IMPLEMENTING SIMULATION OF SIMPLICITY FOR GEOMETRIC DEGENERACIES

W. Randolph Franklin
Electrical, Computer, and Systems Engineering Dept.
Rensselaer Polytechnic Institute
Troy, NY 12180
mail@wrfranklin.org

Salles Viana Gomes de Magalhães
Departamento de Informática
Universidade Federal de Viçosa
Viçosa MG Brazil
sallesviana@gmail.com

ABSTRACT

We describe how to implement Simulation of Simplicity (SoS), which treats geometric degeneracies, or special cases, in map overlay and other 2D and 3D geometric and spatial algorithms. A geometric degeneracy is a coincidence, such as a vertex of one map on an edge of another map, that would have probability zero if the objects were distributed i.i.d. uniformly. However, in real data, they can occur often. Especially in 3D, there are too many types of degeneracies to reliably enumerate. But, if they are not handled, then boolean predicates evaluate wrong, and output topology, such as in a map overlay, may be wrong. SoS pretends to add infinitesimals of different orders to the coordinates, so that there are no longer any coincidences. It then modifies the code to compute the same result, but w/o adding the infinitesimals. The result is a longer code, but with probably the same execution time, because the extra code is executed only when handling a degeneracy. We describe the theory of SoS, and how several algorithms and programs were successfully modified.

***K*eywords** geometric degeneracies · geometric special cases · geometric implementation · boolean intersection · computational geometry

1 Introduction

Handling geometric degeneracies, or special cases, is a nasty part of transforming a beautiful algorithm into useful code. Example 2D degeneracies include an endpoint of one edge on another edge (Figure 1), two overlapping edges, two coincident points from different objects, two edges from different objects with a common vertex, etc. If objects were randomly uniformly and independently and identically distributed, then the probability of any of these would be 0. However they occur frequently in real data. E.g., a CAD designer might place one object tight against another. Two diplomats may agree that the common border between their adjacent countries will coincide with a river shoreline or center line. Two different digital maps created for the same region may have many (but not all) edges in common. We may want to conflate a dataset of contour lines with a dataset of hydrography features, including shorelines, where the shoreline might mostly align with a contour line.

The approach described in this note is better than existing solutions because not handling degeneracies correctly causes erroneous output (examples are shown in some of the references), and allegedly, even can cause commercial CAD systems to crash. Heuristics work only up to a point, and fail for more complicated algorithms. Indeed, then there are too many special cases to grasp. Also, as input datasets get larger, the probability of an error grows.

Why is this difficult? Consider the simple 2D problem of determining whether a point p is contained inside a polygon $\mathcal{P}$ (Figure 2), [16]. The classic Jordan-curve algorithm[19], extends a semi-infinite ray up from p, and counts how many edges $\mathbf{e}_i$ of $\mathcal{P}$ it crosses. p is inside $\mathcal{P}$ iff that number is odd. (We will not here consider what is a legal polygon, and what is the answer if p is on an edge.) It's easiest to consider the $\mathbf{e}_i$ independently; the ray will have 0 or 1 intersection with each $\mathbf{e}_i$; and sum those counts.

But, what if the ray intersects an endpoint of $\mathbf{e}_i$? Is that 0 or 1 intersection? We may not care about one individual $\mathbf{e}_i$. We may not even care about the exact total count. However, we require that the total count be odd or even correctly.

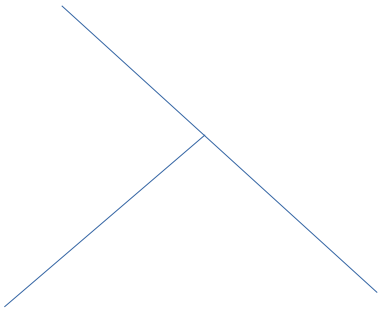

Figure 1: Degeneracy: endpoint on edge interior

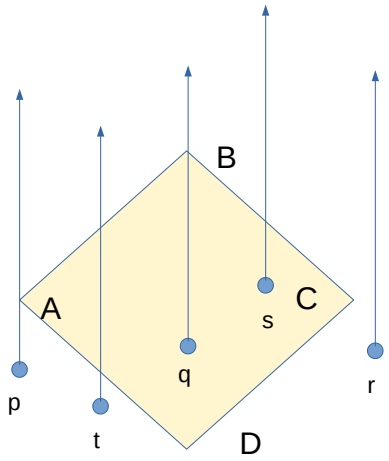

Figure 2: Point in polygon?

This requires that for some rays and $\mathbf{e}_i$, the count be 0 and for others be 1. One heuristic that seems to work to count an intersection iff the ray intersects the right endpoint of $\mathbf{e}_i$. If $\mathbf{e}_i$ is vertical, then count 0 for both endpoints. This heuristic can be justified as follows.

Sort the x-coordinates of the vertices of $\mathcal{P}$, and let d be the smallest nonzero difference between adjacent x-coordinates. Translate the query point right by $d/2$: $p' = p + (d/2, 0)$. Execute the Jordan curve algorithm with p'.

This will give the wrong answer when p is close to an edge of $\mathcal{P}$ and p' is on the other side. However the probability of that error is reduced by reducing how much p was translated. Importantly, the ray run up from p' will never intersect any $\mathbf{e}_i$, so our difficult degeneracy will never occur.

The above analysis illustrates the divide-and-conquer technique. We decomposed the problem of point location in a polygon into a set of smaller problems of testing the point (actually, its ray) separately against each edge. Then we combined the answers to the smaller problems to solve the original problem.

Locating a point in a polygon is so simple that the time spent above on analyzing the special case may seem excessive; we (think that we) can easily make the algorithm work. However, as the problems get bigger, then the number of special cases quickly grows so that an informal analysis will be incomplete and so wrong. Consider the problem of intersecting

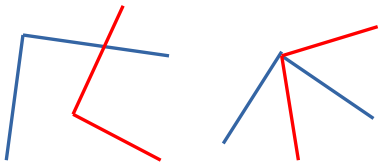

Figure 3: Two crossing polylines: general case, and degeneracy

two polylines $\mathbf{l_0}$ and $\mathbf{l_1}$; see Figure 3. Polyline $\mathbf{l_i}$ is a sequence of n_i vertices v_{ij}, $0 \leq j < n_i$, each defining $n_i - 1$ edges $v_{ij}v_{i,j+1}$ for $0 \leq j < n_i - 1$. The polyline is closed when the last vertex equals the first: $v_{i0} = v_{i,n-1}$.

Assume that each polyline has no two vertices in common; $i \neq k \implies v_{ij} \neq v_{ik}$. In general position, $\mathbf{l_0}$ has no vertices in common with $\mathbf{l_1}$ and no vertex of either polyline is in the interior of any edge of the other. That is, if an edge of $\mathbf{l_0}$ intersects (in a point set sense) an edge of $\mathbf{l_1}$, then the intersection is exactly one point.

If a vertex of one polyline were on an edge of the other, that would be a degeneracy; see Figure 3. One application of polyline intersection would be computing a boolean combination of two polygons, where their boundaries are closed polylines.

Another way of looking at a degeneracy is that it occurs when evaluating a boolean predicate that compares two numbers, e.g., is $a < b$? In this paper, assume computations are exact; we do not consider roundoff errors. This is not to say that roundoff errors are unimportant, merely that they are another topic. SoS assumes exact computation, which can be achieved, e.g., by computing with big rational numbers. Rational numbers represent each number as the ratio of two integers, e.g., $1/3$, and compute exactly, e.g., $1/3 + 2/5 = 11/15$.

Geometric predicate evaluations can be considered as determinant sign evaluations. Consider three 2D points, A, B, C. One possible predicate is whether $\angle ABC < \pi$? That, with the other information, controls whether the convex hull of $OABC$ is $OABC$ or OAC. $\angle ABC < \pi$ is equivalent to

$$\begin{vmatrix} A_x & A_y & 1 \\ B_x & B_y & 1 \\ C_x & C_y & 1 \end{vmatrix} > 0$$

The problem of evaluating whether $a < b$ is that $a == b$ is a third case[1]. Every decision tree that branched out two ways at each decision, now has to branch three ways to handle the degeneracies. After k decisions, 2^k cases grows to 3^k cases. One possible solution is to fold the degenerate case $a == b$ into one of the two nondegenerate cases. That idea is an informal step towards SoS.

In brief, the solution presented in this note is a technique for modifying a geometry algorithm (and so also its source code) so that it will handle degenerate geometric inputs with effectively no increase in execution time.

2 Infinitesimals

The set of real numbers $\Re$ may be partitioned into negative finite numbers, zero, and positive finite numbers. Consider the possibility of a quantity ϵ, which we will call an *infinitesimal*, which is smaller than any positive finite real number. That is $\forall r \in \Re, r > 0 \implies 0 < \epsilon < r$. This is logically possible. Finite multiples of ϵ, such as 2ϵ and $\epsilon/5$ are possible, and obey the obvious ordering. $0 < \epsilon/5 < \epsilon < 2\epsilon < r$. These multiples are called *first order infinitesimals*. ϵ^2 is a *second order infinitesimal*. $0 < \epsilon^2 < \epsilon$. Finite multiples of ϵ^2 operate similarly to first order infinitesimals. Infinitesimals of any positive integral order are possible. If $0 < i < j$, then $\epsilon^j < \epsilon^i$.

The preceding paragraph may sound impossible but it is not. We have just described a *non-Archimedian ordered field*[24].

How is the test $a < b$, where a and b are finite reals, affected by adding infinitesimals to a and b? The test might become $a + \epsilon^i < b + \epsilon^j$. Assume, without loss of generality, that $0 < i < j$. If $a \neq b$, then if $a < b$ is true, then also $a + \epsilon^i < b + \epsilon^j$ is true. Adding the infinitesimals didn't change the result. However, if $a == b$, then the infinitesimals break the tie. If $a == b$, then $a + \epsilon^i < b + \epsilon^j$ reduces to $\epsilon^i < \epsilon^j$, which reduces to $i > j$. So, adding infinitesimals to a and b changes the C++ code to

```
if (a!=b) return a<b;
else return i>j;
```

In the nondegenerate case, the execution time is the same.

3 Simulation of Simplicity

This section describes how to break degeneracies with Simulation of Simplicity (SoS), [12, 10, 11, 20, 23]. Note that the term *simulation* is used in these computational geometry papers with a quite different meaning than used in the modeling and simulation community.

[1]The notations for equality and assignment are not standardized. We will use $==$ for the equality predicate, and $=$ for assignment.

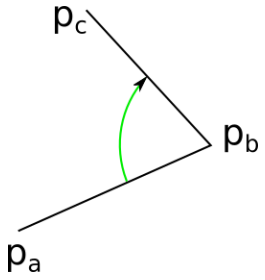

Figure 4: 2D angle

A degeneracy is, in a sense, a dimension reduction in the input, or a constraint between input parameters. SoS has been used for computing contour trees [3], triangulation [2], polyhedral modeling [13], molecular modeling [18], computing line arrangements [4], exact boundary evaluation [21], and polygon overlay [1].

The types of degeneracies that we wish to break with SoS, are these.

1. Two different points having the same value.
2. A point, possibly the endpoint of an edge, being incident on a line, or extended edge. This includes the case of two edges being on the same infinite line.
3. Two edges having the same slope. This degeneracy might not be an immediate problem, but is easy to break.

To do this, we add infinitesimals of different orders to the points' coordinates. The order cannot just be a function of a coordinate's value, but must depend on something unique to the point. Therefore our SoS algorithm goes as follows.

1. Index all the coordinates of all the input points, from 0 up.
2. Let the i-th coordinate be x_i. So, point #k will have coordinates (x_{2k}, x_{2k+1}).
3. Modify the coordinates thus: $x_i \to x_i + \epsilon^{2^i}$.

This will break all degeneracies. For instance, all possible edges between input points must have different slopes.

The above algorithm is general; however sometimes simpler SoS algorithms are adequate in special cases.

For SoS to be valid, we do not want extra constraints on the inputs. E.g., we do not want to have input points a, b, c that are required to be collinear. The SoS technique to be described here would consider that constraints a degeneracy and break it. The solution would be to make a, c, and a real number α the inputs, and then define $b = \alpha a + (1 - \alpha)c$.

3.1 Point in polyhedron test: degenerate cases

Why SoS is needed because we can't hope to enumerate (let alone resolve) all the special cases, consider testing whether a point p is contained in a 3D polygon $\mathcal{P}$ with the Jordan curve algorithm. Run a ray r up from p and test which faces it intersects. p is inside iff that number is odd.

1. p might be on an edge, or on a vertex of $\mathcal{P}$.
2. r might intersect an edge.
3. The local topology of that intersection has variants.
4. r might coincide with an edge, i.e., run along the edge.
5. r might intersect a vertex.
6. That case has several variants.

4 Examples

Here we will see several examples of the SoS algorithm.

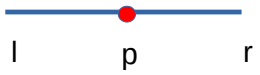

Figure 5: Point on edge

4.1 Point on edge in 1D

This little example illustrates how this technique works.

Consider the 1D case of a point with coordinate p on an edge with left and right coordinates l and r respectively; see Figure 5. The point intersects the edge, in a set theoretic sense, if $(l \leq p) \wedge (p \leq r)$. However, is this what we want when the point is one of the endpoints? We want an answer that makes this a useful subroutine of larger algorithms, such as point in polygon. That is, we want to choose a definition of *intersects* here that makes the larger algorithm "correct", for some useful definition of "correct", such as, not causing topological errors.

In this simple case, it is not necessary to modify all the input, although it would be correct. It is sufficient to modify p thus:

$$p' = p + \epsilon$$

and test point inclusion with the predicate

$$(l \leq p') \wedge (p' \leq r)$$

or

$$(l \leq p + \epsilon) \wedge (p + \epsilon \leq r)$$

Now, if $l == p$ then $l < p'$, so $l \leq p'$ is equivalent to $l \leq p$. Similarly if $p == r$ then $p' > r$ so $p' \leq r$ is equivalent to $p < r$.

So the SoS version of the point inclusion predicate is

$$(l \leq p) \wedge (p < r)$$

4.2 Point in polygon test

Applying this idea to a simple classic problem shows how it works.

The problem is to test whether a point p is contained in a polygon $\mathcal{P}$. (We do not here consider the case of p being on the boundary (i.e., an edge) of $\mathcal{P}$. We also do not worry about complicated types of $\mathcal{P}$ like multiple components and nested holes.) The usual algorithm is the *Jordan curve* method. It runs a semi-infinite ray r up from p, and counts the number of intersections of r) with edges of $\mathcal{P}$. p inside $\mathcal{P}$ is equivalent to there being an odd number of intersections.

(There is another point testing algorithm that is more popular than good, which adds the angles subtended at p by each edge of $\mathcal{P}$. If the sum is 0, p is outside, if 2π, then p is inside. The difficulty is that computing the subtended angle for an edge requires determining whether that edge crosses the positive x-axis, so we're back at the Jordan curve algorithm, but complexified with an arctan evaluation.)

Consider Figure 2, and we want to count intersections of rays from points with the edges of polygon $ABCD$. The ray from point r has 0 intersections; r is outside. The ray from s has 1; s is inside. t induces 2 intersections; it is outside. What about the ray from p? Since p is outside, we need an even number of intersections with the edges DA and AB; either 0 or 2. However, also consider q, which is inside. Its ray must have exactly one intersection with the two edges AB and BC. Which edge should that be? In this case, the solution is to use the previous section's algorithm to determine if the ray will intersect the edge, if the point is also beneath the edge.

4.3 2D angle example

Here is SoS worked out for the 2D angle at b on $\angle p_a p_b p_c$; Figure 4. Without loss of generality, let $p_a = (0, 0)$ (otherwise subtract it out.) ϵ is an infinitesimal, smaller than any positive real number. $\epsilon^2 < \epsilon$ etc. W/o SoS, whether the angle is convex is determined by the sign of

$$D = \begin{vmatrix} x_b & y_b \\ x_c & y_c \end{vmatrix}$$

where $p_b = (x_b, y_b)$ etc. Generally b and c can be considered as integer indices for the points.

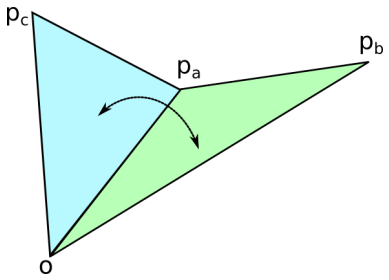

Figure 6: 3D dihedral angle

4.3.1 SoS, version 1

1. Realize SoS by translating
$$(x_b, y_b) \to \left(x_b + \epsilon^{2b}, y_b + \epsilon^{2b+1}\right)$$
etc.
2. However any choice of orders of infinitesimals would have worked;
3. With SoS, the determinant D becomes
$$\begin{aligned}\mathcal{D} &= \begin{bmatrix} x_b + \epsilon^{2b} & y_b + \epsilon^{2b+1} \\ x_c + \epsilon^{2c} & y_c + \epsilon^{2c+1} \end{bmatrix} \\ &= \left(x_b + \epsilon^{2b}\right)\left(y_c + \epsilon^{2c+1}\right) - \left(y_b + \epsilon^{2b+1}\right)\left(x_c + \epsilon^{2c}\right) \\ &= D + \epsilon^{2b} y_c + \epsilon^{2c+1} x_b - \epsilon^{2c} y_b - \epsilon^{2b+1} x_c\end{aligned}$$
4. Define *firstsign(a1,a2,a3,...)* to be sign of the the first argument that is nonzero, or 0.
5. $\sigma = \text{signum}(\mathcal{D}) = 0$ only when all the points are 0.
6. The answer depends on how the points are ordered. Since breaking ties is breaking a symmetry, something has to be nonsymmetric in the rule. We shouldn't care how we break the ties (degeneracies), so long as we do it consistently throughout the whole problem.
7. Nevertheless, this should consistently label angles in a planar mesh.
8. If $p_{\max(b,c)} \neq (0,0)$ then if $y_{\max(b,c)} = 0$ then $x_{\max(b,c)} \neq 0$.
9. Then this can be simplified to
$$\sigma = \text{firstsign}(D, y_{\max(b,c)}, -x_{\max(b,c)}) \tag{1}$$

4.3.2 SoS, version 2

1. Simplify the above even more, by letting the id of p be its order when all the p are sorted lexicographically.
2. Let i be b or c depending on whether y_b or y_c is larger.
3. Let
$$i = \begin{cases} \text{if } y_b > y_c \vee (y_b = y_c \wedge x_b > x_c) & \text{then} \quad b \\ & \text{else} \quad c \end{cases}$$
4. Then Equation 1 above becomes
$$\sigma = \text{firstsign}(D, y_i, -x_i)$$

4.4 3D angle

See Figure 6.

1. Consider two triangles, $\triangle op_a p_b$ and $\triangle op_a p_c$ with a common edge op_a.
2. a, b, c are the integer indices of the points in the points array. $0 \leq a, b, c < n$.
3. We want the answer to the question, is the dihedral angle on that edge convex?

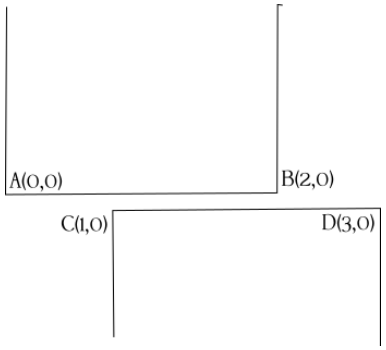

Figure 7: 2 lines, version 1

4. This depends on the sign of the determinant

$$\mathcal{D} = [p_a \quad p_b \quad p_c] = \varepsilon_{ijk}\, p_{ai}\, p_{bj}\, p_{ck}$$

5. That used the Einstein tensor notation:
   (a) $\varepsilon_{ijk} = -1, 0,$ or 1, depending on whether ijk is a positive permutation, have an index duplicated, or is a negative permutation.
   (b) sum over doubled indices.
6. SoS transform $p_{ai} \to p_{ai} + \epsilon^{a+ni}$ (other choices are possible).
7. $$\begin{aligned}\mathcal{D} &= \varepsilon_{ijk}\,(p_{ai} + \epsilon^{a+ni})\,(p_{bj} + \epsilon^{b+nj})\,(p_{ck} + \epsilon^{c+nk}) \\ &= D+ \\ &\varepsilon_{ijk}\left(\epsilon^{a+ni} p_{bj} p_{ck} + \epsilon^{b+nj} p_{ai} p_{ck} + \epsilon^{c+nk} p_{ai} p_{bj}\right) + \\ &\varepsilon_{ijk}\left(\epsilon^{b+c+n(j+k)} p_{ai} + \epsilon^{a+c+n(i+k)} p_{bj} + \epsilon^{a+b+n(i+j)} p_{ck}\right) + \\ &\varepsilon_{ijk}\left(\epsilon^{a+b+c+n(i+j+k)}\right) \\ &= D+ \\ &\epsilon^a(p_{b1}p_{c2} - p_{b2}p_{c1}) + \epsilon^b(p_{a1}p_{c2} - p_{a2}p_{c1}) + \epsilon^c(p_{b1}p_{a2} - p_{b2}p_{a1}) + \\ &\text{higher order terms}\end{aligned}$$
8. Let $a' = \min(a,b,c),\ b' = \text{median}(a,b,c),\ c' = \max(a,b,c)$. Then, apart from possible sign errors, the angle is convex depending on

$$\sigma = \text{firstsign}(D, (p'_b \times p'_c)_2, (p'_c \times p'_a)_2, (p'_a \times p'_b)_2)$$

9. This fails when the vectors are collinear. The omitted higher order terms could handle them.

4.5 2 Intersecting lines

1. See Figure 7.
2. Perturb
   (a) $B(2,0) \to (2+\epsilon, \epsilon^2)$
   (b) $C(1,0) \to (1+\epsilon^3, \epsilon^4)$
   (c) $D(2,0) \to (3+\epsilon^5, \epsilon^6)$
3. The slope of the line through A(0,0) and $B(2+\epsilon, \epsilon^2)$ is

$$\frac{\epsilon^2}{2+\epsilon}$$

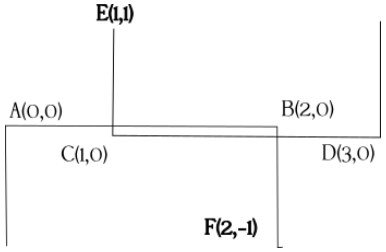

Figure 8: 2 lines, version 2

4. This could probably be simplified to
$$\frac{\epsilon^2}{2}$$
The hi order terms would be relevant only if the lo order terms later cancelled.
5. AB's equation is
$$y = \frac{\epsilon^2}{2+\epsilon}x$$
possibly simplified to
$$y = \frac{\epsilon^2}{2}x$$
6. The slope of the line through $C(1+\epsilon^3, \epsilon^4)$ and $D(3+\epsilon^5, \epsilon^6)$ is
$$\frac{\epsilon^6 - \epsilon^4}{3+\epsilon^5 - 1 - \epsilon^3}$$
7. CD's equation is
$$y = \frac{\epsilon^6 - \epsilon^4}{2+\epsilon^5 - \epsilon^3}x + \frac{3\epsilon^4 - \epsilon^6}{2+\epsilon^5 - \epsilon^3}$$
8. It could probably be simplified to
$$y = -\frac{\epsilon^4}{2}x + \frac{3\epsilon^4}{2}$$
9. Then, using the simplified expressions, AB and CD intersect at
$$\left(-3\epsilon^2, -\frac{3}{2}\epsilon^4\right)$$
10. Since AB's positive slope is greater than CD's negative slope, A and B are above CD, while C and D are below AB. So the above figure is correct.

Now consider Figure 8.

1. Since AB is above CD here, edges AB and CE should intersect.
2. So, in this case, things work out.

4.6 Intersecting squares

What does SoS do to the intersection of two coincident lines that are edges are two squares? Might the intersection after the perturbation be far away? See Figure 9.

1. Let lines A and E be $y = 0$.
2. After SoS, A becomes $y = \epsilon^4 x + \epsilon^8$ and E becomes $y = \epsilon x + \epsilon^2$. E has a slightly greater slope than A.
3. They intersect at
$$(x, y) = \left(-\epsilon\,\frac{1-\epsilon^6}{1-\epsilon^3},\ -\epsilon^5\right)$$

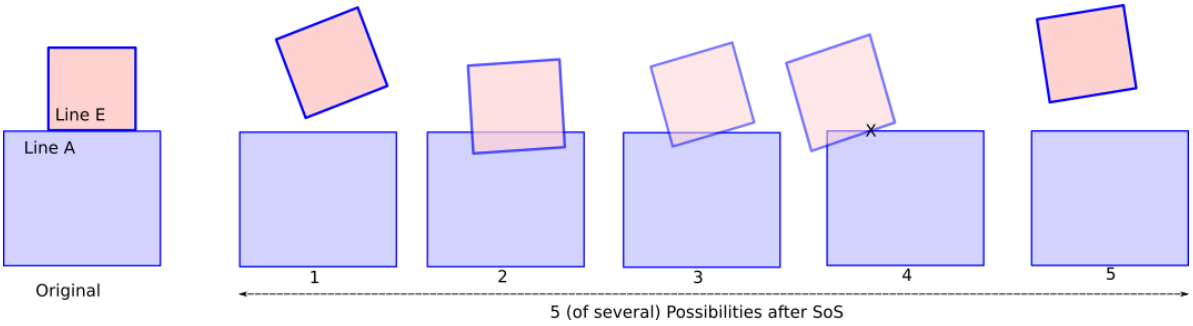

Figure 9: Intersecting squares

4. Approximately,
$$(x, y) = (-\epsilon,\ \epsilon^5)$$
so Case 5 in Figure 9 above obtains.
5. Is it possible that the intersection might sometimes be far away?
6. Two general lines are $y = ax + b$ and $y = cx + d$.
7. They intersect at
$$(x, y) = \left(\frac{d-b}{a-c},\ \frac{ad-bc}{a-c}\right)$$
8. Since the lines are coincident, let $c = a$ and $d = b$.
9. SoS-ify to make the lines
$$y = (a + \epsilon)x + (b + \epsilon^2)$$
and
$$y = (a + \epsilon^4)x + (b + \epsilon^8)$$
.
10. They will intersect at
$$(x, y) = \left(\frac{b + \epsilon^8 - b - \epsilon^2}{a + \epsilon - a - \epsilon^3},\ \frac{\epsilon b - \epsilon^2 a - \epsilon^4 b - \epsilon^6 + \epsilon^8 a + \epsilon^9}{a + \epsilon - a - \epsilon^3}\right)$$
$$= \left(-\epsilon\frac{1-\epsilon^6}{1-\epsilon^2},\ \frac{b - \epsilon a - \epsilon^3 b - \epsilon^5 + \epsilon^7 a + \epsilon^8}{1-\epsilon^2}\right)$$
ignoring higher order terms
$$= \left(-\epsilon,\ b - \epsilon a - \epsilon^5\right)$$
11. Conclusion: the intersection point is never way off in the distance.

4.7 Possible concavity?

Will SoS cause the perturbed polygon to have a concavity at the new intersection?

1. See Case 4 in Figure 9, with the intersection at X.
2. To make X to be on the edge (and not on its extension), ϵ will have to be negative. That is acceptable.
3. Concavities might occur, resulting in a polygon having a spike; Figure 10.
4. Indeed, if the intersection was convex on 1 side then it would have to be concave on the other.
5. Regularization would be needed to fix this, [22]. The problem is the point set unions and intersections of polygons can create isolated edges and gaps. Regularization removes them by replacing a polygon with the closure of its interior.

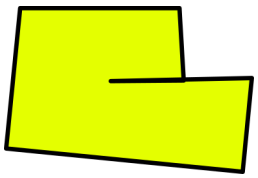

Figure 10: Possible concavity

4.8 Volume of union of cubes

The problem is to compute the volume of the union of tens of millions of identical isothetic cubes. The algorithm optimizes the composition of the union and the volume operations, so that it is not necessary to compute the explicit output union polyhedron. It is sufficient to compute only the set of output vertices, together with their neighborhoods. A vertex's neighborhood consists of the directions of any adjacent edges and faces, and for a face, which side is interior. The neighborhood does not include the other vertices on those adjacent edges and faces. Not needing to compute that is a considerable simplification. Theoretical analysis and implementation and test results are in [14, 15].

Components of the algorithm include finding all the face-face-face intersections and edge-face intersections among the input cubes. SoS is used. The order of infinitesimals added is a function of the indexes of the edges and faces. If the input is independently and identically distributed uniform random, then there is an equation for the expected output volume, and it agrees with what is computed.

4.9 Point location in 3D mesh

PinMesh is a very fast algorithm with implementation to preprocess a polyhedral mesh, also known as a multimaterial mesh, in order to perform 3D point location queries, [5]. PinMesh combines several innovative components to efficiently handle the largest available meshes. Because of a 2-level uniform grid, the expected preprocessing time is linear in the input size, and the code parallelizes well on a shared memory machine. Querying time is almost independent of the dataset size. PinMesh uses exact arithmetic with rational numbers to prevent roundoff errors, and symbolic perturbation with Simulation of Simplicity (SoS) to handle geometric degeneracies or special cases. PinMesh is intended to be a subroutine in more complex algorithms. It can preprocess a dataset and perform 1 million queries up to 27 times faster than RCT (Relative Closest Triangle), the current fastest algorithm. Preprocessing a sample dataset with 50 million triangles took only 14 elapsed seconds on a 16-core Xeon processor. The mean query time was 0.6 microseconds.

4.10 Intersecting 3D triangular meshes

3D-EPUG-Overlay is a fast, exact, parallel, memory-efficient, algorithm for computing the intersection between two large 3-D triangular meshes with geometric degeneracies, [17, 6, 9, 8, 7]. Applications include CAD/CAM, CFD, GIS, and additive manufacturing. 3D-EPUG-Overlay combines 5 separate techniques: multiple precision rational numbers to eliminate roundoff errors during the computations; Simulation of Simplicity to properly handle geometric degeneracies; simple data representations and only local topological information to simplify the correct processing of the data and make the algorithm more parallelizable; a uniform grid to efficiently index the data, and accelerate testing pairs of triangles for intersection or locating points in the mesh; and parallel programming to exploit current hardware. To simplify the symbolic perturbation, the algorithm employs only orientation predicates.

There is a challenge in the mesh intersection problem: the predicates will have not only to handle input vertices (with real or rational coordinates), but also vertices generated from in- tersections. Since the coordinates of a vertex generated from an intersection is a function of five input points (two points defining an edge of one mesh and three points defining a triangle of the other mesh) and these points are perturbed, then the orientation has to be modified to handle these points. The 3D orientation will only be computed using, as argu- ments, three input vertices and another vertex that may be either an input vertex or a vertex from the intersection. Thus, at least two versions of the 3D orientation will have to be implemented.

For the 2D orientation, on the other hand, any of the three pa- rameters may be either an input vertex or a vertex generated by an intersection. Thus, eight versions of the orientation predicate will be required. Since the orientation is computed using a determi- nant, the order of the parameters may be modified as long as the signum of the result is negated for each parameter swap. Then the number of functions actually written can be reduced by sorting the parameters by their type (input vertex or vertex from intersection).

We stress tested 3D-EPUG-Overlay by overlaying a polyhedron with translated or rotated versions of itself. Handling degeneracies correctly is important because some other programs use heuristics with a user-supplied tolerance, and so sometimes fail.

5 Summary

Simulation of Simplicity, adding infinitesimals of different orders to geometric coordinates, is a powerful technique to remove geometric degeneracies. The modified program often has the same length, or only a slight greater length, and has the same or only an insignificantly greater execution time. The main limitation is that computations must be exact; SoS relies on exact equality tests. The only cost of this technique is in the required analysis of the algorithm that this is being applied to.

6 Acknowledgements

This research was partially supported by FAPEMIG, CAPES (Ciencia sem Fronteiras - grant 9085/13-0), CNPq, and a gift from Dr Wenli Li.

References

[1] S. Audet, C. Albertsson, M. Murase, and A. Asahara. Robust and efficient polygon overlay on parallel stream processors. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL'13, pages 304–313, New York, NY, USA, 2013. ACM.

[2] I. Beichl. Dealing with degeneracy in triangulation. *Computing in Science and Engg.*, 4(6):70–74, Nov. 2002.

[3] H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, pages 918–926, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[4] B. Chazelle, H. Edelsbrunner, L. Guibas, M. Sharir, and J. Snoeyink. Computing a face in an arrangement of line segments. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '91, pages 441–448, Philadelphia, PA, USA, 1991. Society for Industrial and Applied Mathematics.

[5] S. V. G. de Magalhães, M. V. A. Andrade, W. R. Franklin, and W. Li. PinMesh – Fast and exact 3D point location queries using a uniform grid. *Computer & Graphics Journal, special issue on Shape Modeling International 2016*, 58:1–11, Aug. 2016. (online 17 May). Awarded a reproducibility stamp, `http://www.reproducibilitystamp.com/`.

[6] S. V. G. de Magalhães, W. R. Franklin, and M. V. A. Andrade. Fast exact parallel 3D mesh intersection algorithm using only orientation predicates. In *25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL 2017)*, Los Angeles Area, CA, USA, 7–10 Nov 2017.

[7] S. V. G. de Magalhães, W. R. Franklin, and M. V. A. Andrade. An efficient and exact parallel algorithm for intersecting large 3-d triangular meshes using arithmetic filters. *J. Computer Aided Design*, 120, Mar. 2020. online 2019-12-19.

[8] M. de Matos Menezes, S. V. G. de Magalhães, M. Aguilar, W. R. Franklin, and B. Coelho. Employing GPUs to accelerate exact geometric predicates for 3D geospatial processing. In J. Krumm, A. Züfle, and C. Shahabi, editors, *Spatial Gems*, volume 1, chapter 11. ACM, 2022.

[9] M. de Matos Menezes, S. V. G. Magalhães, M. A. de Oliveira, W. R. Franklin, and R. E. de Oliveira Bauer Chichorro. Fast parallel evaluation of exact geometric predicates on GPUs. *J. Computer Aided Design*, (103285), Sept. 2022. Special Issue: 28th International Meshing Roundtable: Mesh Modeling for Simulations and Visualization.

[10] Edelsbrunner, Letscher, and Zomorodian. Topological persistence and simplification. *Discrete Comput. Geom.*, 28(4):511–533, Nov. 2002.

[11] H. Edelsbrunner and D. Guoy. Sink-insertion for mesh improvement. In *Proceedings of the Seventeenth Annual Symposium on Computational Geometry*, SCG '01, pages 115–123, New York, NY, USA, 2001. ACM.

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

[13] S. Fortune. Polyhedral modelling with exact arithmetic. In *Proceedings of the Third ACM Symposium on Solid Modeling and Applications*, SMA '95, pages 225–234, New York, NY, USA, 1995. ACM.

[14] W. R. Franklin. Analysis of mass properties of the union of millions of polyhedra. In M. L. Lucian and M. Neamtu, editors, *Geometric Modeling and Computing: Seattle 2003*, pages 189–202. Nashboro Press, Brentwood TN, 2004.

[15] W. R. Franklin. Mass properties of the union of millions of identical cubes. In R. Janardan, D. Dutta, and M. Smid, editors, *Geometric and Algorithmic Aspects of Computer Aided Design and Manufacturing, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 67, pages 329–345. American Mathematical Society, 2005.

[16] W. R. Franklin. PNPOLY - point inclusion in polygon test, 2009. accessed 31-Dec-2009.

[17] W. R. Franklin, S. V. G. de Magalhães, and M. V. A. Andrade. An exact and efficient 3D mesh intersection algorithm using only orientation predicates. In *S3PM-2017: International Convention on Shape, Solid, Structure, & Physical Modeling, Shape Modeling International (SMI-2017) Symposium*, Berkeley, California, USA, 19–23 June 2017. (poster).

[18] D. Halperin and C. R. Shelton. A perturbation scheme for spherical arrangements with application to molecular modeling. In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry*, SCG '97, pages 183–192, New York, NY, USA, 1997. ACM.

[19] M. C. Jordan. *Cours d'analyse de l'École Polytechnic, Tome Troisième, Calcul Intégral, équations différentielles*. Gauthier-Villars, Paris, 1887.

[20] B. Lévy. Robustness and efficiency of geometric programs. *Comput. Aided Des.*, 72(C):3–12, Mar. 2016.

[21] K. Ouchi and J. Keyser. Handling degeneracies in exact boundary evaluation. In *Proceedings of the Ninth ACM Symposium on Solid Modeling and Applications*, SM '04, pages 321–326, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[22] A. A. G. Requicha and H. B. Voelcker. Solid modeling: A historical summary and contemporary assessment. *IEEE Comput. Graph. Appl.*, 2(2):9–24, Feb. 1982.

[23] P. Schorn. An axiomatic approach to robust geometric programs. *J. Symb. Comput.*, 16(2):155–165, Aug. 1993.

[24] Wikipedia contributors. Non-archimedean ordered field — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Non-Archimedean_ordered_field&oldid=972176701`, 2020. [Online; accessed 9-September-2022].