FPGA HARDWARE ACCELERATION FOR VISUALIZATION WITH USE OF THE RAY TRACING ALGORITHM

Liberios VOKOROKOS*, Branislav MADOŠ*, Viktor RUSKA**
*Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics,
Technical University of Košice, Letná 9, 042 00 Košice, Slovak Republic, tel. +421 55 602 3023,
e-mail: {liberios.vokorokos, branislav.mados}@tuke.sk, viktor.ruska@student.tuke.sk

ABSTRACT

This paper deals with the issue of hardware acceleration of photorealistic visualization with use of ray tracing technique in real time. First part of the article briefly introduces ray tracing and presents existing hardware architectures that are accelerating computations with aim to bring real time ray tracing; second part of the article introduces the implementation of the designed solution that represents two modules implemented with use of the VHDL language and the FPGA technology. Modules are accelerating the part of ray tracing algorithm that according to the research requires the most system resources. This part consists of computation needed to find the ray-object intersection. The first module computes ray – sphere intersection, and the second computes ray – triangle intersection. Both modules are composed of number of components, which were optimized and parallel computing was applied in highest possible measure.

Keywords: *ray tracing, rendering, FPGA, VHDL, parallel computing*

1. INTRODUCTION

It is possible to define the ray tracing as the technique used in the computer graphics to produce images with a very high degree of visual realism by calculating the trajectory of the ray of light that is entering imaginary observer's eye. Paths of different rays of light are traced backward, passing from the viewpoint through the each pixel of the plane where image is produced, until they encounter with the surface of the object that is present in the scene or until the ray leaves the scene going into the infinity.

At first glance, process, in which the ray of light is not traced from the source of the light, but vice versa, leading from the observer's eye into the scene, seems like counter-productive and counter-intuitive and goes against the physical reality. This, however, is many times more effective technique compared to the tracing of the trajectory of the ray of light from its source, because the overwhelming majority of light rays from a given light source do not make it directly into the viewer's eye, but leaves the scene in another direction. Calculation of the trajectory and other properties of the ray, which will eventually not reach the observer's eye, is wasteful and the number of rays of light which are unnecessary calculated can be enormous (Fig. 1,2).

Modelled objects are represented by sets of interconnected facets of a wide range of different types, such as triangles, squares, parts of the sphere or more complex surfaces, such as the 3D splines.

Optical properties of object facets, such as the colour, texture, reflectance, transmittance, refraction and also the position, colour, and brightness of light sources, including ambient lights, are also taken into account when the colour and the brightness of the traced ray of light is calculated. Ray tracing is also able to simulate a wide variety of optical effects as the scattering, and dispersion, for example the chromatic aberration. It can produce images with mirrors, shadows and transparent surfaces with great results.

Ray tracing is excellent in its ability to produce a very high degree of visual realism, compared to other visualization methods such as the scanline rendering methods, but this is offset by very high computational cost. That is why the performance can be seen as the biggest disadvantage of the technique in comparison to other techniques, such as the scanline algorithms, and others, which are sharing information when calculating pixels of the image. Ray tracing oppositely, starts calculation from anew for each ray of the light and traces each ray separately.

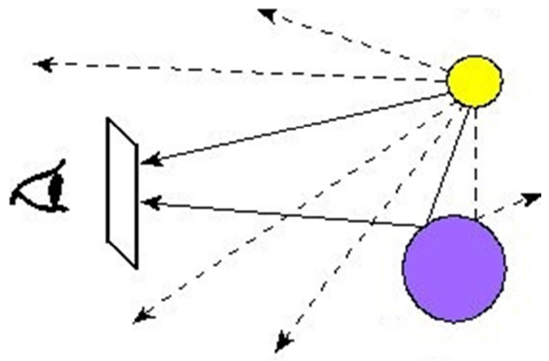

Fig. 1 Tracing light from the source to the observer's eye [1]

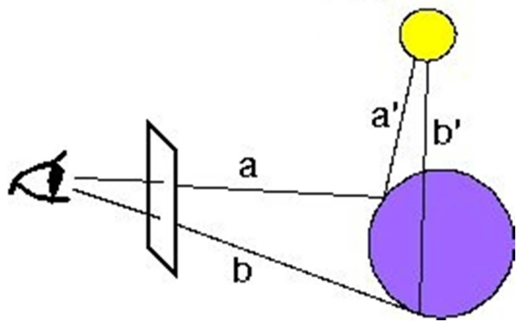

Fig. 2 Tracing light from the observer's eye to the source [1]

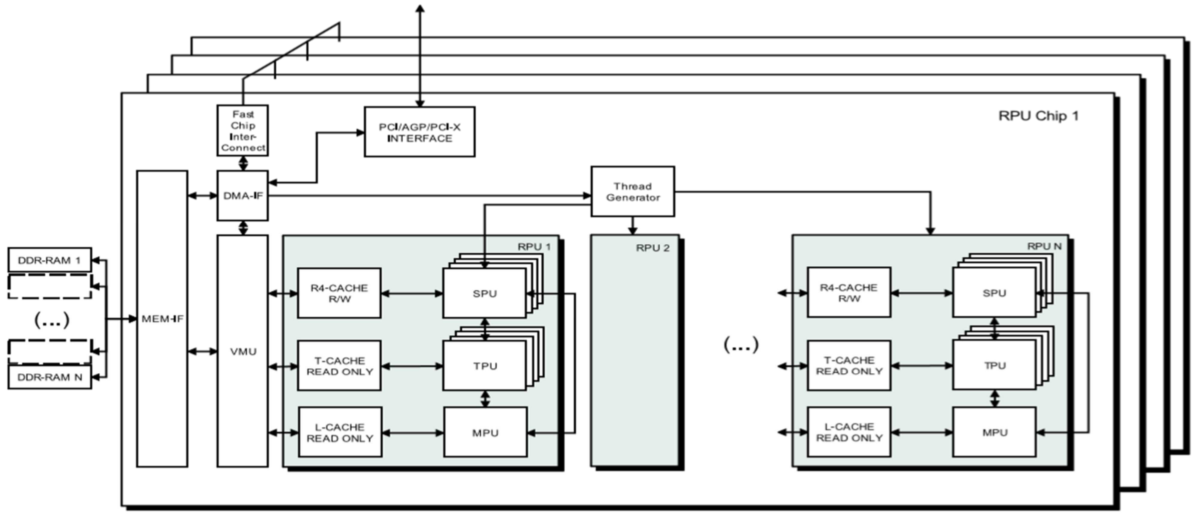

Fig. 3 Ray Processing Unit architecture [5]

This, on the other hand, offers advantage, when it is possible to trace more rays that is needed for example to improve the image quality in areas where needed.

Computational complexity is the reason why this technique is best suited for applications where time or computing power does not play the key role, and the image can be rendered slow. It is ideal for example in production of still images or visual effects and tricks for movies, and is not suited for real-time applications such as the video-games, or different kind of visualizations that are performed in real-time.

On the other hand, the algorithm of ray tracing is ideal for parallel processing since there is huge amount of pixels in the image, and each pixels value is independent from others and thus can be calculated in parallel. That is why there is an effort to design accelerators, from which many are hardware based, that can bring optimal design with enough computing power to make possible ray tracing in real-time. Opportunities for its use can be found in the entertainment industry, in computer games, as well as in the engineering industry, in CAD and other applications where it is necessary to achieve the very high degree of visual realism.

2. REAL TIME RAYTRACING

It is possible to divide ray tracing hardware acceleration architectures into two main groups. First group includes architectures that are designed from scratch for the ray tracing. Another group comprises of multi-core hardware architectures that are optimized for ray tracing. Special attention is paid to the architectures that are intended for the use in mobile devices.

The first implementation of the ray tracing technique in real-time was BRL-CAD solid modelling system, developed by Mike Muuss in 1986, which was the first known parallel network distributed ray tracing system, with ability to render several frames per second [2].

The TigerSHARK is the hardware accelerator of the ray tracing algorithm that was using digital signal processor (DSP) and was proposed in 1996 at the Princeton University [3].

The OpenRT Real Time Ray Tracing Project developed ray tracing software core used for ray tracing and OpenRT-API, similar to OpenGL.

The prototype of the ray tracing hardware based on the SaarCOR (Saarbrücken's Coherence Optimized Ray Tracer) chip was designed at the computer graphics laboratory at Saarland University in 2002.

More advanced processor which combines the flexibility of CPU with effectiveness of GPU in parallel computations was developed at Saarland University in 2005 as the Ray Processing Unit (RPU). It consists of several SPU (Shader Processing Unit), TPU (Traversal Processing Unit) and MPU (Mailboxed List Processing Unit), see Fig. 3.

High-performance ray tracing engine that allowed computer games to be rendered via ray tracing without intensive resource usage was introduced at the University of Saarland on March 2007 [4][5][6].

Copernicus is a ray tracing accelerator based on the multi-core tile architecture which comprises of 128 programmable computing cores [7].

Traversal and Intersection (T&I) architecture, which accelerates traversal and intersection operations of ray tracing algorithm, integrates three novel approaches: an ordered depth-first layout and a traversal architecture using this layout to reduce the required memory bandwidth, three-phase ray-triangle intersection architecture and latency hiding architecture, defined as the ray accumulation unit [8].

TRaX is the multi-processor accelerator that is using multiple instructions multiple data (MIMD) architecture.

Siliconarts designed the mobile MIMD real-time ray tracing hardware accelerator RayCore 2000 GPU IP in 2011 as the semiconductor intellectual property with the performance that allows computation of 300 million rays per second per core [9].

3. SOLLUTION AND RESULTS

In computer graphics, the basic ray tracing algorithm shoots for each pixel on the screen a ray from the camera into the scene. The ray is tested for intersection with each object in the scene. If the ray hits an object the colour of the pixel is set according to the object's surface properties. If the ray does not hit an object the colour of the pixel is set to the background colour. If there are multiple intersections the colour of the pixel is set to the colour of the object which is the closest to the camera. Because the number of bounces that a ray makes before it reaches the light source can be high, there should be set a limit for the number of bounces. The basic ray tracing algorithm in pseudocode:

```
for each pixel do
    compute viewing ray
    if (ray hits an object ) then
          Compute normal
          Evaluate lighting equation, set pixel to that color
    else
          set pixel color to background color
```

According to the research, the most system resource demanding part of the ray tracing algorithm are the ray - object intersection computations. In computer graphics each object's surface can be divided into elemental triangles. This is the reason why this part of the work will be focusing on accelerating the ray - triangle intersection computations.

The numbers x0, y0, z0 represent the coordinates of the camera, the numbers x1, y1, z1 represent the coordinates of the current pixel on the screen and the numbers a1, a2, a3, b1, b2, b3, c1, c2, c3 represent the coordinates of the points defining the triangle.

3.1. Parallelization of the ray - triangle intersection computations

The first step of the ray - triangle intersection computations is to compute the value t which represents the distance between the camera and the intersection of the ray and the plane in which the triangle lies. The second step is to compute the values β,γ and M. The following conditions need to be true for the intersection point to lie in the triangle:

1. $\beta + \gamma < M$
2. The signs of the values β and M must be the same.
3. The signs of the values γ and M must be the same.

If all conditions are true the coordinates of the intersection point are computed according to the value t. The whole process can be parallelized into 3 steps as following (Fig. 4):

1. The computation of the values M, t, β and γ (The computation of the value t is in the end part dependent of the value M. This fact can be ignored because its only one divide operation in the ending part of the computation).

2. The evaluation of the conditions for the intersection point.

3. Setting the error value to true if any of the conditions is false and the computation of the coordinates of the intersection point.

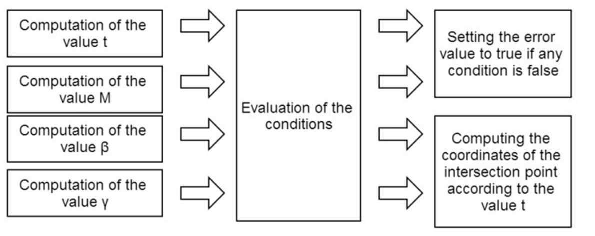
Computation of the value t
Computation of the value M
Computation of the value β
Computation of the value γ
Evaluation of the conditions
Setting the error value to true if any condition is false
Computing the coordinates of the intersection point according to the value t

Fig. 4 Parallelization of the ray - triangle intersection computation.

3.2. Hardware optimalization of the ray - triangle intersection computations

Removing redundant components is a form of acceleration. It makes more space available on the FPGA and makes it possible to implement the same module multiple times on the FPGA. A Virtex6 device XC6VLX75T FPGA was used for implementation. The hardware optimized computations of the values M and β are shown on the data flow diagram at Fig. 5.

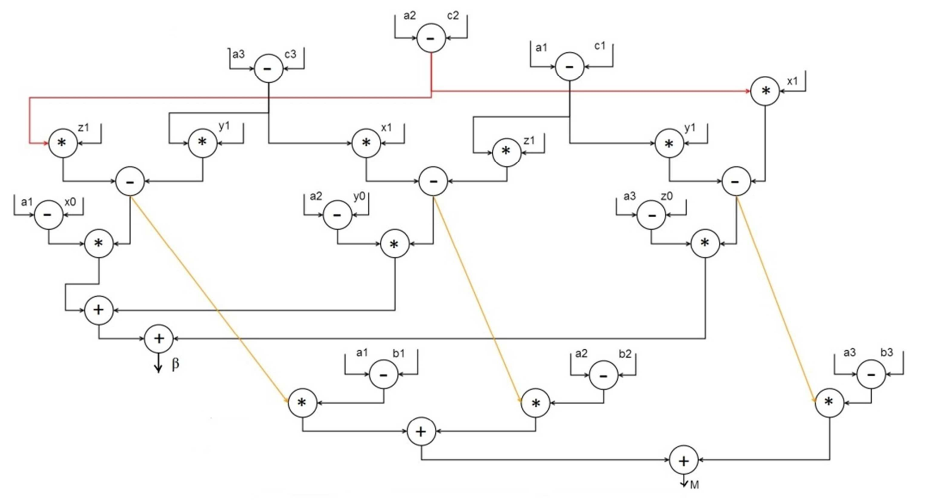

Fig. 5 Optimized data flow diagram that shows the computing of the values M and β

The data flow diagrams on Fig. 5. and Fig. 6. share the operations (a1 − b1),(a2 − b2),(a3 − b3),(a1 − c1),(a2 − c2),(a3 − c3),(a1 − x0),(a2 − y0),(a3 − z0) and can be merged. Greater clarity is the reason why they were split into two data flow diagrams. The hardware optimized computations of the values t and γ are shown on a data flow diagram at Fig. 6.

The number of components before the optimization was 104 and after the optimization it is 50. This represents a reduction of the number of components by 52%.

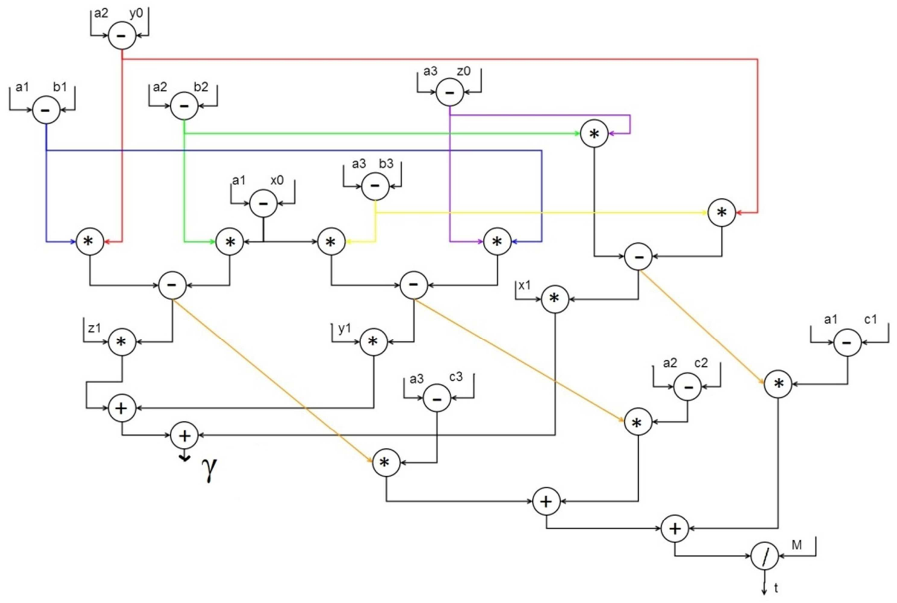

Fig. 6 Optimized data flow diagram that shows the computing of the values t and γ

The table below represents the comparison between the module before and after the optimization (IOB stands for Input Output Blocks and LUT stands for Look - Up Tables).

It shows that the optimized module can be implemented 5 times (5 * 19 = 95 %) on the FPGA. It also shows that 86% of the IOB are used. This means that the left 5% of the LUT should be used for a module that will regulate the input and output traffic for each ray - triangle intersection computing module.

Table 1 Module comparison table

	Used IOB	Total IOB	Usage in %	Used LUT	Total LUT	Usage in %
Before optimization	307	360	86	17164	46560	37
After optimization	307	360	86	8704	46560	19

4. CONCLUSIONS

The goal of this work was to design and implement a module for the acceleration of the computations in the ray tracing algorithm. The most resource demanding part of the algorithm are the ray - object intersection computations. This part of the algorithm was parallelized, optimized on the hardware level and implemented on the FPGA. By the acceleration of this part we achieved acceleration of the whole algorithm.

The module has some disadvantages. It needs a module for regulating the inputs and outputs because of the IOB overload. The interval of numbers with which the modules executing the basic operations can work is limited. These disadvantages will be eliminated in the for future research.

ACKNOWLEDGMENTS

This work was supported by the Slovak Research and Development Agency under the contract No. APVV-0008-10 and project KEGA 008TUKE-4/2013: Microlearning environment for education of information security specialists.

REFERENCES

[1] RADEMACHER, P.: Ray tracing - graphics for the masses, https://www.cs.unc.edu/~rademach/xroads-RT/RTarticle.html [cit. 2013-12-23].

[2] Proceedings of 4th Computer Graphics Workshop, Cambridge, MA, USA, October 1987. Usenix Association, 1987. pp 86–98.

[3] HUMPHREYS, G. – ANANIAN, C. S.: A Hardware Accelerated Ray-tracing Engine, Department of Computer Science, Princeton University, May 14, 1996.

[4] SCHMITTLER, J. – WALD, I. – SLUSALLEK, P.: SaarCOR – A Hardware Architekture for Ray Tracing, in Proceedings of EUROGRAPHICS Graphics Hardware 2002, Saarbrücken, Germany, September 1-2, 2002

[5] SCHMITTLER, J. – WOOP, S. – WAGNER, D. – PAUL, W. J. – SLUSALLEK, P.: Realtime Ray Tracing of Dynamic Scenes on an FPGA Chip, Graphics Hardware 2004, Computer Science, Saarland University, Germany, ACM 2004, p. 95-106.

[6] WOOP, S. – SCHMITTLER, J. – SLUSALLEK, P.: RPU: a programmable ray processing unit for realtime ray tracing, In: ACM Transactions on Graphics (TOG), ACM, 2005, p. 434-444.

[7] GOVINDARAJU, V. – DJEU, P. – SANKARALINGHAM, K. – VERNON, M. – MARK, W. R.: Toward a multicore architecture for real-time ray-tracing. In MICRO 41: Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture, 2008, p. 176–187.

[8] JAE-HO, N. – JEONG-SOO, P. – CHANMIN, P. – JIN-WOO, K. – YUN-HYE, J. – WOO-CNAN, P. – TACK-DON, H.: T&I engine: traversal and intersection engine for hardware accelerated ray tracing, ACM Transactions on Graphics (TOG) 01/2011; 30:160. DOI: 10.1145/2070781.2024194.

[9] JAE-HO, N. – HYUCK-JOO, K. – DONG-SEOK, K. – CHEOL-HO, J. – JINHONG, P. – TACK-DON, H. – DINESH, M. – WOO-CHAN, P. : RayCore: A ray-tracing hardware architecture for mobile devices, ACM Transactions on Graphics (TOG), Vol. 33, Publication date: August 2014.

Received April 23, 2014, accepted June 27, 2014

BIOGRAPHIES

Liberios Vokorokos (prof., Ing., PhD.) was born on 17. November 1966 in Greece. In 1991 he graduated (MSc.) with honours at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University in Košice. He defended his PhD. in the field of programming device and systems in 2000; his thesis title was "Diagnosis of compound systems using the Data Flow applications". He was appointed professor for Computers Science and Informatics in 2005. Since 1995 he is working as an educationist at the Department of Computers and Informatics. His scientific research focuses on parallel computers of the Data Flow type. He also investigates the questions related to the diagnostics of complex systems. He is a dean of the Faculty of Electrical Engineering and Informatics at the Technical University of Košice. His other professional interests include the membership on the Advisory Committee for Informatization at the faculty and Advisory Board for the Development and Informatization at Technical University of Košice.

Branislav Madoš (Ing., PhD.) was born on 20th May 1976 in Trebišov, Slovakia. In 2006 he graduated (MSc.) at the Department of Computers and Informatics at the Faculty of Electrical Engineering and Informatics of the Technical University of Košice. He defended his PhD. in the field of Computers and computer systems in 2009; his thesis title was "Specialized architecture of data flow computer". Since 2010 he is working as a professor assistant at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University in Košice. His scientific research is focused on the parallel computer architectures and architectures of computers with data flow model.

Viktor Ruska (Ing.) was born on 10th June 1990 in Košice, Slovakia. In 2014 he graduated (MSc.) at the Department of Computers and Informatics at the Faculty of Electrical Engineering and Informatics of the Technical University of Košice. His scientific research is focused on computer architectures and architectures of specialized computation accelerator architectures based on FPGA.