International Conference on Artificial Reality and Telexistence
Eurographics Symposium on Virtual Environments (2023)
A. Campbell, C. Krogmeier, and G. Young (Editors)

A GPU Ray Tracing Implementation for Triangular Grid Primitives

Max von Buelow[1], Arjan Kuijper[1,2] and Dieter W. Fellner[1,2,3]

[1]Technical University of Darmstadt, Germany [2]Fraunhofer IGD, Germany
[3]Graz University of Technology, Institute of Computer Graphics and Knowledge Visualization, Austria

Abstract

Triangular grid primitives are a new technique for more efficient handling of memory intensive meshes, also called micro meshes in recent proprietary hardware implementations. This makes it a technique with high potential in the area of virtual environments where hardware capabilities are typically limited. In this poster, we focus on software ray tracing on GPUs and present a novel, easy-to-implement approach that uses a two-level bounding volume hierarchy (BVH) to accelerate these grids. The primary goal of our work is to make the technology more accessible by focusing on standard GPU devices without hardware ray tracing units. With our approach, we are able to encode geometry and BVH with approximately 7.5 bytes per triangle, reducing standard representations by a factor of 3.73 while reducing BVH construction time. Our data structure achieves a peak performance impact of 16 % for a three-level subdivision.

CCS Concepts

• ***Computing methodologies*** → ***Ray tracing;*** *Graphics processors;* • ***Theory of computation*** → ***Data compression;***

1. Introduction

Ray Tracing as a rendering approach [Whi79] is an important technique for transforming arbitrary scene descriptions such as triangle meshes into an image. It is used in many practical applications ranging from virtual reality to video games. Recently, it has become increasingly popular in the real-time domain, replacing classical rasterization, also because of its generally better capabilities for physically correct global illumination approximation on GPUs [PBD*10].

Structured primitives, such as triangular grid primitives, can be used to reduce scene memory consumption in areas where it is feasible to assume static geometry within a coarse domain. Although generating such representations from arbitrary topologies is an active research topic, we want to focus our work on rendering such structures using ray tracing. For structured grid primitives, quad meshes have traditionally been used as the base geometry [BVW21], more recently also in a triangular *micro mesh* representation [BP23]. While there are very recent works that implement these primitives [BP23], none of them focus on a pure software implementation that uses the GPU as a ray tracing device. The drawback of these is that they require newer hardware based on proprietary implementations, making the idea less accessible to older or less specialized devices such as smartphones or self-contained virtual environment devices that also benefit from ray tracing [OSR09].

In this poster, we present a data structure optimized for ray tracing triangular grid primitives on the GPU. The main idea of our data structure is to drastically reduce the memory footprint of connectivity data by exploiting the internal structure of such a grid. This basic structure is accelerated by a secondary level BVH using semi-implicitly stored bounding spheres that are built around the geometry during the subdivision recursion.

In summary, our contributions are:

- A joint geometry and BVH data structure for triangular grid primitives.
- Increased availability due to a simple design that is easy to implement on arbitrary GPU architectures.

2. Ray Tracing of Triangular Grids

As the name suggests, triangular grids can be thought of as primitives in our ray tracer, similar to standard triangles in a ray tracing pipeline. Consequently, we use a two-level BVH, where the top-level BVH is built on top of the list of primitives using the *surface area heuristic* (SAH) [MB90] and the bottom-level BVH resides inside a grid primitive to speed up the subdivision geometry. The geometric structure of our grid primitive data structure is largely based on the recursive definition of subdivision surfaces and can be seen in fig. 1, while the vertices define the geometry and the connectivity is implicitly given by the static structure of a grid. For the lowest level BVH, we construct bounding spheres around each of the four triangles in each subdivision level.

The main idea of our bottom-level hierarchy is to use a semi-implicitly derived sphere at position p and radius r for each triangle

DOI: 10.2312/egve.20231341

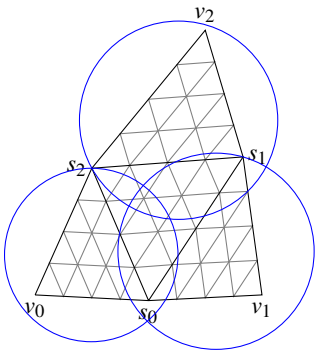

Figure 1: *Two-dimensional sketch of the geometric structure for three subdivisions. The blue circles indicate the bounding spheres for the first subdivision. Gray triangles indicate further subdivisions without spheres. The bounding sphere of the center triangle is omitted for a less cluttered visualization.*

spanned by vertices (v_0, v_1, v_2) as an enclosing volume and to align the BVH with the recursive subdivision aspect, which produces the set of vertices vertices V:

$$p = 1/3\,(v_0 + v_1 + v_2) \tag{1}$$

$$r = \max_{\hat{v}_i \in V} ||\hat{v}_i - p||_2 \tag{2}$$

As seen in eq. (1), p is constructed in such a way that it is a trivial arithmetic operation that can be computed efficiently, while r requires several iterations as defined in eq. (2). Therefore, we only explicitly store r as one single floating-point number and recompute p during traversal. All radii are stored in memory in a breadth first search (BFS) traversal order, which ensures that equal levels of detail are within the same region of memory. In addition, since all subtrees have the same subdivision depth, the BVH is a complete tree, making implicit indexing trivial. We have evaluated that it is optimal to exclude the last two layers containing 16 triangles from the BVH in order to take advantage of the tradeoff between intersecting unnecessary triangles and the additional sphere intersection overhead [MB90].

The geometry is largely based on an static indexed triangle list residing in constant memory. Since constant memory benefits from L1 caching due to high coherence on relatively small memory areas, accessing this single list is almost as fast as using registers. Inner vertices are stored per grid primitive, and boundary vertices are stored in a separate memory area to avoid redundancy.

The traversal of our structure is based on the *while-while* approach [AL09], which we implement slightly differently for the two-level BVH. The first inner while loop traverses the top-level hierarchy and the second traverses the bottom-level hierarchy and its leaves, rather than separating between inner and leaf nodes. Other configurations, such as three separate while loops for bottom-level, top-level, and leaves, proved less efficient and resulted in less device occupancy. The top-level hierarchies are traversed as usual.

Table 1: *Our results on five meshes with three levels of subdivision. We compare the ray tracing run-time performance impact, the compression rate gain and the construction time gain on multiple meshes.*

Mesh	RT impact	Comp. gain	Constr. gain
Bike	65.78 %	3.73	47.9
Dragon	34.95 %	3.73	45.28
Head	37.21 %	3.73	35.78
Armadillo	16.27 %	3.73	40.88
Sponza	68.25 %	3.73	42.65

3. Conclusion

In this poster, we presented a novel data structure that exploits the structure of triangular grid primitives and compresses their connectivity information to a minimum. Since our data structure implicitly preserves the subdivision recursion information, it is possible to limit to intermediate subdivision depths without regenerating the data structure, which could be useful in computationally constrained environments such as those typically used in virtual reality. Our results in table 1 show that our approach reduces standard representations by a factor of 3.73 for geometry and BVH, while we are able to speed up the construction time by a factor of 42 compared to SAH in the lowest levels for a subdivision depth of three. Our data structure has a performance impact between 16 % to 68 % for the same three-level subdivision depending on the rendered mesh, but the tradeoff of losing runtime performance for more efficient storage is a common behavior.

References

[AL09] AILA, TIMO and LAINE, SAMULI. "Understanding the efficiency of ray traversal on GPUs". *Proceedings of the Conference on High Performance Graphics 2009*. HPG '09. ACM, Aug. 2009. DOI: 10.1145/1572769.1572792 2.

[BP23] BENTHIN, CARSTEN and PETERS, CHRISTOPH. "Real-Time Ray Tracing of Micro-Poly Geometry with Hierarchical Level of Detail". *Computer Graphics Forum* (Aug. 2023). DOI: 10.1111/cgf.14868 1.

[BVW21] BENTHIN, CARSTEN, VAIDYANATHAN, KARTHIK, and WOOP, SVEN. "Ray Tracing Lossy Compressed Grid Primitives". *Eurographics 2021 - Short Papers*. The Eurographics Association, 2021. DOI: 10.2312/EGS.20211009 1.

[MB90] MACDONALD, J. DAVID and BOOTH, KELLOGG S. "Heuristics for ray tracing using space subdivision". *The Visual Computer* 6.3 (May 1990), 153–166. DOI: 10.1007/bf01911006 1, 2.

[OSR09] ODOM, CHRISTIAN N. S., SHETTY, NIKHIL J., and REINERS, DIRK. "Ray Traced Virtual Reality". *Advances in Visual Computing*. Springer Berlin Heidelberg, 2009, 1031–1042. DOI: 10.1007/978-3-642-10331-5_96 1.

[PBD*10] PARKER, STEVEN G., BIGLER, JAMES, DIETRICH, ANDREAS, et al. "OptiX. a general purpose ray tracing engine". *ACM Transactions on Graphics* 29.4 (July 2010), 1–13. DOI: 10.1145/1778765.1778803 1.

[Whi79] WHITTED, TURNER. "An improved illumination model for shaded display". *ACM SIGGRAPH Computer Graphics* 13.2 (Aug. 1979), 14. DOI: 10.1145/965103.807419 1.