3D Res (2018) 9:7
https://doi.org/10.1007/s13319-018-0159-x

3DR EXPRESS

# Parallel Computer System for 3D Visualization Stereo on GPU

**Anas M. Al-Oraiqat · Sergii A. Zori**

Received: 21 November 2017 / Revised: 25 January 2018 / Accepted: 26 January 2018 / Published online: 3 February 2018

**Abstract** This paper proposes the organization of a parallel computer system based on Graphic Processors Unit (GPU) for 3D stereo image synthesis. The development is based on the modified ray tracing method developed by the authors for fast search of tracing rays intersections with scene objects. The system allows significant increase in the productivity for the 3D stereo synthesis of photorealistic quality. The generalized procedure of 3D stereo image synthesis on the Graphics Processing Unit/Graphics Processing Clusters (GPU/GPC) is proposed. The efficiency of the proposed solutions by GPU implementation is compared with single-threaded and multithreaded implementations on the CPU. The achieved average acceleration in multi-thread implementation on the test GPU and CPU is about 7.5 and 1.6 times, respectively. Studying the influence of choosing the size and configuration of the computational Compute Unified Device Archi-tecture (CUDA) network on the computational speed shows the importance of their correct selection. The obtained experimental estimations can be significantly improved by new GPUs with a large number of processing cores and multiprocessors, as well as optimized configuration of the computing CUDA network.

**Keywords** 3D visualization · Ray tracing · 3D stereo image · Synthesis · Graphics processor · Computer system

A. M. Al-Oraiqat (✉)
Department of Computer Sciences and Information, Taibah University, P.O. Box 2898, Medina, Kingdom of Saudi Arabia
e-mail: anas_oraiqat@hotmail.com

S. A. Zori
Department of Computer Sciences and Technologies, SHEE «Donetsk National Technical University», Donetska oblast, P.O. Box 85300, Pokrovsky, Ukraine
e-mail: sa.zori1968@gmail.com

## 1 Introduction

The organization of computer systems for realistic 3D spatial visualization provides a fundamentally new organization of the computational process, as compared to the standard 3D graphics pipeline. In the standard 3D graphics pipeline, the use of complex methods of synthesis and visualization (such as ray tracing, etc.) increases the realism of spatial 3D synthesis. In this regard, modern computer systems for image synthesis and visualization of the environment require a high-quality and effective application. Those methods combine realistic 3D methods with both traditional visualization mechanism and non-traditional methods of 3D visualization. It should be noted that, the methods and algorithms used in such systems provide high realism of synthesized images but with a great time complexity, graphics multiprocessors requirements and a compulsory modern hardware support for synthesis on multi-core CPUs. This confirms that the speed and realism of synthesis in 3D systems visualization is still a challenging and promising problem [2–4, 12].

3D spatial visualization assumes such a computer transformation of a scene model (spatial 3D synthesis, rendering). It creates a fully 3D visual 3D image in a specific 3D display device (3D visualization device, 3D monitor). The 3D image (spatial visual image) is then displayed in some way in the 3D display space and the person perceives it (possibly using additional interfaces) in bulk.

Currently, the vast majority of 3D synthesis and 3D imaging/display information systems use the method of stereoscopic 3D visualization that gives a reasonable compromise between quality, speed and cost of 3D visualization [2–4, 12, 16]. At the same time, the most applicable implementation approach, from the processor core cost point of view for implementing parallel 3D display information/hardware support systems, is the implementation on the bases of Graphical Processing Units (GPUs).

This paper investigates organization of a 3D-stereo visualization parallel computing system based on the modified ray tracing method developed by the authors [2, 3, 14, 16, 17] for fast search of tracing rays intersections with scene objects and the Compute Unified Device Architecture (CUDA), i.e., parallel computing platform and programming model developed for general computing on GPUs.

## 2 Organization of the 3D-Stereo Visualization System Based on the Parallel Graphic Processor

From the practical view point implementation of the 3D stereo visualization system on the GPU, we recall that the synthesis of 3D stereo images of improved quality is proposed as a double ray tracing of independent stereo pair images [1, 2, 15]:

- Parallel independent implementation of the synthesis of frames “left”–”right” on the GPU;
- Parallel “intraframe” implementation of the ray tracing method on the GPU resources allocated for each channel.

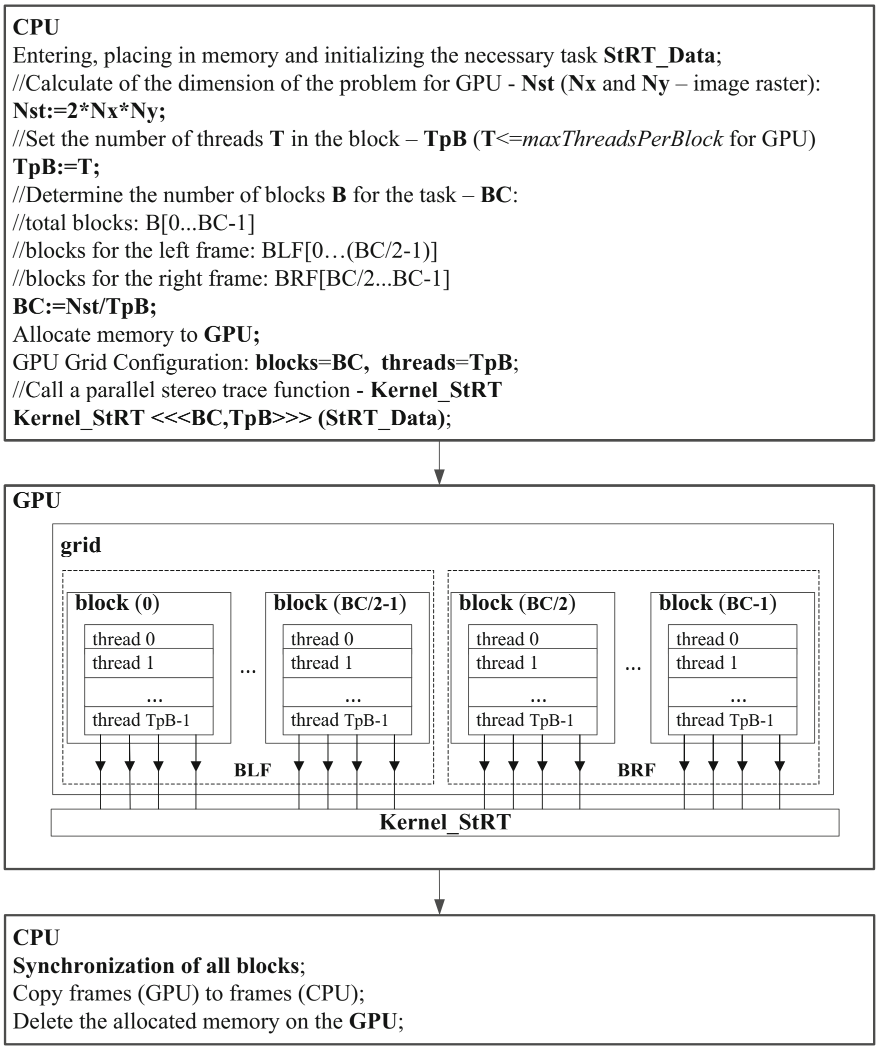
Further (if necessary, taking into account the specifics of the 3D-display device), the process of post-processing of the received stereo pair images (conversion of frames into one of the standard output formats for 3D-visualization devices) can also be performed on the GPU. In this regard, the general organization of the computational process of the 3D stereo imaging procedure by ray tracing on the GPU architecture (or GPC graphics cluster) is illustrated in Fig. 1.

CUDA blocks perform a stereo trace procedure (StRT) in parallel for the left and right image frames. All images which are divided into the same number of raster cluster blocks (BLF–BRF sub-arrays) in which SIMT ray tracing is also performed in parallel.

It should be noted that the topological organization of the network and the blocks is not unique and can be the subject of optimization for a particular GPU architecture. To preconfigure the configuration, the CUDA GPU Occupancy Calculator may be used [7]. The mapping of the 3D synthesis procedure to the GPC can be represented similarly, for example by placing the BLF and BRF blocks in different clusters.

The main basic procedure for 3D stereo imaging is the ray tracing [5, 6, 13]. When mapping the architecture of ray tracing to the GPU architecture, the main idea is to divide the ray tracing algorithm into several kernel cores, each one performs only a single specific task. Also, it is necessary to take into account the peculiarities of the specific microarchitecture of the GPU, as well as the features of the ray tracing algorithm itself. The parallel implementation of the ray tracing algorithms is complicated due to a large number of factors, such as possible completion of the execution threads at different times (in the event that a part of the rays is absorbed or not crossed surface of the scene), incoherence in the calculation of secondary rays and leads to a branching of the streams (if different rays intersect surfaces that have different properties and the model of light scattering). It should also be taken into account that, in order to search for the intersection of the beam with the surfaces of the scene and calculate the illumination at the founded point, the execution flows require regular access to the description of the scene. So, an efficient organization of the accelerated structure, the data storage location in memory as well as the access to it is also an important factor.

Notice that, in the urgency view of the parallel organization problem of ray tracing, and primarily on the GPU, many practical solutions have been proposed



```
CPU
Entering, placing in memory and initializing the necessary task StRT_Data;
//Calculate of the dimension of the problem for GPU - Nst (Nx and Ny – image raster):
Nst:=2*Nx*Ny;
//Set the number of threads T in the block – TpB (T<=maxThreadsPerBlock for GPU)
TpB:=T;
//Determine the number of blocks B for the task – BC:
//total blocks: B[0...BC-1]
//blocks for the left frame: BLF[0…(BC/2-1)]
//blocks for the right frame: BRF[BC/2...BC-1]
BC:=Nst/TpB;
Allocate memory to GPU;
GPU Grid Configuration: blocks=BC, threads=TpB;
//Call a parallel stereo trace function - Kernel_StRT
Kernel_StRT <<<BC,TpB>>> (StRT_Data);
```

```
CPU
Synchronization of all blocks;
Copy frames (GPU) to frames (CPU);
Delete the allocated memory on the GPU;
```

**Fig. 1** Algorithm of the display 3D stereo synthesis on a GPU/GPC

in the literature. The most successful in the sense of universality and qualitative indicators is the global concept where proposed in [5]. However, the authors in [5] pointed out that one of the most laborious stages of visualization is the search for the nearest intersection of the ray with the objects of the scene and use accelerated structures to increase the productivity.

In connection with the foregoing, it seems inexpedient to consider and develop approaches for GPU architecture implementation to general ray tracing technique (it is sufficient to use the universal approach, or more narrowly specialized, with the adopted simplifications), but rather focusing on the acceleration of the method steps bottlenecks by displaying them on the GPU architecture.

The general idea of searching for ray intersection and Axis-Aligned Bounding Box (AABB) is schematically shown in Fig. 2 [3, 14, 16]. The general mechanism for mapping the intersection algorithm proposed in [1, 3, 15] to the GPU architecture is shown in Fig. 3.

## 3 Proposed Modified Ray Tracing Method

In order to accelerate the detection of intersection during ray tracing, a modified ray crossing detection algorithm with accelerating technique based on the two-level hierarchy of bounding volumes and AABB for accelerating multithreaded calculations is proposed. In addition, as a modification of the "classical"

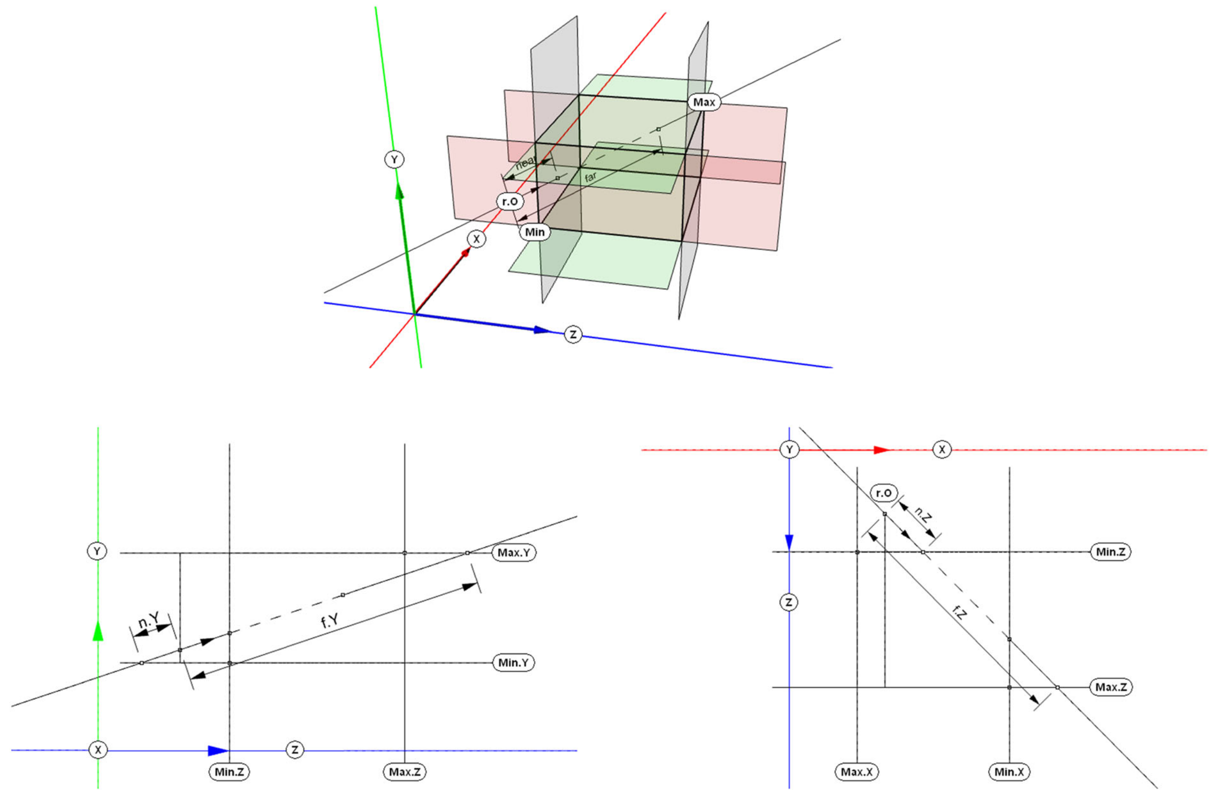


**Fig. 2** Principle of operation of the ray intersection search algorithm with AABB

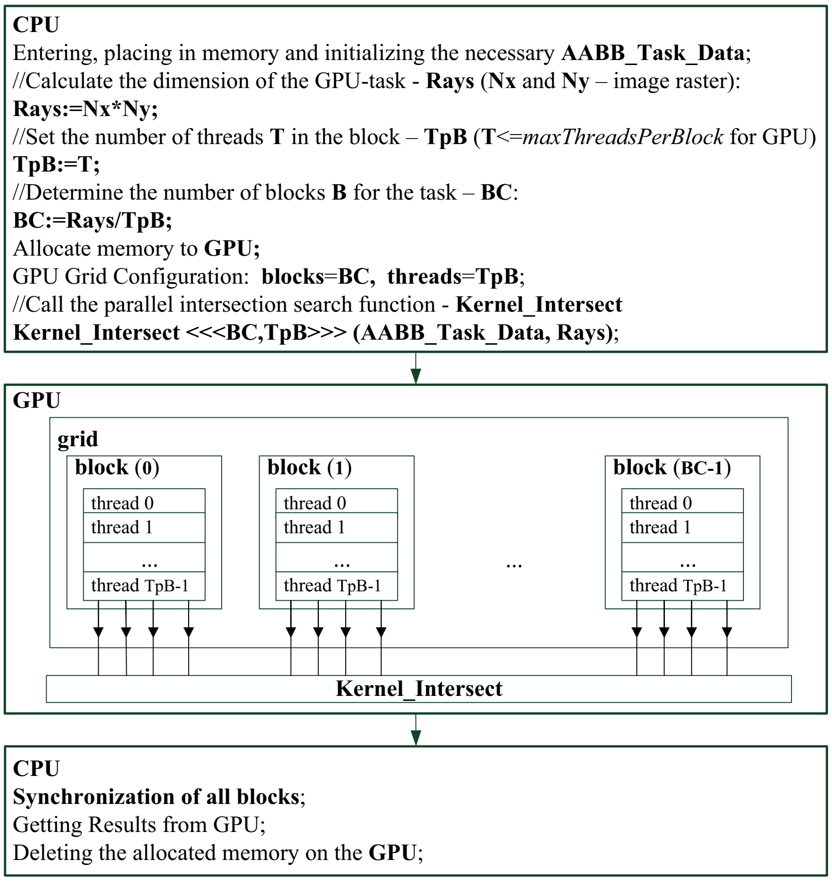
ray intersection search algorithm [3, 4, 6, 14, 16] to improve the efficiency of its implementation on the GPU and multithreaded computing devices, the calculation order in the base algorithm is revised and branches are eliminated when searching for intersections with AABB (with all intersection predicates transferred to the end of the procedure) as illustrated in Fig. 4.

The modified scheme effectiveness was investigated in a single-threaded uniprocessor implementation. The test configuration platform is shown in Table 1. As detailed later, comparing the results of proposed modified method to the methods of [1, 3, 14–16], the proposed modification of the search accelerating technique for beam intersections with scene objects on single-threaded implementation, although it gives a positive effect (about 3–7%), it still does not work fast enough for complex scenes with high resolution. However, the essence of the modifications made allows more significant acceleration in multi-threaded implementations on multi-core CPUs and GPU.

Figures 5 and 6 give the experimental results for the speed evaluation of the proposed modified algorithm. We study the multithreaded implementation on the CPU for the test stand in test scenes 2 and 3 (given in Table 2) as well as various areas of the scene projection onto the picture plane (mostly non-overlapping and “close” objects in relation to the observer—”a large area of the projection”, and vice versa—”small area”).

Figure 7 shows the average achieved acceleration of multithreaded implementations in comparison with a single-threaded CPU on a test bench. Let us consider the experimental results of the proposed modified algorithm speed estimation for the intersection search algorithm when it is implemented on the GPU. The algorithm modifications are developed in accordance with recommendations for optimization. Among of these recommendations, the most important is the reduction of the branching within each individual block and warp.

It should be noted that configuring the GPU grid is in general an ambiguous task since the solution of this



```
CPU
Entering, placing in memory and initializing the necessary AABB_Task_Data;
//Calculate the dimension of the GPU-task - Rays (Nx and Ny – image raster):
Rays:=Nx*Ny;
//Set the number of threads T in the block – TpB (T<=maxThreadsPerBlock for GPU)
TpB:=T;
//Determine the number of blocks B for the task – BC:
BC:=Rays/TpB;
Allocate memory to GPU;
GPU Grid Configuration:  blocks=BC,  threads=TpB;
//Call the parallel intersection search function - Kernel_Intersect
Kernel_Intersect <<<BC,TpB>>> (AABB_Task_Data, Rays);
```

```
CPU
Synchronization of all blocks;
Getting Results from GPU;
Deleting the allocated memory on the GPU;
```

**Fig. 3** General mechanism for computing the intersections on the GPU

```
bool Intersect(AABB &bbox, Ray3 &ray) {
   Vector3 nearPlaneDistances = (bbox.Min - ray.Origin) * ray.InversedDirection;
   Vector3 farPlaneDistances = (bbox.Max - ray.Origin) * ray.InversedDirection;
   float nearDistance, farDistance;
   nearDistance = min(nearPlaneDistances.X, farPlaneDistances.X);
   nearDistance = max(nearDistance, min(nearPlaneDistances.Y, farPlaneDistances.Y));
   nearDistance = max(nearDistance, min(nearPlaneDistances.Z, farPlaneDistances.Z));
   farDistance = max(nearPlaneDistances.X, farPlaneDistances.X);
   farDistance = min(farDistance, max(nearPlaneDistances.Y, farPlaneDistances.Y));
   farDistance = min(farDistance, max(nearPlaneDistances.Z, farPlaneDistances.Z));
   return farDistance >= nearDistance && farDistance >= 0;
}
float max(float a, float b)
 {    return (abs(a-b) + a + b) / 2); }

float min(float a, float b)
{ …. }
```

**Fig. 4** Modified algorithm for checking the intersection between the AABB and the ray tracing

task may directly affect the performance of the program executed by the GPU. In order, to determine the theoretical estimation of the multiprocessor loading degree and the configuration of the GPU computing network, the CUDA SDK Occupancy Calculator software has been used [7]. According to the theoretical estimate, the maximum load of the multiprocessor (100%) is possible when less than 10 registers per string are used with a block sizes of 192, 256 or 384 threads. The GPU multiprocessor can be loaded in

**Table 1** Tested configuration

| OS | CPU | RAM | GPU |
|---|---|---|---|
| Windows 8 Ultimate, x64 | Intel Core 2 Quad Q9100 @ 2.26 GHz | 3 GB DDR3 | NVIDIA GeForce G110 M (16 CUDA cores) |

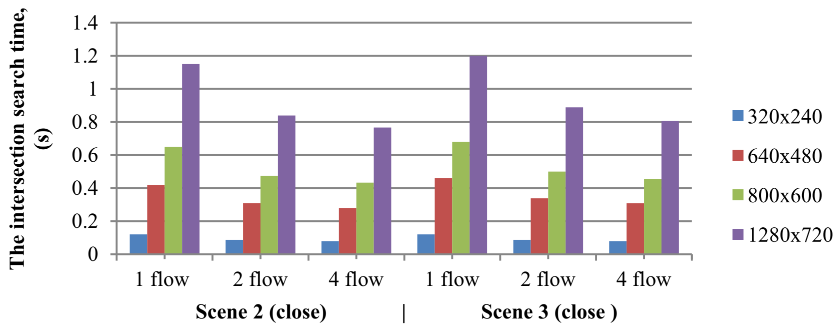


**Fig. 5** Intersection search time: large area of the scene projection on the display with CPU

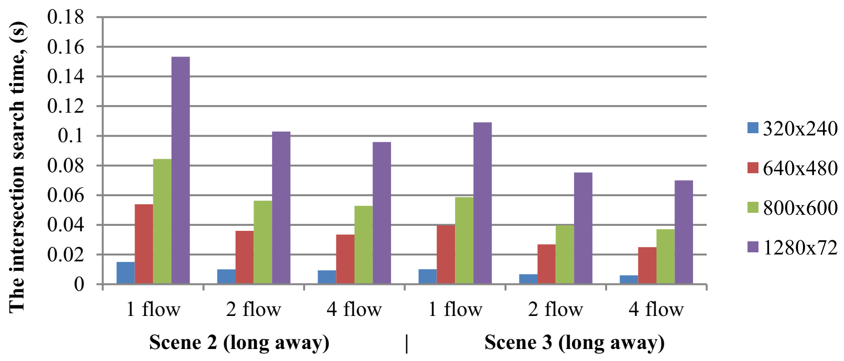


**Fig. 6** Intersection search time: small area of the scene projection on the display with CPU

**Table 2** Characteristics of test scenes

| Characteristic | Scene 1 | Scene 2 | Scene 3 |
|---|---|---|---|
| Form of objects | Sphere | Pyramid | Icosahedron |
| Number of objects | 200 | 210 | 70 |
| Amount triangles in the form | 0 | 6 | 20 |

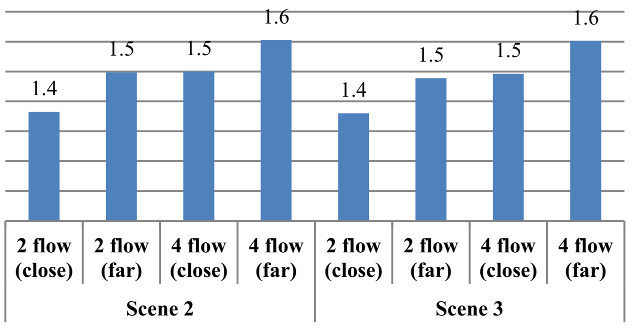


**Fig. 7** Average achieved acceleration compared to single-threaded implementation with CPU

**Table 3** GPU network configuration characteristics (100% congestion)

| | | | |
|---|---|---|---|
| Threads per block | 192 | 256 | 384 |
| Registers per thread | 10 | 10 | 10 |
| Active threads per multiprocessor | 768 | 768 | 768 |
| Active warps per multiprocessor | 24 | 24 | 24 |
| Active thread blocks per multiprocessor | 4 | 3 | 2 |
| Occupancy of each multiprocessor | 100% | 100% | 100% |

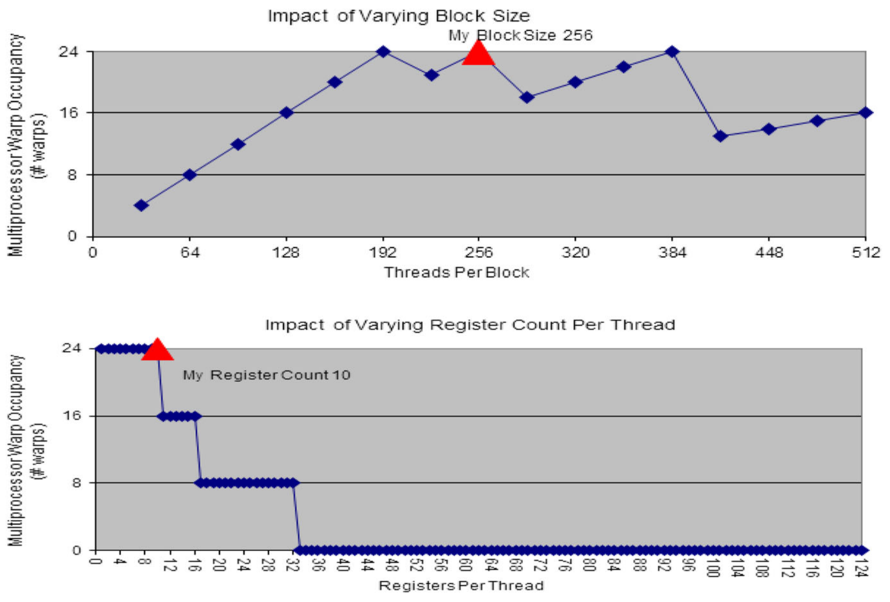


**Fig. 8** Impact of varying the block size and register count per thread on warps for a multiprocessor load of 100%

**Table 4** GPU network configuration characteristics (67% congestion)

| | | | |
|---|---|---|---|
| Threads per block | 128 | 256 | 512 |
| Registers per thread | 16 | 16 | 16 |
| Active threads per multiprocessor | 512 | 512 | 512 |
| Active warps per multiprocessor | 16 | 16 | 6 |
| Active thread blocks per multiprocessor | 4 | 2 | 1 |
| Occupancy of each multiprocessor | 67% | 67% | 67% |

67% by using 16 registers per string and a block sizes of 192, 256 or 512 threads. The greater number of dedicated registers per string leads to a decrease in the load of the GPU multiprocessor to 50% when the registers number is between 17 and 20. However, the lower one is about 33% when the registers number is between 21 and 32 (as shown in Table 3 and Fig. 8 as well as Table 4 and Fig. 9).

The experimental results performed on these two configurations show that, with a theoretical multiprocessor load of 100%, the execution speed of the entire algorithm is less than the case of the GPU network configurations with a 67% load (it was chosen for further experiments). One of the reasons is the intensive use of the global memory with a small number of allocated registers. However, it should be noticed that, this doesn't mean that its speed is optimal. Experimental studies on the optimization of the computational process in the CUDA network (as used for the GPU experiment, and on other more powerful GPU architectures) have not been carried out yet. Therefore, the obtained experimental estimates can be further improved when implemented on new GPUs with a large number of processing cores and multiprocessors with an optimization of the configuration of the computing CUDA network.

To perform and accelerate the formation of a standard stereo frame (anaglyph conversion of

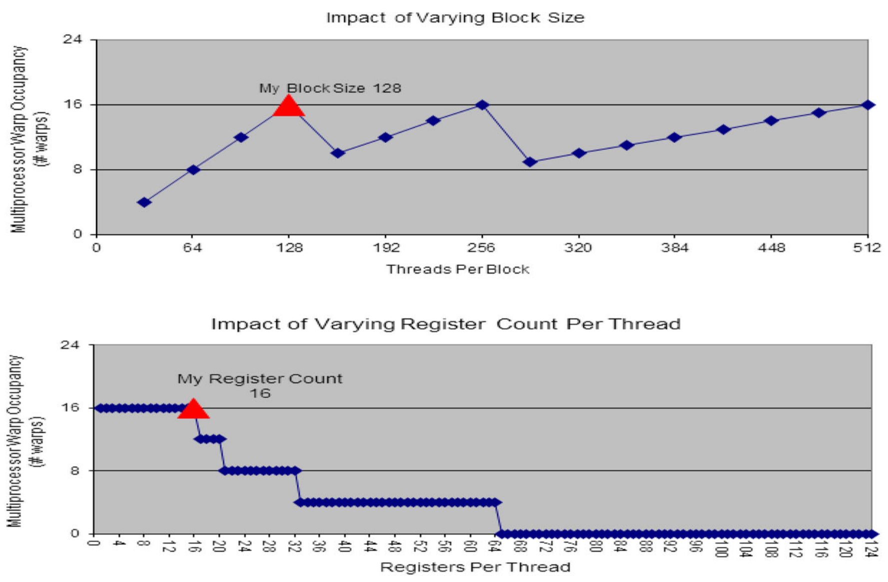


**Fig. 9** Impact of varying the block size and register count per thread on warps for a multiprocessor load of 67%

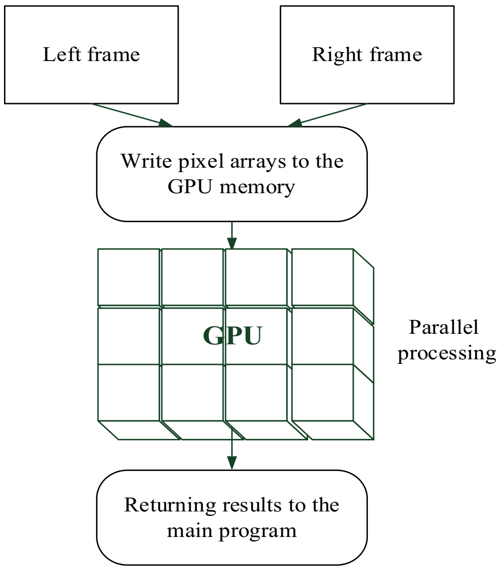

Left frame
Right frame
Write pixel arrays to the GPU memory
GPU
Parallel processing
Returning results to the main program

**Fig. 10** Schematic diagram of the anaglyph transformation module

separate frames of a pair into a stereo frame) and actual visualization, they have been implemented on the GPU.

At the first stage, the obtained separate stereo pair frames are converted into a combined anaglyph image. To do this, we need to do the following (as illustrated in Fig. 10):

- Generate an initial data in the GPU memory.
- Perform post processing calculations on the GPU.
- Return results to the main program of the visualization node.

The computing scheme is shown in Fig. 11. The computations on the GPU occur in parallel with the use of all available graphics card capacities. Parallel processing of byte arrays of synthesized left and right images is performed as well as the process of merging two arrays into one by their anaglyph transformation. After that, the result of the last operation is stored in the original byte array of the left image and returned to the main program of the module.

An experimental study of the created software module for anaglyph transformation has been carried out. To do this study, test images (left and right angles) of the scene rendering (a light source and 3 spheres) are computed with 3D Max application software. The obtained result, shown in Fig. 12, confirms the performance of the program module.

## 4 Experimental Results

The experimental results for the speed increasing estimate of the modified algorithm on the test system, as compared to the base version implemented on the GPU architecture [1, 14, 15], are shown in Figs. 13 and 14. The experiment and estimates are given for the case when the area of the scene projection on the

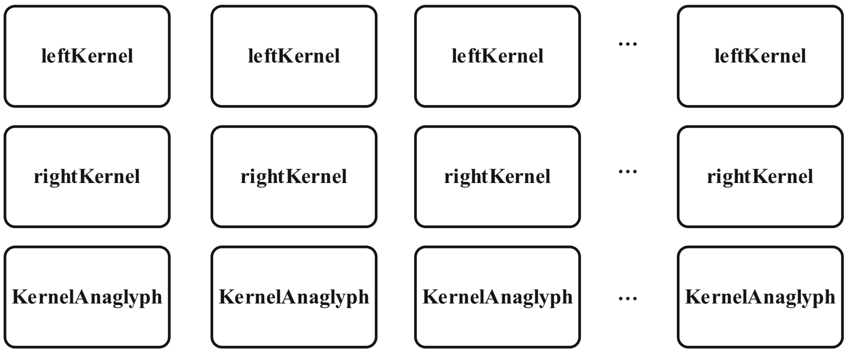


**Fig. 11** Anaglyph conversion to GPU

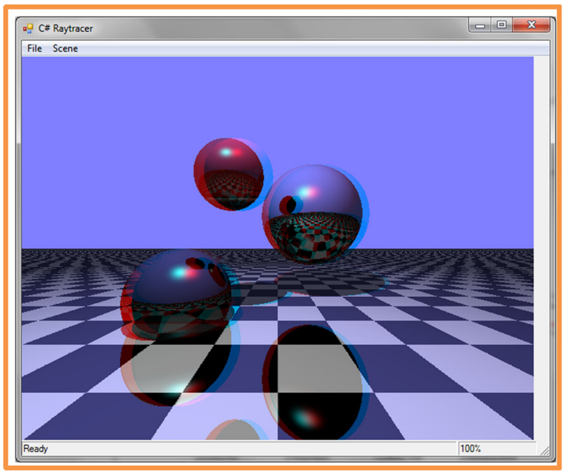


**Fig. 12** The result of the GPU–anaglyph transformation

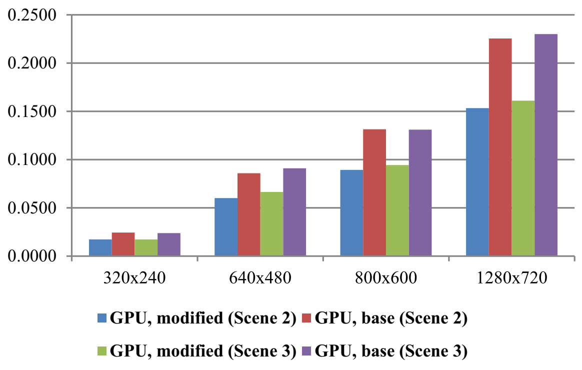


**Fig. 13** Intersection search time (modified-base implementation), GPU

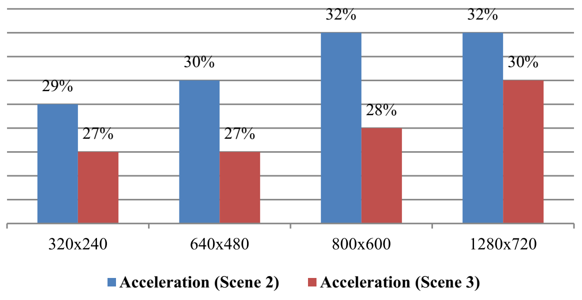


**Fig. 14** Efficiency of modification, GPU

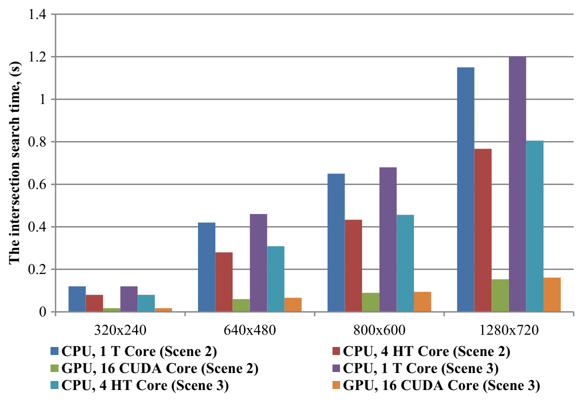


**Fig. 15** Intersection search time (small area of the scene projection on the display)

picture plane is large, since previous experiments and performance measurements have shown that this situation is the most computationally complex.

It can be seen that the modified algorithm gives a performance gain in relation to the basic version starting from 27 to 32% when it is implemented on the GPU of the test bench (single-threaded CPU implementation showed a performance gain of 3–7% [3], while the optimized implementation on the specific architecture of the GPU is not conducted.

The results of experimental studies of the implementation time on the CPU/GPU test system and the achieved acceleration of the proposed modification of the intersection search algorithm are shown in

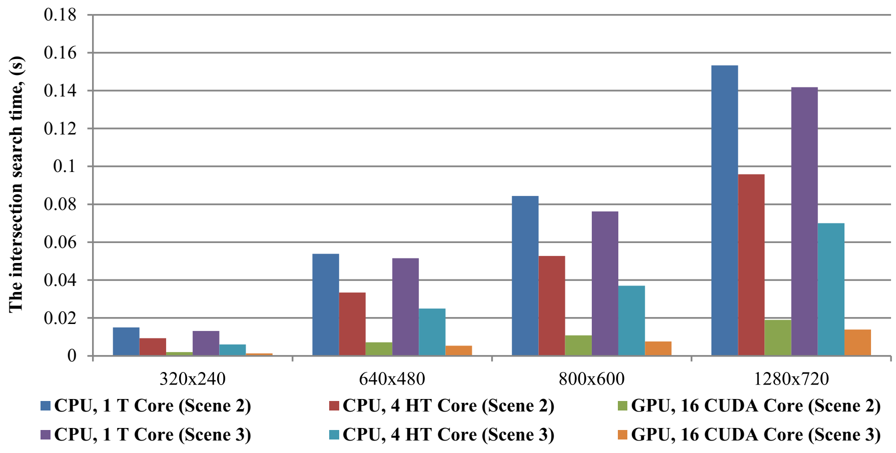


**Fig. 16** Intersection search time (large area of the scene projection on the display)

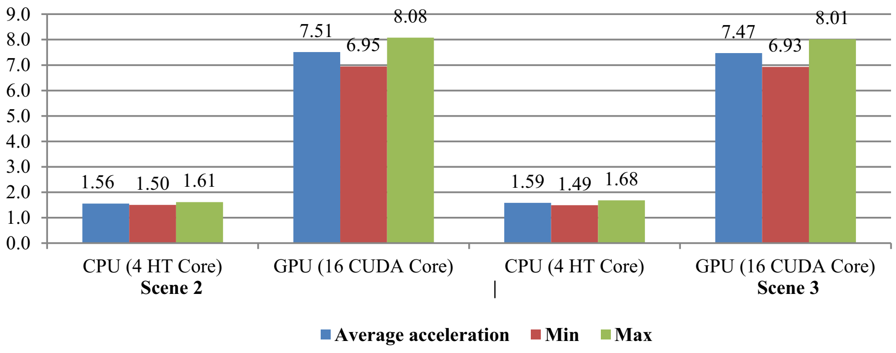


**Fig. 17** The achieved acceleration compared to the CPU in 1T Core Mode

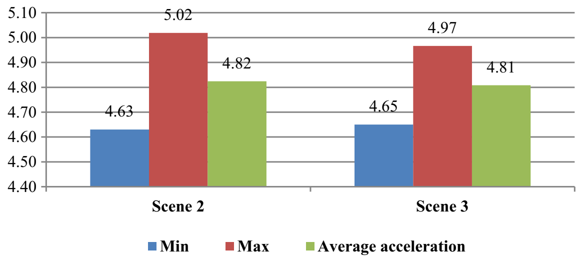


**Fig. 18** Acceleration of the GPU (16 cores) in relation to the CPU (4 cores)

Figs. 15, 16, 17 and 18. Figures 17 and 18 summarize the performances evaluation of the software implementation of the modified beam intersection search algorithm with AABB and allow a comparison of the time scores that were obtained when it was executed on four threads on a CPU (4 cores) and on a test GPU (16 CUDA cores).

The implementation on the test GPU allowed obtaining an average acceleration of 75 times to the variant without the use of Axis Aligned Bounding Box (AABB) techniques. Once again, it should be noted that the obtained experimental estimates can be significantly improved when implemented on new GPUs with a large number of processing cores and multiprocessors. Also, it can optimize the computing CUDA network configuration.

An experimental prototype of the 3D-stereo visualization system by the ray tracing method has been carried out using the developed algorithm and architecture means successfully solving the problem of 3D stereo images synthesizing.

To perform a comparative evaluation of the visual realism of the obtained experimental images, it is necessary to generate reference images of test scenes by ray tracing in one of the software packages using physically correct ray tracing renderers. The developed prototype of the 3D stereoscopic synthesis software system uses the ray tracing method based on the Whitted algorithm [13]. It does not generate some highly realistic global lighting effects realized by powerful physically correct renderers, such as FryRender, Indigo Renderer, Kerkythea Rendering System, Maxwell Render and so on. The main task of the development is to test the efficiency of the proposed algorithmic tools on the level of proposed architectural implementation of the GPU synthesis for 3D stereo images by ray tracing and on the level of the developed modification for the search technology accelerating for ray intersections with AABB.

In this regard, to determine the comparative evaluation of the obtained images visual realism, the generation of "reference" images of the test scene (separately for the left and right frames) and the method for determining the normalized mean square error of the NMSE of the visual difference between the reference and experimental images [10] by the mental ray rendering are used by ray tracing in the 3Ds Max package [11]. The render settings correspond basically to the settings of the software implementation, the developed prototype of the software system. Standard point and area light sources are used for realization of caustics rectilinear trace taking into account the refractive index. The final collection method is implemented using the Final Gathering module.

According to this technique, the visual difference between the two images can be estimated by the normalized mean-square error (NMSE) that can be calculated as follows:

$$NMSE = \frac{\sum_i \left(R(i)_1 - R(i)_2\right)^2 + \left(G(i)_1 - G(i)_2\right)^2 + \left(B(i)_1 - B(i)_2\right)^2}{\sum_i \left(R(i)_1\right)^2 + \left(G(i)_1\right)^2 + \left(B(i)_1\right)^2} \quad (1)$$

where $i$ denotes the pixels number in the image and $(R_1(i), G_1(i), B_1(i))$, $(R_2(i), G_2(i), B_2(i))$ are the color intensities of the red, green and blue components in the $i$th pixel in both formed images (left and right).

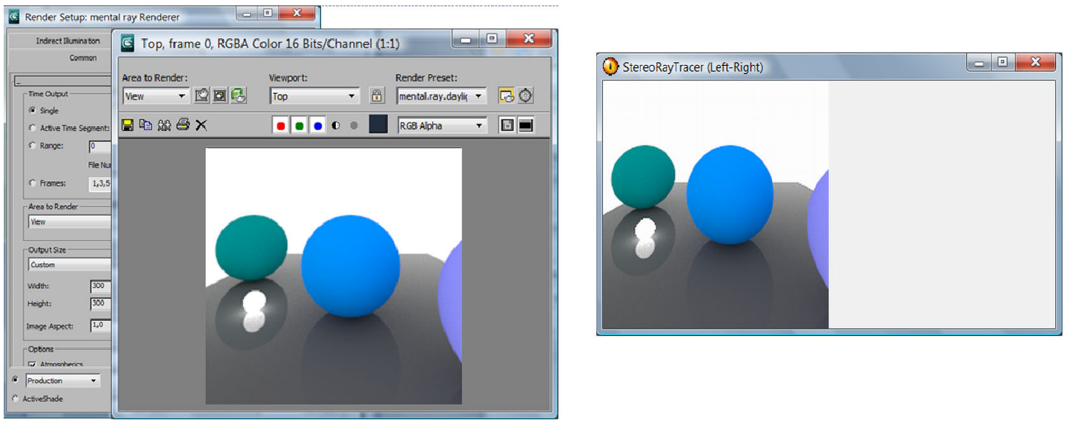
In computer graphics, when such testing images estimates are used [11]: if the NMSE value is not more than 0.0001, then the visual images do not differ from each other; if the NMSE is in the range [0.0001–0.00025], then the two images are slightly different; if the NMSE is in the range [0.00025–0.001], then the images have visually noticeable differences; if the NMSE is greater than 0.001, then the two images are significantly different from each other.

Comparison of this criterion of the reference and experimental images generated by the system (given in Fig. 19) has showed a NMSE value of 0.00021. This means that the images have insignificant visual differences and are explained by the impossibility of complete identity of the render settings in the reference and experimental systems with the ray tracing synthesis (closed, non-disabling rendering optimization and the implementation of the applied shaders in 3Ds Max, the built-in mechanism for generating "soft" area shadows and anti-aliasing in mental ray, etc.).

Examples of some images formed by the system are shown in Fig. 20.

A system operation study showed that:

1. The developed prototype program system 3D-stereo visualization using the proposed algorithmic and architectural means successfully solves the problem of synthesis of 3D-stereo images.
2. Synthesized image views have minor visual differences as compared to the standard and calculated 3D's Max rendering engine, related to the impossibility of a complete identity of the render settings of the reference and experimental systems.
3. About 70% of the time is spent on solving the computational problem while a considerable part of the time (up to 30%) is spent on transferring data between the central processor and the graphics card for computation and performing the visualization process.
4. With the increase of the computational complexity of the scene, the time for processing increases as well and the dependency is almost linear.
5. When using a single video card and GPGPU computations, the processes of scene computation, post-processing and visualization that actually occur sequentially, ultimately slow down the whole process.
6. The study of the influence of the choice of the size and the configuration of the computational CUDA network on the computational speed shows the importance of their correct assignment.


Render Setup: mental ray Renderer
Top, frame 0, RGBA Color 16 Bits/Channel (1:1)
StereoRayTracer (Left-Right)

Area to Render: View
Viewport: Top
Render Preset: mental.ray.daylig
RGB Alpha
Output Size: Custom
Width: 300
Height: 300
Image Aspect: 1,0
Production

Fig. 19 Reference and generated image by the software system

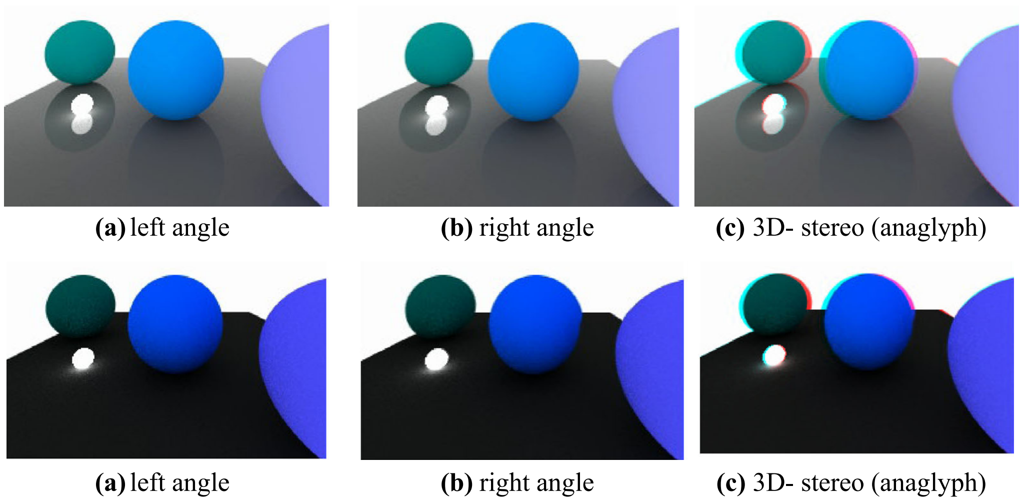


Fig. 20 Synthesized 3D stereo images of test scenes

## 5 Conclusions

The proposed architecture of a parallel computer system for realistic 3D stereo visualization of 3D scenes is accelerated by the method of ray tracing on the GPU and a generalized procedure for 3D stereo synthesis is displayed on the GPU/GPC.

The efficiency of the proposed solutions under the GPU implementation is compared to the single-threaded and multithreaded implementations on the CPU test stands. The results of the studies mainly showed the following:

The modified algorithm gives a performance gain in relation to the base version from 27 to 32% when implemented on the GPU test bench (single-threaded CPU implementation showed a performance gain of 3–11%), while optimized implementation on a specific architecture used by the GPU was not carried out. The generalized results of software performance evaluations showed that the overall average achieved acceleration for multi-threaded implementation on test GPUs and CPUs is about 7.5 and 1.6 times, respectively. The developed prototype program system 3D-stereo visualization proposed algorithm and architecture means successfully solves the problem of 3D stereo images synthesis. In this case, the synthesized images have an insignificant visual differences in comparison to the standard and the calculated 3Ds Max raisers (NMSE = 0.00021). It is probably due to the impossibility of obtaining complete identity of the render settings of the reference and experimental systems. When using a single video card and GPGPU calculations, the processes of scene calculation, post-processing and visualization actually occur sequentially. This behavior ultimately slows down the whole process. It is noted that the obtained experimental estimates can be significantly improved when new GPU architectures with a large number of processing cores and multiprocessors are implemented, as well as optimizing the configuration of the computing CUDA network. Nonetheless the study of the influence of the choosing the size and configuration of the computational CUDA network on the computational speed shows the importance of their correct assignment [1, 8, 9, 15].

## References

1. Al-Oraiqat, A., & Zori, S. (2016). 3D-visualization by raytracing image synthesis on GPU. *International Journal of Engineering Science and Technology (IJEST), 8*(6), 97–104.
2. Al-Oraiqat, A., Bashkov, E., & Zori, S. (2018). *Spatial visualization via real time 3D volumetric display technologies* (p. 120). Saarbrücken: LAP LAMBERT Academic Publishing.
3. Bashkov, E., & Zori, S. (2016). Systems of spatial visualization of environment. *Science Bulletin of Donetsk National Technical University, Krasnoarmiysk, 1*(1), 20–45.
4. Blundell, B., & Schwarz, A. (2002). The classification of volumetric display systems, characteristics and predictability of the image space. *IEEE Transactions on Visualization and Computer Graphics, 8*(1), 66–75.
5. Bogolepov, D., Ulyanov, D., Sopin, D., & Turlapov, V. (2013). Optimization of the bidirectional path trace method for modeling the optical experiment on a graphics processor. *Scientific Visualization, 2*(5), 1–15.
6. Cook, R., Porter, T., & Carpenter, L. (1984). Distributed ray tracing. *ACM SIGGRAPH Computer Graphics*, *18*, 137–145. https://dl.acm.org/citation.cfm?id=808590.
7. CUDA. (2007). Occupancy Calculator Helps pick optimal thread block size. https://devtalk.nvidia.com/default/topic/368105/cuda–occupancy–calculator–helps–pick–optimal–thread–block–size/. Accessed Jul 11, 2017.
8. Hong, G.-S., Hoe, W., Kim, B.-G., Beak, J.-W., & Kwon, K.-K. (2016). Stereo matching performance analysis of cost functions on the graphic processing unit (GPU) for pervasive computing. *Journal of Engineering and Applied Sciences, 11*(7), 1480–1487.
9. Hong, G.-S., & Kim, B.-G. (2017). Novel local stereo matching technique based on weighted guided image filtering (WGIF). *Displays (Elsevier), 49,* 80–87. https://doi.org/10.1016/j.displa.2017.07.006.
10. Meyer, L., & Klassen, R. (1998). A comparison of two image quality models. In *Human Vision and Electronic Imaging III* (Vol. 3299, pp. 98–109). SPIE. https://doi.org/10.1117/12.320101.
11. Murdock, K. L. (2010). *3Ds Max 2010 Bible* (p. 1312). Mission: SDC Publications. https://dl.acm.org/citation.cfm?id=1795691.
12. Samarin, A. (2005). Modern three-dimensional image display technologies. *Modern Electronics, 2,* 2–7.
13. Shirley, P., & Morley, R. (2003). *Realistic ray tracing* (Vol. 235). Boca Raton: CRC Press.
14. Zori, S. A. (2015). Volumetric visualization by ray tracing algorithm with two-level hierarchy of limiting volumes and AABB. *Bulletin of DonNTU, Donetsk, 2*(21), 5–10.
15. Zori, S. A. (2016). GPU-implementation of parallel computing system 3D stereo imaging using the ray tracing method. *Information Processing Systems: Collected Works, Kharkiv, 6*(143), 201–204.
16. Zori, S., & Porfirov, P. (2015). Productivity increasing of realistic ray tracing stereo-image synthesis. *Journal of Qafqaz University, Mathematics and Computer Science, 3*(1), 30–38.
17. Zori, S., Zaporozhchenko, I., & Grigorev, M. (2013). Analysis of ways to reduce the computational complexity of ray tracing algorithm and methods of its parallel implementation. *Digital Signal and Image Processing, 4*, 352–357.