International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
DOI: https://doi.org/10.17762/ijritcc.v11i9.8341
***Article Received:** 16 July 2023 **Revised:** 02 September 2023 **Accepted:** 19 September 2023*

GPU Accelerated Simulation of Scene Generation of 3D Photonic Mixer Device Camera

Sangita Gautam Lade[1], Sanjesh Pawale[2], Aniket Patil[3]
[1]Department of computer Engineering,
Vishwakarma University, Pune, India
sangita.lade@vit.edu
[2]Department of computer Engineering,
Vishwakarma University, Pune, India
sangita.lade@vit.edu
[3]IFM engineering pvt ltd, Pune, India
Aniket.patil@ifm.com

Abstract—Simulation of Photonic Mixer Device (PMD) sensors have the capability to create virtual environment to test 3D camera design. This simulation comprises of multiple steps like scene generation using ray tracing, power calculation, raw data generation and raw data processing. However, each step-in situation process takes longer time to implement and they are simulation process, simulators need to be faster. In this paper, we propose parallel implementation method for scene generation using GPGPUs. The feasibility of the method is confirmed using Amdahl's law before implementation. The method is implemented and tested on GeForce 820M, GeForce 750Ti and Volta V100.Tthe highest speed up obtained is 219.913 using Volta (GV100) GPU for block size 1024. Thus, parallel method optimizes the scene generation time as compared to serial processing and the implemented results are better than the state of the art in the literature.

Keywords-GPU; CUDA; simulation; PMD sensor; Ray tracing; scene generation.

I. INTRODUCTION

Simulation plays a vital role in industries, academics and in research. The PMD sensor simulator has ability to reproduce the essential sensor characteristics. Also, for dynamic scene setups, it becomes essential to carry out experiments under reproducible conditions. E.g., if we want to customize the camera for outdoor applications, then building prototype for it is very expensive, but if we develop a simulator for it, we can generate the dynamic scenes using the simulator. Therefore, the simulation results must reflect major sensor characteristics in order to produce results representative of and comparable to real sensor data.

The simulation process of PMD sensor comprises of multiple steps like scene generation using ray tracing, power calculation, raw data generation and raw data processing. Therefore, to optimize time for simulation process, simulators need to be faster.

Generation of scene is the basic step in simulating PMD sensors. Generally, when the snap is actually captured using 3D camera, the scene is in front of us. But when we are simulating it, we have to generate the scene. Here ray tracing is used to generate the scenes. Ray Tracing is a method of generating photo realistic images of the 3D scenes [1]. Here the path of light is traced through each pixel in an image plane. In ray tracing, the intersection of the rays and each pixel in the 3D object is found out. The scene may consist of a set of geometric primitives like polygons, spheres, cones etc. Also, generating scenes using random shapes is really a challenging job, so very basic 3D objects are used for scene generation i.e. sphere, box, plane and triangle.

The major contribution of this research paper are the proposed parallel method for scene generation on three different GPUs viz GeForce 820M, GeForce 750Ti and Volta V100 and time optimization using parallel implementation using GPU and improved performance over multiple algorithms like Hierarchy traversal algorithm [2] , Optimized ray tracing algorithm using CUDA library [3], A uniform grid accelerated GPU ray tracer [4], A Parallel ray tracer on GPU [5], Traversal of a kd-tree without stack [6], Bounding Volume Hierarchies [7], Algorithms to manage scene complexity using cache [8], Parallel ray tracer [9] and Parallelized version of ray tracing in CUDA[10].

This paper covers the parallel implementation for scene generation. Section 1 provides the introduction. Section 2 briefs about the literature survey & related work. Section 3 describes the methodology. In section 4 the results of sequential and parallel implementation for scene generation are discussed. Section 5 ends with conclusion.

II. LITERATURE SURVEY AND RELATED WORK

A new hierarchy traversal algorithm for speeding up ray-object intersection calculations are presented in [2]. But no parallel algorithm was proposed for the same. Optimized ray tracing algorithm using CUDA library is proposed in [3]. But they used GT840M graphics card for it. Since, Cuda's library prepares threads to computation based on power of graphic card, therefore they can't activate some of top and down threads. So, some threads were not utilized for parallel implementation. A uniform grid accelerated GPU ray tracer was implemented but there is an overhead incurred due to state-based programming [4]. A GPU ray tracer where the load on GPU is transferred to CPU using partitioning, was implemented in [5]. But the problem of CPU-GPU communication bottleneck is not solved as the data needs to be transferred from CPU memory to GPU memory. Two algorithms were implemented viz kd-restart and kd-backtrack. These variations of kd tree are designed without using stack for the traversal. kd-restart uses top-down approach while kd-backtrack uses bottom-up approach. But they achieved only 10% of the performance using GPU as compared to its counterpart using CPU [6]. Tracing static models can be efficiently done by Bounding Volume Hierarchies (BVHs) but they can be used to ray trace deformable models with little loss of performance. Also, they have not used GPU to enhance the speed [7].

The algorithms based on caching to manage scene complexity are developed. These algorithms are used for scene generations where scene contains millions of primitives but only ten percent of the scene description is stored in memory. This task is also embarrassingly parallel in nature but no parallel architecture like GPU was used for the same [8]. A sequential ray tracer on CPU and parallel ray tracer on GPU was designed which achieved the speed up of 185.241% for producing images using GPU over CPU [9]. A typical run of the serial C code for ray tracing on Lincoln took 90.56 seconds. Compared to the serial code, the naive implementation achieved a speedup of 52 times using CUDA on GPU. The cyclic implementation and the dynamic implementation achieved a speedup of 211 compared to the serial code using GPU [10]. The techniques and algorithms known for ray tracing were studied and the results concluded that ray tracing will be the most revolutionary technology ever witnessed in the field of animation and graphics [11].

III. METHODOLOGY

A. *Scene Generation*

In Scene Generation, Ray Tracing is carried out by generating and tracing rays through each pixel of the picoflex camera having a resolution of 172 X 224 and finding the nearest ray-object intersection. Five output matrices are generated which stores the data of Intersection Point, Distance, Normal, Reflectivity and Visibility. The Intersection Point matrix stores the nearest Ray-Object intersection for each ray that is casted into the scene. The distance matrix holds the Euclidean distance from the sensor to the point of the nearest intersection for each ray. The normal matrix stores the normal at the point of intersection for every ray. The reflectivity matrix stores the reflectivity of the object at the point of intersection whereas the visibility matrix stores the information of whether or not the point of intersection is illuminated by the light source.

To get the intersection point of a ray and any object is to solve the two equations i.e., the equation of a ray and the equation of an object. For each of the object, the intersection with the ray is calculated. For the plane, A, B, C are the x, y and z components of the plane normal and D is the perpendicular distance from ray origin to the plane. Ray's parametric variable t is calculated by substituting the equation of ray in the equation of plane. The variable t is bounded by $[0, \infty]$. If t is negative, the object lies behind the ray's origin, thus invalidating the obtained intersection point. If t is positive, the intersection point can be obtained by re-substituting t in the ray's equation.

The Ray- Triangle intersection is divided into two parts. First, the intersection is computed with the plane containing the triangle. The normal of the plane containing the triangle is computed by taking cross product of any two edges of the triangle. The perpendicular distance between the plane and the ray origin is calculated by taking dot product of the normal and any vertex of the triangle. Ray's parametric variable t is calculated by substituting the equation of ray in the equation of plane. The variable t is bounded by $[0, \infty]$. If t is negative, the object lies behind the ray's origin, thus invalidating the obtained intersection point. If t is positive, the ray-plane intersection point can be obtained by re-substituting t in the ray's equation. Further, Moller Trumbore's algorithm is used is to find if the intersection point lies within the boundaries of the triangle. For finding the intersection with the sphere, the ray's equation is substituted in the equation of sphere and the value of t (parametric variable) is computed. Since the equation of the sphere is quadratic, the value of t is calculated using determinant. If the determinant is less than zero, then there is no intersection of the ray and the sphere. If it is positive, the intersection point can be calculated by resubstituting the t in the equation of ray. Before that, the value of t is checked to be greater than zero to avoid behind the ray, Ray-Object intersections.

Ray-Box intersection is found out by slab method proposed by Kay and Kajiya. Slab is the space between two parallel planes. So, the intersection of a set of slabs defines a bounding volume or a box. The method looks at the intersection of each pair of slabs by the ray. It finds tfar and tnear for each pair of slabs. If the overall largest tnear value i.e., intersection

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
DOI: https://doi.org/10.17762/ijritcc.v11i9.8341
***Article Received:** 16 July 2023 **Revised:** 02 September 2023 **Accepted:** 19 September 2023*

with the near slab, is greater than the smallest tfar value (intersection with far slab) then the ray misses the box, else it hits the box. In order to calculate the normal at the intersection point, a vector is generated from the center of the cube to the point of intersection. The vector is then divided by the length of the box along x, y and z dimension to get a unit vector. The normal vector is obtained by extracting only the integer form of the obtained vector.

B. Therotical SpeedUp using Amdhl's Law

Before implementing the parallel method, the theoretical speed up is computed using Amdahl's law to check the feasibility of the method as the scene generation using multiple objects is time consuming and embarrassingly parallel. The speed up calculated using Amdahl's law is presented in fig. 1. The computation for each ray with each object pixel is considered here.

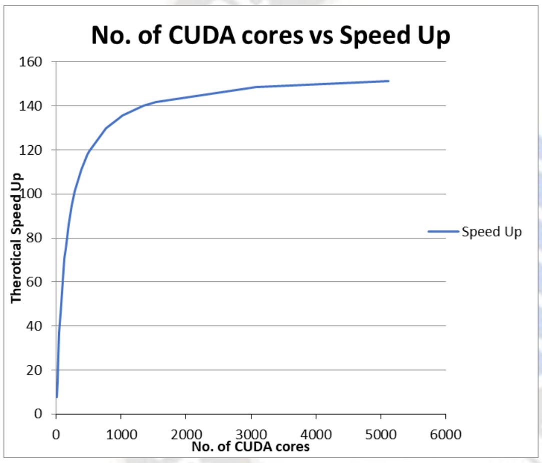

Figure 1. No. of CUDA cores V/s Theoretical Speedup

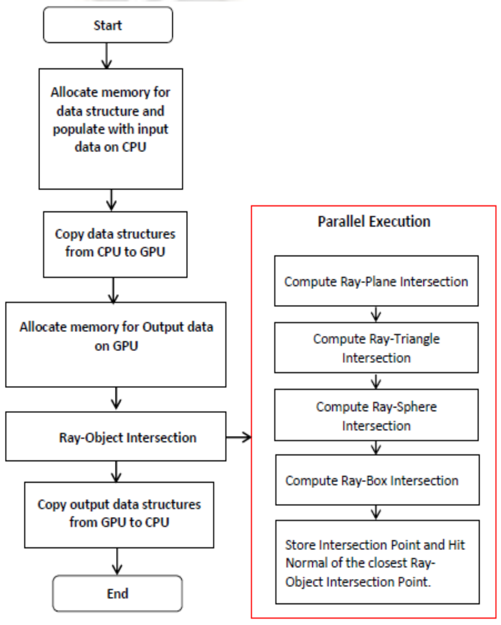
Start
Allocate memory for data structure and populate with input data on CPU
Copy data structures from CPU to GPU
Allocate memory for Output data on GPU
Ray-Object Intersection
Copy output data structures from GPU to CPU
End
Parallel Execution
Compute Ray-Plane Intersection
Compute Ray-Triangle Intersection
Compute Ray-Sphere Intersection
Compute Ray-Box Intersection
Store Intersection Point and Hit Normal of the closest Ray-Object Intersection Point.

Figure 2 : Flowchart for Parallel Ray Tracing

Here in this method, the intersection of 38,528 rays (172 * 224) with the number of pixels of the objects are computed. When the number of objects in the scene are increased, the computations are increased exponentially (in the multiples of the number of objects). So, to use sequential algorithm poses a restriction on the number of iterations, memory as well as time. Some sequential algorithms even cannot be executed completely if the number of objects is increased beyond certain limit. So, this task of calculation can be done parallelly with the help of GPU as shown in fig. 2.

IV. EXPERIMENTATION AND RESULTS

Generally, when the repetitive task is done, in sequential method loops are used. But GPU has multi core architecture. So assigning task to each core is a critical task. So, thread organization is to be done properly so that each streaming multiprocessor can evenly get the number of threads to execute.

Here a total number of thread generated are 172 * 224. For the experimentation three GPGPUs are used viz GeForce820M, GeForce850Ti and Volta V100. The basic configuration of all these GPUs is given in Table 1.

To assign a task to every core, the number of threads are created which is called as a grid. The task to be given is to be assigned by writing a kernel. The kernel is written for the grid of 38528 (172 * 224) threads here. Further these threads are organized as 2D blocks for scheduling it on the number of streaming multiprocessors available in each GPU. The warp size is 32 but the block size is varied from 32 to 1024 in the multiples of 32 and the time for scene generation is measured. Finally, the actual speedup is calculated for all the 15 different scenes. Initially the scene is generated using a single 3D object. Then the complex scenes are generated using all possible combinations of multiple objects. E.g. initially the scene is generated using individual objects like plane, box, sphere & triangle. Then two objects are combined to generate the scene with all permutations and combinations. There are 15 cases tested on each GPU for the scene generation using multiple block sizes varying from 32 to 1024. Deciding the optimum block size for scene generation is very crucial. That's why the experimentation is done with varying block sizes from 32 to 1024 in the multiples of 32. The time taken by parallel method for various block sizes on three different GPUs is shown in Figure 2. The least execution time for the scene generation of all the four objects was taken by block sizes 1024 for GeForce 750Ti and Volta respectively.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
DOI: https://doi.org/10.17762/ijritcc.v11i9.8341
***Article Received:** 16 July 2023 **Revised:** 02 September 2023 **Accepted:** 19 September 2023*

TABLE I. SPECIFICATIONS OF THREE DIFFERENT GPUS

GPU Architecture Parameters	GPU Architecture		
	GeForce 820M	**GeForce 750Ti**	**Volta V100**
Cores	96	640	5120
Memory Capacity	2 GB	2 GB	16 GB
Memory Bus	64 Bit	128 Bit	4096 Bit
Power Consumption	15 W	60 W	300 W

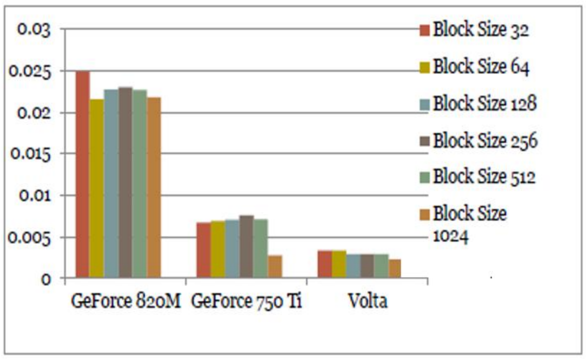

Figure 3. Block size V/s Time taken by GPU for execution

The time taken by sequential method is measured on CPU and for parallel method, it is measured on GPU using CudaEvents. Sequential implementation on CPU takes 0.5080 seconds to execute. A gradual reduction of execution time is seen when the same is implemented in parallel on GPUs. The execution time required by GeForce 820M, GeForce 750Ti and Volta is 0.0215, 0.002804 and 0.00231 seconds respectively for block size of 1024. The time taken by sequential method on CPU and parallel method on GPU is shown in fig. 3.

The speed up is one of the important metrics to evaluate the performance of a parallel algorithm where the speedup of GPU over CPU is calculated by considering the time taken by CPU & GPU for the execution. Figure 4 shows the speedup obtained using GPU over CPU. A speed up of 23.3445, 181.1697 and 219.9134 was obtained on GeForce 820M, GeForce 750Ti and Volta respectively. As the number of cores are increased from 96 to 640, the speedup has also increased from 23.35 to 181.17. For the increased number of cores from 640 to 5120 , the speedup has also increased from 181.17 to 219 .91.

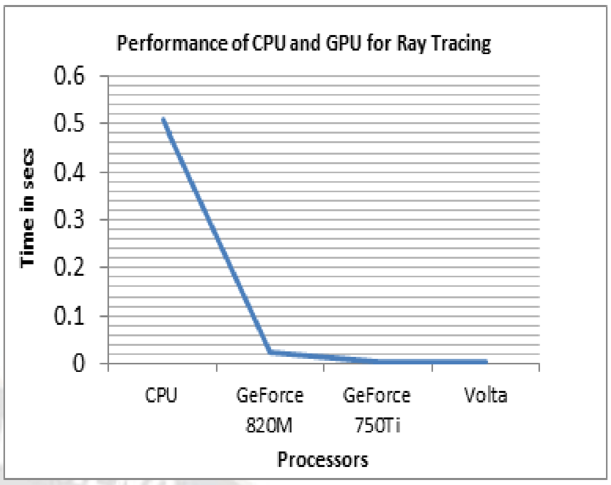

Figure 4. Processor (GPUs) V/s Execution Time in secs

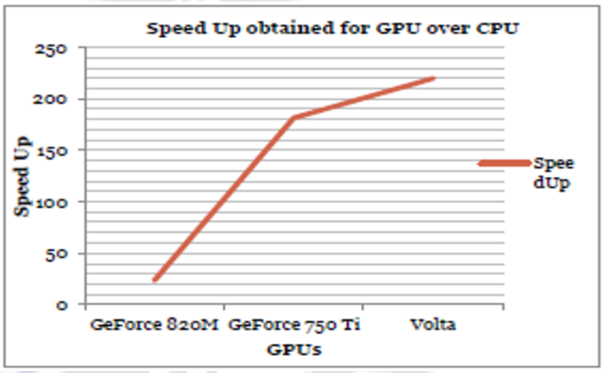

Figure 5. GPUs V/s Speed Up

The results obtained are also compared with the algorithms listed in table 2 from the literature. It is observed that the implemented parallel method out performs all the algorithms presented in table 2. The parallel algorithm like Optimized ray tracing algorithm, A uniform grid accelerated GPU ray tracer , Parallel ray tracer on GPU, Traversal of a kd-tree without stack

Have their own limitations in terms of speedup & thread organization. All these limitations are overcome by the proposed parallel method.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
DOI: https://doi.org/10.17762/ijritcc.v11i9.8341
***Article Received:** 16 July 2023 **Revised:** 02 September 2023 **Accepted:** 19 September 2023*

TABLE II. COMPARISON OF PROPOSED METHOD WITH EXISTING ALGORITHMS

Ref No	Algorithms for scene generations	Type of Algorithm	Results
2	Hierarchy traversal algorithm	Sequential	NA
3	Optimized ray tracing algorithm using CUDA library	Parallel on GT840M	Some threads were not used due to inbuilt library.
4	A uniform grid accelerated GPU ray tracer	Parallel	overhead incurred due to state-based programming
5	A Parallel ray tracer on GPU	Parallel	Bottleneck due to CPU-GPU communication
6	Traversal of a kd-tree without stack	Parallel	10 % Speed Up
7	Bounding Volume Hierarchies	Sequential	NA
8	Algorithms to manage scene complexity using cache	Sequential	NA
9	Parallel ray tracer	Parallel	185.241%
10	Parallelized version of ray tracing in CUDA	Parallel	52%
	Proposed Parallel Method	**Parallel**	**219 %**

V. CONCLUSION

The acceleration of scene generation is demonstrated in this paper by making use of GPGPUs. The kernel is written organizing the grid of 38,528 by into 2D blocks. Parallel implementation was tested on three GPUs viz- GeForce 820M, GeForce 750Ti and Volta V100 having 96, 640 and 5120 CUDA cores respectively. Highest speed up of 219.913 is obtained on Volta GPU for the block size 1024. The parallel algorithm implemented gives the highest speedup.

Acknowledgment

This work is sponsored by ifm engineering private limited, Pune. It is a privately-held global manufacturer of sensors and controls for industrial automation, producing more than nine million sensors annually. Ifm has more than 70 subsidiaries located in all major countries including North America, South America, Asia, Europe and Africa. We would like to thank Mr. Jitesh Butala, Aniket Patil and their team for providing the hardware details of 3D camera and providing the environment to test the results.

REFERENCES

[1] Lade, S., Kulkarni, M., Patil, A. Ray Tracing Algorithm for Scene Generation in Simulation of Photonic Mixer Device Sensors. In: Swain, D., Pattnaik, P.K., Athawale, T. (eds) Machine Learning and Information Processing. Advances in Intelligent Systems and
[2] Computing. 2021; vol 1311. Springer, Singapore. https://doi.org/10.1007/978-981-33-4859-2_25
[3] Timothy L. Kay James T. Kajiya. Ray Tracing Complex Scenes. California Institute of Technology, © 1986; A CM 0-89791-196-2/86/008/0269
[4] Sayed Ahmadreza Raziana, Hossein Mahvash Mohammadi. Optimizing Raytracing Algorithm Using CUDA. Italian Journal of Science and Engineering. October, 2017; Vol. 1, No. 3, http://www.ijournalse.org/
[5] Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan. Ray tracing on programmable graphics hardware. 2002; In Proc. SIGGRAPH.
[6] Hussain Bukhari, S. N. . (2021). Data Mining in Product Cycle Prediction of Company Mergers . International Journal of New Practices in Management and Engineering, 10(03), 01–05. https://doi.org/10.17762/ijnpme.v10i03.127
[7] Nathan A. Carr, Jesse D. Hall, and John C. Hart. The ray engine. In Proc. Graphics Hardware Sep 2002; pages 37–46.
[8] Vlastimil Havran. Heuristic Ray Shooting Algorithms. Ph.d. thesis, Dept. of CSE, Fac. of EE, Czech Technical University in Prague, Nov. 2000.
[9] Wald, I., Boulos, S., and Shirley, P. Ray tracing deformable scenes using dynamic bounding volume hierarchies. ACM Trans. Graph. 26, 1, Article 6. January 2007; 18 pages. DOI=10.1145/1186644.1186650 http://doi.acm.org/10.1145/1186644.1186650
[10] Matt Pharr, Craig Kolb, Reid Gershbein and Pat Hanraha. Rendering Complex Scenes with Memory-Coherent Ray Tracing. Association for Computing Machinery, Inc.1997
[11] Pitkin, Thomas A. GPU ray tracing with CUDA. EWU Masters Thesis Collection. (2013); 94. http://dc.ewu.edu/theses/9
[12] Liang Chen Hirakendu Das Shengjun Pan. An Implementation of Ray Tracing in CUDA CSE 260 Project Report. December 4, 2009
[13] Dhruv Dhote, Charu Virmani, Gopi Krishna, Shivansh Raghav. The Science of Ray Tracing. International Journal of Computer Applications (0975 – 8887). July 2020; Volume 176 – No. 42.