An exact and efficient 3D mesh intersection algorithm using only orientation predicates

Salles V. G. Magalhães[1,2], W. Randolph Franklin[2], Marcus V. A. Andrade[1]
[1]Universidade Federal de Viçosa, Brazil [2]Rensselaer Polytechnic Institute, USA

Intersecting meshes

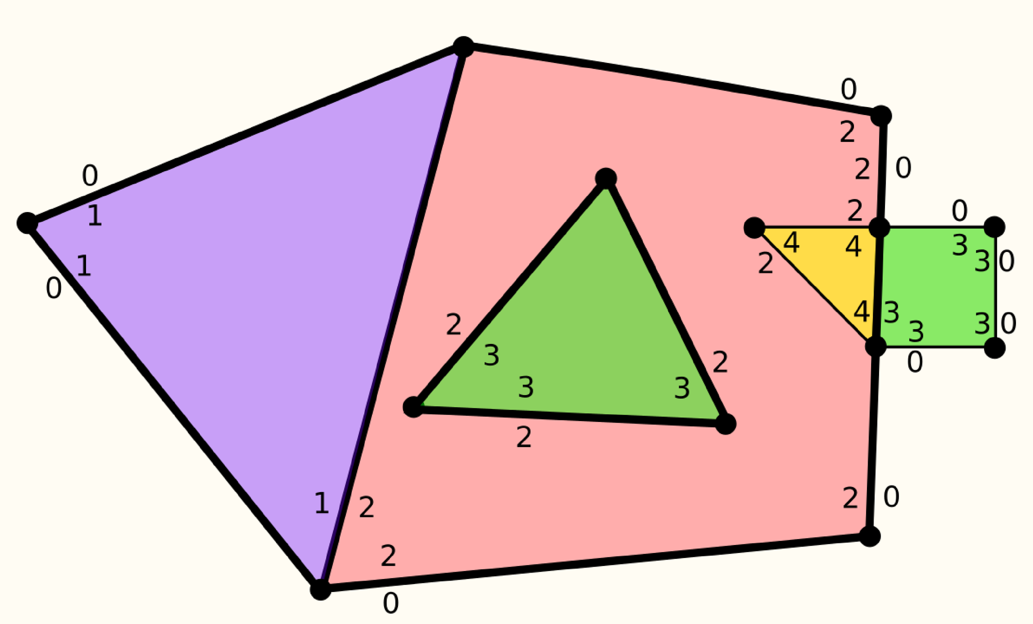
- Objective: Efficiently compute the exact intersection between two triangular meshes.
- Applications in CAD, GIS, Additive Manufacturing, etc.
- Example: 3D mesh may represent objects in a CAD system.
- Challenges
  - Special cases and roundoff errors
  - Applications may give inconsistent results or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

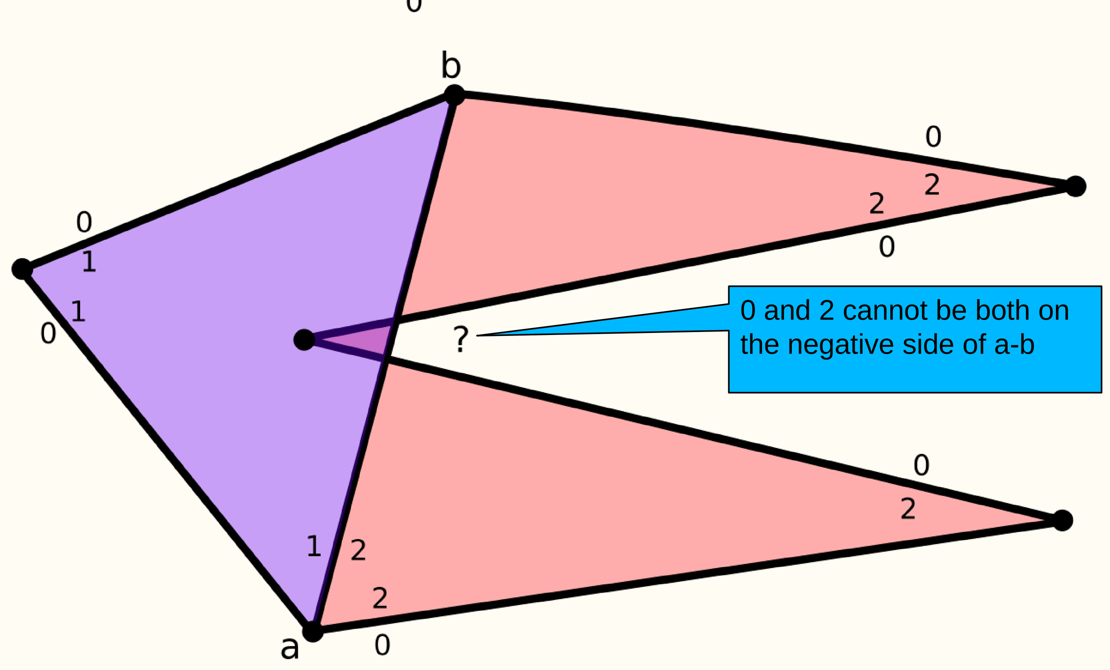
Novelties

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

Data representation

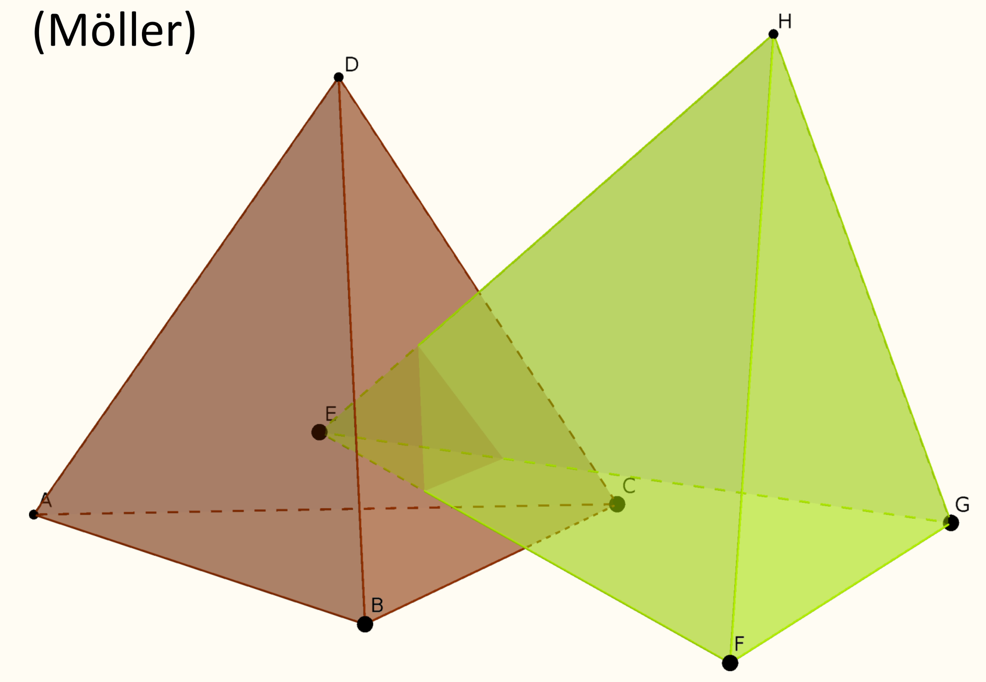
- Triangular soup:
  - Oriented triangles.
  - Each triangle stores the ids of the two objects it bounds (on the negative and positive sides).
- Supports:
  - ✓Multiple components
  - ✓Components with different ids ("materials")
  - ✓Non-manifoldness
  - ✓Nested components
  - × Self intersections →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



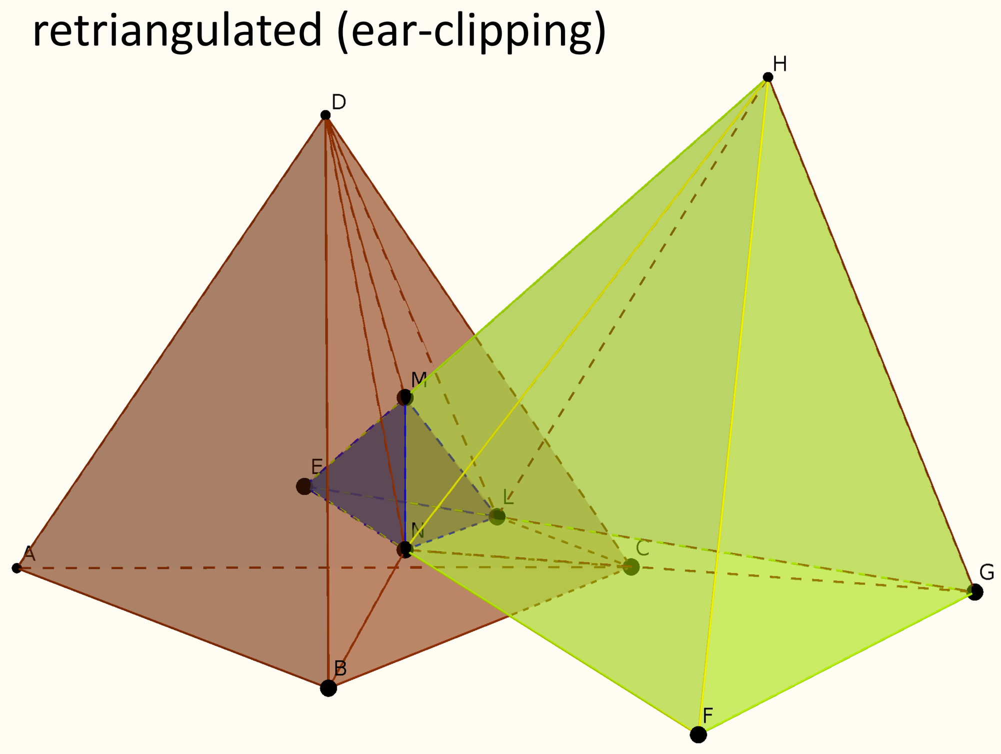


The algorithm

- Tries to process triangles independently (→ parallelism)
- Intersect pairs of triangles
  - Grid index
  - Fast triangle-triangle intersection algorithm (Möller)



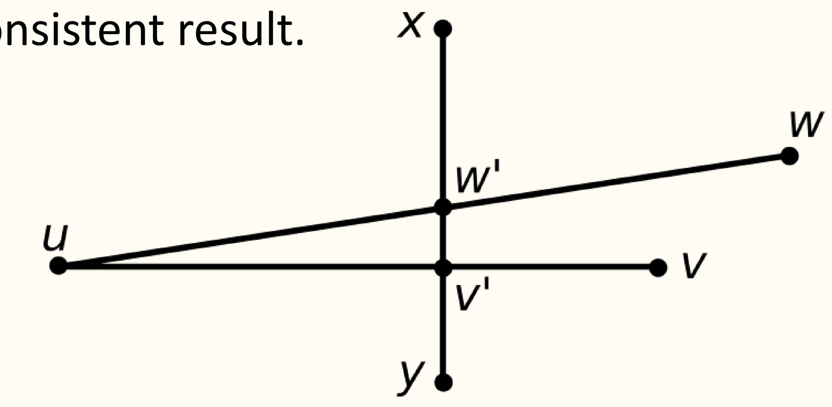
- Retesselation
  - Triangle split at intersection edges
  - Polygonal subdivision is created and retriangulated (ear-clipping)



- Triangle classification
  - Input and new triangles are classified.
  - If *t* was bounding objects (*a*,*b*) and is inside object c of the other mesh, in the output *t* will bound ($a \cap c$, $b \cap c$) (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

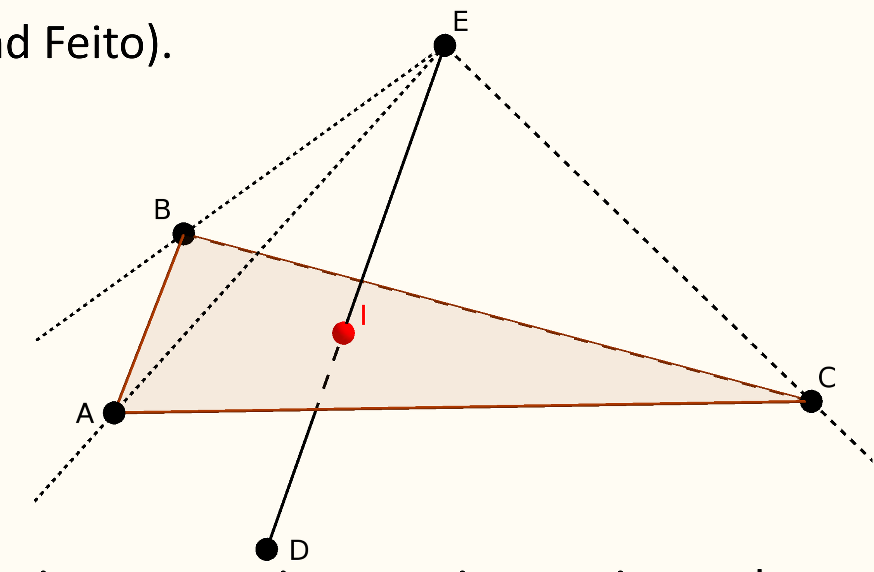
- Challenging, hard to treat
- Solution: Simulation of Simplicity
  - Points symbolically perturbed with infinitesimals (ε does not "exist", simulated effect)
- If input is non-degenerate → no change.
  - Performance
- Otherwise → SoS → no coincidence & globally consistent result.



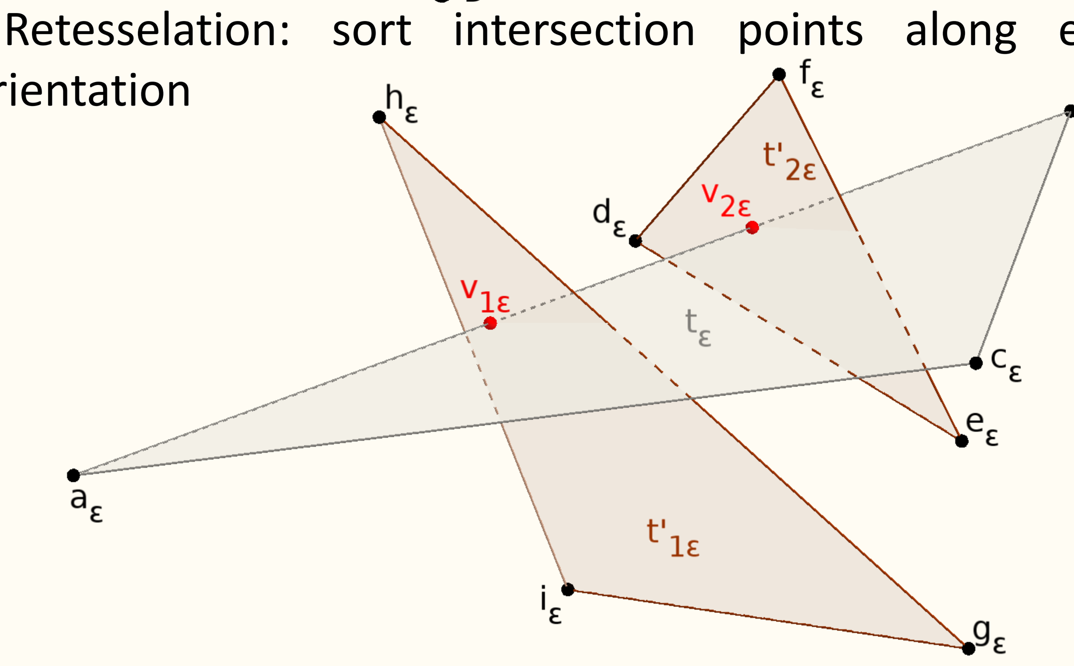
- Meshes are symbolically perturbed
  - Mesh 0: (x,y,z) → (x,y,z)
  - Mesh 1: (x,y,z) → $(x+\varepsilon, y+\varepsilon^2, z+\varepsilon^3)$
- After perturbation:
  - A vertex from one mesh will never be on the plane of a triangle from the other mesh.
  - An edge from one mesh will never intersect an 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: 1D and 2D 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).

Performance experiments

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

Input		Thousand faces		Times (s)		Speedup
Mesh 0	Mesh 1	Mesh 0	Mesh 1	Our alg.	LibiGL	
BumpySphere	BumpyTorus	11	34	0.27	4.49	16.4
RoundOcta	SharpSphere	33	21	0.09	1.74	18.4
Camel	Camel	69	69	20.04	16.71	0.8
Bimba	NewYear	150	10	0.24	5.13	21.5
Camel	Armadillo	69	331	0.41	14.31	35.0
Armadillo	Armadillo	331	331	94.16	75.81	0.8
Th10k:461112	Th10k:461115	805	822	2.79	64.69	23.1
Kitten	RedCircBox	274	1402	1.23	36.28	29.5
Bimba	Vase	150	1792	1.86	65.44	35.2
Th10k:226633	Th10k:461112	2452	805	3.22	119.97	37.3
Ramesses	Ram.Transl	1653	1653	4.61	102.72	22.3
Ramesses	Ram.Rotated	1653	1653	6.55	122.46	18.7
Horse	Neptune	97	4008	4.42	103.25	23.4
Neptune	Ramesses	4008	1653	5.49	150.03	27.3
Neptune	Nept.Transl.	4008	4008	10.84	247.90	22.9

- For the largest dataset (last row): 5 million pairs of triangles tested for intersection, 78 thousand pairs of triangles intersect, 389 thousand triangles generated from retesselation.

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 NSF grant IIS-1117277

CAPES FAPEMIG NSF