**Performance Evaluation of CUDA Based Ray Tracers**

A Parallel and Distributed Computing Project Report  
CSE4001

Submitted by  
**Project Members**  
Nitin Garg 19BCE0239  
Aadarsh N Prasad 19BCE2168  
Suyash 19BCE2297  
Ananya Anand 19BCE2332

Submitted to  
Prof. Anto S

In partial fulfilment for the award of the degree of

**Bachelor of Technology**  
in  
**Computer Science**



School Of Computer Science & Engineering  
January 2022

**Table of Contents:**

1. Abstract

2. Problem statement

3. Existing solutions for this problem

4. Issues faced by existing solutions

5. Proposed solution

6. Details about data

7. Performance metrics under consideration

8. References

**1. Abstract**

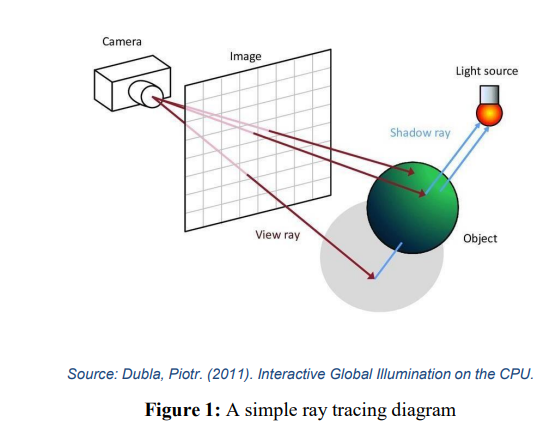
In the field of computer graphics, Ray Tracing is a rendering technique used to generate an image by tracing the path of light rays as pixels on a plane and simulating various effects and encounters with virtual objects. This technique is capable of producing high degrees of visual realism but at the sacrifice of computational costs. In this project we make an attempt at reducing the time taken to render an image using parallelization techniques. The aim of our project is to present an implementation and analysis of a CUDA based parallel Ray Tracer. CUDA is a parallel computing platform and application programming interface model created by Nvidia. CUDA parallelization paradigms allow us to use multi-threading to achieve maximum speedups up to 18 times against the serial implementation.

**2. Problem Statement**

Enhancing ray tracers and tracing algorithms by varying various parameters and determining the right combination of parallelization techniques and using distributed tracing, and selecting the right variation of hyperparameters to further optimize the renderers.

**3. Existing solutions for this problem**

By tracing the pathways of light rays via pixels in a view image, ray tracing creates a photo-realistic 3D image. In ray tracing, there are two approaches for tracing the light: forward tracing and backward tracing. Forward tracing is when rays are traced from the camera to the light source, and backward tracing is when rays are traced from the light source to the camera. Backward tracing is more computationally intensive since every ray emitted by the light source must be tracked (even if it does not reach the camera). Figure 1 shows a scene that will be ray traced.



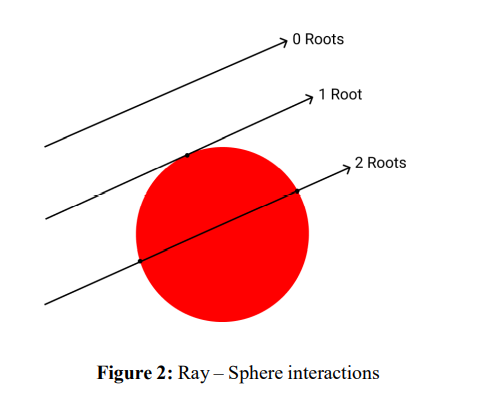
Algorithm 1 is a simple ray tracer algorithm wherein, the rays sent out by camera (for forward ray tracing) can be represented as a line (ray) with an origin (point A) and a direction (Vector B). At any time, a point on the line (ray) can be represented with

P(t) = A + B ⋅ t (1)

 where P is a 3D point lying along a line in 3D. The ray parameter t is the distance of the point from the ray origin.

 Using the origin and direction of the ray, all points lying on that ray can be computer. Using these points on a ray and the 3D position of a sphere, we can compute whether the ray intersects with any sphere in our scene at any points.

The ray-sphere interactions can be summarized using figure 2.



A sphere in 3D space can be defined vectors as:

             (P – C) ⋅ (P – C) = r ^2 (2)

Where P is any point on the surface of the sphere and C is the center of the sphere, with r being the radius of the sphere. Using equations 1 and 2, the ray sphere interactions can be defined as:

(P(t) – C) ⋅ (P(t) – C) = r 2 (3) ((A + B ⋅ t) – C) ⋅ ((A + B ⋅ t) – C) = r^2  (4)

The quadratic equation (4) can be solved for t to get different possibility in number of real roots, as depicted in Figure 2.

**4. Issues faced by existing solution**

The most serious flaw in this ray tracing method is that it results in aliasing. When the ray does not collide with a sphere (no roots), the background colour is returned, and so a deterministic colour is returned for each pixel in the view frame, which is overwritten by interactions of subsequent rays with that pixel. The solution is to add more rays to the picture and increase the randomness of the ray-object interactions.

By introducing sampling to eliminate aliasing, distributed ray tracing improves on the standard ray tracing approach. Anti-aliasing an image improves shadow quality and improves reflections, resulting in a more photo-realistic image. The single ray that computes a pixel's colour is replaced with many rays, and the anti-aliased pixel is rendered using an average of some of these randomly sampled rays.

**5. Proposed Solution**

In the CUDA version, the processing of the image is done in square batches of pixels (e.g., 8 by 8 batch). Four kernels are called, the first two of which are used for initializing and setting up the type of scene that needs to be rendered and the next two kernels have been used to initialize the random numbers for each thread while rendering the required image respectively.

In the serial implementation, we will use nested for loops to iterate over all of the pixels. In CUDA, the scheduler takes blocks of threads and schedules them on the GPU for us. CUDA allows us to maintain a Unified Memory frame buffer that is written by the GPU and read by the CPU.

**6. Details about data**

The rendered scene which we used to test and analyse our serial and CUDA based parallel ray tracers consists of spheres placed in a 3D plane, with light sources casting rays into the scene while the camera captures the reflected and refracted rays. The rendered scene is anti-aliased by random sampling of rays through a pixel and then averaging them to compute a colour. Three different types of materials have been implemented, namely diffuse or Lambertian, metal and dielectric. The metal objects show fuzzy reflection while the dielectrics show physically accurate phenomenon such as refraction and total internal reflection. The rendered scene is also gamma corrected to get accurate colour intensities.

**7. Performance metrics under consideration**

- Samples per pixel vs Rendering time  
- Maximum recursive depth vs Rendering time  
- Block size vs Rendering time  
- Dimensions vs Rendering time

**8. References**

* Adewale AE. (2021). Performance Evaluation of Monte Carlo Based Ray Tracer, Journal of Computational Science
* Xing, Qiwei & Chen, Chunyi & Li, Zhihua. (2021). Progressive path tracing with bilateral-filtering-based denoising. Multimedia Tools and Applications. 80. 1-16. 10.1007/s11042-020-09650-7.