An efficient algorithm for computing the exact overlay of triangulations

Salles Viana Gomes Magalhães
Supervised by: W. Randolph Franklin
Rensselaer Polytechnic Institute
Troy, NY, USA
vianas2@rpi.edu

ABSTRACT

This paper presents a proposal for the development of 3D-EPUG-Overlay, an efficient algorithm for exactly computing the intersection of 3D triangulations. 3D-EPUG-Overlay will be innovative because of two reasons. First, it will use arbitrary-precision rational numbers to represent and process spatial data, thereby completely avoiding round-off errors caused by floating-point numbers, what could create topological impossibilities. The use of rationals goes beyond merely using existing packages, which are inefficient when used in parallel on large problems. Second, for efficiency, 3D-EPUG-Overlay will use high performance computing and a uniform grid to index the data.

In order to validate these ideas, we developed and implemented EPUG-Overlay, a version of 3D-EPUG-Overlay for overlaying polygonal maps. EPUG-Overlay uses a two-level uniform grid for indexing the edges in the maps and high performance computing to accelerate the computations. Preliminary experiments showed that EPUG-Overlay was faster than GRASS GIS, even though GRASS uses inexact arithmetic. This suggests that the 3D version will also be efficient.

Categories and Subject Descriptors

F.2.2 [**Nonnumerical Algorithms and Problems**]: Geometrical problems and computations

General Terms

Algorithms, Experimentation, Performance

Keywords

Map overlay, high performance computing, exact arithmetic

1. INTRODUCTION

Computing intersections is a very important boolean operation supported by many GIS and CAD systems. Given a pair of maps A and B, that are composed of sets of faces or polygons representing partitions of the E^2 plane, the intersection of A with B is a map C where each polygon is the intersection of a polygon of A with a polygon of B. For example, the intersection of a map representing the states of the United States with a map representing the American drainage basins is another map where the polygons represent the portion of each basin that is in each state. Map intersections can be computed using algorithms such as the ones based on the plane sweep paradigm [16].

SIGSPATIAL'15, November 03-06 2015, Bellevue, WA, USA
ACM 978-1-4503-3980-3/15/11.
http://dx.doi.org/10.1145/2855680.2855840

Boolean operations also extend to 3D objects. For example, if a set of 3D solids A represents the layers of soil in a given region and another set B contains one polyhedron representing a section of the soil that will be excavated, the intersection between A and B will represent the different layers of soil that will be extracted during the digging.

The ability of representing and processing 3D data in GIS is important in several area such as urban planning, transportation and millitary planning, etc [19]. A popular way to represent these objects is with triangular meshes [5], where the boundary of the solids are represented by a set of triangles that are connected in their common edges and vertices.

However, according to Feito et al. [5], although 3D models have been widely used in computer science, processing them is still a challenge. Due to the algorithm implementation complexity (that usually need to handle several special cases), the necessity of processing big volumes of data and precision problems caused by floating point arithmetic, software packages occasionally "fail to give a correct result, or they refuse to give a result at all" [5]. This is a challenge particularly in the problem of overlaying triangulations.

There are several possible strategies to compute 3D overlays. For example, when only a simple approximation of the overlay is enough, a common technique consists in converting the objects to a volumetric representation using voxels and, then, computing the voxels overlay. For computing the exact overlay, a common strategy is to use indexing to accelerate spatial operations performed during the overlay process (such as computing the triangle-triangle intersection). For example, Feito et al. [5] and Mei et al. [14] use octrees to intersect triangulations while Jing et al. [18] use Oriented Bounding Boxes trees (OBBs).

While these methods are usually considered exact (in the sense that they do not use approximation techniques such as voxels) robustness cannot be always guaranteed due to floating point errors. For example, the algorithm presented in [5] uses a tolerance to process floating-point numbers what, as it will be mentioned in section 2, is not guaranteed to always work.

Even though in some situations an approximate algorithm is acceptable, it is often important to have an algorithm that is both efficient and robust. For example, overlay algorithms are frequently used as subroutines for other algorithms and, therefore, if their outputs are not exact or they are too slow, these problems may propagate to the algorithms using them.

The main goal of this work is to develop an efficient and robust algorithm for exactly computing the overlay between objects represented by triangulations. We intend to use exact arithmetic to completely avoid errors caused by floating point numbers. Special cases will be treated using *Simulation of Simplicity* (SoS) [4]. Since the use of exact arithmetic is expected to add an overhead to the program, we intend to use efficient indexing techniques and High Performance Computing (HPC) to mitigate this overhead.

As a proof of concept, we have already developed a 2D map intersection algorithm (named EPUG-OVERLAY), that can efficiently compute the intersection between 2 maps using exact arithmetic. EPUG-OVERLAY uses the same techniques we intend to use for 3D triangulations overlay. That is, to mitigate the performance overhead associated with the exact arithmetic we propose to use a uniform grid for indexing spatial data, SoS for efficiently treating the special cases and parallelism to explore the parallel computing capability of current hardware. As a next step, our objective is to extend the ideas used in EPUG-OVERLAY to compute the intersection between 3D triangulations.

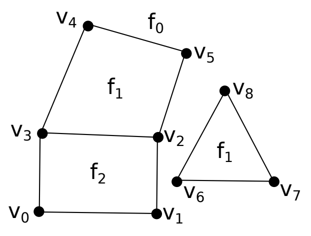
2. ROUNDOFF ERRORS IN GIS

Usually, non-integer numbers are approximately represented in computers with floating-point numbers. The difference between the value of a non-integer number and its approximation is often referred as roundoff error. Even though these errors are usually small, arithmetic operations frequently create more errors and, thus, a set of operations performed in sequence usually leads to larger errors.

The presence of roundoff errors in computer programs often create serious consequences in diverse fields such as the failure of the first Ariane V rocket [1], the failure of the Patriot missile defense system [17], etc. In geometry, roundoff errors can generate topological inconsistencies causing globally impossible results for predicates like point inside polygon.

Several techniques have been proposed in order to overcome this problem. The simplest one consists of using an ϵ tolerance to consider two values x and y are equal if $|x - y| \leq \epsilon$. However this is a formal mess because equality is no longer transitive, nor invariant under scaling.

Boissonat [2] describes a robust implementation of the plane sweep approach for intersecting segments using triple of the precision of the input data. Li [13] presents the Exact Geometric Computation (EGC) model, which represents mathematical objects using algebraic numbers to perform computations without errors. This model has some interesting features but its main drawback is the performance penalty. Even determining the sign of an expression is nontrivial.



Map representation:
(v_0,v_1,f_2,f_0) ; (v_1,v_2,f_2,f_0);
(v_2,v_3,f_2,f_1) ; (v_3,v_0,f_2,f_0);
(v_3,v_4,f_0,f_1) ; (v_4,v_5,f_0,f_1);
(v_5,v_2,f_0,f_1) ; (v_6,v_7,f_1,f_0);
(v_7,v_8,f_1,f_0) ; (v_8,v_6,f_1,f_0);

Figure 1: Example of a 2D map: each edge e is represented by a quadruple (v_i,v_j,f_l,f_r), where v_i and v_j are the vertices of e and f_l and f_r are, respectively, the faces to the left and to the right of e.

Another technique is snap rounding (SP) [10], whose basic idea is to use some rounding method to convert an arbitrary precision arrangement of segments into a fixed-precision representation. Snap rounding has been used in GIS packages such as GRASS, but it can generate some inconsistencies by changing the map topology.

The formally proper way to effectively eliminate roundoff errors and guarantee algorithm robustness is to use exact computation based on rational number with arbitrary precision [11, 12]. In this work we intend to develop 2D and 3D overlay algorithms that are efficient enough to perform the computations using rationals.

3. DATA REPRESENTATION

We intend to represent the maps using simple representations that do not keep global topology information. This is important because the more explicit topological information is stored the more difficult it is to guarantee robustness [5].

A 2D map will be composed of a set of faces labeled with identification numbers. A face does not need to be a connected set and, therefore, the face representing the USA in a map of countries could include Alaska and Hawaii. The faces are represented implicitly by a set of oriented line segments representing their boundaries. Each segment stores the identification numbers of the faces on its right and left sides. By convention the exterior face has identification number 0. Figure 1 presents an example of map

In this work, we represent 3D maps using a similar strategy. A 3D map is composed of a set of objects each one labeled with a unique identification number. Similarly to the 2D maps, the objects do not need to be connected sets and they are represented implicitly by a set of oriented triangles. Each triangle stores the identification number of the objects above (that is, on the side of its normal) and below it.

4. USING A UNIFORM GRID FOR INDEXING MAPS

EPUG-OVERLAY uses a uniform grid (also known as regular grid) [8] for indexing the map edges. The idea is to create a $G \times G$ grid and superimpose it to the map. The edges are, then, inserted into the grid cells it intersects.

Uniform grids are very parallelizable and can be quickly constructed by doing one pass through the data: for each edge e in the set of edges to be indexed e is rasterized and inserted into the grid cells it intersects. This simplicity is important mainly when exact computation is used, since more complex indexing techniques (such as quadtrees) usually require several arithmetic operations to be constructed, what could be slow if these operations are performed with rationals.

Furthermore, regular grids can efficiently handle uneven data [3, 7, 6]. To improve the overlay performance for very uneven data, EPUG-Overlay supports the use of multi-level uniform grids. The idea is to refine cells containing more edges than a threshold, what created a nested grid. This refinement could be recursively repeated in a process similar to a quad-tree creation (but with a larger branching factor). However, according to our preliminary experiments, for the overlay problem a good performance is obtained by using only one or two levels.

Since the use of uniform grids in the EPUG-Overlay led to a good performance, we intend to extend this strategy to solve the 3D version of the intersection problem. The idea is to create a 3D grid and insert in the grid cells the triangles intersecting them.

5. 2D INTERSECTION

As mentioned in section 1, we have developed the exact algorithm EPUG-Overlay for computing the intersection between two maps A and B. EPUG-Overlay represents all coordinates using GMPXX [9] rational numbers and all the computations are performed exactly. All the special cases were treated using SoS.

Since the faces are stored implicitly, that is, we represent the map as a unstructured set of edges that contains information about the neighbor faces, the overlay operation is performed by computing the intersection between the edges from the two maps and classifying them. Algorithm 1 summarizes the overlay process.

Algorithm 1 Computes the overlay of two maps A and B

```
1: Create the uniform grid
2: Compute the intersection points between all edges of maps $A$ and $B$
3: Locate all vertices of map $A$ in map $B$ and vice-versa
4: Classify the edges and create the resulting map
```

Initially, the uniform grid is created by inserting in its cells the edges from both maps that intersect them. This process is easily parallelized over the edges. If the algorithm is configured to use more than one level in the grid, the cells containing too many edges are refined creating a nested grid in them. The decision about whether or not refine a cell is taken verifying if the number of pairs of edges from both maps in the cell is greater than a given threshold.

After creating the index, the intersections between edges in each grid cell are computed. For each grid cell c, the pairs of edges from both maps in c are tested for intersection and the intersection points are computed. It is worth mentioning that some intersections may be detected more than once since edges may be in more than one grid cell. Again, this step is highly parallelizable over the grid cells.

In the next step, we compute the face from one map in which each vertex from the other map is (and vice-versa). This process is performed by creating, for each vertex v, a semi-infinite vertical ray starting in v and determining what is the lowest edge e directly above v. The face containing v is one of the two faces adjacent to e. If there is no edge above v, v is in the outside face.

Similarly to computing the intersections, the process of locating the face f in which a vertex v is located can be efficiently performed using the uniform grid. The idea is to interactively process the grid cells on and above v trying to locate the lowest edge above v. Once an edge above v is found and all grid cells that could contain the lowest edge have been processed, the process is interrupted. If the uniform grid is sized so that the expected number of edges per cell is constant, the average time to locate each vertex will be constant. Furthermore, this step is highly parallelizable over the vertices being located.

Finally, in the next step the output edges are computed. If an input edge e from map A does not intersect any edge from the map B, it will be an output edge if and only if it is inside a face f (that is not the exterior face) of B (this can be determined by checking in what face of B one of its two vertices is) and its left and right faces will be the intersection between f and the faces from B that are around e (as mentioned in section 1, this information is stored in e).

If e intersects n edges from the other map, e is divided in $n + 1$ sub-edges e_1, e_2, ... ,e_{n+1} and each sub-edge e_i is classified using a strategy similar to the one used when an edge does not intersect other edges: e_i will be an output edge iff it is inside a face of the other map and the new faces to its left and right sides will be determined based on the face of the other map where e_i is and on the faces of e's map that are around e.

Since there is no data dependency between the edges, they can be easily processed in parallel.

It is worth mentioning that preliminary experiments have showed that simply using rational numbers in an implementation previously created for floating-point numbers does not lead to a good performance. For example, in our implementation we had to avoid recreating temporary variables in expressions since the creation of rational numbers usually lead to memory allocations on the heap, what would significantly slow down the performance of the algorithm mainly when these expressions are evaluated in parallel. Thus, EPUG-Overlay needed to be carefully implemented to be efficient.

The current parallel implementation is very efficient and our experiments showed that its performance is comparable to the performance of the overlay module present in GRASS GIS. For example, in our largest experiment using maps with 20 and 30 million edges, EPUG-Overlay was 50% faster than GRASS GIS that, even though is sequential, does not use exact arithmetic.

6. 3D INTERSECTION

As mentioned in section 1, we intend to extend the ideas used in EPUG-Overlay to compute the intersection between two triangulations. The initial steps of the new method, named 3D-EPUG-Overlay, were already implemented and will be described in this section.

Initially, a 3D uniform grid is created and the triangles are inserted into the grid cells they intersect. For performance purposes, the axis-aligned bounding boxes (*AABBs*) of the triangles are computed in parallel and, then, the triangles are inserted in the grid cells intersecting their AABBs.

We intend to use the uniform grid as a filtering structure to accelerate the computation of the intersection, that is, the presence of a triangle in a grid cell c that does not actually intersect c does not affect the correctness of the algorithm (but it may affect its performance). Thus, the choice of inserting the triangles into the grid basing on their AABBs simplifies the creation of the uniform grid while it adds an overhead for the steps of the algorithm that relies in the grid to index the data. If the triangles are inserted into the cells they actually intersect, the cost for computing the grid will be larger but the cost of the other computations steps will be reduced. As a future work we intend to study these overheads in order to choose the grid creation strategy that leads to the best performance.

The most critical step of the overlay is computing the intersection between the triangles. This step was implemented using a strategy similar to the one used in EPUG-Overlay: for each grid cell, the intersections between pairs of triangles from the two triangulations are computed. For efficiency, the pairs of triangles are intersected using the algorithm presented by Möller [15], that uses several techniques to avoid unnecessary computation by detecting as soon as possible if the triangles do not intersect.

The next steps (computing and classifying the output triangles basing on the intersections computed in the previous step) are still under development and, thus, they will be implemented as future work. Furthermore, the current version of this algorithm does not handle the special cases and, thus, using *SoS* to handle these cases is also object of future work.

6.1 Conclusions

This paper presented a proposal for the development of 3D-EPUG-Overlay, an efficient algorithm for exactly computing the overlay of triangulations. 3D-EPUG-Overlay will use arbitrary-precision rational numbers to represent and process the spatial data. In order to mitigate the performance overhead added by the rational numbers, we intend to use high performance computing and a uniform grid for indexing the data.

As a proof of concept, we have developed EPUG-Overlay, a 2D version of 3D-EPUG-Overlay. Our preliminary experiments have showed that EPUG-Overlay was able to efficiently compute overlays, even though it uses exact arithmetic. Therefore, we expect that 3D-EPUG-Overlay will be also efficient.

The initial steps of 3D-EPUG-Overlay were already implemented and future works include finishing this implementation and performing experiments to validate the algorithm's performance and correctness.

Acknowledgment

This research was partially supported by NSF grant IIS-1117277, by CAPES (Ciencia sem Fronteiras) and FAPEMIG.

7. REFERENCES

[1] E. S. Agency. Ariane 501 inquiry board report. http://ravel.esrin.esa.it/docs/esa-x-1819eng.pdf (accessed on Jun-2015).

[2] J.-D. Boissonnat and F. P. Preparata. Robust plane sweep for intersecting segments. *SIAM J. Comput.*, 29(5):1401–1421, 2000.

[3] L. Cucu, M. Dragan, V. Negru, and D. Mangu. Three dimensional Delaunay triangulation using an uniform grid. In *11th European Workshop Comput. Geom.*, pages 21–23. Universität Linz, 1995.

[4] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM TOG*, 9(1):66–104, 1990.

[5] F. Feito, C. Ogayar, R. Segura, and M. Rivero. Fast and accurate evaluation of regularized boolean operations on triangulated solids. *Computer-Aided Design*, 45(3):705 – 716, 2013.

[6] W. R. Franklin, N. Chandrasekhar, M. Kankanhalli, M. Seshan, and V. Akman. Efficiency of uniform grids for intersection detection on serial and parallel machines. In N. Magnenat-Thalmann and D. Thalmann, editors, *Proc. Computer Graphics International'88*, pages 288–297. Springer-Verlag, 1988.

[7] W. R. Franklin, V. Sivaswami, D. Sun, M. Kankanhalli, and C. Narayanaswami. Calculating the area of overlaid polygons without constructing the overlay. *Cartography and Geographic Information Systems*, 21(2):81–89, 1994.

[8] W. R. Franklin, D. Sun, M.-C. Zhou, and P. Y. Wu. Uniform grids: A technique for intersection detection on serial and parallel machines. In *Proc. of Auto Carto 9*, pages 100–109, Baltimore, Maryland, April 1989.

[9] T. Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 6.0.0 edition, 2014. http://gmplib.org/ (accessed on Jun-2015).

[10] J. D. Hobby. Practical segment intersection with finite precision output. *Comput. Geom.*, 13(4):199–214, 1999.

[11] C. M. Hoffman. The problems of accuracy and robustness in geometric computation. *Computer*, 22(3):31–40, 1989.

[12] L. Kettner, K. Mehlhorn, S. Pion, S. Schirra, and C. Yap. Classroom examples of robustness problems in geometric computations. *Comput. Geom. Theory Appl.*, 40(1):61–78, May 2008.

[13] C. Li. *Exact geometric computation: theory and applications,*. PhD thesis, Department of Computer Science, Courant Institute - New York University, January 2001.

[14] G. Mei and J. C. Tipper. Simple and robust boolean operations for triangulated surfaces. *CoRR*, abs/1308.4434, 2013.

[15] T. Möller. A fast triangle-triangle intersection test. *Journal of graphics tools*, 2(2):25–30, 1997.

[16] J. Nievergelt and F. P. Preparata. Plane-sweep algorithms for intersecting geometric figures. *Commun. ACM*, 25(10):739–747, Oct. 1982.

[17] R. Skeel. Roundoff error and the patriot missile. *SIAM News*, 25. Jul. 1992.

[18] J. Yongbin, W. Liguan, B. Lin, and C. Jianhong. Boolean operations on polygonal meshes using obb trees. In *ESIAT 2009*, volume 1, pages 619–622. IEEE, 2009.

[19] S. Zlatanova, A. A. Rahman, and M. Pilouk. Trends in 3d gis development. *Journal of Geospatial Engineering*, 4(2):71–80, 2002.