Multi-Core Data Streaming Architecture for Ray Tracing

Yoshiyuki Kaeriyama, Daichi Zaitsu, Kenichi Suzuki, Hiroaki Kobayashi
Graduate School of Information Sciences, Tohoku University
yoshi@sc.isc.tohoku.ac.jp, zaitsu,suzuki@archi.is.tohoku.ac.jp, koba@isc.tohoku.ac.jp

Nobuyuki Ohba
IBM Research, Tokyo Research Laboratory, IBM Japan, Ltd.
ooba@jp.ibm.com

Abstract

Ray tracing is a computer graphics technique to generate photo-realistic images. Although it can generate precise and realistic images, it requires a large amount of computation. The intersection test between rays and objects is one of the dominant factors of the ray tracing speed.

We propose a new parallel processing architecture, named RAPLAS, for accelerating the ray tracing computation. RAPLAS boosts the speed of the intersection test by using a new algorithm based on ray-casting through planes. A data streaming architecture offers highly efficient data provision in a multi-core environment. We estimate the performance of a future SoC implementation of RAPLAS by software simulation, and show 600 times speedup over a conventional CPU implementation.

1 Introduction

Three dimensional computer graphics is highly demanded in many industrial and consumer electronics areas, such as architecture, automotive, aerospace, and game businesses. In these areas, the quality of generated images is important. Ray tracing is a well known method to generate photo realistic images. It is based on geometrical optics accurately simulating physical lighting, and illustrates specular reflection/refraction, fuzzy reflection, shadow, and so on. Figure 1 shows our test scenes generated by ray tracing using an accurate soft shadow technique. Although ray tracing generates photo-realistic images, it requires a large number of computations, which have made ray tracing be used in limited areas of computer graphics.

The intersection test between rays and objects is one of the factors dominating the processing speed of ray tracing. Fortunately, we can find a lot of parallelism in the intersection test. The key factor for boosting the speed of intersection test is to exploit plenty of parallelism. Recent GPUs [5] have high performance SIMD engines. However, they are specialized for conventional image generation. The programmability of the GPU is not enough for ray tracing, and memory usage is also strongly limited. Therefore, it is quite difficult to implement the processing of ray tracing onto a GPU. The GPU will not be the best solution for accelerating the ray tracing now and in near future. Ray tracing on a conventional CPU, equipped with multiple cores, has been widely studied [8][11]. In particular, CELL [6], which has multiple calculation engines providing high capability and flexibility, may be thought as a good candidate for the ray tracing processor. However, even these CPU and CELL cannot fully exploit the huge amount of and several types of the parallelism in the intersection test, because they are equipped with only several processor cores.

In this paper, we focus on the design of ray-tracing hardware. The advance in VLSI technologies makes it possible to integrate many more functions into a single chip. Taking account of processing speed and power efficiency, developing a new architecture and creating an SoC specialized for ray tracing is the best solution to achieve ultimate perfor-

Figure 1. Test scenes generated by ray tracing

(a) Z33 (b) Bunny (c) Conference
mance of ray tracing. To fully exploit the parallelism in the intersection test, we propose a hardware architecture, named RAPLAS (Ray tracing Architecture based on PLane And Sphere intersection). We present a novel intersection test method using ray-plane and bounding sphere, which can utilize plenty of parallelism in the processing. The proposed architecture consists of multiple floating-point calculation engines specialized for the intersection test to exploit different types of the parallelism. To keep high scalability of the engines and low hardware cost, our architecture employs a data-streaming memory system.

This paper is organized as follows. Section 2 presents a new algorithm for the intersection test. Section 3 proposes a novel architecture for ray tracing. Section 4 demonstrates the performance estimation of the proposed architecture implemented as a future SoC chip. Section 7 offers concluding remarks.

2 Intersection Tests using Ray-Plane and Bounding Spheres

Ray tracing shoots rays from an origin point (an eye) to an object space. A ray traverses the space until it intersects an object. During the traverse, intersection tests between the ray and objects are performed. When the closest object blocking the ray is found, plenty of new rays to illustrate various lighting effects such as reflection, refraction, and shadow are re-generated. To find the nearest object for origin point of each ray, the intersection test between all the rays and objects must be done. Therefore, the intersection test needs a large amount of computation.

To reduce the number of computations, a table look up method is often used. Before shooting a ray, approximate locations of all the objects are calculated and stored in a location table. By referring to the table, we can find the objects that obviously do not intersect the ray, and thus we do not perform the intersection test for them. However, in case of parallel processing which can accelerate the ray tracing dramatically, as the number of PEs (Processing Elements) increases, the accesses to the look-up table will cause a serious memory contention, which decreases the PE utilization.

In a ray tracing process, a series of rays diffused from a same origin are widely used as shown in Figure 2. We have focused on the process of the series of rays, and proposed an accelerating method of the intersection test using ray-planes and bounding spheres. The principal idea of the method is to cast ray-planes toward the grid points instead of casting rays individually. Figure 3 shows the six steps of the intersection test method.

Step 1 Generate bounding spheres, each of which includes several triangles.

Step 2 Cast rays passing through the grid points on a certain plane above the origin, even if the rays essentially need to cast completely random directions.

Step 3 Make a plane which includes the origin and aligned grid points. We call it "ray-plane".

Step 4 By repeating Step 3 for vertical and horizontal directions, all the orthogonal planes are obtained.

Step 5 Perform the intersection test between the bounding spheres and all the ray-planes.

Step 6 For each bounding sphere which has intersected any two orthogonal planes, the intersection test are performed between the objects in the bounding sphere and the original ray reproduced from the two intersected ray-planes.

Let us assume that N x N rays are cast to the space. The conventional methods require N x N intersection tests per bounding sphere. The proposed method, on the other hand, uses planes instead of rays and requires only 2N intersection tests per bounding sphere. The method not only reduces the number of computations but also easily exploits the parallelism residing in the
intersection test. The intersection tests are performed between sequential objects and different rays or ray groups at the same time, as shown in Figure 4. In the processing, we can find three types of parallelism.

**Type I** Different ray groups can be tested in parallel (coarse grain spatial parallelism).

**Type II** Different ray-planes can be tested in parallel (fine grain spatial parallelism).

**Type III** A large number of spheres or triangles are calculated in parallel (temporal parallelism).

Our architecture introduced in the next section is designed to utilize all of them.

### 3 Ray Tracing Architecture RAPLAS

**3.1 Multi-core Data-Streaming Architecture**

Although the parallel processing boosts the intersection test speed, simply increasing the number of PEs is not necessarily effective. This is because that the data must be supplied to all the PEs. In the case of our intersection method, only the triangles included in the intersected sphere are used. Unfortunately, since each PE works on different spheres and rays, it is impossible to know when and where the data for the triangle is needed before starting the operation. In other words, memory accesses coming from the PEs become random.

A shared memory system, as shown in Figure 5(a), has a limited scalability owing to the memory access conflict. A distributed memory system, as shown in Figure 5(b), has a good scalability, but requires high hardware cost to implement. To avoid the access conflict and yet to save the hardware cost, we propose a multi-core data-streaming architecture for ray tracing, as shown in Figure 5(c). The proposed memory system has a broadcasting bus instead of a shared memory bus. It can accommodate many PEs without causing access conflicts, and thus has a good scalability.

The sphere and triangle data are sequentially broadcasted to all the PEs. Each PE has a data selector and a FIFO. The selector passes through only the objects used by its corresponding PE and stores the objects in the FIFO.

The proposed memory system has following features: 1) When the number of PEs is small, PEs often disregard the data on the broadcasting bus because they simply do not need them at that time. When the number of PEs is large, PEs have more chances to receive the data required for the intersection test. Let us think about an extreme case where the number of PEs is as large as that of rays. The data broadcasting is completed in one turn. 2) If one of the FIFOs becomes full, the broadcasting operation must be ceased. For this reason, we have to pay a special attention to the FIFO size.

**3.2 Structure of RAPLAS**

Figure 6 shows the detail structure of RAPLAS. It consists of multiple ICCs (Intersection Calculation Cores), and two memory modules (Triangle Memory and Sphere Memory). Each ICC has two different types of fully pipelined floating-point calculation engines (Plane-Sphere Intersector and Ray-Triangle Intersector), Selector, and FIFO.

The first advantage of RAPLAS is that it has a good scalability while keeping the hardware cost of the memory system low. Two data broadcasting buses, Sphere Stream Link and Triangle Stream Link, supply the data of bounding spheres and triangles to all the PEs without break. Each ICC chooses only the data necessary for the intersection tests in charge. In addition, this simple data transfer does not need additional hardware for bus arbitration which is necessary for typical shared memory systems.

The second advantage of RAPLAS is that it can utilize all the three types of parallelism described at the end of Section 2. Type I parallelism (coarse grain spatial) is utilized by having multiple ICCs in the system. Each ICC is in charge of several ray groups and has no data to share with the other ICCs. Type II parallelism (fine grain spatial) is utilized by multiple Plane-Sphere Intersectors in an ICC. Each ray group has a number of ray-planes, which are
processed by Plane-Sphere Interectors in parallel. Type III parallelism (temporal) is utilized by the pipeline operation in the Intersectors.

At the first step of an image generation, all the data on spheres and triangles are generated by the host computer, and stored in the Triangle and Sphere Memories. In the second step, the sphere and triangle data are broadcasted through the stream links to the ICCs. Each ICCs stores the data in its incoming FIFO and performs plane-sphere intersection tests by using its multiple Plane-Sphere Interectors. For any intersected spheres, the ICC performs the intersection tests between the rays in the ray-planes and triangles included in the sphere.

4 Implementation Details

This section describes the detail implementation of RAPLAS. Figure 7 shows the block diagram of an accelerating board based on the RAPLAS architecture. The board is compliant to a PCI form factor and supposed to be installed in a PCI slot of PC. The board has a multi-core SoC called BUICC (BUnch of Intersection Calculation Cores) and two off-chip DRAM modules storing sphere and triangle data.

BUICC has a general purpose CPU core and multiple ICC cores. The general purpose CPU is in charge of image synthesis operations other than the intersection test. TheICC consists of Plane-Sphere Intersector, Ray-Triangle Intersector, FIFOs, Decoder, Triangle Selector, and Ray Data Fetcher. The ICC shown in Figure 7 has eight Plane-Sphere Interectors and a Ray-Triangle Intersector, so that it is optimized for the operation of 4 × 4 rays.

Plane-Sphere Intersector executes the intersection test between planes and bounding spheres. The Intersector calculates the distance between the plane and the sphere of interest to check whether the sphere intersects the plane or not. Triangle Selector selects the triangle data, which are included in the intersected sphere. If it finds no intersected planes, the triangle data are discarded; otherwise, the triangle data are written into Triangle FIFO. Ray Data Fetcher reads the ray data corresponding to the plane-IDs, and inputs the ray data into Ray-Triangle Intersector. Ray-Triangle Intersector performs the intersection test of rays and triangles. After the intersection test, the triangle-ID and the distance between the intersection point and the start point of each ray are stored in the memory.

Plane-Sphere Intersector and Ray-Triangle Intersector are the key components of the ICC as they directly affect the system performance. We designed custom-made floating-point adder and multiplier. The Intersectors use a 24-bit floating-point format, which we found optimal for RAPLAS in terms of the speed and accuracy. We have done a lot of simulations and confirmed that the 24-bit format is enough to generate images with good quality.
The Plane-Sphere Intersector executes the intersection test as follows. Given the center coordinate of sphere \((c_x, c_y, c_z)\), and the plane equation \(n_x x + n_y y + n_z z + p_w = 0\), the distance between the plane and the center of sphere is calculated by:

\[
d = n_x x + n_y y + n_z z + p_w
\]  

(1)

The floating-point operations for this calculation are three additions and three multiplications. In the image synthesis process, to increase the throughput is more important than to shorten the latency. Taking this into consideration, we designed the Plane-Sphere Intersector with a 10-stage pipeline. It should be noted that this operation is to simply check whether the plane intersects the sphere or not. Therefore, we do not have the floating-point normalization to save the hardware resources.

The Ray-Triangle Intersector calculates the distance between the origin point and the intersection point, and calculates the position of the intersection point. Let us define the ray vector \(D\) and the origin of ray \(O\). The triangle consists of location vectors \(V_0, V_1\) and \(V_2\), edge vectors \(E_1\) and \(E_2\), and normal vector \(E\). The intersection point on the triangle defined by two barycentric coordinates \(u\) and \(v\) is calculated by the following equation.

\[
\begin{bmatrix}
  t \\
  u \\
  v
\end{bmatrix} = \frac{1}{D \cdot E} \begin{bmatrix}
  N \quad \begin{pmatrix} O \\ V_0 \end{pmatrix} \\
  D \quad \begin{pmatrix} O \\ V_0 \end{pmatrix} \\
  D \quad \begin{pmatrix} O \\ V_0 \end{pmatrix}
\end{bmatrix} (D \quad E) (t \ u \ v)
\]  

(2)

When conditions \(u > 0, v > 0\) and \(u+v < 1\) are all met, the intersection point is positioned inside the triangle. However, division in Equation (2) requires more CPU cycles than addition or multiplication. To avoid this division, we use new condition parameters as \((t', u', v') = (D \cdot E)(t \ u \ v)\). The new intersection conditions are defined as \(u' > 0, v' > 0\) and \(u' + v' < N \cdot D\).

The floating-point operations for this calculation are 20 additions and 15 multiplications. In the same way as the Plane-Sphere Intersector, we designed the Ray-Triangle Intersector with a 25-stage pipeline.

## 5 Performance Evaluation

### 5.1 Effect of Parallelization

In order to estimate the performance of our data-streaming hardware, we did software simulations. We made a software simulator by using C++ and SystemC. We used Microsoft Visual C++ .Net and SystemC Version 2.0.1 for Microsoft Windows. The simulation consists of two steps. 1) A software renderer written in C++ generates profile information. 2) Based on the profile information, a SystemC simulator performs the bus-cycle accurate simulation.

The hardware parameters for the simulation are shown in Table 1. Three images in Figure 1 and Table 2 were used as test scenes. Image size is 64 × 64. To process 16 rays in parallel, each ICC has 32 Plane-Sphere Intersectors pipelines and one Ray-Triangle Intersector. The sizes of Sphere FIFOs and Triangle FIFOs were set to 10, 100, and 1,000. We investigated the total number of execution cycles by varying the number of ICCs and FIFO size.

As shown in Figure 8, when the size of the FIFOs is 1,000 (it requires several dozen KB memory for each ICC), RAPLAS achieves over 90% of scalability in all the test scenes for 48 ICCs. However, increasing the number of ICCs does not contribute to the performance when the size of FIFOs is 10 or 100. It is due to the FIFO full condition, which stops the object data broadcasting to all the ICCs. Based on the results, we conclude that RAPLAS can realize highly effective parallel processing of ray tracing.

### 5.2 Performance Comparison with Other Data Path Designs

To show the superiorities of RAPLAS, we compare its performance with a shared memory system without streaming and a distributed memory system. The experimental condition is same as the previous sub-section, and the FIFO size is assumed to be 1,000.

In the distributed memory system, the performance increases linearly with the number of ICCs, but the area cost of memories also increases linearly. In contrast, although the scalability in RAPLAS is limited as shown in Figure 8, the area cost of RAPLAS increases slowly compared with that of the distributed memory system. To estimate the trade-off between the area cost and performance, we calculate the product of the die size and the processing time for
Figure 8. Scalabilities

(a) Z33  
(b) Bunny  
(c) Conference

Figure 9. Product of die size and processing time

Figure 10. Total number of execution cycles for triangle intersection

the intersection test. Figure 9 shows the results normalized by the case of a distributed memory system. We assume the area ratio of PE and memory is 1:2. When the number of ICCs is 16 or more, the product of the area cost and the processing time of RALPAS becomes lower than that of the distributed memory system in all the test scenes.

In the shared memory system without streaming, the total volume of data transfer is reduced as the number of ICC increases, and processing time will be shorter. To confirm this, we investigated the total number of execution cycles for the ray and triangle intersection test as shown in Figure 10, when the PE of the shared memory system is assumed to be fast enough. When the number of ICC is 16 or more, we can see that the total number of the execution cycles of RALPAS is less than the shared memory system in two of the three test scenes.

However, in the Conference scene, even if the number of ICC is 48, the total number of the execution cycles is still large than that of the shared memory system. The reason is that the number of triangles in this scene is larger than that of the other scenes. It is not a defect of RALPAS, because we have seen the stream link is effective if the number of ICCs is large enough, and for scenes like Conference, RALPAS outperforms shared memory systems when a larger number of ICCs are equipped. We believe that RALPAS will outperform the shared memory system if the number of ICC goes greater than 64, which could not be simulated this time because of the memory size of the SystemC simulator.

5.3 Performance Estimation of BUICC

To evaluate the performance of BUICC, we developed a prototype of the ICC by using FPGA. FPGA is suited for prototyping ICC, because recent FPGA devices are
equipped with hardware DSP macros, which are very attractive to designing custom-made floating-point calculators. Table 3 summarizes the specification of the board we used for prototyping. The board has a Xilinx Virtex-4 SX35 FPGA, which has 192 DSP hardware macros. Table 4 shows the hardware resources for the implementation and the frequencies. We confirmed that the prototype runs at 160MHz or faster.

Based on the result of the FPGA prototype implementation, we estimate the performance of BUICC in an ASIC implementation. Kuon et al. discussed the performance and cost of FPGA in comparison with ASIC [4]. They reported that an ASIC of 90nm technology runs at 2.8 times higher operating frequency than its FPGA implementation and saves the hardware cost to 1/21. Based on their results, BUICC ASIC will have 16 ICC cores and run at 400MHz. We assumed that the throughput of ICC is one task per cycle and the memory system supplies the data quickly enough to accommodate the speed.

The same ray tracing operation was written in Visual C++.Net and run on an X86 CPU to evaluate the performance of ASIC BUICC. We used single-precision floating-point calculation in the ray tracing program. It was run on an PC with 2.4GHz Intel Core 2 Duo E6600 Processor and 2GB DDR2 800MHz SDRAM. To fully exploit the SIMD and multi-thread function in the X86 processor, the program was compiled and optimized by Intel C++ Compiler Version 9.0.

The performance estimation results are shown in Table 5. It is seen that one ICC already gives 40 times faster performance than the X86 CPU. The reasons are as follows. The SIMD engine in an X86 CPU can execute only four floating-point operations in a cycle. In addition, the SIMD engine requires register transfer operations and load/store operations. In contrast to the CPU, ICC is highly optimized for the intersection test, so that it can execute 80 floating-point operations in parallel, which means that it can work on eight ray-planes at a time. BUICC equipped with 16 ICCs achieves more than 600 times speed up over the CPU. This is because the RAPLAS effectively exploits Type I (coarse grain) parallelism in the intersection test.

Let us discuss the data transfer rate on the stream links. Assuming the BUICC runs at 400MHz, it requires 6.4GB/sec and 15.6GB/sec bandwidths for Sphere Stream Link and Triangle Stream Link, respectively. We believe these are feasible transfer rates using recent DRAM and serial link technologies. For example, recent GPU [5] has a 100GB/sec memory bus. Note that the transfer rate is constant with respect to the number of ICCs and the burst transfer of SDRAM can be used for the sphere and triangle data broadcast. The size of the FIFO in front of the ICC should be 1,000, which can be implemented with 40KB SRAM.

### 5.4 Performance Comparison with Other Hardware System

There are several approaches in the hardware acceleration of ray tracing.

The GPU, which is primarily designed for conventional image synthesis, can also be used for ray tracing. The acceleration of ray tracing using the programmable unit embedded in the GPU was reported in [7]. The authors estimated that the GPU gives 10 times speed up over the conventional CPU. However, GPU has drawbacks for ray tracing. The local memory of GPU is not so flexible to manipulate various ray and object data. The ray and object data must be fetched from the memory through the CPU-GPU bus, which is a bottleneck of the data transfer.

A ray tracing hardware using CELL was reported in [1]. The paper showed that the 2.4GHz CELL processor gives 6.5 times better performance than the 2.4GHz AMD Opteron. The CELL processor has eight SPUs, each of
which has four floating-point pipelines. This configuration is not enough to fully exploit parallelism residing in the intersection test. In addition, a 256KB local memory for each SPU is not large enough to sustain the ray and object data.

Several hardware architectures for ray tracing were proposed. Sanchez-Elez et al. presented a mapping scheme of an optimized octree-based ray tracing algorithm and its implementation on a SIMD reconfigurable architecture, MorphoSys [9]. In addition, DCGiRAM integrates functional memories with processing elements [3]. It uses virtual 3D intelligent memories for divided object spaces, and performs the ray-object intersection tests in a fully distributed manner. However, these proposals were difficult to be applied to animated scenes, because they require a recalculation of the object mapping in every frame of animation scenes. In contrast, RAPLAS does not need such a recalculation for dynamic scenes and can easily be applied to them.

Slusallek et al. also proposed an architecture named SaarCor for accelerating ray tracing in [10]. The performance estimation based on a future ASIC implementation was also reported in [12]. The authors described that the future ASIC implementation with eight calculation units running at 400MHz achieves approximately 80 times better performance over the conventional CPU. It requires 25.6GB/sec bandwidth in the interconnection bus. Because the test scenes and experimental conditions are different from ours, we cannot directly compare the performance of SaarCor and RAPLAS. We think that RAPLAS has more scalability than SaarCor because of the RAPLAS memory system with broadcasting buses.

6 Concluding Remarks and Future Work

We have proposed a new SoC architecture named RAPLAS for ray tracing. The RAPLAS is based on a ray-plane casting algorithm and specialized for the intersection test. It has fully exploited three types of parallelism in the processing of the plane-sphere intersection test, and kept the scalability of the calculation engine high by using broadcasting memory buses. The performance of RAPLAS has been estimated by software simulations. We conclude that RAPLAS will give a hundreds times speed up over a ray tracing system using a conventional CPU.

In the near future, we will design and build ASIC version of RAPLAS to realize real-time ray tracing animations.

Acknowledgements

The authors would like to thank to Professor Emeritus Tadao Nakamura, Professor Ryusuke Egawa, and Mr. Kazuhiro Komatsu of Tohoku University for their comments for this research. This research is partially supported by Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research(B) 18300011.

References