An efficient algorithm for computing the exact overlay of triangulations

Salles V. G. de Magalhães (salles@ufv.br)
Marcus V. A. Andrade (marcus@ufv.br)
Universidade Federal de Viçosa, Brazil

W. Randolph Franklin (mail@wrfranklin.org)
Wenli Li (liw9@rpi.edu)
Rensselaer Polytechnic Institute, USA

ABSTRACT

This paper presents a proposal for the development of 3D-EPUGO (from *3D Exact Parallel Uniform Grid Overlay*), an efficient algorithm for exactly computing the intersection of 3D triangulations. 3D-EPUGO will be innovative because of two reasons. First, it will use rational numbers to represent spatial data, thereby completely avoiding roundoff errors, 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-EPUGO will use high performance computing and a uniform grid to index the data.

To validate these ideas, we developed EPUGO, a version of 3D-EPUGO for overlaying polygonal maps. Preliminary experiments showed that EPUGO was faster than GRASS GIS, even though GRASS uses inexact arithmetic. This suggests that the 3D version will also be efficient.

1. INTRODUCTION

Computing intersections is a very important boolean operation supported by many GIS and CAD systems. This kind of operation is important not only in polygonal maps but also in 3D solids. However, according to Feito et al. [1], although 3D models have been widely used in computer science, processing them is still a challenge. Due to the algorithm implementation complexity, 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" [1].

There are several possible strategies to compute 3D overlays. A common strategy is to use indexing to accelerate spatial operations performed during the overlay process (such as computing the triangle intersections). For example, Feito et al. [1] uses octrees to intersect triangulations while Jing et al. [7] use Oriented Bounding Boxes trees (OBBs). However, because of floating-point errors robustness cannot be always guaranteed in the traditional methods.

Even though in some situations an approximate algorithm is acceptable, algorithms that are both efficient and robust are a requirement in operations such as overlay, that are frequently used as subroutines for other algorithms and, therefore, errors and performance problems may propagate to these algorithms.

The goal of this work is to develop an efficient algorithm for overlaying 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). Efficient indexing techniques and parallel computing will be used for performance purposes.

As a proof of concept, we have developed a polygonal intersection algorithm (named EPUGO), that can efficiently intersect 2D maps using exact arithmetic. EPUGO uses the same techniques we intend to use for 3D triangulations overlay. As a next step, our objective is to extend the ideas used in EPUGO 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 number and its approximation is often referred as roundoff error. Even though these errors are usually small, arithmetic operations frequently create more errors and a sequence of operations usually leads to larger errors.

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 compare floating-point numbers. However this is a formal mess because equality is no longer transitive, nor invariant under scaling.

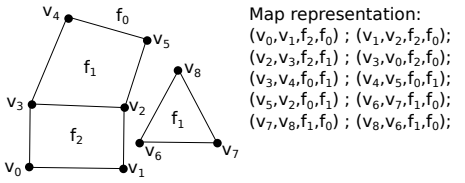

Figure 1: 2D map: each edge e is represented by a quadruple (v_i,v_j,f_l,f_r), where v_i and v_j are the vertices of e and f_l and f_r are the left and right faces of e.

Another techniques are the Exact Geometric Computation (EGC)[5] model, which represents mathematical objects exactly using algebraic numbers, and snap rounding (SP) [3], whose basic idea is to use some rounding method to convert an arbitrary precision arrangement of segments into a fixed-precision representation. While *EGC* has interesting features, its main drawback is the performance penalty (even determining the sign of an expression is nontrivial). 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 is to use exact computation based on rational number with arbitrary precision [4]. 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 [1].

A 2D map will be composed of a set of faces. 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. 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. 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

EPUGO uses a uniform grid 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 they intersect.

Uniform grids are very parallelizable and can be quickly constructed by doing one pass through the data: for each edge e in the input, e is rasterized and inserted into the grid cells it intersects. This simplicity is important mainly when rationals are used, since more complex indexing techniques usually require several arithmetic operations to be constructed.

Furthermore, these grids can efficiently handle uneven data [2]. To improve the performance for very uneven data, EPUGO supports the use of multi-level uniform grids. The idea is to re-

fine 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, according to our preliminary experiments, for the overlay problem a good performance is obtained by using only one or two levels.

Since uniform grids performed well in EPUGO, we intend to extend this strategy to 3D. The idea is to create a 3D grid and insert in the grid cells the triangles intersecting them.

5. 2D INTERSECTION

We have developed the algorithm EPUGO for exactly overlaying maps. All coordinates are represented with GMP rationals. Special cases are treated handled using SoS.

Since the map is represented as a unstructured set of edges that contain information about the neighbor faces, the overlay operation is performed by computing the intersection between the edges 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 between all edges of the maps
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. 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.

After creating the index, the intersections between edges in each grid cell are computed. For each cell c, the intersection between pairs of edges from both maps in c are computed. Again, this step is highly parallelizable over the grid cells.

Then, we compute the face from one map in which each vertex from the other map is. 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. This process also uses a uniform grid to index the edges that need to be tested for intersection with the vertical rays. 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.

If e intersects n edges from the other map, e is divided in $n+1$ sub-edges 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 rationals in an implementation created for floating-point numbers isn't fast. For example, in our implementation we avoid recreating temporary variables in expressions since the creation of rational numbers usually lead to memory allocations on the heap, what is significantly slow when these expressions are evaluated in parallel. Thus, EPUGO 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, EPUGO was 50% faster than GRASS GIS that, even though is sequential, does not use exact arithmetic.

6. 3D INTERSECTION

We intend to extend the ideas used in EPUGO to compute the intersection between two triangulations. The initial steps of the new method, named 3D-EPUGO, 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. 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. 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 EPUGO: 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 [6], that uses several techniques to avoid unnecessary computations.

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-EPUGO, an efficient algorithm for exactly computing the overlay of triangulations. 3D-EPUGO will use arbitrary-precision rational numbers to represent and process the spatial data.

As a proof of concept, we have developed EPUGO, a 2D version of 3D-EPUGO. Our preliminary experiments have showed that, due to the techniques we used, EPUGO was able to efficiently compute overlays. Thus, we expect that 3D-EPUGO will be also efficient.

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

7. REFERENCES

[1] F. Feito and C. Ogayar et al. Fast and accurate evaluation of regularized boolean operations on triangulated solids. *Computer-Aided Design*, 2013.

[2] W. R. Franklin and N. Chandrasekhar et al. Efficiency of uniform grids for intersection detection on serial and parallel machines. In *Proc. Comp. Graphics Int.* 1988.

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

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

[5] C. Li. *Exact geometric computation: theory and applications,*. PhD thesis, Department of Computer Science, Courant Institute - NYU, 2001.

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

[7] J. Yongbin and W. Liguan et al. Boolean operations on polygonal meshes using obb trees. In *ESIAT*, 2009.