Raytracer Implementation using cuda

ADITYA SINGH RATHORE, Indraprastha Institute of Information Technology, India

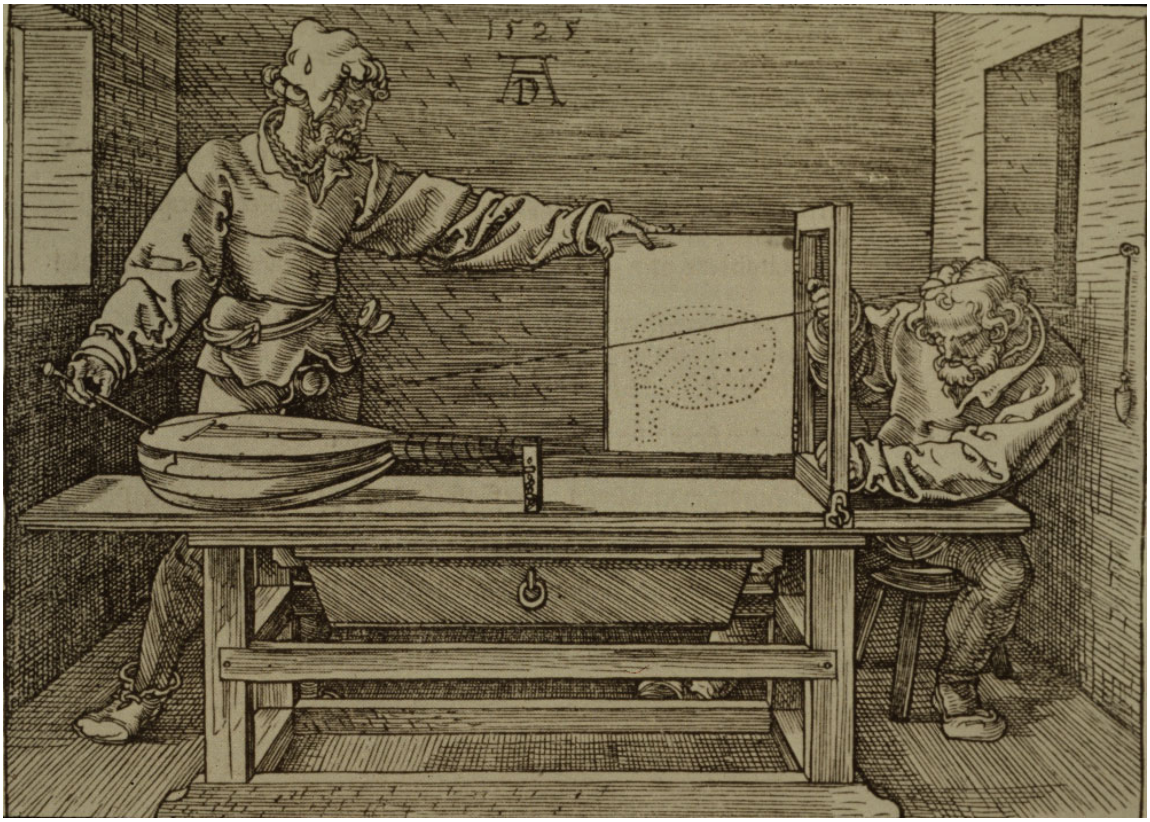

Fig. 1. Albrecht Dürer, the inventor of ray tracing

Ray tracing is an embarrassingly parallel algorithm that is being increasingly used for rendering in modern games. A key factor in success of ray tracing has been the growth in GPUs. In this project, we implement a ray tracer on cuda using acceleration data structures.

Additional Key Words and Phrases: Ray tracing, cuda, GPU computing

1 LITERATURE REVIEW

Work on Ray tracing has been going on for decades. [1] suggested a process of shooting rays into the scene to get pixel colors and generate an image. [9] introduced a simple recursive Ray tracing model in 1980s. The original rays, called primary rays hit and object, a secondary ray may emit from these points and go in a direction depending on the angle of incidence of rays. In a recursive way, by processing the secondary rays, we get pixel colors simulating the effects like shadows, refractions and reflections.

1.1 Acceleration structures

The general algorithm for ray tracing is shown below:

```
#Loop-1
for pixel in pixels:
    ray = get_ray(camera, pixel)
    color = init_color()
    #Loop-2
    for object in objects:
        intersection = get_intersection(ray, object)
        if (instersection):
            color += get_color(intersection)
    pixel.set(color)
```

The actual bottleneck in performance of ray tracer comes from the inner loop-2. There are lots of redundant ray-object intersections. Work has been done in the field of using spatial data structures to reduce the number of redundant intersections. The general idea is to group objects into groups based on their spatial orientation. Intersection if first checked for a group. If the ray is not hitting the group, we can ignore all objects of that group. The algorithm is now modified to:

```
#Loop-1
for pixel in pixels:
    ray = get_ray(camera, pixel)
    color = init_color()
    #Loop-1a
    for group in groups:
        intersection = get_intersection(ray, group)
        if (instersection):
            #Loop-2
            for object in group:
                obj_intersection = get_intersection(ray, object)
                if obj_intersection:
                    color += get_color(obj_intersection)
    pixel.set(color)
```

1.1.1 Kd-tree. Kd-tree was introduced by [2] as a space partitioning data structure in a k-dimensional space. Kd-tree is a binary tree with each node as a k-dimensional point. Every non-leaf node is a hyperplane that divides the hyperspace into to half-spaces. The first kd-tree traversal algorithm was the Sequential Traversal [5]

1.1.2 BVH. BVH was proposed in [3]. BVH traversal is done through recursive descent. The performance of traversal is impacted by the order in which nodes are traversed. [7] compares [6], [4] and suggests a new method of traversal.

1.2 Performance Studies

[7] compares the GPU based traversal methods on kd-trees and uniform grids with bounding volume hierarchy traversal scheme. In all scenes, BVH is upto 9 times faster than uniform grid and kd-tree. Further it is the most memory efficient also.

[8] compares the performance of kd-tree with BVH on GPUs. The conclusions are:

- **normal to moderately complex scenes** BVH outperforms kd-tree
- **Complex scenes** kd-tree outperforms BVH

2 ALGORITHM DETAILS

Based on the comparisons, I will be exploring ray-tracing using BVH as acceleration data structure. I will explore multiple traversal techniques starting with the traversal algorithm explained in [7].

3 AVAILABLE IMPLEMENTATIONS

3.1 Cuda based Ray tracer

- https://github.com/ttsiodras/renderer-cuda
- https://github.com/boonemiller/CudaRayTracer
- https://github.com/magnificus/Ray

3.2 Cuda OpenGL interperatibility

- https://github.com/Forceflow/cuda2GLcore

4 MILESTONES FOR IMPLEMENTATION

The goal of the project is to implement a Ray tracer that would read a standard mesh representation of an object and render it using Cuda and OpenGL.

4.1 Evaluation-2

Milestone for second evaluation:

- **Ray-object Intersection**: Rendering the model.

A lot of logistic setup would be needed for this evaluation. Majority of that would include:

- **OpenGL setup**: Creating window and writing to OpenGL framebuffers from Cuda efficiently.
- **Mesh setup**: Setup a library for reading a mesh.

4.2 Evaluation-3

- **Raytracing acceleration**:
  - Acceleration data structures.
  - Using Cuda Profiler.
- **Shading and lighting** Implementing lighting and shading.

REFERENCES

[1] Arthur Appel. 1968. Some Techniques for Shading Machine Renderings of Solids. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference* (Atlantic City, New Jersey) *(AFIPS '68 (Spring))*. Association for Computing Machinery, New York, NY, USA, 37–45. https://doi.org/10.1145/1468075.1468082

[2] Jon Louis Bentley. 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM* 18, 9 (sep 1975), 509–517. https://doi.org/10.1145/361002.361007

[3] James H. Clark. 1976. Hierarchical Geometric Models for Visible Surface Algorithms. *Commun. ACM* 19, 10 (oct 1976), 547–554. https://doi.org/10.1145/360349.360354

[4] Jeffrey Goldsmith and John Salmon. 1987. Automatic Creation of Object Hierarchies for Ray Tracing. *IEEE Computer Graphics and Applications* 7, 5 (May 1987), 14–20. https://doi.org/10.1109/MCG.1987.276983

[5] Vlastimil Havran. 2000. *Heuristic Ray Shooting Algorithms*. Ph. D. Dissertation.

[6] Timothy L. Kay and James T. Kajiya. 1986. Ray Tracing Complex Scenes. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '86)*. Association for Computing Machinery, New York, NY, USA, 269–278. https://doi.org/10.1145/15922.15916

[7] Niels Thrane, Lars Ole Simonsen, and Advisor Peter Ørbæk. 2005. *A comparison of acceleration structures for GPU assisted ray tracing*. Technical Report.

[8] Marek Vinkler, Vlastimil Havran, and Jiří Bittner. 2016. Performance Comparison of Bounding Volume Hierarchies and Kd-Trees for GPU Ray Tracing: Bounding Volume Hierarchies versus Kd-Trees. *Computer Graphics Forum* 35, 8 (Dec 2016), 68–79. https://doi.org/10.1111/cgf.12776

[9] Turner Whitted. 1980. An Improved Illumination Model for Shaded Display. *Commun. ACM* 23, 6 (jun 1980), 343–349. https://doi.org/10.1145/358876.358882