An exact and efficient 3D mesh intersection algorithm using only orientation predicates
Salles V. G. Magalhães¹,², W. Randolph Franklin³, Marcus V. A. Andrade¹
¹Universidade Federal de Viçosa, Brazil ²Rensselaer Polytechnic Institute, USA

Intersecting meshes

- Objective: Efficiently compute the exact intersection between two triangular meshes.
- Applications in CAD, GIS, Additive Manufacturing, etc.
- Example: 3D mesh may represent objects in a CAD system.
- Challenges
  - Special cases and roundoff errors
  - Applications may give inconsistent results or even crash
- People want exactness and performance.

The algorithm

- Tries to process triangles independently (→ parallelism)
- Intersect pairs of triangles
- Grid index
- Fast triangle-triangle intersection algorithm (Möller)
- Retesselation
  - Triangle split at intersection edges
  - Polygonal subdivision is created and retriangulated (ear-clipping)

Triangular classification

- Input and new triangles are classified.
- If t is bounding objects (a,b) and is inside object c of the other mesh, in the output t will bound (a∩b∩c) (other booleans → similar strategy)
- How to determine in what object of the other mesh t is? → traverse mesh and label accordingly
- Start with an input vertex: point location → location of triangle containing it.
- Two triangles share a “regular edge” → they are in the same object.
- Two triangles share an edge generated from an intersection → they are in different objects (triangle labels give the locations).

Special cases

- Challenging, hard to treat
- Solution: Simulation of Simplicity
- Points symbolically perturbed with infinitesimals (ε does not “exist”, simulated effect)
- If input is non-degenerate → no change.
- Performance
- Otherwise → SoS → no coincidence & globally consistent result.

- Meshes are symbolically perturbed
  - Mesh 0: (x,y,z) → (x,y,z)
  - Mesh 1: (x,y,z) → (x+ε,y+ε,z+ε)
- After perturbation:
  - A vertex from one mesh will never be on the plane of a triangle from the other mesh.
  - An edge from one mesh will never intersect an edge from the other mesh.
  - Two coplanar triangles from different meshes will never intersect.

Implementation

- Two versions of each algorithm: one using only orientation predicates.
- Tri-tri intersection: 5 3D orientations for each edge-triangle (Segura and Feito).
- Retesselation: sort intersection points along edges: 3D orientation
- Extracting faces from retesselation: 2D and 3D orientation.
- Ear-clipping: detecting convex vertices and point in triangle → 2D orientation
- Challenge: vertices generated from intersection may be argument of the predicates → represent them as pairs (edge-triangle).

Data representation

- Triangular soup:
  - Oriented triangles.
  - Each triangle stores the ids of the two objects it bounds (on the negative and positive sides).
- Supports:
  - Multiple components
  - Components with different ids (“materials”)
  - Non-manifoldness
  - Nested components
- Self intersections → contradictions

Novelties

- Parallel: for multi-core computers
- Grid indexing: efficient parallel uniform grid
- Special cases: carefully treated using Simulation of Simplicity (SoS).
- All computation: exact (GMP rational)
- For triangulated meshes:
  - Widely used
  - Simple representation
  - Supports multi-material and “internal structure”

Performance experiments

- Dual 8-core Xeon, 128 GB of RAM
- Algorithm still under development (can be improved)
- Comparison with LibiGL (exact algorithm, resolves self intersections)

Conclusions, limitations, future work

- Parallel and efficient machines → we can afford exact algorithms.
- Future work:
  - Improve efficiency
  - Validate results
  - Experiments with huge meshes, tetrahedral meshes, etc.
  - Compare with more methods (CGAL, QuickCSG, etc)
  - Floating-point input → exact and more efficient predicates
  - Result is valid for the symbolically perturbed input
  - If output is considered without the perturbation → it may contain polyhedra with volume 0, triangles with area 0, etc.
  - Perturbed output: also useful
  - Future work: how to remove perturbation from output?
  - Source code: freely available (soon on Github)

Acknowledgements

This work was partially supported by Capes (Ciência Sem Fronteiras), FAPEMIG and NST
grant 15-1112777