British Journal of Mathematics & Computer Science
11(3): 1-11, 2015, Article no.BJMCS.19879
ISSN: 2231-0851

SCIENCEDOMAIN *international*
www.sciencedomain.org

Enhanced Ray Tracing Algorithm for Depth Image Generation

Hanan Ahmed[1*], Howida A. Shedeed[1] and Doaa Hegazy[1]

[1]Department of Scientific Computing, Faculty of Computer and Information Systems 6, Ain Shams University, 11566, Cairo, Egypt.

Article Information

DOI: 10.9734/BJMCS/2015/19879
Editor(s):
(1) Dariusz Jacek Jakóbczak, Chair of Computer Science and Management in This Department, Technical University of Koszalin, Poland.
Reviewers:
(1) Anand Nayyar, KCL Institute of Management and Technology, Jalandhar, Punjab, India.
(2) Anonymous, China University of Mining and Technology, China.
Complete Peer review History: http://sciencedomain.org/review-history/11188

Original Research Article

Received: 01 July 2015
Accepted: 31 July 2015
Published: 31 August 2015

Abstract

Ray tracing is a method to convert 3D image to high quality 2D realistic image. In traditional Ray tracing technique generating an image is an expensive process due to the large number of transmitted rays and the intersection tests of these rays with the scene primitives. This paper introduces an enhanced ray tracing (Enhanced RT) algorithm. In the proposed algorithm, merge sort algorithm is used to order triangles according to the minimum x coordinate. Then Binary Search algorithm is used to find the end index of the first triangle that has minimum x coordinate greater than the pixel x coordinate. This search limits the subset of the triangles that may intersect the ray, and hence, reduces the intersection calculation time. Experimental results show that the proposed algorithm decreased the execution time by 99.8% than the traditional ray tracing algorithm with high quality for the produced depth images for a standard Benchmark models. The implementation was done on an ordinary hardware without need to use the highly expensive parallel architecture hardware (as GPUs or Clusters) as in the other research in the same application. The proposed algorithm also achieved the highest successful hit rate in comparing to the most recent ray tracing algorithms.

Keywords: Depth images; ray tracing, merge sort algorithm; binary search algorithm.

1 Introduction

Recently, with large evolution in computer graphics and the increasing in display resolution and quality, the generated synthetic scenes have become more complex. Hence; the images generated from these scenes have become more complex. Ray tracing [1] is considered to be the most popular technique for rendering complex images, as it simulates the real vision process. Moreover, Ray Tracing is able to visualize highly realistic graphics effects. However, the main drawback of the ray tracing technique is its computational cost which is

**Corresponding author: Email: hanan.ahmed@fcis.asu.edu.eg, hanan.ahmed20100@cis.asu.edu.eg;*

highly expensive. Some recent researches appeared that aimed to enhance the computational cost of the ray tracing or to enhance its optics performance. Most of the computational cost enhancements were concentrated on using special data structure [2,3], special hardware [4,5] or decreasing the amount of the rays to be transmitted [6,7,8].

This paper proposes an enhancement of the Ray Tracing performance. The proposed method concentrates on reducing the computational cost by reducing the ray object intersections using merge sort and binary search algorithms. The discussion and evaluation in this research are restricted to the models that consist completely of opaque surfaces. For these models, only rays from the eye to the first surface (ray-tracing with no secondary rays) will be considered to generate depth images. The resultant depth images of our proposed method have the same quality as images generated using native ray tracing but with very low execution time. The execution time decreased with 99.8% by implementing the proposed algorithm on CPU.

The rest of the paper is organized as follows: Section 2 provides a brief to the previous work. Section 3 explains the proposed method. The results and conclusion will be in Section 4 and 5 respectively.

2 Materials and Methods

2.1 Previous Work

Ray tracing is the most popular image rendering algorithms. However, many researches appeared that aimed to improve its computational performance.

Many of these methods used acceleration data structures before tracing the rays. Clark [9] proposed the use of hierarchical bounding volumes to speed both clipping and visibility calculations.

Rubin and Whitted [2] used Bounding Volume Hierarchies (BVHs). Then Weghorst [10] proposed the use of different types of bounding volumes in a single hierarchy. The BVH has drawbacks as it depends on the selection of suitable bounding volume shape which may differ from object to anther in the same scene. The BVHs supposes that, the scene consists of more than one object but if a scene consists of single complex object the BVHs algorithm will be inefficient and loss its functionality. The efficiency of BVHs is dependent on how well an object fills the space of the bounding volume, in such a way that, if the empty volume around the object is large in proportional to the object's volume, the complexity of the algorithm will increase and the algorithm will be inefficient. In addition, The BVHs search time depends on the scene complexity.

Fujimoto [11] studied the problem from different approach, they divided the space into a uniform gird. Glassner [12] and Kaplan [13] divided their objects into octree voxels. Bentley [14] proposed the use of kd-trees as acceleration data structure. The main drawback of using spatial subdivision is that, an extremely unbalanced tree may occur. This will increase the complexity of searching the tree. The octree has the same problem of BVHs that the voxels may contain a single object with inappropriate bounding shape which will cause unnecessary intersection calculations. On the other hand kd-tree requires a highly construction time and the uniform grid may be only sparsely filled or the scene's geometry may still be clumped.

Real-time ray tracing has been a goal of the computer-graphics community for many years. For real-time ray tracing, the acceleration data structure must be built or updated for each frame, which impacts on the runtime of the ray tracing. Wald [15], Lagae [16] and Kalojanov [17] used grids as accelerated data structure but Hunt [18], Shevtsov [19], Zhou [20] and Wu [21] used kd-trees and Wald [22,23], Lauterbach [24] and Garanzha [25] used BVHs.

Another approach appeared based on using GPUs to accelerate the ray tracing process. GPU ray tracing offers significant performance gain over the CPU ray tracing, but it requires a customizing process for the algorithm to be able to operate on GPUs. Scientific researches in this area can be classified as follows:

1) Parallelizing the ray tracing process itself and compare the performance with the sequential ray tracing model as in [26,27,28].
2) Using the GPUs for parallelizing the construction process for the data structures that can be used in the ray tracing. Other research work in this trend, designed new optimized structures suitable for GPU computing as [29,30].

However, the last approach is neglecting some rays as pixel differential method [31], pixel averaging method [31] and advanced pixel averaging method [7] which are based on using the upper and lower pixels to get the color of the intermediate pixels. These methods enhanced the computational time of the algorithm but the image quality was going to be low. Another method is to recognize the rays that will not intersect the primitives and neglect them as in [6]. The main drawback of this method is that it is complicated as it requires performing several preprocessing steps as building BVH and determining active rays and sample them to calculate the cost function, the distribution of active rays in the bounding volume, and the traversal order of child nodes.

2.2 Ray Tracing Algorithm

Ray tracing algorithm is a well-known algorithm for rendering 3D scenes by modeling light reflection and refraction. The main idea of ray tracing algorithm is tracing the light rays through the scene. The objective is to determine the color of each light ray that strikes the view window in more simple words the objective is finding the color of each pixel.

In ray tracing algorithm as a theory, the rays were traced backward; they will be started from the eye position (camera position) instead of light source to avoid the effort of tracing rays that will not reach the eye. Any pixel its color is given by the color of the light ray that passes through that point on and reaches the eye. So For each pixel, the ray extends from the eye to it called primary ray. This ray will be followed into the scene and as it bounces off of different objects to limit the ray bounces the following approximation will be made: every time a ray hits an object, a single new ray will be followed from the point of intersection directly towards the light source The final color of the pixel is given by the colors of the objects hit by the ray as it travels through the scene.

From practical point of view ray tracing algorithm fundamentally one of those algorithms that make sure the appropriate object is “seen” through each pixel, and that the pixel color is shaded based on that object’s material properties, the surface normal seen through that pixel, and the light geometry that what is known with basic ray tracing, naïve ray tracing or traditional ray tracing [1].

To add shadow, a shadow ray will be added which is ray transmitted from the point (pixel) will be shaded to the light source if this ray hit an object the point will be in shadow and one of the shading models will be used. The most used one is Phong shading model [32].

Finally, refraction and reflection will be added according to the object properties, that means if the primary ray hits object like a mirror the ray will be reflected and if it hits transparent object like water or glass refraction will be applied. Both refraction and reflection depends on the objects materials [32].

The Shadow, refraction and reflection rays are called secondary rays which add realistic effects to the scene.

2.3 Enhanced Ray Tracing Algorithm (Enhanced RT)

The main idea of the proposed Enhanced RT algorithm is to reduce the ray object intersections testing for each primary ray and turn off the secondary ray to get fast depth images without aliasing. The secondary rays were neglected because we test the method on totally opaque models without environment around so, there are no shadows, reflections and refractions.

In the proposed Enhanced RT, merge sort algorithm is used to order triangles according to the minimum x coordinate of each one. Then Binary Search finds the end index of the first triangle that has minimum x coordinate greater than the pixel x coordinate. This search limits the subset of the triangles that may intersect the ray, and hence, reduces the intersection calculation time. After filtering the triangles with the minimum x coordinate of each triangle we will filter the resulting subset with y coordinate to avoid calculating the ray triangle intersection to all the previous subset, the algorithm tests whether the x and y coordinates of the pixel within the minimum and the maximum x and y coordinates of triangles in the subset instead of sorting the subset by the minimum y coordinate of each triangle and use Binary Search to find the end index of the first triangle that has minimum y coordinate greater than the pixel y coordinate, The complexity of using sorting the subset and applying binary search on it equals O(nlgn +lgn) but if the linear test will be O(n).This test causes that part of triangles will be neglected from calculation without affecting the quality of the rendered image.

The process sequence of our proposed Enhanced RT algorithm is explained in (Algorithm 1) in pseudo code form while the naïve ray tracing algorithm [1] is explained in (Algorithm 2). The main steps of our proposed algorithm can be summarized as follows:

1) The algorithm takes the camera position **eye_pos**, and the set of triangles **T** as input.
2) Determines the bounding volume of **T**, the minimum and the maximum x and y coordinates of each triangle **t** in **T**.
3) Triangles are sorted using Merge Sort algorithm based on the minimum x value of each triangle.
4) For each ray **r** in **R** which is set of rays from **eye_pos** to the screen pixels, P which lies inside the bounding volume of **T**.
    a. Initialize **P** depth with ∞.
    b. Assign the x coordinate of pixel **P** to **V**
    c. Apply binary search to get **endIndex**. **endIndex** is the index of the first triangle in **T** which its minimum x coordinate greater than **V**.
    d. For each triangle t in the subset which ends at **endIndex**
        i. Check if the x and y coordinates of P within the minimum and the maximum x and y coordinates of **t**.
            1. If **P** coordinate within the triangle coordinate calculate the intersection of the ray **r** with triangle **t** to get the current **depth** and update the depth of the pixel **P** in case that the **depth** is less than the current depth.

Algorithm 1

Proposed Enhanced Ray tracing Algorithm for calculating depth image. T is a set of triangles in the scene. eye_pos is the eye position to the scene.

```
1.  Fast_Depth_Image(eye_pos,T)
2.  For each triangle t in T
3.  Get min. and max. x and y coordinate of t vertices
4.  Merge_Sort(T) // sort triangles according to their min. x values
5.  For each ray  r from  eye_pos to each pixel P
6.  P. depth= ∞
7.  V=P.x
8.  endIndex=Binary_Search(T,V) //Search triangles according to their min. x value of the  triangle and
    get the index of  triangle which his min. x value is the first greater than V
9.  For each triangle t in T from 0 to endIndex
10. If P.x within t min. x and max. x and P.y within t min. y and max. y
11. depth= GetIntersection(r,t)
12. If(depth < P.depth)
13. P.depth=depth
```

Algorithm 2

Naïve ray tracing without secondary ray. T is a set of triangles in the scene. eye_pos is the eye position to the scene.

1. RT (eye_pos,T)
2. For each ray r from eye_pos to screen pixel P
3. P.depth= ∞
4. For each triangle t in T
5. If r intersect t
6. depth= get_ray_triangle_intersection (r,t)
7. If depth < P.depth
8. P. depth=depth

Algorithm 3

Naïve z-buffer without interpolation. T is a set of triangles in the scene. eye_pos is the eye position to the scene.

1. Z_buffer (eye_pos,T)
2. For each pixel P on screen
3. P. depth= ∞
4. For each triangle t in T
5. For each pixel P on t
6. get r the ray from eye_pos to P
7. depth= get_ray_triangle_intersection (r,t)
8. If depth < P.depth
9. P. depth=depth

3 Results and Discussion

The experiments are designed to test the following:

1- Compare the performance of the proposed Enhanced RT with the Traditional RT algorithms in terms of the quality of the generated images.
2- Compare the performance of the proposed Enhanced RT to different RT algorithms in terms of the successful hit rate which is the ration between the visible pixels and total intersection tests.

The algorithms that implemented and used for performance comparison are: naïve z-buffer without interpolation [33] shown in (Algorithm 3), naïve ray tracing [1] without secondary ray shown in Algorithm 2 and our proposed enhanced ray tracing algorithm. SHREC 2011 benchmark dataset [34] was used in all experimental evaluations. SHREC is a benchmark created for shape retrieval contest of Non-rigid 3D. The benchmark consists of 600 models. Models are represented as watertight triangle meshes and the file format is selected as the ASCII Object File Format (*.off). Watertight triangle meshes have average size 18950 triangles. We used T1.off and T6.off which are models in SHREC benchmark to evaluate the performance of our Enhanced RT algorithm. The algorithm implemented on a standard PC with an intel core i7-4702MQ 2.20GHz CPU and 8GB memory.

3.1 Results

We tested the proposed Enhanced RT by generating the depth images for two different models from SHREC benchmark, each image of size 256×256. The quality of those images are compared with the quality of the images generated for the same models using z-buffer without interpolation and naïve RT. The images

generated using z-buffer have many holes as the z-buffer can deal only with pixels that completely located inside the triangle (see Figs. 1 and 2). These holes made the images very low quality. The images generated using naïve RT shown in (Figs. 3 and 4) are smooth, without holes and aliasing. But the images generated using our proposed Enhanced RT shown in (Figs. 5 and 6) have the same quality as those generated with naïve RT but with very low execution time for our proposed method.

(Table 1) displays the execution time of the three different algorithms. As shown from the results, z-buffer algorithm has the lowest execution time. As the size of the mesh is less than the size of the image, which equals the number of rays that will be transmitted in naïve RT. The complexity of z-buffer is O(T) where T is the number of triangles and naïve RT is O(RT) where R is the number of rays so, the Naive RT has the highest execution time. The Enhanced RT has a very low execution time, with decrease by 99.8% compared to the time of the naive RT. However, the low execution time of the naive z-buffer is considered to be of less importance compared to the quality of its generated image as shown in (Figs. 1 and 2). The images generated using z-buffer contain many holes as the method can deal only with pixels that completely located inside the triangle. For the proposed Enhanced RT algorithm, the gain in the quality of the generated images is a very promising advantage. The proposed algorithm achieved a very good compromise between the quality of the generated depth images and the algorithm's execution time.

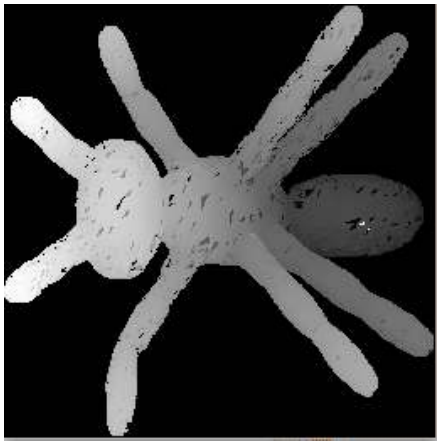

Fig. 1. Depth image of T1.off with z-buffer

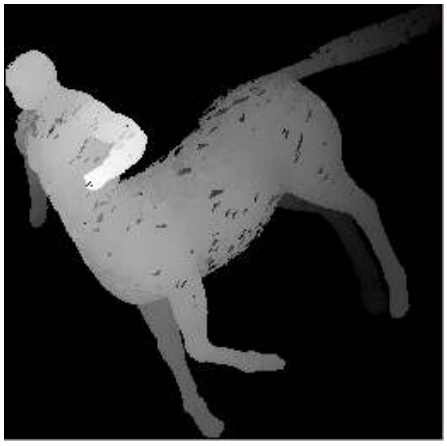

Fig. 2. Depth image of T6.off with z-buffer

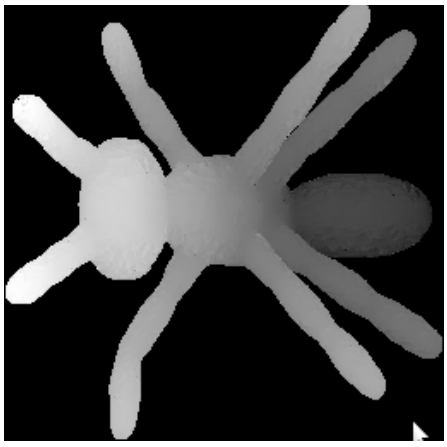

Fig. 3. Depth image of T1.off with naïve RT

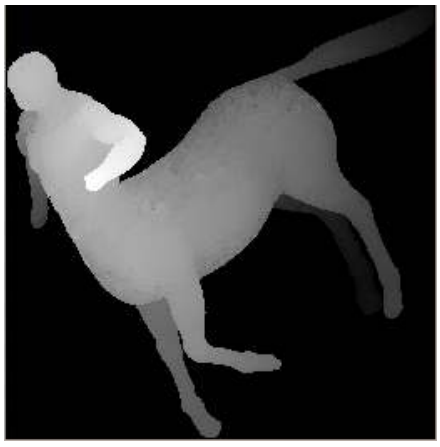

Fig. 4. Depth image of T6.off with naïve RT

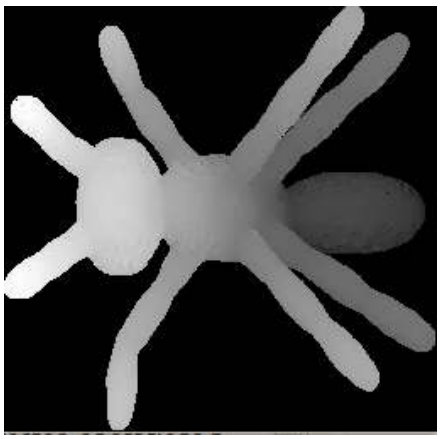

Fig. 5. Depth image of T1.off with our proposed Enhanced RT algorithm

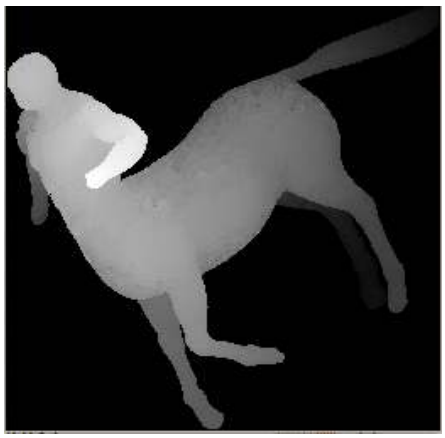

Fig. 6. Depth image of T6.off with our proposed Enhanced RT algorithm

Table 1. The run time of each model using Naïve RT (naïve ray tracing), Enhanced RT (our proposed method) and Z-buffer

Model name	Mesh size	Naïve RT (seconds)	Enhanced RT includes merge sort time (seconds)	Enhanced RT excludes merge sort time (seconds)	Enhanced RT excludes binary search time (seconds)	Acceleration ratio of enhanced RT in worst case	Z-buffer (seconds)
T1.off	18998	1716.6	1.66235	1.64638	1.636	99.9%	0.33
T6.off	18876	1816.59	1.88094	1.850	1.84929	99.896%	0.33

A comparison performed between our proposed algorithm and one of the most recent ray tracing algorithms, the fast kd-tree construction for ray tracing based on efficient ray distribution [35], was down using successful hit rate as shown in (Table 2 and Fig. 7). (Figs. 8 and 9) show the high quality of the produced depth images for the same two models used in [35]. (Table 2) shows that our proposed algorithm, Enhanced RT, has the highest successful hit rate which means that, it is the most efficient algorithm.

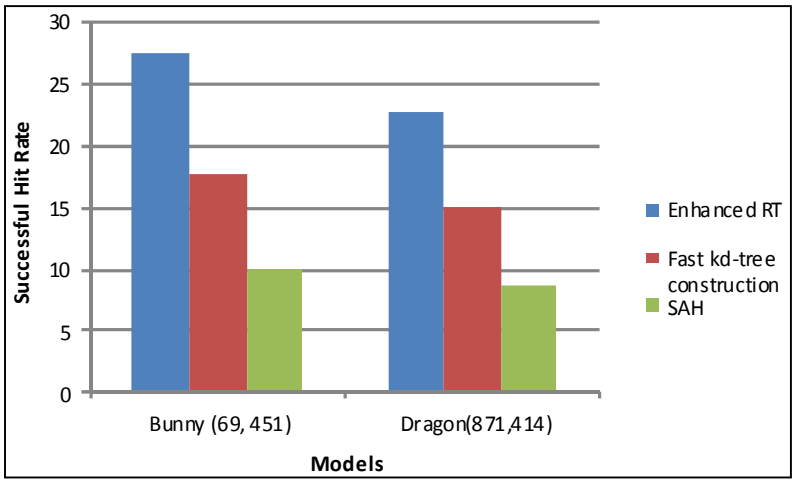

Fig. 7. The successful hit rate of different algorithms for different mesh sizes

As a summarization Enhanced RT achieved better results than naïve RT in terms of running time which decreased by 99.8% with the same image quality. Also Enhanced RT achieved better image quality than z-buffer but z-buffer achieved better running time. In comparison with fast kd-tree construction for ray tracing algorithm based on efficient ray distribution [35], Enhanced RT achieved better successful rate which indicates that Enhanced RT succeeded in decreasing the total ray object intersection.

Table 2. The comparison between our algorithms and the fast kd-tree construction for ray tracing based on efficient ray distribution

Model	Mesh size	Method	Used hardware	Number of ray primitive intersections	Successful hit rate
Bunny	69, 451	Enhanced RT	Intel core i7-4702MQ 2.20GHz CPU and 8GB memory	154,409	27.5%
Dragon	871,414	Enhanced RT	Intel core i7-4702MQ 2.20GHz CPU and 8GB memory	148,710	22.7%
Bunny	69, 451	Fast kd-tree construction for ray tracing based on efficient ray distribution [35]	Intel i3-2100 3.00 GHz CPU, equipped with 4.0G memory and an NVidia GeForce GTX 450 graphics card	999,885	17.68%
Dragon	871,414	Fast kd-tree construction for ray tracing based on efficient ray distribution [35]	Intel i3-2100 3.00 GHz CPU, equipped with 4.0G memory and an NVidia GeForce GTX 450 graphics card	1,090,330	15.11%
Bunny	69, 451	SAH [36]	Undefined	1,637,046	9.89%
Dragon	871,414	SAH [36]	Undefined	1,720,314	8.65%

Fig. 8. Depth image of Bunny with our proposed Enhanced-RT algorithm

Fig. 9. Depth image of Dragon with our proposed Enhanced-RT algorithm

4 Conclusion

In this paper we introduced an enhanced ray tracing (Enhanced RT) algorithm, and its implementation, which used to get the depth images for retrieving non-rigid 3D objects based on rendering the 3D model as a set of depth images from multiple view directions. Enhanced RT is based on sorting the mesh's triangles according to its minimum x coordinate and get the range of triangles that may intersect with each ray using

binary search. The proposed algorithm enhanced the execution time of the ray tracing, with the same quality as the naïve ray tracing, by reducing the number of ray triangle intersections.

The most recent researches concentrated on using accelerating data structure or implement algorithms on GPUs. But In this work the implementation was done on an ordinary hardware with high quality results which reflect the great efficiency for the proposed algorithm. The implementation was done using, Intel core i7-4702MQ 2.20GHz CPU and 8GB memory. The proposed algorithm achieved very good run time with excellent accuracy without using acceleration data structure and using the simple well known helper algorithms, merge sort and binary search, to reduce the number of ray triangles intersection tests. Experimental results show that the proposed Enhanced RT algorithm decreased the execution time by 99.8% than the execution time for the traditional ray tracing algorithm with the same quality for the produced depth images. The proposed algorithm achieved a very good compromise between the quality of generated depth images and the algorithm's execution time. Also as shown in the experimental result, our proposed algorithm, Enhanced RT, has the highest successful hit rate which means that, it is the most efficient algorithm.

For future work we would like to implement our algorithm with secondary rays to add reflection and refraction effects to make the scenes more realistic and compare it with other ray tracing algorithms which used for static and dynamic scenes in terms of running time and image quality. We would like to study the effect of increasing the scene mesh size and the number of objects in the scene. Also we intended to parallelize the implementation of our proposed model using GPUs to achieve more performance enhancement.

Authors' Contributions

Author HA designed the study, performed the statistical analysis, wrote the protocol, and wrote the first draft of the manuscript. Author HAS managed the analyses of the study and manuscript revision. Author DH managed the literature search. All authors read and approved the final manuscript.

Competing Interests

Authors have declared that no competing interests exist.

References

[1] Appel A. Some techniques for shading machine rendering of solids. Spring Joint Computer Conference. Washington DC. 1968;37-45.

[2] Rubin SM, Whitted T. A 3-dimensional representation for fast rendering of complex scens. The 7th Annual Conference on Computer Graphics and Interactive Techniques. New York. 1980;110-6.

[3] Kay TL, Kajiya JT. Ray tracing complex scenes. 13th Annual Conference on Computer Graphics and Interactive Techniques. 1986;269-78.

[4] Zhu X, Deng Y. Evaluation and improvement of GPU ray tracing with a thread migration technique. International Journal of Signal Processing Systems. 2013;1:111-5.

[5] Áfra AT, Szirmay-Kalos L. Stackless multi-BVH traversal for CPU, MIC and GPU ray tracing. Computer Graphics Fourms. 2014;33:129–40.

[6] Nabata K, Iwasaki K, Dobashi Y, Nishita T. Efficient divide and conquer ray tracing using ray sampling. The 5th High-Performance Graphics Conference. 2013;129-35.

[7] Patoliya D, Shah V, Ghodasara B. Reducing Computational time in ray tracing. International Journal of Advance Engineering and Research Development. 2014;1:1-5.

[8] Stokes D. Divide and conquer G-buffer ray tracing: Eastern Washington University; 2014.

[9] Clark JH. Hierarchical geometric models for visible surface algorithms. Communications of the ACM. 1976;19:547-54.

[10] Weghorst H, Hooper G, Greenberg DP. Improved computational methods for ray tracing. ACM Transactions on Graphics. 1984;3:52-69.

[11] Fujimoto A, Tanaka T, Iwata K. ARTS: Accelerated Ray-Tracing System. Computer Graphics and Applications, IEEE. 1986;6:16-26.

[12] Glassner AS. Space subdivision for fast ray tracing. Computer Graphics and Applications, IEEE. 1984;4:15-24.

[13] Kaplan MR. The uses of spatial coherence in ray tracing. Techniques for Computer Graphics. 1987;173-93.

[14] Bentley JL. Multidimensional binary search trees used for associative searching. Communications of the ACM. 1975;18:509–17.

[15] Wald I, Ize T, Kensler A, Knoll A, Parker SG. Ray tracing animated scenes using coherent grid traversal. ACM Transactions on Graphics. 2006;25:485–93.

[16] Lagae A, Dutré P. Compact, fast and robust grids for ray tracing. Computer Graphics Forum. 2008;27:1235–44.

[17] Kalojanov J, Slusallek P. A parallel algorithm for construction of uniform grids. HPG '09 Proceedings of the Conference on High Performance Graphics. 2009;23-8.

[18] Hunt W, Mark WR, Stoll G. Fast kd-tree construction with an adaptive error-bounded heuristic. Interactive Ray Tracing 2006, IEEE Symposium on. Salt Lake City. 2006;81-8.

[19] Shevtsov M, Soupikov A, Kapustin E. Highly parallel fast kd-tree construction for interactive ray tracing of dynamic scenes. Computer Graphics Forum. 2006;26:395–404.

[20] Zhou K, Hou Q, Wang R, Guo B. Real-time kd-tree construction on graphics hardware. ACM Transactions on Graphics. Asia; 2008.

[21] Wu Z, Zhao F, Liu X. SAH kd-tree construction on GPU. ACM SIGGRAPH Symposium on High Performance Graphics. 2011;71-8.

[22] Wald I, Boulos S, Shirley P. Ray tracing deformable scenes using dynamic bounding volume hierarchies. ACM Transactions on Graphics. 2007;26:2-18.

[23] Wald I. On fast construction of SAH-based bounding volume hierarchies. Interactive Ray Tracing. 2007;33-40.

[24] Lauterbach C, Garland M, Sengupta S, Luebke D, Manocha D. Fast BVH construction on GPUs. Computer Graphics Forum. 2009;28:375–84.

[25] Garanzha G, Pantaleoni J, McAllister D. Simpler and faster HLBVH with work queues. ACM SIGGRAPH Symposium on High Performance Graphics. 2011;59-64.

[26] Britton AD. Full CUDA implementation of GPGPU recursive ray-tracing. Purdue University; 2010.

[27] Budge BC, Anderson JC, Garth C, Joy KI. A straightforward CUDA implementation for interactive ray-tracing. Interactive Ray Tracing. Los Angeles: IEEE Symposium on; 2008;178.

[28] Segovia A, Li X, Gao G. Iterative layer-based raytracing on CUDA. Performance Computing and Communications Conference. Scottsdale; 2009.

[29] Carr NA, Hoberock J, Crane K, Hart JC. Fast GPU ray tracing of dynamic meshes using geometry images. Graphics Interface. Toronto. 2006;203-9.

[30] Santos AL, Teixeira JMXN, de Farias TSMC, Teichrieb V, Kelner J. KD-Tree traversal implementations for ray tracing on massive multiprocessors: A comparative Study. Computer Architecture and High Performance Computing. Sao Paulo. 2009;41-8.

[31] Behmanesh AA, Pourbahrami S, Gholizadeh B. Reducing render time in ray tracing by pixel averaging. International Journal of Computer Graphics & Animation. 2012;2:15-24.

[32] Peter S, Gleicher M, Marschner MR, Reinhard E, Sung K, Thompson WB, et al. Fundametals of Computer Graphics. Second Edition. 2005;201-237.

[33] Joy KI. The depth buffer visible surface algorithm: One line computer graphics notes. University of California; 1999. Accessed25-September-2014.
Available: http://www.idav.ucdavis.edu/education/GraphicsNotes/Z-Buffer-Algorithm.pdf

[34] NIST. SHREC 2011 - Shape Retrieval Contest of Non-rigid 3D Watertight Meshes; 2011. Accessed: 10 Jan 2015. Available: http://www.itl.nist.gov/iad/vug/sharp/contest/2011/NonRigid/data.html

[35] Liang X, Yang H, Qian Y, Zhang Y. A Fast Kd-tree construction for ray tracing based on efficient ray distribution. Journal of Software. 2014;9:596-604.

[36] Wald I, Havran V. On building fast kd-trees for ray tracing, and on doing that in o (n log n). Interactive Ray Tracing. IEEE. 2006;61-9.

Peer-review history:
The peer review history for this paper can be accessed here (Please copy paste the total link in your browser address bar)
http://sciencedomain.org/review-history/11188