Evaluating the usage of exact queries on 3D spatial databases

Matheus A. de Oliveira[1], Marcelo de M. Menezes[1], Salles V. G. de Magalhães[1],Bruno F. Coelho[1]

[1]Departamento de Informática, Universidade Federal de Viçosa (UFV)
Campus da UFV, Viçosa, MG, Brazil

{matheus.a.aguilar, marcelo.menezes, salles, bruno.f.coelho}@ufv.br

Abstract. *The availability of big geospatial databases has increased the necessity of having efficient algorithms for processing them. Furthermore, as datasets grow, the chance of having failures due to rounding-errors increases, which makes exact (but typically slower) algorithms more important. This paper presents an evaluation of PostGIS exact and approximate backends. In our experiments, the exact backend was up to* 27 *times slower than the approximate one. We also observed that the straightforward usage of some spatial queries may lead to a poor performance, what requires more care when they are programmed. These results suggest applications requiring exact computation could benefit from the development of faster exact backends, which is the long-term goal of the research project this paper is part of.*

1. Introduction

The ability of storing and processing 3D data in Geographic Information Systems (GISs) has become very important. This type of modeling is especially necessary in areas such as urban planning, environmental monitoring, telecommunications, rescue operations, landscape planning, geology and mining [Zlatanova et al. 2002].

Despite such importance, this processing still faces a major challenge: performing robust computation while maintaining a good performance. This is fundamental for current data sets, since their size and complexity have been increasing, which makes them more prone to roundoff errors caused by floating point arithmetic. This kind of error is especially problematic, since it can propagate and generate inconsistent results or even cause systems to crash [Goodchild and Gopal 1989] .

This work has been developed in the scope of a project whose long-term goal is to optimize (using *GPUs*) exact geometric algorithms and spatial databases.

This paper describes our first case studies: the calculation of exact 3D intersection between segments and triangulated meshes and exact 3D intersection between triangles, using PostGIS, an extension for spatial data from the PostegreSQL Database Management System (DBMS), through its exact backend SFCGAL. This DBMS was chosen because it is open source, widely used and is one of the systems with the best support for spatial data with both exact and approximate arithmetic [Real et al. 2019]. The goal was to evaluate the support for 3D spatial data and the performance obtained by different queries.

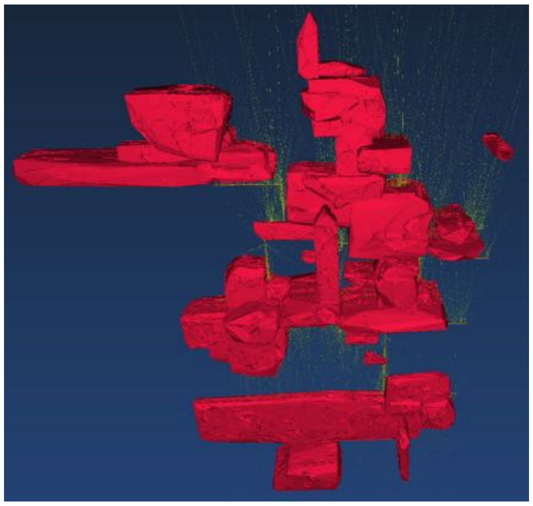

Figure 1. An example of a synthetic mine model. Source: [Real 2020]

2. Case Studies

As a case study, the following problems were considered: given a set of line segments and a set of triangulated meshes, both in 3D, to detect which pairs of segments and triangles do intersect and given two sets of triangles, also in 3D, find the pairs of intersecting triangles. These problems have several applications in computational geometry, GIS, CAD, etc. An example of application is studying the interaction of geological objects in a 3D mining model.

Thus, a 3D mining model provided by [Real et al. 2019] was employed in our experiments (Figure 1 illustrates this dataset, drill holes are in green and minerals are in red). In this domain, the intersection of segments with 3D objects is employed by geologists for studies related to the intersections between survey drill holes (represented by segments) and mineral layers (represented by triangulated meshes). The goal is to verify which layers of minerals were reached by each one of the drill holes in order to estimate the amount of ore that can be extracted.

The intersection of pairs of triangles, on the other hand, may be employed to to intersect a mining model with a shape representing an excavation area. This computation would result in shapes representing the minerals that could be extracted by the excavation.

Geologists typically use spatial queries (such the ones provided by PostGIS) to do these studies. However, the processing time spent by these systems is often prohibitive [Real et al. 2019].

2.1. PostGIS

The tool used to perform the experiments was PostGIS, since it has a large amount of resources for 3D geometries, such as: geometric data types, spatial indexes and intersection functions, unlike other database management systems [Real et al. 2019].

In addition, PostGIS has another important functionality to obtain robustness in spatial data: a backend capable of performing operations with exact arithmetic, the SFCGAL. This backend employs the CGAL library to achieve exactness.

The following query illustrates a problem that occurs when computation is performed using floating point arithmetic (subject to errors). This query checks whether the intersection point between two segments intersects one of the segments (clearly such an intersection occurs). However, as mentioned by [Mercier 2013], this specific intersection is not detected when the (inaccurate) GEOS backend is used (on the other hand, it is correctly computed by SFCGAL). While these errors may be rare, they make software relying on floating-point arithmetic unreliable.

```
SELECT ST_Intersects(ST_Intersection(
  'LINESTRING(0 0,2 1)'::geometry,'LINESTRING(1 0,0 1)'::geometry),
  'LINESTRING(0 0,2 1)'::geometry);
```

SFCGAL uses the *kernel* of CGAL which ensures that both geometric constructions (for example, points constructed as the intersection of two segments) and predicates (e.g., detecting whether two segments do intersect) are performed exactly. However, such an implementation has some challenges, such as the difficulty of parallelization (the *kernel* is not *thread-safe*) and the computational cost is higher than that obtained with floating point arithmetic[The CGAL Project 2019].

In the experiments, both GEOS (inaccurate) and SFCGAL (exact) backends were evaluated in order to assess the impact of exact arithmetic on the performance of the queries.

The data was modeled using tables with the following internal structure: the table of drillholes was composed of objects of type *LineString Z* whereas the mineral layers were divided into a set of indexed triangles, which were stored in a table of objects of type *Polygon Z*.

In addition, two types of spatial indices were evaluated in the table geometries, a 2D GiST index (PostGIS default), that drops the Z coordinate and is applied to a projection of the objects in a plane, and a GiST 3D index, that uses all the coordinates. Thus, it focused on assessing which index was most suitable for this data set, since the 3D index is slightly more computationally expensive, but allows for greater filtering on queries.

Another strategy employed in the experiments was to change the cost value of the intersection functions to $100,000$, in order to force the query planner to perform parallel scans instead of sequential ones (as suggested by [Real et al. 2019]).

3. Results and Discussion

The main idea of the experiments was simulate queries that would be useful in the field of mining. First, we considered the problem of detecting intersections between line segments (drill holes) and 3D objects (minerals) represented by triangles. In the segment/object intersection experiments we employed the dataset provided by [Real et al. 2019], which contains $7,846$ segments, 71 objects and the objects are composed of a total of $3,215,052$ triangles. In the triangle/triangle intersection experiments we employed two datasets (generated with the syntetic mine maker available at [Real 2020]) representing mines and containing, respectively, $125,258$

and 138,964 triangles. All the experiments were performed on a machine with a Ryzen 5 1600 AMD CPU with 6 cores at 3.2 GHz, 16*GB* of RAM, Kingston A400 SSD, Ubuntu Linux 20.04, PostgreSQL 12.4 and PostGIS 2.5.5. We evaluated both the exact (SFCGAL 1.3.8) and approximate (GEOS 3.8.0) PostGIS backends.

This could be evaluated by selecting pairs of segments and objects that do intersect. However, PostGIS spatial index would index the objects, which leads to a poor performance because, after culling the pairs of segments/objects that may intersect, PostGIS would process each pair, potentially testing the segment for intersection against all the triangles in each object. To improve this performance, we created a table of triangles (each row contains a triangle, its id and *objectId* the id of the 3D object it belongs to) with a spatial index on the geometry. Since the spatial index is now on the triangle level, it can perform a better culling. Considering this table, the intersections can be found with the query *SELECT DISTINCT s.id, t.objectId FROM Triangles t, Segments s WHERE ST_3DIntersects(s.geom, t.geom)* (this approach will be referred as *DISTINCT* in this section).

A drawback with the previous strategy is that, given a segment s, the query planner tests s for intersection with many triangles from the same $3D$ object and, only after all intersections are detected, the unique intersections are filtered. In order to try to obtain a better performance, one could try to employ an approach using an exists clause in order to try to avoid this. Thus, we also evaluated the following approach: *SELECT s.id, o.id FROM Segments s, Objects o WHERE o.geom && s.geom AND EXISTS(SELECT 1 FROM Triangles t WHERE t.objectId = g.id AND s.geom ST_3DIntersects(t.geom, s.geom))* (this approach will be referred as *EXISTS*). This strategy employs two levels of indexing: the *select* clauses selects pairs of segments and objects that may intersect (using an indexed bounding-box check on the object level). Then, for each pair of segment s and object o that may intersect, the exists clause checks if there exists a triangle t from o intersecting s (this step employs the indexing at the triangle level).

The previous two queries present a pitfall that may degrade the performance of naive solutions: even 3D predicates (such as *ST_3DIntersects*) employ 2D indices in PostGIS by default (even when a 3D index is available). Internally, *ST_3DIntersects* is implemented using a 2D bouding-box intersection test using the && operator (which employs a 2D index) followed by a call to the _ST_3DIntersects function that tests the pair for intersection after the bounding-box detects a potential intersection. Thus, the culling is performed by evaluating the projection of the geometric data onto the xy plane.

In order to actually use a 3D index, one should add to the query a 3D bounding-box intersection test (using the &&& operator). We evaluated queries using both the 2D and 3D indices in order to show how the performance of a naive solution could be affected.

Table 1 presents the times (in seconds) obtained by these two approaches using the two kinds of indices and the two options of backend.

As it can be seen, the 3D index significantly reduces the running times in comparison with the 2D index. The performance improvement is higher when the

	2D index		3D index	
Query	GEOS	SFCGAL	GEOS	SFCGAL
seg/tri (DISTINCT)	71.7	2352.8	26.7	220.4
seg/tri (EXISTS)	238.1	26354.8	167.6	318.0
Triangle/triangle	25.8	1625.0	3.1	83.4

Table 1. Times (in seconds) for detecting segment/triangle intersections (first and second rows) using the two different queries (DISTINCT and EXISTS) and for testing pairs of triangles for intersection (third row).

Index	Without the Index	With the index
2D	25,225,297,992	89,523,915
3D	25,225,297,992	7,571,816

Table 2. Number of pairs of segments and triangles tested for intersection considering the 2D and 3D indices

exact backend is employed (for example, considering the *DISTINCT* queries, using the 3D index leads to a running time 11 times smaller when SFCGAL is employed, while the difference when GEOS is employed is 3 times). This can be explained because of the higher cost of evaluating geometric predicates using SFCGAL associated to the better culling of the 3D index, which performs a more significant reduction (in comparison with the 2D index) in the number of intersection predicates that actually need to be evaluated after the culling.

Considering the 3D index and the fastest query (the *DISTINCT* one), the exact backend was 8 times slower than the inexact one. While in some applications this difference is acceptable, in big datasets and applications requiring fast answers this may not be suitable.

Table 2 presents the number of intersection tests performed when the 2D and 3D indices are employed. As it can be seen, the $3D$ index reduces in 12 times the number of pairs being evaluated (this reduction explains the performance improvement obtained with the 3D index) in comparison with the 2D index (the default one employed by PostGIS).

Considering the triangle/triangle intersection tests (third row of Table 1), we employed a straightforward query for the tests: *SELECT COUNT(*) FROM triangles1 AS t1, triangles2 AS t2 WHERE t1.geom OP t2.geom AND _st_3dintersects(t1.geom, t2.geom);*, where OP is && for the 2D index and &&& for the 3D index.

Since testing a pair of triangles for intersection employs more arithmetic operations than intersecting segments with triangles, the performance difference between the exact and approximate backends was more noticeable than in the segment/triangle tests. Considering the $3D$ index, SFCGAL was 27 times slower than GEOS.

4. Conclusions and future work

This paper presented a performance analysis of PostGIS over 3D spatial data on a mining dataset. Our experiments have showed that some naive queries performed on PostGIS could present a high performance penalty. For example, applying a trivial intersection test on 3D data uses, by default, 2D bounding-box tests and indices, which was up to 83 times slower than the query employing the 3D index.

We also evaluated PostGIS exact and approximate backends. Experiments in two kinds of analysis showed a performance difference of 8 and 27 times between the two backends (when a 3D index was employed). This suggests employing faster techniques for exact computation could benefit applications demanding both exactness and performance. This could be particularly important for big datasets containing millions of features.

Researchers have recently been employing GPUs for accelerating queries employing the approximate backend [Real et al. 2019]. Similarly, in a previous paper we have proposed the use of GPUs for accelerating the exact evaluation of geometric predicates [Menezes et al. 2019], which led to a performance improvement of up to 40 times over the sequential implementation. As future work, we intend to combine the two ideas, i.e., accelerate the exact PostGIS backend with GPUs. Thus, GIS applications requiring exactness could benefit from this performance improvements while also benefiting from the modularity and simplicity of a DBMS.

References

Goodchild, M. F. and Gopal, S. (1989). *The accuracy of spatial databases*. CRC Press.

Menezes, M. M., Magalhães, S. V. G., Franklin, W. R., de Oliveira, M. A., and Chichorro, R. E. O. B. (2019). Accelerating the exact evaluation of geometric predicates with GPUs. In *28th International Meshing Roundtable*, Buffalo, NY, USA.

Mercier, H. (2013). 3d and exact geometries for PostGIS. `https://wiki.postgresql.org/images/3/36/Postgis_3d_pgday2013_hm.pdf`. (Retrieved on 09/02/2020).

Real, L. C. V. (2020). Synthetic mine maker. `https://github.com/lucasvr/synthetic-mine-maker`. (Retrieved on 09/24/2020).

Real, L. C. V., Silva, B., Meliksetian, D. S., and Sacchi, K. (2019). Large-scale 3d geospatial processing made possible. In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 199–208.

The CGAL Project (2019). *CGAL User and Reference Manual*. CGAL Editorial Board, 4.14.1 edition.

Zlatanova, S., Rahman, A., and Pilouk, M. (2002). 3d gis: current status and perspectives. *International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences*, 34(4):66–71.