientifique and Institut National de Recherche en Informatique et en Automatique.

[2] Sirac laboratory – Joint laboratory of Institut National Polytechnique de Grenoble, Université Joseph Fourier and Institut National de Recherche en Informatique et en Automatique.

So we have parallelized our prototype in two ways:

- Processes share a scene and each of them computes one image and saves it to the disk. Only the objects that differ between two images are duplicated.
- Processes share a scene and each of them computes a small part of one image. Another process collects results to generate the complete image and then save it to the disk.

The SciOS/SciFS prototype implements a distributed shared memory system on a SCI cluster of Intel PCs with Linux 2.0 or 2.2 kernels and Dolphin's 32-bit (D310) or 64-bit (D321) PCI-SCI adapters [2]. SciOS and SciFS are both implemented as kernel modules and extend the early SciOS prototype described in [3]. Figure 2 shows the SciOS/SciFS architecture and how it interfaces with *Stingray*.

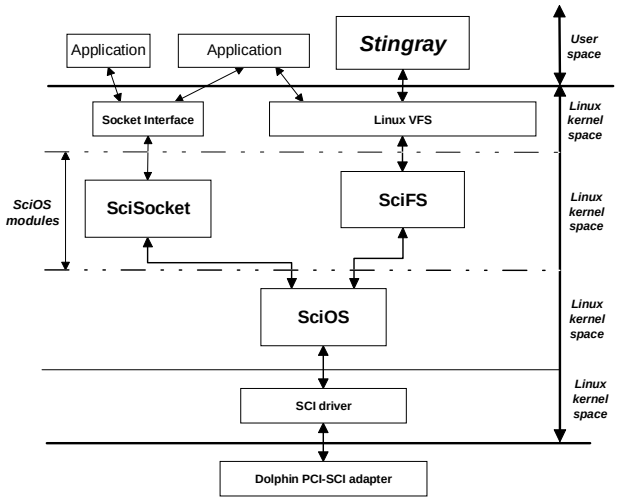

Figure 2. SciOS/SciFS architecture

SciFS really implements the software DSM providing the memory management and coherency protocols. SciFS relies on a lower layer, SciOS, which we also have developed. The SciOS layer is based on Dolphin's PCI-SCI adapters [2] and offers basic services for SCI clusters, such as messages, remote procedure calls and physical memory management. SciFS is implemented as a Linux distributed file system. We find that a file system interfaced with the file mapping mechanism allows a good integration of SCI with the operating system's virtual memory system. The main abstraction in SciFS is a memory-mapped file, which is created and deleted independently of the processes. SciFS interfaces to the Linux Virtual File System (VFS) facility. Shared memory segments are presented to the user as files and are accessed by using file operations such as the *open*, *mmap*, and *close* system calls. A process can open a file, map it in its virtual address space and use normal load and store instructions on the mappings thereby reading and modifying the file's contents. Multiple processes, possibly on different nodes, that open the same file and map it in their address space, share the data contained in the file. Stingray uses the SciFS file system to create map (i.e. share) and destroy the shared memory segments.

In the next sections, we briefly describe Stingray and SciFS features (section 2) and then the implementation of Stingray on SciFS (section 3). In section 4, we evaluate Stingray performance on the cluster and compare the results with those obtained on the SGI Origin 2000. Finally, we discuss related work in section 5 before presenting ongoing work (section 6) and concluding in section 7.

2. Stingray & SciFS features

We start with a small introduction about computer graphics, especially about computation of an image using the ray/cone tracer algorithm. People familiar with this subject can jump to the second part of this section. Technical discussions about ray tracing algorithms can be found at

http://www.acm.org/tog/resources/RTNews/html

2.1. Stingray

To compute an image we start from a 3D scene that contains primitives: polygons, cones, cylinders, spheres, camera, etc. We also may describe how all primitives change during the time if we want to compute an animation. This scene construction is done at modeling time. Then, to compute images, we use a rendering algorithm like our cone tracing. The goal of a rendering algorithm is to compute an image of the scene as viewed from the camera. It has to solve the visibility problem and then to compute the shading of each visible object.

The principle of the ray-tracing algorithm is to compute, for each pixel of the image, the intersection between a ray and the primitives of the scene to keep only the object visible in this pixel. A ray is a virtual line starting from the point of view and crossing the pixel whose color we are computing. This is the visibility problem. Then the pixel color is given by the computation of an illumination model on the surface of the nearest object. This is the shading computation.

Considering a pixel as punctual is an approximation that entails the aliasing problem. In reality pixels are small surfaces that are covered by more than one object. To solve this problem we don’t have to trace lines but beams.

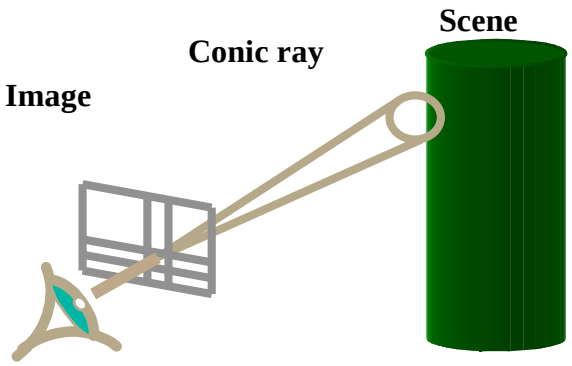

Figure 3. Cone tracing principle

Classical ray tracing algorithm traces several rays per pixel (up to 64 or more per pixel) and average results. On the other hand the cone tracing algorithm trace one conic ray per pixel. Analytical computations of intersection are more complex but the most important in our case of distributed algorithm is that the algorithm always traces one ray per pixel, so scene data are read less time than with a classical ray tracing algorithm.

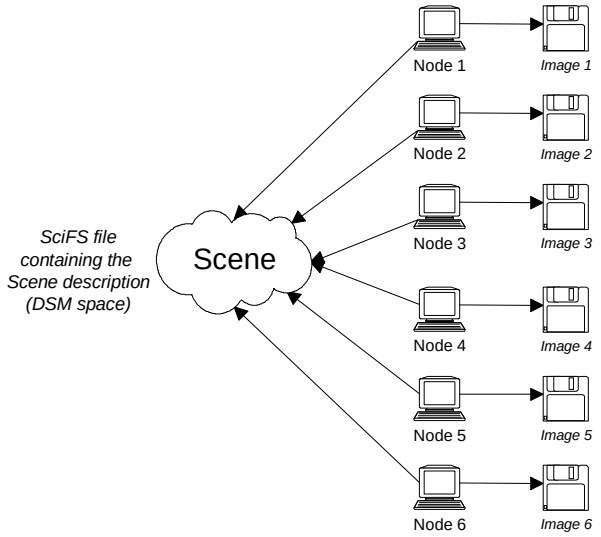

Figure 4. Animation calculation parallelization

The Stingray prototype is a distributed implementation of the cone tracing rendering algorithm. Scenes are often complex by the number of primitives (several millions) but as we use an instantiation technique the size of memory used during the computation is reasonable (several tens of MB to one or two hundred MB). The principle of the instantiation is to re-use the same object in different places of the scene (for example a same tree is re-used thousands of times to make a forest).

As explained in the introduction, we have implemented two kinds of parallelized computation:

- Each image of an animation is computed in parallel. All processes share the scene; only moving objects are duplicated (see Figure 4).
- Each pixel of an image is computed in parallel and one process collects the results to generate the image (see Figure 5).

Note that our scene is only touched in reading mode during computation. Only the image segment is touched in writing mode. All temporary memory needed for computation is allocated in the local memory of each node.

2.2. SciFS

SciFS implements a DSM system and manages physical memory as in a NUMA architecture. The average memory access time is lowered by using several techniques: a relaxed memory consistency model, dynamic page migration and replication mechanism, combined with the use of idle remote memory instead of disk swap.

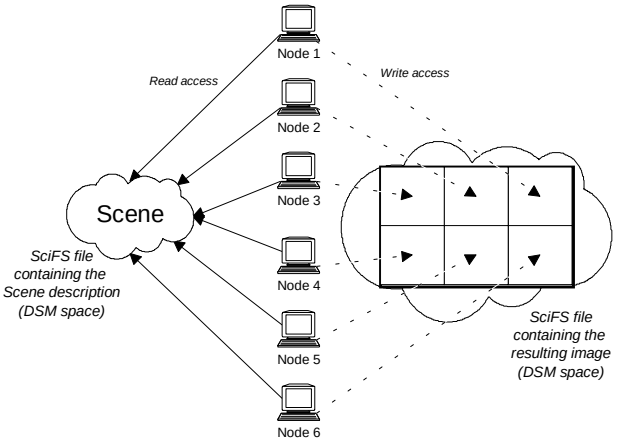

Figure 5. Image calculation parallelization

In SciFS, an application can request different memory management protocols depending on the sharing pattern for a given file. We will describe three of them: the *fixed*, *1st touch* and *global* memory management protocols.

On each page fault, the *fixed* protocol allocates the physical memory on the node that created the SciFS file, i.e., the memory segment. The memory pages are never migrated or replicated but always accessed remotely through SCI mappings.

The *1st touch* protocol allocates the physical memory page on the node that performs the first fault on the page. Like for the *fixed* protocol, the page placement is definitive and pages are accessed remotely using the addressing capabilities of the network.

The *global* protocol allocates the memory like the *1st touch* protocol, but on further accesses pages are either automatically replicated or migrated to lower remote memory references and optimize data locality. The application must be data race free and respect the Lazy Release Consistency (LRC) coherency protocol [1] using the synchronization primitives provided by SciFS.

The SciOS/SciFS prototype sources and documentation are freely available for download from our web site: http://sci-serv.inrialpes.fr.

3. Implementation

In this section we explain the implementation of Stingray on the SciOS/SciFS prototype.

For the two schemes of parallelization (see Section 2) we have to load the scene in a shared memory segment. SciOS/SciFS provides a memory mapped file paradigm to do that. We had to rewrite the malloc/new functions and a STL allocator to use this distributed shared memory instead of the local hardware shared memory.

The scene description is stored in a shared memory segment using the *global* memory management protocol. As the scene is only read shared, SciFS automatically replicates the data on each accessing node. The DSM offers several advantages:

- The programmer does not have to handle data locality and just accesses the objects he needs in the scene. The DSM dynamically replicates only the needed data.
- The scene is only read by one node from the disk to be stored in the DSM and all other nodes access the scene through the DSM. In the case of a large cluster, we have no problem of network file system scalability because the file is only accessed by one node and the replication is the DSM's responsibility. The SciFS replication mechanism is far much faster than accessing the data in a file using NFS over a FastEthernet network.
- As data in SciFS files are persistent, when a process terminates on one node, a new process can re-use all the data that were already replicated in the local memory by the previous process. It reduces communication induced by replication and provides an efficient support on SMP nodes.

In the case where several nodes compute one image, we use a second shared memory segment where image data are stored. Each process stores its pixels in this shared memory segment. One particular process continuously reads the segment and draws the resulting image in a window. Figure 6 shows the architecture used for this experiment. All computing nodes write their data in the fixed memory segment allocated on the viewer node. The throughput of the SCI network in remote write is over 80 MB/s. With a sufficient computing power, it would be realistic to transfer and display images for a real time animation.

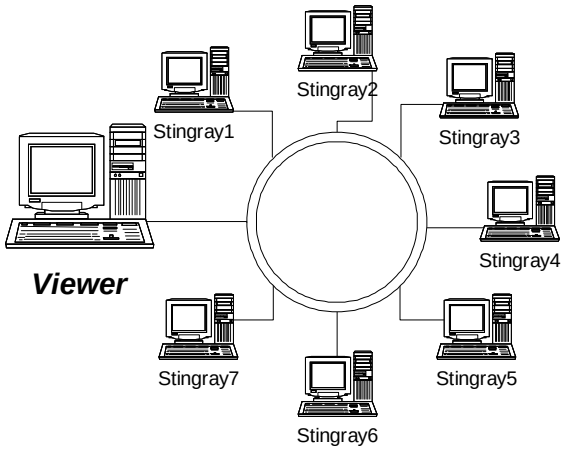

Figure 6. Architecture for image display

As SciFS does not provide support for remote thread creation, the most important problem to do this port was to emulate thread creation calls with "fork; rsh" calls. When multithreading a process you keep all memory initializations that were done before, whereas when you start a new process on another node you have to take care to reinitialize the whole environment: scene data are read from the shared memory but runtime data like the C++ virtual function tables (VTBL) have to be initialized doing a fake reading of the scene because this table is not stored in a shared segment. The port of Stingray from the SGI Origin 2000 to the cluster took a small week with most of the time spent to emulate the remote thread creation using heavy weight processes on each node. SciFS lacks a support for remote thread creation to provide a transparent port of multithreaded parallel applications on the cluster.

4. Evaluation

We have tested and compared Stingray implementations using the SCI cluster and the SGI Origin 2000. The cluster consists of 12 PC equipped with Pentium II-450Mhz, 512 MB of memory, 440BX PCI chipset and Dolphin D321 64 bits SCI adapters and running the software SciOS/SciFS DSM version 2.2 release 3. The SGI Origin 2000 has six R12000 processors at 300Mhz and 4 GB of memory. Our test scene has a hundred pine-trees and around 3 millions of primitives. We use level of detail and shaders presented in [5].

We report on two series of experiments corresponding to the two schemes of parallelization described in 2.1:

- The first scheme is the computation of an animation. It is composed of 160 images that represent an aerial view of the wood described in the scene. The size of each image is 640x400 pixels.
- The second scheme is the computation of a 1200x1000 pixels image by several processes. The camera is in the middle of the wood so there are many objects, resulting in complex calculations. Figure 1 shows the image computed in this experiment.

Table 1 presents the results obtained for the calculation of the animation.

Table 1. Computation of an animation

	Animations of 160 images (640x400)	Speedup
Standalone node	6h31min7s	**1**
6 nodes with SciOS/SciFS	1h03min42s	**6.1**
12 nodes with SciOS/SciFS	31min48s	**12.2**
Origin 2000 1 CPU	4h29min54s	**1**
Origin 2000 6 CPU	44min59s	**6**

In this experiment, we obtain a super linear speedup with the cluster because we use our dedicated implementation of malloc/new functions and our own STL allocator for the distributed version. These implementations are slightly more efficient than the standard ones because they are dedicated to our problem. These good results are also due to the simple sharing pattern because we touch the scene segment only in reading mode. Then we quickly reach the configuration where all nodes have their own copy of the scene in their local memory and process the images locally.

The SGI Origin 2000 obtains a linear speedup too. But a processor of the SGI is 40 to 45% faster than a Pentium II 450 MHz because most of the calculation uses the floating-point unit that is the weak point of the Pentium processor. The cluster using 12 nodes is about 30% faster than the Origin 2000 with 6 processors.

The second test performed is the computation of one image by several processes. Each node writes remotely its part of the image in a shared memory segment. Table 2 shows the results obtained for this experiment.

SCI provides a large bandwidth and a low latency that allow the DSM to perform the remote write of each process data to the viewer node with very low overhead. But the processor remains busy during the remote writes. The sequential version is still 47% faster on the SGI O2K than on the PC. But it is very interesting to observe the cluster using SciFS scales better than the Origin 2000. We achieve a speedup of 11.2 with a 12 nodes cluster when we only obtain a speedup of 3.2 with the full power of the SGI. Finally the 12 nodes cluster performs 2.31 times better than the SGI O2K with 6 processors.

On the Origin 2000 we use the PThread implementation that seems to be not as optimized as the SPROC implementation. Perhaps an implementation with SPROC will decrease the overhead but for compatibility reason with other Unix we keep this PThread version.

Table 2. Parallel computation of one image

	Image 1200x1000	Speedup
Standalone node	51min38s	**1**
6 nodes with SciOS/SciFS	9m32s	**5.4**
12 nodes with SciOS/SciFS	4m39s	**11.2**
Origin 2000 1 CPU	33min55s	**1**
Origin 2000 6 CPU	10min45s	**3.2**

In our case where only arithmetic processors are used the SGI Origin 2000 with 6 processors performs as well as a SCI cluster with 9 nodes. Our 12 nodes SCI cluster cost about $36000, which is 5 times less than an Origin 2000 with 6 processors (about $180000 without graphic board). With little effort, the Stingray application was ported to the SCI cluster using the SciOS/SciFS software DSM. We were able to achieve better performance on this small cluster than on the SGI supercomputer. The cluster can be easily upgraded and benefit from newer generation processors reusing the same network interconnects. Upgrading an Origin 2000 is an heavy investment and often requires complete hardware replacement.

The programming ease is very close on both architectures due to the shared memory paradigm. Anyway the

SciOS/SciFS prototype lacks a support for distributed multithreaded application to really offer the same facilities to the end-user. On one hand we showed that the performance/price ratio is clearly better with the cluster. On the other hand, it is important to note that the Origin 2000 has been designed for graphic applications using graphic board and not only processors like in our experiments. So the Origin remains a major choice for graphic applications but for more computational applications the SCI cluster is an excellent alternative.

5. Related Work

The ray tracing algorithm was introduced by Whitted [7] and improved [4,5,15] to fix the aliasing problem. Later, several works have tried to limit the number of rays in an image or in an animation by taking care of scene coherence (for example [14]). Other works have used these results and parallelization results to develop a real time ray tracer (for example [10,11]).

Because of the simplicity of parallelizing the ray tracing algorithm many distributed implementations have been done, which may be divided in two categories:

- Ray tracing on one computer with several processors using classical shared memory and thread processes.
- Ray tracing on network of workstations using a standard Ethernet network where processes use socket and network tools to communicate.

One computer with several processors gives efficient results but cost quickly limits the number of processors. The cost of several workstations using a legacy network is reasonable but communication overhead with network tools (without SCI technology) limits the performance.

Remote mapped network technology like SCI provides an efficient hardware support for software DSMs like SciOS/SciFS. We can take advantage of the good price/performance/programming ease of SCI clusters to develop efficient rendering algorithms. Other research projects work on parallel ray tracing using high performance clusters. For example, the PM2POV-Ray project is a parallelization of the public domain Persistence Of Vision Ray-Tracer using the PM² environment [12]. Unlike our approach it uses message passing and performs load balancing using thread migration. To our knowledge there are no other works about cone tracing using a software DSM for clusters.

6. Ongoing and future work

A 200-processor SCI cluster will be set up at INRIA Rhône-Alpes in 2001. With this computational power, we will be able to investigate real time animation problems.

As the SCI network allows any part of a node's physical address space to be mapped, it is possible for one node to remotely map the memory of the video adapter of another node. This mechanism is illustrated on Figure 7.

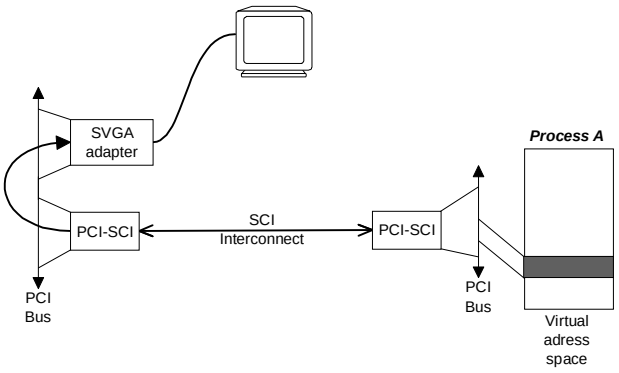

Figure 7. Remote access to a video adapter

We think that we can reduce the image transfer overhead due to the data and computation distribution, by letting each node write directly its data in the video board memory of the displaying node through SCI mappings. This will solve one of the main issues the DDDDRRaW project had to face [13].

Another promising idea is to use a video boards with hardware 3D features in each node. Using other algorithm like Z-Buffer techniques, we can compute the images in the video adapters and use direct SCI transfers from a local video memory to a remote video memory to merge the results on one node.

7. Conclusion

Our experience with the Stingray parallel cone tracer shows that a cluster of PC using a software DSM like SciOS/SciFS is a valid alternative to a supercomputer. With this kind of application, our 12 nodes cluster is 5 times cheaper and can perform up to 2.3 times better than the SGI Origin 2000 with 6 processors. It took one week to adapt the code from the SGI to the cluster, and most of this work was to emulate a remote thread creation that was not provided by SciOS/SciFS.

It is true that our application is well-suited for clusters: easy to parallelize, few communications between nodes and large use of arithmetic processors. Anyway we believe that clusters using a software DSM can provide an excellent performance/price/programming ease tradeoff for a large range of applications.

There is still much research to do in this way with other parallel algorithms. But this first step with a cone tracer is a very promising start and forecasts a good future for the use of software DSMs on SCI clusters for parallel algorithms in computer graphics and other disciplines.

8. References

[1] Pete Keleher, Alan L. Cox, Sandhya Dwarkadas, and Willy Zwaenepoel. TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems – *Proceedings of the Winter 94 Usenix Conference*, pp. 115-132, January 1994

[2] Dolphin Interconnect Solutions. PCI-SCI cluster adapter specification, May 1996. Version 1.2.

[3] Povl T. Koch, E. Cecchet and X. Rousset de Pina. Global Management of Coherent Shared Memory on an SCI Cluster - *Proceedings of SCI Europe'98*, pp. 51-57, September 1998

[4] Paul S. Heckber and Pat Hanrahan. Beam Tracing Polygonal Objects - *Proceedings of Computer Graphics SIGGRAPH'84*, pp. 119-127, July 1984

[5] John Amanatides. Ray Tracing with Cones – *Proceedings Computer Graphics SIGGRAPH'84*, pp. 129-135, July 1984

[6] Cook, Robert L., Thomas Porter and Loren Carpenter. Distributed Raytracing – *Proceedings Computer Graphics SIGGRAPH'84*, pp. 137-145, July 1984

[7] Turner Whitted. An Improved Illumination Model for Shaded Display – *Communications of the ACM*, pp. 343-349, June 1980

[8] Alexandre Meyer and Fabrice Neyret. Multiscale Shaders for the Efficient Realistic Rendering of Pine-Trees – *Graphics Interface*, May 2000

[9] E. Reinhard, A. Chalmers and F.W. Jansen. Overview of parallel photo-realistic graphics – *Eurographics '98 State of the Art Reports*, August 1998

[10] S. Parker, W. Martin, P. Sloan, P. Shirley, B. Smits, and C. Hansen. Interactive ray tracing – *Symposium on Interactive 3D Computer Graphics*, April 1999

[11] B. Walter, G. Drettakis and S. Parker. Interactive Rendering using Render Cache – *Eurographics Workshop on Rendering '99*, June 1999

[12] Olivier Aumage, Luc Bougé, and Raymond Namyst. A portable and adaptative multi-protocol communication library for multithreaded runtime systems. *Proceedings of the 4th Workshop on Runtime Systems for Parallel*, pages 1136–1143, Cancun, Mexico, May 2000

[13] Thu D. Nguyen and John Zahorjan. Image Layer Decomposition for Distributed Rendering on NOWs - University of Washington, Department of Computer Science & Engineering *Technical Report UW-CSE-99-10-01*, October 1999

[14] D. Jevans, Object Space Temporal Coherence for Ray Tracing – *Proceedings of Graphics Interface '92*, Toronto, May 1992

[15] Dipplé A. Mark and Erling Henry Wold, Antialiasing Through Stochastic Sampling – *SIGGRAPH 85*, pp. 69-78