EXACT INTERSECTION OF 3D GEOMETRIC MODELS

Salles V. G. de Magalhães, UFV/RPI
Marcus V. A. Andrade, UFV
W Randolph Franklin, RPI
Wenli Li, RPI
Mauricio G. Gruppi, UFV
Mesh intersection

- Polygonal map overlay/intersection: important GIS problem

- 2D intersection also extends to 3D.

- Examples:
  - CAD: intersection of industrial parts.
  - GIS: terrain models (layers of soil in a mine, volume that will be dug, city models, etc)

- Our focus: 3D triangulated meshes
Challenge

- Finite precision of floating point $\rightarrow$ roundoff errors.
- Common techniques (snap rounding, epsilon tweaking, etc): no guarantee.
- Big amount of data & 3D $\rightarrow$ increase problem.
- Exactness and performance: very important (e.g. guaranteed subroutine)

Source: Kettner et al., Classroom examples of robustness problems in geometric computations
Examples from CGAL mailing list (there are several other similar threads): People want exactness and performance!

---

I have implemented boolean operation using nef polyhedra. The performance however leaves something to be desired. A simple union between two spheres constructed from roughly 400 triangles each, take almost 8 seconds to solve (in release mode). Is this expected or might i be doing something to inhibit the performance. I am using an epec kernel which i know might impact performance. I have however been unable to get it working with other kernels. Even so, 8 seconds seems excessive for a simple union.

Are...
Key techniques

- We've been using a combination of 5 techniques
  - Arbitrary precision rational numbers: for exactness.
  - Simulation of Simplicity: for ensuring all the special cases are properly handled.
  - Simple data representation and local information: parallelization and correctness.
  - Parallel programming: explore better the computing capability of current hardware.
  - Two-level uniform grid: accelerate computation; quickly constructed in parallel.
Example: computing intersections

- “Brute force”: \( O(|A| \times |B|) \)
- Other possible techniques:
  - Sweep-line
  - Complicate and doesn't parallelize
- Uniform grid
  - Tests: very efficient
Uniform grid

• Insert edges in grid cells (edge may be in several cells).
• For each grid cell $c$, compute intersections in $c$.
  • 3D version is analogous

4x7 uniform grid.
Blue map: 8 edges
Black map: 16 edges
Uniform grid

- Uniform Grids work well for uneven data.
- For very uneven data: 2-level uniform grid.
Simulation of Simplicity

- Special/degenerate cases
  - Usually difficult to handle
  - Mainly in 3D

- How to handle them efficiently and effectively?

- Simulation of Simplicity (SoS), Edelsbrunner and Mücke:
  - Simple and efficient general purpose technique.
  - Globally consistent
  - Basic idea: if points are perturbed, the degeneracies in geometrical problems will disappear and do not need to be treated.
Simulation of Simplicity

• Perturbation
  • Points are perturbed using orders of infinitesimals $\varepsilon^i$
  • Infinitesimal: indeterminate (code simulates the effect of the infinitesimals – we do not actually use specific infinitesimals).
Our previous works using these techniques

- **EPUG-OVERLAY**
  - Exact.
  - Parallel.
  - Uniform Grid.

- **PinMesh**
  - Exact and efficient point location
  - Point location: subproblem of the mesh overlay

- **EPLSimp**
  - Map simplification
  - Exact, topologically correct and parallel
3D Point Location - PinMesh

• Input:
  • A mesh (set of triangles, each one with labels of the region on its positive and negative sides)
  • A set of query points
• Objective determine where the query points are.
3D Point Location - PinMesh

• Idea:
  • Trace a vertical ray from each point
  • Find the lowest triangle above the point
  • Use orientation to locate the point

• Techniques:
  • SoS: no ray hits edges, vertices or starts on triangle.
  • Uniform grid: reduce ray-triangle intersection tests.
  • Parallel programming: grid creation and queries.
  • Rational numbers: exact computation.

• Result: PinMesh is very efficient and robust
2D map overlay algorithm - EPUG-OVERLAY

- Given two polygonal maps, compute their intersection

- Idea:
  - Find all intersections using a uniform grid.
  - Split edges at intersection points.
  - Locate vertices/edges in the other map (using grid).
  - Compute output polygons.
2D map overlay algorithm - EPUG-OVERLAY

- Given two polygonal maps, compute their intersection

- Idea:
  - Find all intersections using a uniform grid.
  - Split edges at intersection points.
  - Locate vertices/edges in the other map (using grid).
  - Compute output polygons.
2D map overlay algorithm - EPUG-OVERLAY

• Given two polygonal maps, compute their intersection

• Idea:
  • Find all intersections using a uniform grid.
  • Split edges at intersection points.
  • Locate vertices/edges in the other map (using grid).
  • Compute output polygons.

• (u, w) divided into 7 segments.
  • 5 will be in output.
2D map overlay algorithm - EPUG-OVERLAY

• Given two polygonal maps, compute their intersection

• Idea:
  • Find all intersections using a uniform grid.
  • Split edges at intersection points.
  • Locate vertices/edges in the other map (using grid).

• Compute output polygons.

• (u,w) divided into 7 segments.
  • 5 will be in output.
2D map overlay algorithm - EPUG-OVERLAY

• Given two polygonal maps, compute their intersection

• Idea:
  • Find all intersections using a uniform grid.
  • Split edges at intersection points.
  • Locate vertices/edges in the other map (using grid).

• Compute output polygons.

Case 2

(i₆,w) → outside other map

• (u,w) divided into 7 segments.
• 5 will be in output.
Current work: 3D-EPUG-OVERLAY

- Apply the same techniques, but for 3D mesh intersection
  - Rational numbers
  - “3D maps” represented by a set of triangles
  - Triangles: left/right objects
  - 3D uniform grid for intersection and point in polygon
  - Simulation of Simplicity
  - Algorithm designed to be parallel

Source: Autodesk

Source: Rockworks

source: wikipedia
First step: triangle-triangle intersections

- A 3D uniform grid is created.
- Triangles from both meshes are inserted into the cells their AABB intersect.
- Cells with “too many” triangles are refined, creating a second level grid.
- Pairs of triangles in each cell are tested for intersection → “Too many” = number of pairs of triangles.
- Intersection tests: Moller's algorithm for performance.
- Cells do not influence each other → process them in parallel.
Second step: retessellation

- Triangles are, then, split at the intersections.
- Similar to splitting edges in EPUG-OVERLAY.
- Intersection on each triangle → planar subdivision → retriangulation.
- Again, this step can be done in parallel on the triangles.

Red intersecting triangle: split into 2 polygons → 7 triangles
Second step: retessellation

- Retessellated mesh: equivalent to the original
  - Union of each split triangle is equal to the original triangle
  - Non split triangles will also be in retessellated mesh

- After retessellation: intersections will only happen at common vertices/edges.
Third step: classification

- Finally, triangles are classified.
  - Similar to edge classification in EPUBUG-OVERLAY.
  - Only two basic cases for each triangle \( t \) (bounding \( A, B \)):
    - \( t \) outside other mesh \( \rightarrow t \) will not be in the output.
    - \( t \) inside region \( R \) of the other mesh \( \rightarrow t \) will bound \( R \cap A \) and \( R \cap B \).
Third step: classification

- How to locate a triangle?
  - Simple and fast solution: PinMesh

Outside green region → not in the output

Inside green region → bound (Green ∩ Red), exterior
Special cases

- Under development
- Proposed solution: SoS
- SoS was successfully employed in EPUO-CLICKLAY
  - Idea: translate one of the maps by \((\varepsilon, \varepsilon^2)\) → no common edges/intersection at endpoints
- Example: two coincident polygons → translation \((\varepsilon, \varepsilon^2)\) → non coincident
Special cases

- Example: two coincident polygons $\rightarrow$ translation $(\epsilon, \epsilon^2) \rightarrow$ non-coincident
- **Intersection** computation: two intersections $u$ and $v
Special cases

- Example: two coincident polygons → translation \((\varepsilon, \varepsilon^2)\) → non coincident
  - Intersection computation: two intersections \(u\) and \(v\)
- **Retesselation**: a-d split into a-u, u-d
- Classification:
  - a-u is outside the other polygon → not in output
  - u-d is inside the other polygon → u-d in the output
    - u-d will bound the interior and the exterior of the output polygon
Special cases

- Example: two coincident polygons $\rightarrow$ translation $(\varepsilon, \varepsilon^2) \rightarrow$ non coincident
- Intersection computation: two intersections $u$ and $v$
- Retessellation: a-d split into a-u, u-d
- Classification:
  - a-u is outside the other polygon $\rightarrow$ not in output
  - u-d is inside the other polygon $\rightarrow$ u-d in the output
    - u-d will bound the interior and the exterior of the output polygon
Special cases

- Example: two coincident polygons → translation \((\varepsilon, \varepsilon^2)\) → non coincident
  - But… a-b-c-d is equal to e-f-g-h
  - u-f-v-d should also represent the same polygon!
  - The translation is only conceptual! It only affects the conditionals
  - \(u=a=e\) and \(v=c=g\) → the polygons are the same!
Special cases

• In PinMesh we also employ a similar idea: all the query points are translated by \((\varepsilon, \varepsilon^2, \varepsilon^3)\)
• We believe this same perturbation scheme will be suitable for intersecting 3D meshes.
Example of result

- Intersection of two big meshes from AIM@SHAPE:
  - Ramesses: 1.7 million triangles
  - Neptune: 4 million triangles
Example of result

- Hard to process triangles → roundoff errors
Example of result

- Hard to process triangles → roundoff errors
Example of result

- Hard to process triangles $\rightarrow$ roundoff errors
Current work

- Employ techniques successfully applied in our previous works.

- This algorithm → few data dependency → very parallelizable.
  - Uniform grid creation: edges in parallel.
  - Locate vertices in polyhedra.
  - Compute intersections: cells in parallel.
  - Compute output triangles: process input triangles in parallel.

- Most of computers: multicore → OpenMP.

source: wikipedia
Conclusions

- 3D-EPUG-OVERLAY
  - Exact
  - Parallel
  - Uniform grid

- Part of a bigger project
  - Exact and parallel geometric algorithms
  - Applications in GIS, CAD and AM

- Ongoing/future work
  - SoS perturbation scheme
  - Code optimizations
  - Application of these ideas to other algorithms
Thank you!

Acknowledgement:
Simulation of Simplicity

- Example: how to check if a point $q$ is directly “below” the interior of a triangle $t$?
- Project $q$ and $t$ to $z=0$, check if $q'$ is inside $t'$ (also check $q_z$).
- Is $q'$ inside $t'$? $\rightarrow$ barycentric coordinates $\rightarrow 0 < \lambda_i < 1$ for $i=1,2$ and 3?

$$
\lambda_0 = \frac{(t'_{1y} - t'_{2y}) \times (q'_{x} - t'_{2x}) + (t'_{2x} - t'_{1x}) \times (q'_{y} - t'_{2y})}{\text{det}}
$$

$$
\lambda_1 = \frac{(t'_{2y} - t'_{0y}) \times (q'_{x} - t'_{2x}) + (t'_{0x} - t'_{2x}) \times (q'_{y} - t'_{2y})}{\text{det}}
$$

$$
\lambda_2 = 1 - \lambda_0 - \lambda_1
$$

$$
\text{det} = (t'_{1y} - t'_{2y}) \times (t'_{0x} - t'_{2x}) + (t'_{2x} - t'_{1x}) \times (t'_{0y} - t'_{2y})
$$
Simulation of Simplicity

- Degeneracies: \( \det = 0 \rightarrow \) vertical triangle
- Point on boundary of \( t' (\lambda_i = 0 \text{ or } 1) \).

\[ \text{SoS} \rightarrow q(x,y,z) \rightarrow q_\varepsilon(x+\varepsilon,y+\varepsilon^2,z+\varepsilon^3), \quad q'(x,y) \rightarrow q'_\varepsilon(x+\varepsilon,y+\varepsilon^2) \]

- \( q'_\varepsilon \) will never be on a vertex or edge of \( t' \).
  - \( q' \) is not on vertex/edge \( \rightarrow q'_\varepsilon \) is also not on vertex/edge (infinitesimal).
  - \( q' \) is on vertex/edge \( \rightarrow q'_\varepsilon \) is not on vertex/edge (infinitesimal/slope).
    - Ex: \( q' \) is on an edge \( \rightarrow q'_\varepsilon \) cannot be on the same edge (slope would be infinitesimal)
Simulation of Simplicity

- SoS implementation:
  - $q'_\varepsilon$ will never be on a vertex or edge of $t'$ → if $\det=0$ → false
  - Replace $q'$ with $q'_\varepsilon$

\[
\lambda_0 = \frac{(t'_{1y} - t'_{2y}) \times (q'_x - t'_{2x}) + (t'_{2x} - t'_{1x}) \times (q'_y - t'_{2y})}{\det}
\]

\[
\lambda_1 = \frac{(t'_{2y} - t'_{0y}) \times (q'_x - t'_{2x}) + (t'_{0x} - t'_{2x}) \times (q'_y - t'_{2y})}{\det}
\]

\[
\lambda_2 = 1 - \lambda_0 - \lambda_1
\]

\[
\det = (t'_{1y} - t'_{2y}) \times (t'_{0x} - t'_{2x}) + (t'_{2x} - t'_{1x}) \times (t'_{0y} - t'_{2y})
\]
Simulation of Simplicity

- SoS implementation:
  - $q'_\varepsilon$ will never be on a vertex or edge of $t'$ $\rightarrow$ if $\det=0$ $\rightarrow$ false
  - Replace $q'$ with $q'_\varepsilon$ $\rightarrow$ $\lambda_i$ with $\lambda_{\varepsilon i}$
  - E.g.: is $0 < \lambda_{\varepsilon 0}$?
    - $\lambda_0 \neq 0$ $\rightarrow$ check $\lambda_0$
    - $\lambda_0 = 0$ $\rightarrow$ check $t'_1 y - t'_2 y$
    - $t'_1 y - t'_2 y = 0$ $\rightarrow$ check $t'_2 x - t'_1 x$
  - Both can't be 0.

\[
\begin{align*}
\lambda_{\varepsilon 0} &= \lambda_0 + \frac{(t'_1 y - t'_2 y) \times \varepsilon + (t'_2 x - t'_1 x) \times \varepsilon^2}{\det} \\
\lambda_{\varepsilon 1} &= \lambda_1 + \frac{(t'_2 y - t'_0 y) \times \varepsilon + (t'_0 x - t'_2 x) \times \varepsilon^2}{\det} \\
\lambda_{\varepsilon 2} &= 1 - \lambda_{\varepsilon 0} - \lambda_{\varepsilon 1}
\end{align*}
\]