Point in Polygon Tests Using Hardware Accelerated Ray Tracing

Moritz Laass
moritz.laass@tum.de
Technical University of Munich Department of Aerospace and Geodesy
Big Geospatial Data Management
Germany

ABSTRACT

Recent generations of GPUs have seen the introduction of hardware-accelerated ray tracing algorithms that are suitable for real-time use. They provide hardware for massively parallel ray-geometry intersection computations, indicating a highly optimized spatial data structure derived from arbitrary triangle-based geometries. Spatial join is an ubiquitous problem in spatial databases, GIS applications, spatial statistics, and similar applications. On a fundamental level, spatial joins are based on point in polygon tests (PIP). We suggest exploiting the capabilities of ray tracing hardware to perform fast parallel point in polygon tests in order to implement hardware-accelerated spatial joins.

CCS CONCEPTS

• **Information systems** → **Computing platforms**; **Geographic information systems**.

KEYWORDS

point in polygon test, RTX, spatial join, ray-tracing, GPU

ACM Reference Format:
Moritz Laass. 2021. Point in Polygon Tests Using Hardware Accelerated Ray Tracing. In *29th International Conference on Advances in Geographic Information Systems (SIGSPATIAL '21), November 2–5, 2021, Beijing, China.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3474717.3486796

1 INTRODUCTION

Spatial data has many applications in science and industry. One important and ubiquitous operation on spatial data is spatial join [1] an operation to combine two or more spatial datasets using a spatial predicate. One fundamental part of spatial join is a point in polygon join algorithm, although spatial join is much more. The classical algorithm for point in polygon tests is known as Jordan test [3], which works by intersecting rays with polygon edges. While there are existing techniques using GPUs to accelerate spatial join, we propose and investigate the exploitation of current generation GPUs with hardware enabled *ray tracing* to achieve highly performant point in polygon tests on large numbers of points and polygons in parallel. We introduce a novel technique that promises great performance, show the limitations and solutions, as well as promising experimental results and end with a conclusion where we propose some vendor side improvements that can be useful for this and other scientific applications.

2 MAPPING PIP TO 3D RAY TRACING OPERATIONS

In order to solve the point in polygon problem on a GPU optimized for 3D rendering, the data needs to be transformed in an appropriate way and the problem needs to be formulated in a way, so that it can be executed on this hardware in the most efficient way.

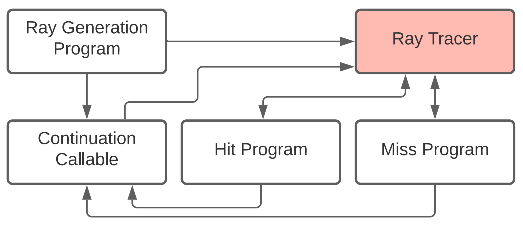


Figure 1: Simplified ray tracing architecture.[2]

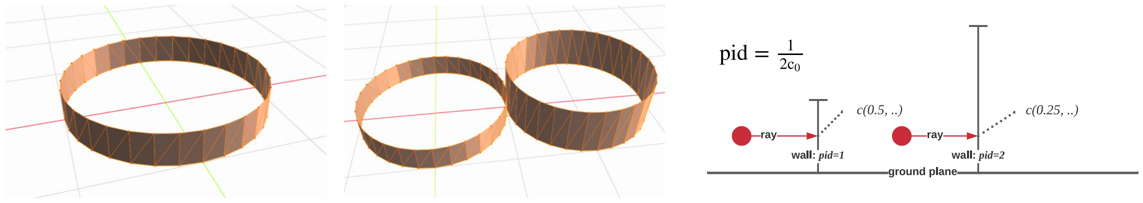
GPU programs are small independent functions that execute at different steps in the ray tracing pipeline, see figure 1.

To do point in polygon tests, a ray is generated for each point in parallel to the ground-plane and polygons are transformed into 3D representation by extrusion along the 3rd spacial dimension, thus creating walls see figure 2(a). The ray tracer will call the *closest hit program* for rays intersecting a wall or the *miss program* for any ray that is not intersecting with geometry. The *closest hit program* can decide between front-facing and back-facing intersection, which is used to detect whether the point is inside the polygon or not.

To minimize non-local data access, the polygon id is encoded as the wall-height (see figure 2(b)) and reconstructed by the *closest hit program* using the barycentric coordinates of the intersection point, see figure 2(c). While this limits the amount of polygons in the experiments to 2^{16}, before floating point precision leads to inaccuracies, the alternative of using a triangle index, not only is slower due to memory access patterns, but also comes with a higher memory footprint.

3 IMPLEMENTATION CHALLENGES

The implementation posed several challenges, some of which will be discussed here. Additionally we ran into problems implementing data streaming which led to hard to track down data alignment issues, using available driver versions.



(a) Extruded polygon on the ground plane. (b) Polygons with two different extrusion heights. (c) Wall height encoding of polygon ids

Figure 2: 3D geometry generation from 2D polygons.

3.1 Z-Fighting

Traditional rasterization-based pipelines have a well known problem called z-fighting, which happens when the the distance between triangles is smaller than the precision of the z-buffer, that is used to enforce drawing depth on a per pixel basis. A similar effect can be observed in the ray tracing pipeline, which is a problem for polygons that share borders. The following is a collection of possible solutions, some of which have been implemented successfully

3.1.1 Stochastic Multi Rays. For each point several rays are launched in random directions or an evenly distributed star pattern. If any of the rays suggests that the point is inside a polygon it can be taken as truth, only if it classified as outside a certain percentage is wrong. This percentage is dependant on the number of rays used. Outside points can either be discarded or retested using traditional methods. For spatial statistics it could be regarded as noise.

3.1.2 Negative Buffer. A practical solution for many scenarios is to add a preprocessing step which introduces a negative buffer operation on the polygon data. This will shrink all polygons by a predefined amount that should be slightly higher than the precision error of the the ray intersection algorithm. This approach works well, though it bears the problem of points falling into the buffer, which can be handled as in 3.1.1.

3.1.3 Ray Continuation. Ray continuation is the idea of continuing a ray, once it intersects with a wall that suggests we are outside a polygon. This new ray should then either intersect with the wall of the polygon we are inside, which will be very close, which means we can safely assume that we are inside the given polygon, or it intersects with the opposite side wall of the polygon that was hit from the outside, which suggests the starting point is outside a polygon. This again is a technique that is prone to error due to inaccuracies in the ray tracer. As ray continuation is slow, an *all hit program* or *many hit program* for the *n* closest hits would enable this idea to great effect.

3.2 Nested Polygons

To account for nested polygons the continuation callable can be exploited, continuing the ray until the miss program is called, counting the intersections along the way. A vendor side addition of an *all hit program* in addition to the *any hit program* could improve this step as well. Currently the maximal nesting depth needs to be predefined as ray payloads have a fixed size, and ray continuation is generally slow.

4 EXPERIMENTS

Experiments suggest that scaling is linear for polygon geometries and points. Due to problems streaming larger datasets, experiments are incomplete as of yet and have only been performed using synthetic data consisting of random voronoi cells and randomly generated points.

5 CONCLUSION

We show that ray tracing enabled GPUs can be exploited to perform parallel point in polygon tests on large numbers of points and polygons. While single batches worked great, a streaming implementation came across problems that might need resolution on vendor side. We also propose the additional feature of an *all hit program* or *many hit program* to efficiently solve nested polygons and problems caused by inaccuracies in the ray tracer.

REFERENCES

[1] Edwin H. Jacox and Hanan Samet. [n.d.]. Spatial join techniques. 32, 1 ([n. d.]), 7–es. https://doi.org/10.1145/1206049.1206056
[2] Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. [n.d.]. OptiX: a general purpose ray tracing engine. 29, 4 ([n. d.]), 66:1–66:13. https://doi.org/10.1145/1778765.1778803
[3] M. Shimrat. [n.d.]. Algorithm 112: Position of point relative to polygon. 5, 8 ([n. d.]), 434. https://doi.org/10.1145/368637.368653

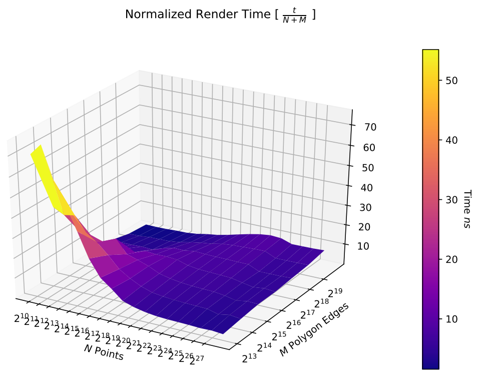


Figure 3: Performance matrix per point per polygon edge of point in polygon tests. Comparing to QGIS on the same machine, reveals magnitudes of performance difference.