Hybrid Ray/Voxel-Tracing Fixed Capacity Grids

Baktash Abdollah-shamshir-saz
baktash@toomuchvoltage.com
TooMuchVoltage Software Inc.

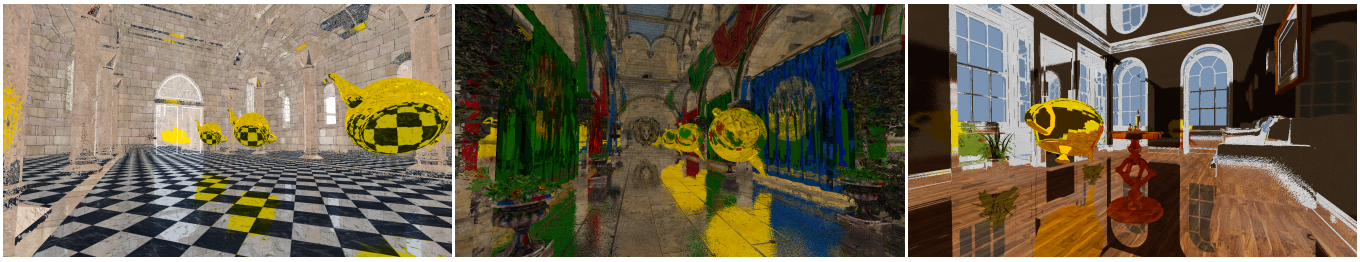

Figure 1: (From left to right) Sibenik, Sponza and Fireplace scenes rendered with a single glossy bounce off of a gather buffer using our technique. Sibenik and Sponza featured three animated teapots while Fireplace featured one. All scenes were rendered at a resolution of 1920x1080. We have achieved average frame costs ranging from 15 to 23 milliseconds on the above scenes on an AMD Radeon VII. Environments provided by the McGuire Computer Graphics Archive[McGuire 2017].

ABSTRACT

With the commercial availability of NVIDIA Co.'s RTX line of video cards capable of hardware accelerated ray tracing, a resurgence of interest surrounding ray tracing has appeared in the games industry. The caveat of NVIDIA's solution being tied to NVIDIA hardware has prompted some efforts of replication in video cards lacking any explicit hardware ray tracing acceleration. A notable attempt is that of CryTek GmbH in their Neon Noir demo. However, their solution as of this writing is only briefly discussed publicly and relies on geometry proxies. Our approach aims to provide a comprehensive solution that avoids reliance on lower resolution replicas.

CCS CONCEPTS

• **Computing methodologies** → **Computer graphics**.

KEYWORDS

ray tracing, grids, voxels, rendering

ACM Reference Format:
Baktash Abdollah-shamshir-saz. 2020. Hybrid Ray/Voxel-Tracing Fixed Capacity Grids. In *i3D '20: ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, May 05–07, 2020, San Francisco, CA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/1122445.1122456

i3D '20, May 05–07, 2020, San Francisco, CA

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Since the introduction of ray tracing for computer graphics by [Whitted 1980] researchers have been investigating various approaches for fast scene traversal. One of the earliest methods proposed was uniform grids by [Fujimoto et al. 1986]. Given its impracticality for large sparsely-populated environments, it has resulted in several refinements such as Two-Level Grids [Kalojanov et al. 2011] and Irregular Girds [Pérard-Gayot et al. 2017]. However, its simplicity may have contributed to its adoption by CryTek GmbH in their Neon Noir showcase – described in [Vladimir Kajalin 2019] – demonstrating highly interactive ray-tracing on modern GPUs lacking ray-tracing hardware. Inspired by Neon Noir we present this technique as a complete Vulkan implementation and address CryTek's reliance on geometry proxies which we find to be cumbersome or infeasible in scenarios such as procedural destruction.

2 ALGORITHM OVERVIEW

Our algorithm is straightforward and consists of three components: **scene build**, **scene update** and **scene trace**. The scene build phase constructs the grid for all static components of the scene and happens mostly once. It may be repeated in the event of a major update to the static scene. The scene update phase clears all references to dynamic scene components from the grid and re-adds them to account for motion. The scene trace step proceeds after scene update and produces the traced imagery.

2.1 Scene Build

We commence by creating a Vulkan 3D image representing a grid covering the entire scene with every texel being a potential triangle reference. The 'x' dimension is stretched to contain a scene-dependent maximum number of triangle references per cell. The grid granularity and the cell capacity are currently chosen manually. The image has the single channel 32 bit unsigned integer format to enable atomic operations. The lower 16 bits of every reference

index a triangle within a geometry Shader Storage Buffer Object (SSBO) while the higher 16 bits index an instance within a variable count descriptor set. A value of 0xFFFFFFFF indicates a blank reference. The build phase is a raster pass and in addition to its common vertex stage transformations, every vertex's instance ID (gl_InstanceIndex) and triangle ID (gl_VertexIndex/3) are passed down as well with a flat qualifier. The flat qualifier ensures correctness of values further in the pipeline and reduces pressure on hardware interpolators. Grid building happens inside geometry shader invocations enabling wider integration with our implementation of [Crassin and Green 2012] while presenting a familiar rasterization interface to the operation. The practitioner may opt to use compute shaders for performance considerations. The transformed triangle is written into its corresponding geometry SSBO and subsequently all cells overlapping its AABB are tested for overlap with its plane using slightly enlarged cell bounding spheres. This enlargement helps with our non-watertight trace loop later. Plane intersecting cells commence a search to write the triangle's reference ID using imageAtomicCompSwap() into an empty cell. An additional reserved texel can be used to flag saturated cells though this would not reduce intersection cost during trace as we iterate through all non-blank references.

2.2 Scene Update

Scene updates start off by launching a compute shader that clears dynamic primitive references from cells. This is achieved by providing a 'comparison offset' – which is the total number of static geometry instances shifted to the left by 16 bits – to the shader and every reference higher or equal to this in integer value is blanked using imageAtomicExchange(). We use a small Uniform Buffer Object (UBO) to supply this to the shader though a push constant may be more performant. The clear loop is terminated when encountering the first blank value. We ultimately determined a workgroup size of 4x4x4 as optimal for this compute shader. Once dynamic primitives are cleared, a procedure nearly identical to the build step is launched with two exceptions: a) it targets the same 3D image created during the build step and b) every reference's instance ID is increased by the number of static instances. In some cases it may be preferred to only clear certain references from the grid (e.g. only objects that have moved or changed) and re-submit those alone for update. We are considering this as future improvement.

2.3 Scene Trace

We initiate our ray tracing by casting a ray segment the size of a cell edge bounced off of a gather buffer. Visited cells are tracked to prevent potential revisits. Majority of our ray advancements result from single blank reference reads as most scenes are sparsely populated. Since our trace steps are not watertight we avoid generating hierarchies of occupancy and larger steps. Variable count descriptor sets representing transformed instance geometry and instance materials are attached alongside instance property SSBOs for static and dynamic instances. Both variable count descriptor sets are collations of dynamic atop static variable count descriptor sets. All primitives in an occupied cell are tested against an elongated version of the ray segment using [Möller and Trumbore 2005]. The instance ID of a successful hit is extracted and its instance properties are checked for the alpha keying flag. If this flag is enabled, a diffuse map sample fetch is performed to determine whether the hit should be ignored. Our choice for computing barycentric coordinates is Cramer's Rule detailed in [Ericson 2004]. If a hit is not ignored it counts towards minimizing the current intersection time. An intersection time below 1.0 reports a closest hit. If the cell queried happens to be saturated and no intersections were reported, a false intersection is reported at the halfway point of the ray segment using the last encountered primitive. This is our gap-filling technique intended to fill holes with textured voxels. This procedure does not kick in unless the ray segment has travelled far enough to avoid intersecting the voxels covering the current surface. This is determined via projecting the current ray segment onto the normal of the surface it was launched from. Various primitive spillover schemes were tried to avoid gap-filling. However, the performance penalties appeared unacceptable.

3 RESULTS AND CONCLUSION

We have presented a comprehensive ray tracing solution, hybrid ray/voxel-tracing fixed capacity grids inspired by CryTek's Neon Noir ray tracing demo. This approach has achieved highly interactive frame rates on AMD Radeon VII ranging from 15 to 23 milliseconds while avenues of improvement have been considered for all components involved. Our technique can serve as a foundation for ray-tracing on capable GPUs without hardware ray-tracing support.

ACKNOWLEDGMENTS

The author would like to thank Matthäus G. Chajdas of AMD for assisting in debugging some of the issues encountered on AMD Radeon VII. Special thanks are extended to Azam Khan for reviewing this writing.

REFERENCES

Cyril Crassin and Simon Green. 2012. . CRC Press, Patrick Cozzi and Christophe Riccio. http://www.seas.upenn.edu/~pcozzi/OpenGLInsights/OpenGLInsights-SparseVoxelization.pdf

Christer Ericson. 2004. *Real-Time Collision Detection.* CRC Press, Inc., Boca Raton, FL, USA.

Akira Fujimoto, Takayuki Tanaka, and Kansei Iwata. 1986. ARTS: Accelerated Ray-Tracing System. *IEEE Computer Graphics and Applications* 6 (1986), 16–26.

Javor Kalojanov, Markus Billeter, and Philipp Slusallek. 2011. Two-level grids for ray tracing on GPUs. *Comput. Graph. Forum* 30 (04 2011), 307–314. https://doi.org/10.1111/j.1467-8659.2011.01862.x

Morgan McGuire. 2017. *Computer Graphics Archive.* https://casual-effects.com/data

Tomas Möller and Ben Trumbore. 2005. Fast, Minimum Storage Ray/Triangle Intersection. In *ACM SIGGRAPH 2005 Courses (SIGGRAPH '05).* ACM, New York, NY, USA, Article 7. https://doi.org/10.1145/1198555.1198746

Arsène Pérard-Gayot, Javor Kalojanov, and Philipp Slusallek. 2017. GPU Ray Tracing Using Irregular Grids. *Comput. Graph. Forum* 36, 2 (May 2017), 477–486. https://doi.org/10.1111/cgf.13142

Ron Frölich Vladimir Kajalin. 2019. How we made Neon Noir - Ray Traced Reflections in CRYENGINE and more! https://www.cryengine.com/news/view/how-we-made-neon-noir-ray-traced-reflections-in-cryengine-and-more. Accessed: 2019-12-10.

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