Rensselaer Polytechnic Institute, Troy NY USA
Universidade Federal de Vicosa, MG, Brasil

RPI

UFV

Fast exact parallel 3D mesh intersection algorithm using only orientation predicates

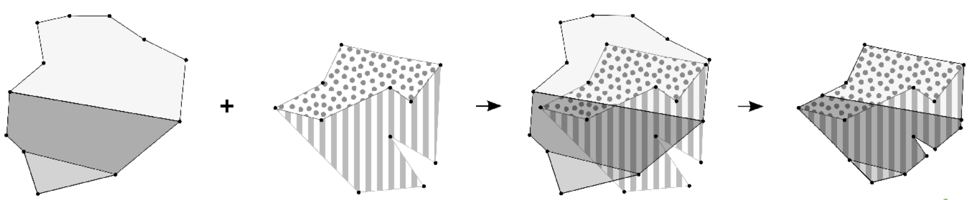
W Randolph Franklin, RPI
Salles V. G. de Magalhães, UFV/RPI
Marcus V. A. Andrade, UFV

Society for Industrial and Applied Mathem

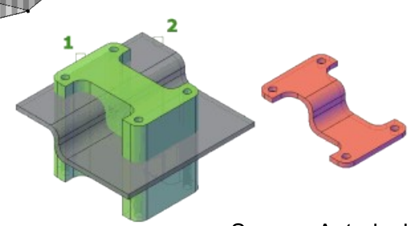
FAST, EXACT, PARALLEL MESH INTERSECTION ALGORITHM

Mesh intersection

- Polygonal map overlay/intersection: important CAD/GIS problem



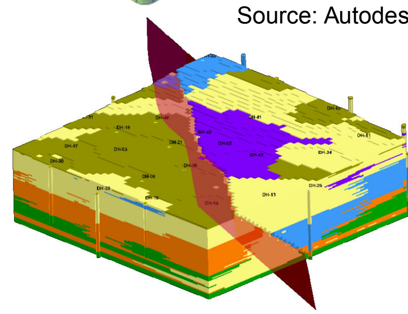
- 2D intersection also extends to 3D.



Source: Autodesk

- Applications: CAD, Additive Manufacturing, GIS, cross-interpolation after remeshing in CFD

- Our focus: 3D triangulated meshes



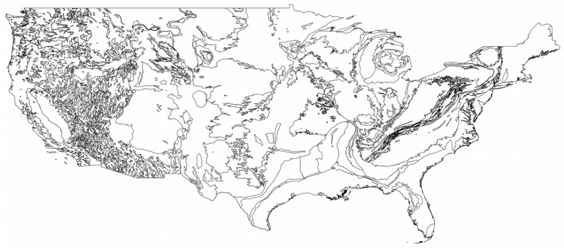
Source: Rockworks

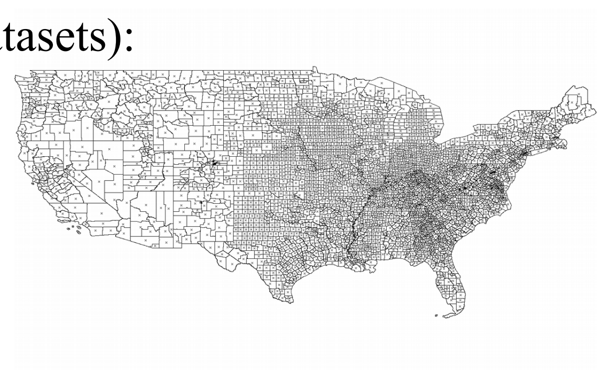
EPUG-Overlay: 2D planar graph overlay

Previous step, presented at 2015 ACM BIGSPATIAL

Biggest example:

- USWaterBodies: 21,652,410 vertices, 219,831 faces, with
- USBlockBoundaries: 32,762,740 vertices, 518,837 faces.
- (Images are of simpler similar datasets):



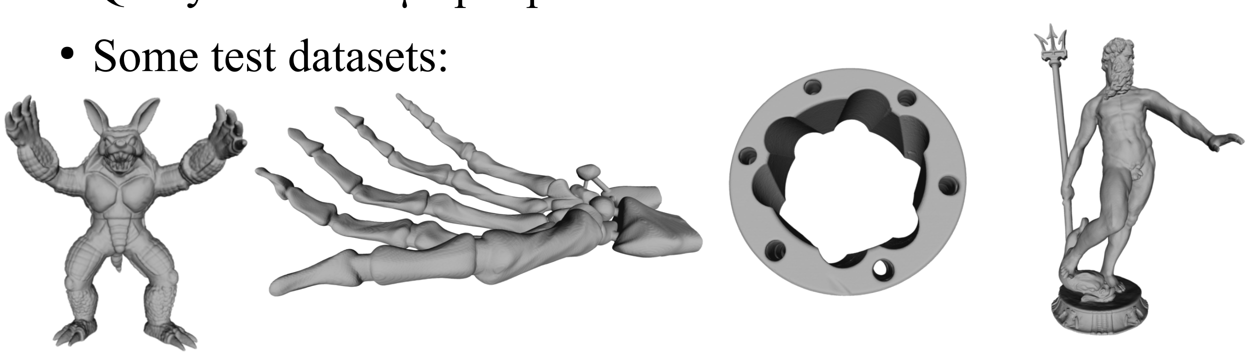


Time (w/o I/O):

- 1342 secs (1 thread)
- 149 secs (16 cores, 32 threads). 9X parallel speedup

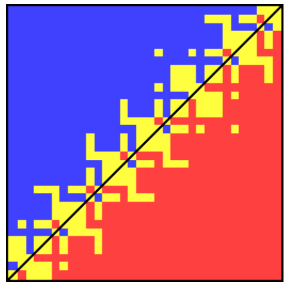
PINMESH: 3D point location

- Previous step, presented at 2016 Berlin Geometry Summit
- Uses rational numbers, Simulation of Simplicity, uniform grid, parallelism, simple data structures
- Biggest example: sample dataset with 50 million triangles.
  - Preprocessing: 14 elapsed seconds on 16-core Xeon processor.
  - Query time: 0.6 µs per point.
  - Some test datasets:



Roundoff Challenge

- Finite precision of floating point →roundoff errors.
  - Common techniques (snap rounding, epsilon tweaking, etc): no guarantee.



Source: Kettner et al., Classroom examples of robustness problems in geometric computations

- Big amount of data & 3D→ increase problem.

- Exactness and performance: very important (e.g. guaranteed subroutine)

Examples from CGAL mailing list (there are several other similar threads): People want exactness and performance!

Nov 10, 2015; 4:36am **Boolean performance** Reply | Threaded | M

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
tim

--
Re
Ta
So

--
Yo
To
http

dekosser Dec 07, 2009; 10:50am **Re: RE: Performance of boolean operations on Nef_polyhedron_3**

19 posts

> I have found and evaluated another GPL library that specializes in boolean
> operations on Polyhedra. This library (CARVE) performed on average 100
> (ONE
> HUNDRED) times faster than CGAL with the typical use-cases that apply to
> our
> application. It also proved completely computationably stable with our
> tests.

For the record, I also evaluated Carve for our project, and found similar performance results (at least for low volumes of data). It's mostly imputable to the overhead of using an exact arithmetic kernel.

On the other hand I rapidly ran into instability and crashes even with some simple use cases. Our project required perfect robustness, and so CGAL/Nef3 was ultimately retained for this reason.

Fred

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.

Rational numbers

- Each component of each coordinate is a ratio of integers
  - No rounding or finite precision errors.
  - Each integer: array of groups of digits
  - Uses GMPXX
  - Rationals double in size with each operation: 2/3+4/5=22/15
  - However depth of computation tree is small
  - Problem: GMPXX liberally constructs new objects on heap
  - Heap is superlinear time in number of objects, and parallel hostile.
  - We minimize heap constructions.
  - Increased execution time is tolerable.

Current hardware

Massive shared memory

- is an underappreciated resource.
- External memory algorithms not needed for many problems.
- Virtual memory is obsolete.
- $40K buys a workstation with 80 cores and 1TB of memory.

Parallel computing

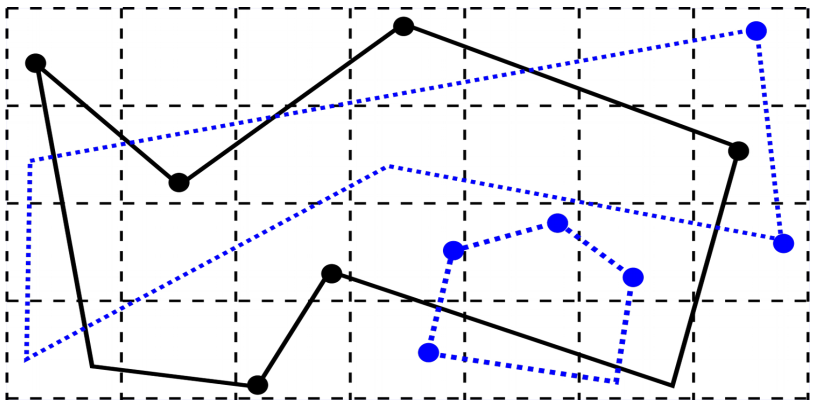
- Almost all processors, even my smart phone, are parallel.
- Algorithms that don't parallelize are obsolete.
- Nvidia GPUs are almost ubiquitous.
- However, 1 Xeon core is 20x more powerful than 1 CUDA core.

Component: computing 2D intersections

- "Brute force": O(|A| x |B|)
- Other possible techniques:
  - Sweep-line
  - Complicated and doesn't parallelize
- Uniform grid
  - Theoretical and experimentally: very efficient

Uniform Grid

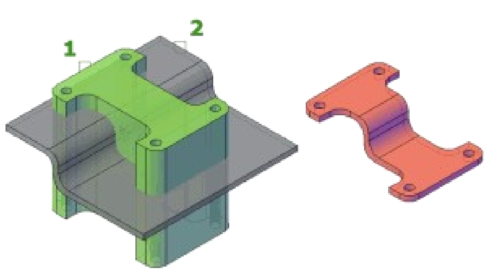
- Insert edges in grid cells (edge may be in several cells).
- For each grid cell *c*, compute intersections in *c*.
- 3D version is analogous
- Provably efficient for I.i.d. input
- Experimentally more efficient on irregular data than octrees



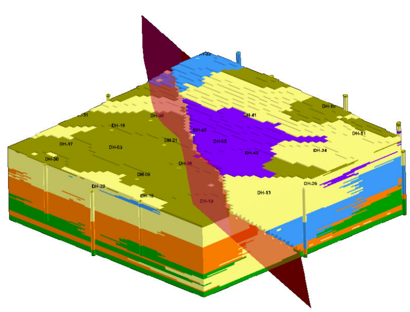
4x7 uniform grid.
Blue map: 8 edges
Black map: 16 edges

3D-EPUG-OVERLAY

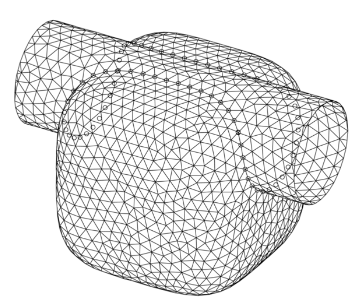
- Apply the key techniques mentioned before for 3D mesh intersection
  - Rational numbers
  - “3D maps” represented by a set of triangles
  - Triangles: left/right objects
  - 3D uniform grid for intersection and point location
  - Simulation of Simplicity
  - Algorithm designed to be **parallel**



Source: Autodesk



Source: Rockworks

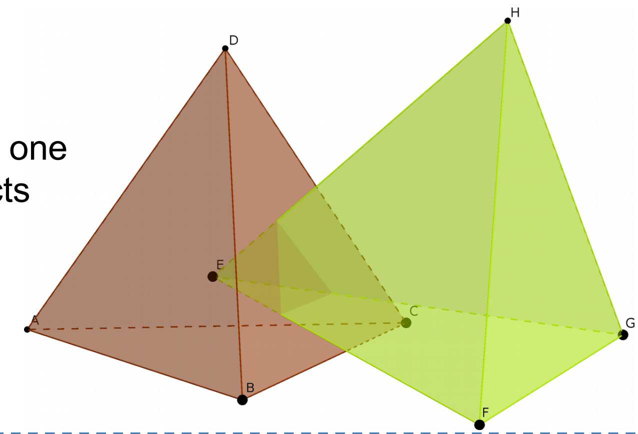


source: wikipedia

First step: triangle-triangle intersections

- A 3D uniform grid is created.
- Triangles from both meshes are inserted into the cells an enclosing cube intersects.
- Cells with “too many” pairs of triangles are refined, creating a second level grid (because the enclosing cube above is suboptimal).
- Intersection tests: Moller's algorithm for performance.
- Cells do not influence each other → process them in parallel

Red mesh: only one triangle intersects green

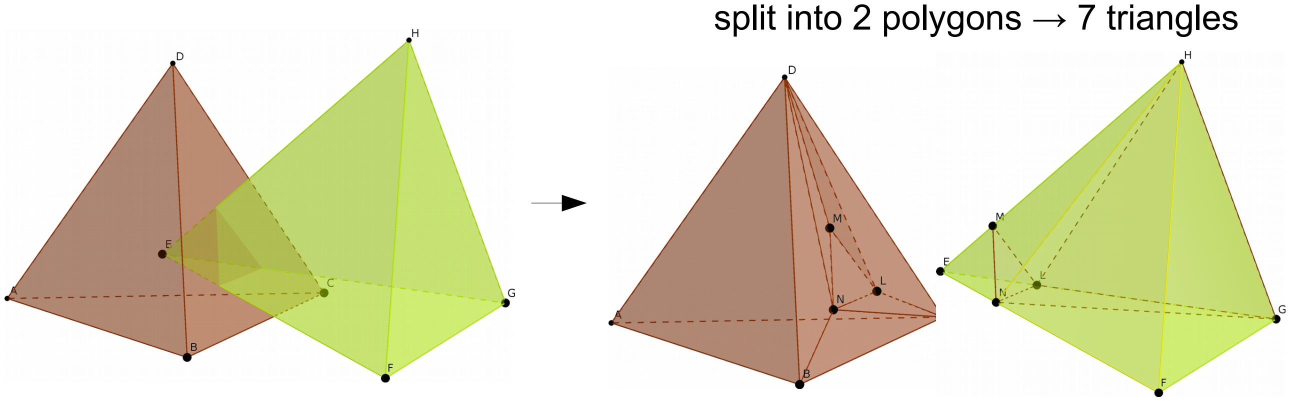


FAST, EXACT, PARALLEL MESH INTERSECTION ALGORITHM

Second step: retesselation

- Triangles are then split at the intersections.
  - 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

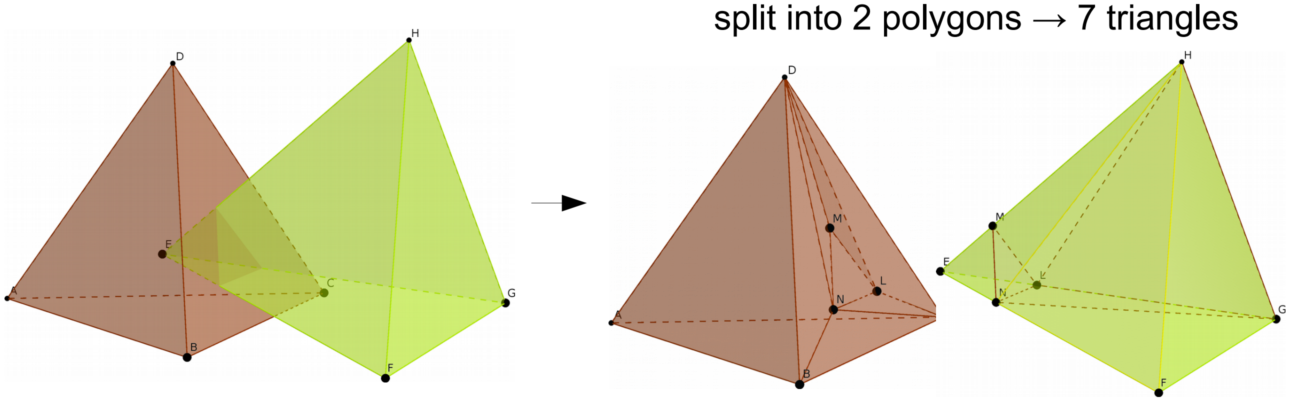


Second step: retesselation

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

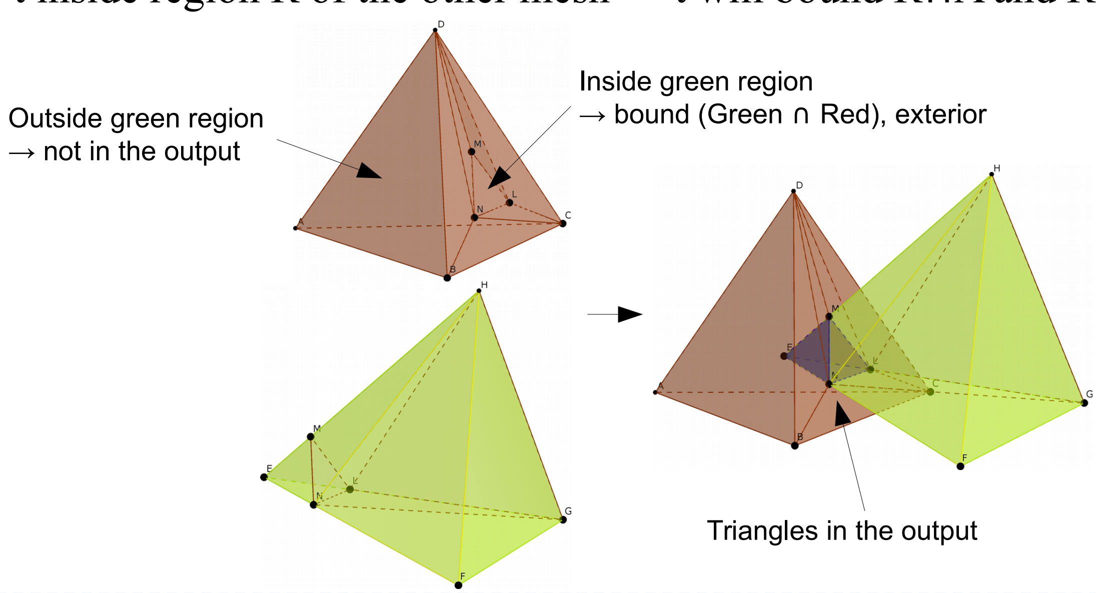
- After retesselation: intersections will only happen at common vertices/edges.

Red intersecting triangle:
split into 2 polygons → 7 triangles



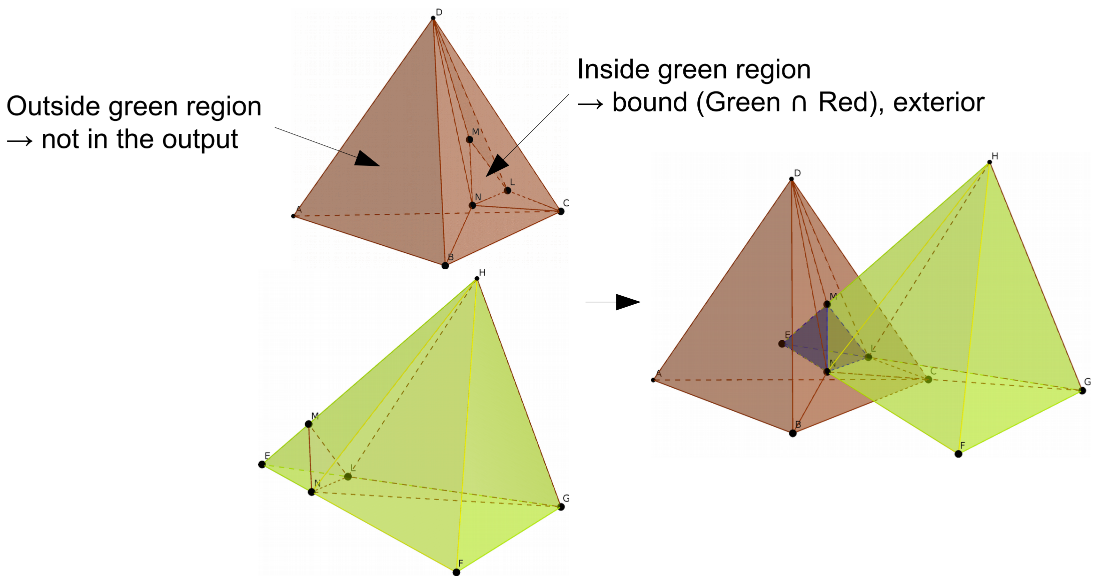
Third step: classification

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



Third step: classification

- How to locate a triangle?
  - Simple and fast solution: point location (PinMesh)



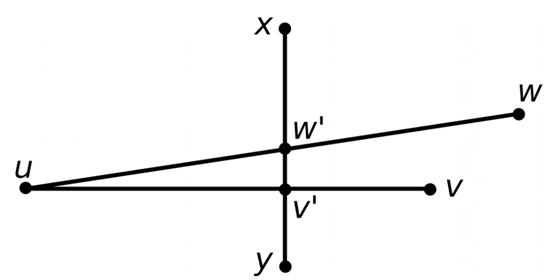
Special cases (geometric degeneracies)

- Ad-hoc enumerating special cases is error-prone.
- How many ways can a line intersect a polyhedron?
- Local rules must lead to a globally consistent result.
- Testing a point against a line must give a consistent result when comparing two polylines.
- Existing programs can get complicated cases wrong.
- Need a general solution.

Simulation of Simplicity

- 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.

Global consistency (*uw*, *uv* were coincident):
- w' is on the positive side of uv
- w' is closer to x than v' is



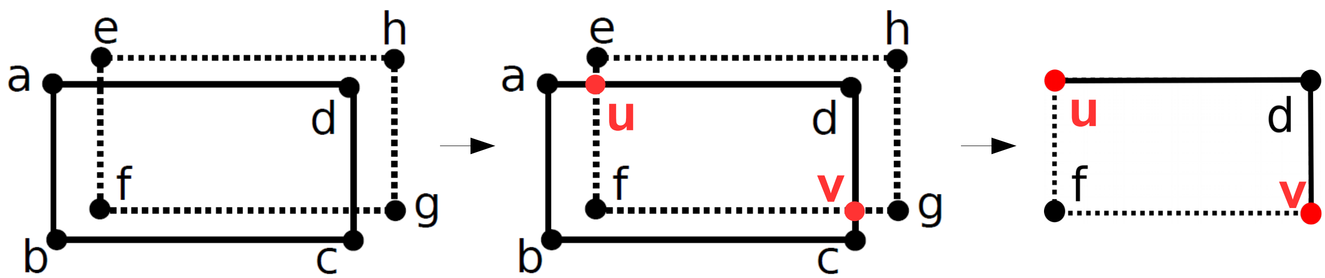
Simulation of Simplicity ctd

- Perturbation
  - Points are perturbed using orders of infinitesimals ε^i
  - Infinitesimal: indeterminate (code simulates the *effect* of the infinitesimals – we do not actually use specific infinitesimals).

Simulation of Simplicity - 3

- SoS has been successfully employed in the 2D version of the problem
  - Idea: translate one of the maps by $(\varepsilon, \varepsilon^2)$ → no common edges/intersection at endpoints

- Example: two coincident polygons → translation $(\varepsilon, \varepsilon^2)$ → no coincidence.
  - Perturbation is only <u>conceptual</u> → resulting rectangle is actually equal to input triangles!



FAST, EXACT, PARALLEL MESH INTERSECTION ALGORITHM

SoS + 3D

- Mesh 0 is not perturbed, mesh 1 is translated by $(\varepsilon, \varepsilon^2, \varepsilon^3)$

- This perturbation presents some properties:
  - Examples:
    - A vertex from a mesh will never be on a triangle of the other one.
    - Two co-planar triangles from distinct meshes never intersect.
  - These properties →no coincidence between the two meshes.
  - Example of consequence: intersection of two triangles (if exist) is always a line segment with non-zero length.

Implementing SoS

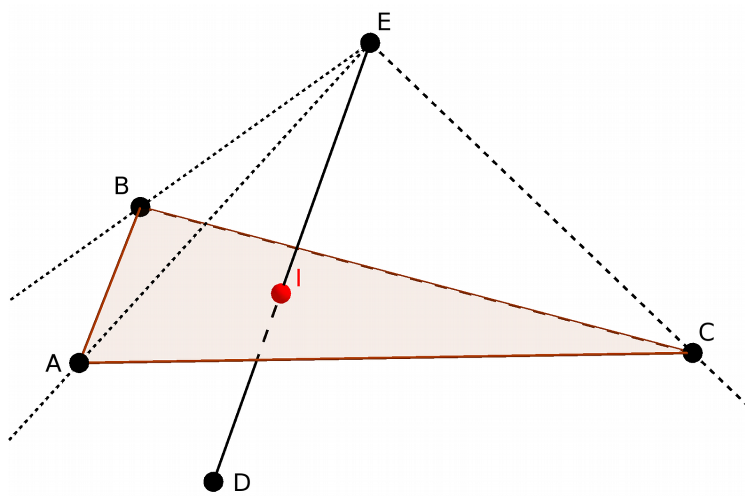
- In a predicate:
  - No coincidence → unperturbed result = perturbed result ≠ 0
  - Coincidence → unperturbed result = 0, unperturbed result ≠ 0

- For performance:
  - Two versions of each predicate:
    - One developed for efficiency (standard algorithms from literature)
    - One for simplicity (using as few predicates as possible).

    - The simpler version: used when a coincidence is detected.
    - Consequence: implement SoS only in few predicates.

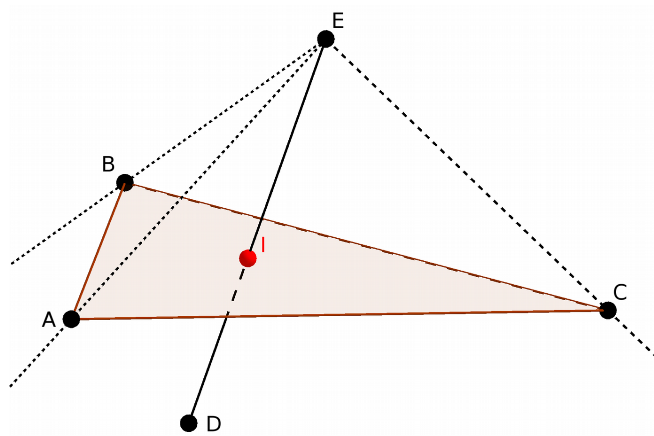
Implementing SoS

- It is possible to implement all the steps of the algorithm employing only orientation (1D, 2D and 3D) predicates.
- Example: intersection of two triangles → check if each edge of one triangle intersects the other triangle.
  - Intersection of line ED with ABC?
  - orientation(A,B,E,D)=orientation(B,C,E,D)=orientation(C,A,E,D) ?



Implementing SoS

- Challenge:
  - If a vertex of mesh 0 has coordinates (x,y,z), what is its perturbed coordinate? Ans: (x,y,z)
  - If a vertex of mesh 1 has coordinates (x,y,z), what is its perturbed coordinate? Ans: $(x+\varepsilon, y+\varepsilon^2, z+\varepsilon^3)$
  - If a vertex generated by an intersection of a triangle with an edge has coordinates (x,y,z), what is its perturbed coordinate?
    - Ans: ???

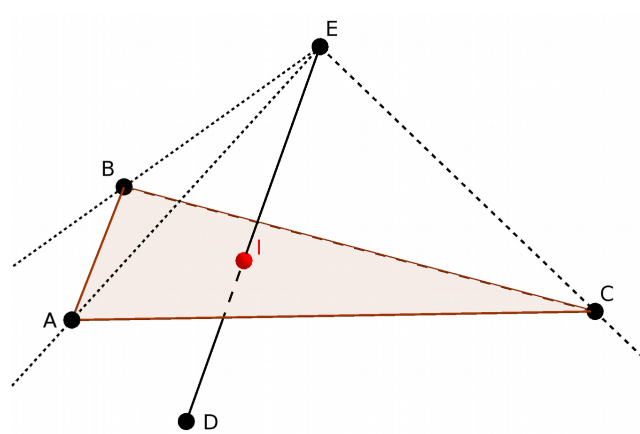


Implementing SoS

- Challenge:
  - If a vertex of mesh 0 has coordinates (x,y,z), what is its perturbed coordinate? Ans: (x,y,z)
  - If a vertex of mesh 1 has coordinates (x,y,z), what is its perturbed coordinate? Ans: $(x+\varepsilon, y+\varepsilon^2, z+\varepsilon^3)$
  - If a vertex generated by an intersection of a triangle with an edge has coordinates (x,y,z), what is its perturbed coordinate?
    - Ans: ???

    → store these coordinates implicitly

    → process implicit coordinates in the predicates



Experiments

- Algorithm implemented in C++.
  - OpenMP (parallel) + GMPXX (exact coordinates)
- Experiments on a workstation
  - Dual Intel Xeon E5-2687 processors, 8 cores, 2 threads/core
  - 128 GB of RAM.
  - Ubuntu Linux 16.04.
- Comparison with:
  - LibiGL: recent, exact, parallel and resolves self-intersections.
  - CGAL Nef Polyhedra: exact
  - QuickCSG: fast, parallel, but may fail (floating-point errors/do not handle special cases).

Experiments

- Up to 37x faster than LibiGL
- Up to 281x faster than CGAL (935x including conversion)

		Triangles (thousands)				Running times (s)				
								CGAL		
Mesh 0	Mesh 1	Mesh 0	Mesh 1	Out	Inter.tests	3D-EPUG	LibiGL	Convert	Intersect	QuickCSG
Casting10kf	Clutch2kf	10	2	6	8	0.1	1.4	4.5	1.1	**0.1***
Armadillo52kf	Dinausor40kf	52	40	25	42	0.2	2.9	38.5	21.2	0.1
Horse40kf	Cow76kf	40	76	24	50	0.2	3.1	50.6	24.1	0.1
Camel69kf	Armadillo52kf	69	52	16	54	0.2	3.3	51.0	26.0	0.1
Camel	Camel	69	69	81	1181	20.0	16.7	60.8	228.6	**1.0***
Camel	Armadillo	69	331	43	33	0.4	14.3	189.9	80.0	0.3
Armadillo	Armadillo	331	331	441	5351	94.2	75.8	339.7	1198.2	**3.9***
461112	461115	805	822	808	876	2.8	64.7	753.2	473.2	1.0
Kitten	RedCircBox	274	1402	246	27	1.2	36.3	819.8	329.6	1.0
Bimba	Vase	150	1792	724	122	1.9	65.4	971.7	455.7	1.0
226633	461112	2452	805	1437	307	3.2	120.0	1723.7	905.5	**2.0***
Ramesses	Ramess.Trans.	1653	1653	1571	866	4.6	102.7	1558.8	946.1	**2.2***
Ramesses	Ramess.Rot.	1653	1653	1691	2275	6.6	122.5	1577.3	989.8	2.2
Neptune	Ramesses	4008	1653	1112	814	5.5	150.0	3535.5	1535.6	3.5
Neptune	Nept.Transl	4008	4008	3303	2924	10.9	247.9	5390.7	2726.2	5.4
ArmadilloTetra	ArmadilloTetraTransl	1602	1602	61325	259436	420.3	-	-	-	-
518092_tetra	461112_tetra	5938	8495	23181	255703	333.0	-	-	-	-

Experiments

- Slightly slower than LibiGL when a mesh is intersected with itself: too many SoS calls (non-optimmized, future work)

						Running times (s)				
		Triangles (thousands)						CGAL		
Mesh 0	Mesh 1	Mesh 0	Mesh 1	Out	Inter.tests	3D-EPUG	LibiGL	Convert	Intersect	QuickCSG
Casting10kf	Clutch2kf	10	2	6	8	0.1	1.4	4.5	1.1	**0.1***
Armadillo52kf	Dinausor40kf	52	40	25	42	0.2	2.9	38.5	21.2	0.1
Horse40kf	Cow76kf	40	76	24	50	0.2	3.1	50.6	24.1	0.1
Camel69kf	Armadillo52kf	69	52	16	54	0.2	3.3	51.0	26.0	0.1
Camel	Camel	69	69	81	1181	20.0	16.7	60.8	228.6	**1.0***
Camel	Armadillo	69	331	43	33	0.4	14.3	189.9	80.0	0.3
Armadillo	Armadillo	331	331	441	5351	94.2	75.8	339.7	1198.2	**3.9***
461112	461115	805	822	808	876	2.8	64.7	753.2	473.2	1.0
Kitten	RedCircBox	274	1402	246	27	1.2	36.3	819.8	329.6	1.0
Bimba	Vase	150	1792	724	122	1.9	65.4	971.7	455.7	1.0
226633	461112	2452	805	1437	307	3.2	120.0	1723.7	905.5	**2.0***
Ramesses	Ramess.Trans.	1653	1653	1571	866	4.6	102.7	1558.8	946.1	**2.2***
Ramesses	Ramess.Rot.	1653	1653	1691	2275	6.6	122.5	1577.3	989.8	2.2
Neptune	Ramesses	4008	1653	1112	814	5.5	150.0	3535.5	1535.6	3.5
Neptune	Nept.Transl	4008	4008	3303	2924	10.9	247.9	5390.7	2726.2	5.4
ArmadilloTetra	ArmadilloTetraTransl	1602	1602	61325	259436	420.3	-	-	-	-
518092_tetra	461112_tetra	5938	8495	23181	255703	333.0	-	-	-	-

Experiments

- Up to 3x slower than QuickCSG (tests without reported failures), but exact.

						Running times (s)				
		Triangles (thousands)						CGAL		
Mesh 0	Mesh 1	Mesh 0	Mesh 1	Out	Inter.tests	3D-EPUG	LibiGL	Convert	Intersect	QuickCSG
Casting10kf	Clutch2kf	10	2	6	8	0.1	1.4	4.5	1.1	**0.1***
Armadillo52kf	Dinausor40kf	52	40	25	42	0.2	2.9	38.5	21.2	0.1
Horse40kf	Cow76kf	40	76	24	50	0.2	3.1	50.6	24.1	0.1
Camel69kf	Armadillo52kf	69	52	16	54	0.2	3.3	51.0	26.0	0.1
Camel	Camel	69	69	81	1181	20.0	16.7	60.8	228.6	**1.0***
Camel	Armadillo	69	331	43	33	0.4	14.3	189.9	80.0	0.3
Armadillo	Armadillo	331	331	441	5351	94.2	75.8	339.7	1198.2	**3.9***
461112	461115	805	822	808	876	2.8	64.7	753.2	473.2	1.0
Kitten	RedCircBox	274	1402	246	27	1.2	36.3	819.8	329.6	1.0
Bimba	Vase	150	1792	724	122	1.9	65.4	971.7	455.7	1.0
226633	461112	2452	805	1437	307	3.2	120.0	1723.7	905.5	**2.0***
Ramesses	Ramess.Trans.	1653	1653	1571	866	4.6	102.7	1558.8	946.1	**2.2***
Ramesses	Ramess.Rot.	1653	1653	1691	2275	6.6	122.5	1577.3	989.8	2.2
Neptune	Ramesses	4008	1653	1112	814	5.5	150.0	3535.5	1535.6	3.5
Neptune	Nept.Transl	4008	4008	3303	2924	10.9	247.9	5390.7	2726.2	5.4
ArmadilloTetra	ArmadilloTetraTransl	1602	1602	61325	259436	420.3	-	-	-	-
518092_tetra	461112_tetra	5938	8495	23181	255703	333.0	-	-	-	-

Experiments

- Up to 3x slower than QuickCSG (tests without reported failures), but <u>exact</u>.
  - * → QuickCSG failed and reported failure
  - If a failure is not reported → result may still have errors

		Triangles (thousands)				Running times (s)				
								CGAL		
Mesh 0	Mesh 1	Mesh 0	Mesh 1	Out	Inter.tests	3D-EPUG	LibiGL	Convert	Intersect	QuickCSG
Casting10kf	Clutch2kf	10	2	6	8	0.1	1.4	4.5	1.1	**0.1***
Armadillo52kf	Dinausor40kf	52	40	25	42	0.2	2.9	38.5	21.2	0.1
Horse40kf	Cow76kf	40	76	24	50	0.2	3.1	50.6	24.1	0.1
Camel69kf	Armadillo52kf	69	52	16	54	0.2	3.3	51.0	26.0	0.1
Camel	Camel	69	69	81	1181	20.0	16.7	60.8	228.6	**1.0***
Camel	Armadillo	69	331	43	33	0.4	14.3	189.9	80.0	0.3
Armadillo	Armadillo	331	331	441	5351	94.2	75.8	339.7	1198.2	**3.9***
461112	461115	805	822	808	876	2.8	64.7	753.2	473.2	1.0
Kitten	RedCircBox	274	1402	246	27	1.2	36.3	819.8	329.6	1.0
Bimba	Vase	150	1792	724	122	1.9	65.4	971.7	455.7	1.0
226633	461112	2452	805	1437	307	3.2	120.0	1723.7	905.5	**2.0***
Ramesses	Ramess.Trans.	1653	1653	1571	866	4.6	102.7	1558.8	946.1	**2.2***
Ramesses	Ramess.Rot.	1653	1653	1691	2275	6.6	122.5	1577.3	989.8	2.2
Neptune	Ramesses	4008	1653	1112	814	5.5	150.0	3535.5	1535.6	3.5
Neptune	Nept.Transl	4008	4008	3303	2924	10.9	247.9	5390.7	2726.2	5.4
ArmadilloTetra	ArmadilloTetraTransl	1602	1602	61325	259436	420.3	-	-	-	-
518092_tetra	461112_tetra	5938	8495	23181	255703	333.0	-	-	-	-

Experiments

- Can process meshes with millions of triangles in few seconds.
- Can handle tetra-meshes (461112_tetra: 8 M triangles, 4 M tetrahedra).

		Triangles (thousands)				Running times (s)				
								CGAL		
Mesh 0	Mesh 1	Mesh 0	Mesh 1	Out	Inter.tests	3D-EPUG	LibiGL	Convert	Intersect	QuickCSG
Casting10kf	Clutch2kf	10	2	6	8	0.1	1.4	4.5	1.1	**0.1***
Armadillo52kf	Dinausor40kf	52	40	25	42	0.2	2.9	38.5	21.2	0.1
Horse40kf	Cow76kf	40	76	24	50	0.2	3.1	50.6	24.1	0.1
Camel69kf	Armadillo52kf	69	52	16	54	0.2	3.3	51.0	26.0	0.1
Camel	Camel	69	69	81	1181	20.0	16.7	60.8	228.6	**1.0***
Camel	Armadillo	69	331	43	33	0.4	14.3	189.9	80.0	0.3
Armadillo	Armadillo	331	331	441	5351	94.2	75.8	339.7	1198.2	**3.9***
461112	461115	805	822	808	876	2.8	64.7	753.2	473.2	1.0
Kitten	RedCircBox	274	1402	246	27	1.2	36.3	819.8	329.6	1.0
Bimba	Vase	150	1792	724	122	1.9	65.4	971.7	455.7	1.0
226633	461112	2452	805	1437	307	3.2	120.0	1723.7	905.5	**2.0***
Ramesses	Ramess.Trans.	1653	1653	1571	866	4.6	102.7	1558.8	946.1	**2.2***
Ramesses	Ramess.Rot.	1653	1653	1691	2275	6.6	122.5	1577.3	989.8	2.2
Neptune	Ramesses	4008	1653	1112	814	5.5	150.0	3535.5	1535.6	3.5
Neptune	Nept.Transl	4008	4008	3303	2924	10.9	247.9	5390.7	2726.2	5.4
ArmadilloTetra	ArmadilloTetraTransl	1602	1602	61325	259436	420.3	-	-	-	-
518092_tetra	461112_tetra	5938	8495	23181	255703	333.0	-	-	-	-

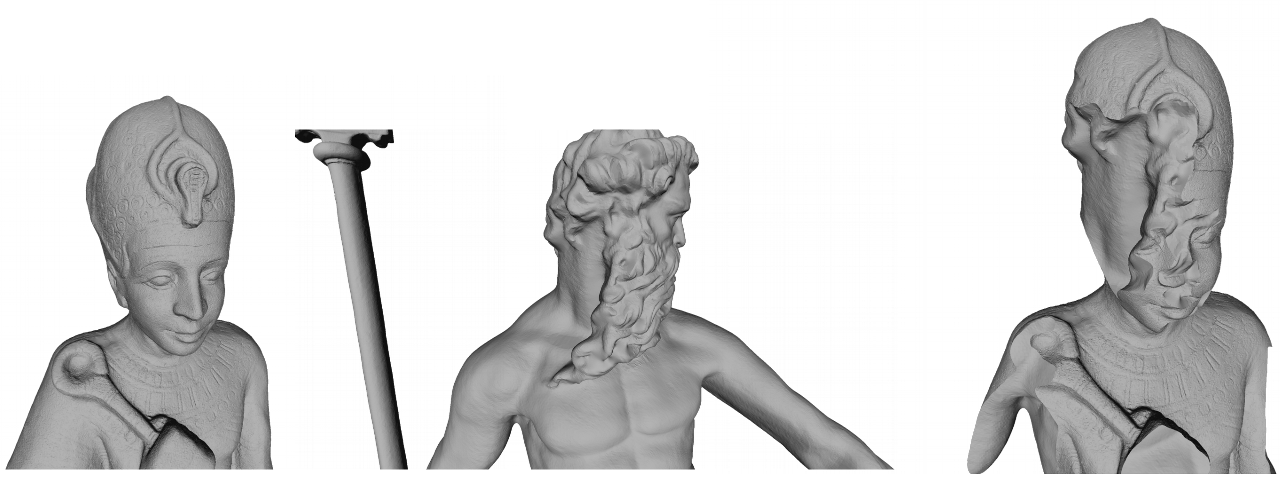
Experiments

- Memory efficient:
  - Neptune vs Neptune translated: 3D-EPUG: 5GB of RAM, LibiGL: 22.5GB, CGAL: 110GB, QuickCSG: 4.5GB

		Triangles (thousands)				Running times (s)				
								CGAL		
Mesh 0	Mesh 1	Mesh 0	Mesh 1	Out	Inter.tests	3D-EPUG	LibiGL	Convert	Intersect	QuickCSG
Casting10kf	Clutch2kf	10	2	6	8	0.1	1.4	4.5	1.1	**0.1***
Armadillo52kf	Dinausor40kf	52	40	25	42	0.2	2.9	38.5	21.2	0.1
Horse40kf	Cow76kf	40	76	24	50	0.2	3.1	50.6	24.1	0.1
Camel69kf	Armadillo52kf	69	52	16	54	0.2	3.3	51.0	26.0	0.1
Camel	Camel	69	69	81	1181	20.0	16.7	60.8	228.6	**1.0***
Camel	Armadillo	69	331	43	33	0.4	14.3	189.9	80.0	0.3
Armadillo	Armadillo	331	331	441	5351	94.2	75.8	339.7	1198.2	**3.9***
461112	461115	805	822	808	876	2.8	64.7	753.2	473.2	1.0
Kitten	RedCircBox	274	1402	246	27	1.2	36.3	819.8	329.6	1.0
Bimba	Vase	150	1792	724	122	1.9	65.4	971.7	455.7	1.0
226633	461112	2452	805	1437	307	3.2	120.0	1723.7	905.5	**2.0***
Ramesses	Ramess.Trans.	1653	1653	1571	866	4.6	102.7	1558.8	946.1	**2.2***
Ramesses	Ramess.Rot.	1653	1653	1691	2275	6.6	122.5	1577.3	989.8	2.2
Neptune	Ramesses	4008	1653	1112	814	5.5	150.0	3535.5	1535.6	3.5
Neptune	Nept.Transl	4008	4008	3303	2924	10.9	247.9	5390.7	2726.2	5.4
ArmadilloTetra	ArmadilloTetraTransl	1602	1602	61325	259436	420.3	-	-	-	-
518092_tetra	461112_tetra	5938	8495	23181	255703	333.0	-	-	-	-

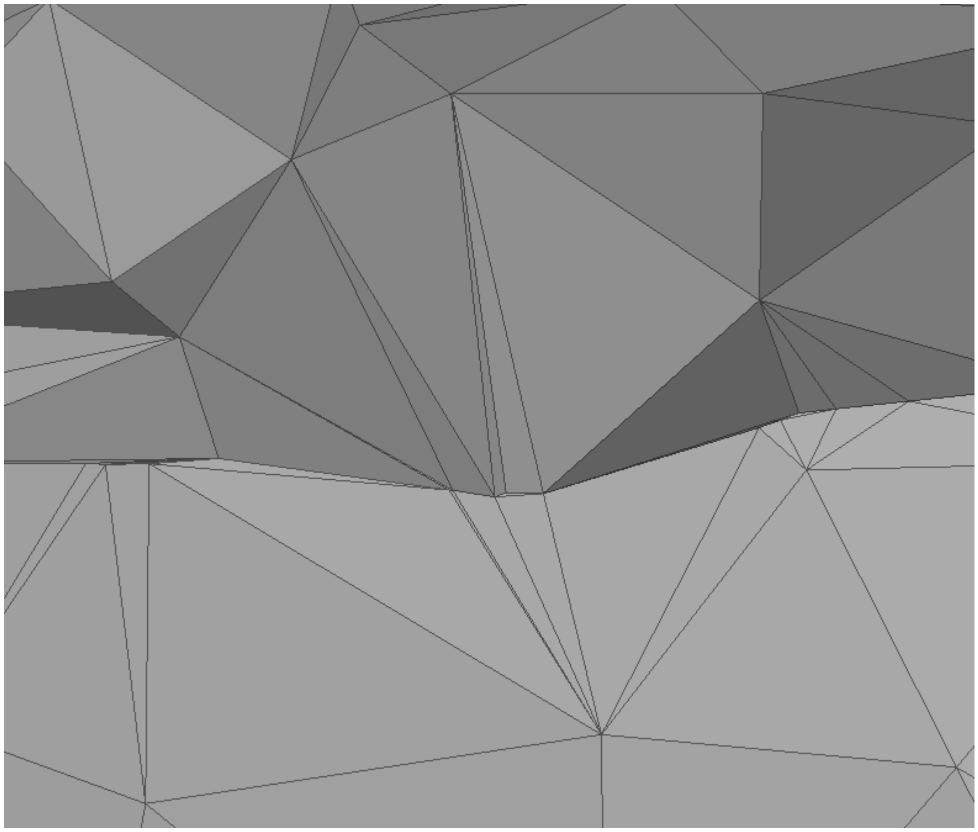
Example of result

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



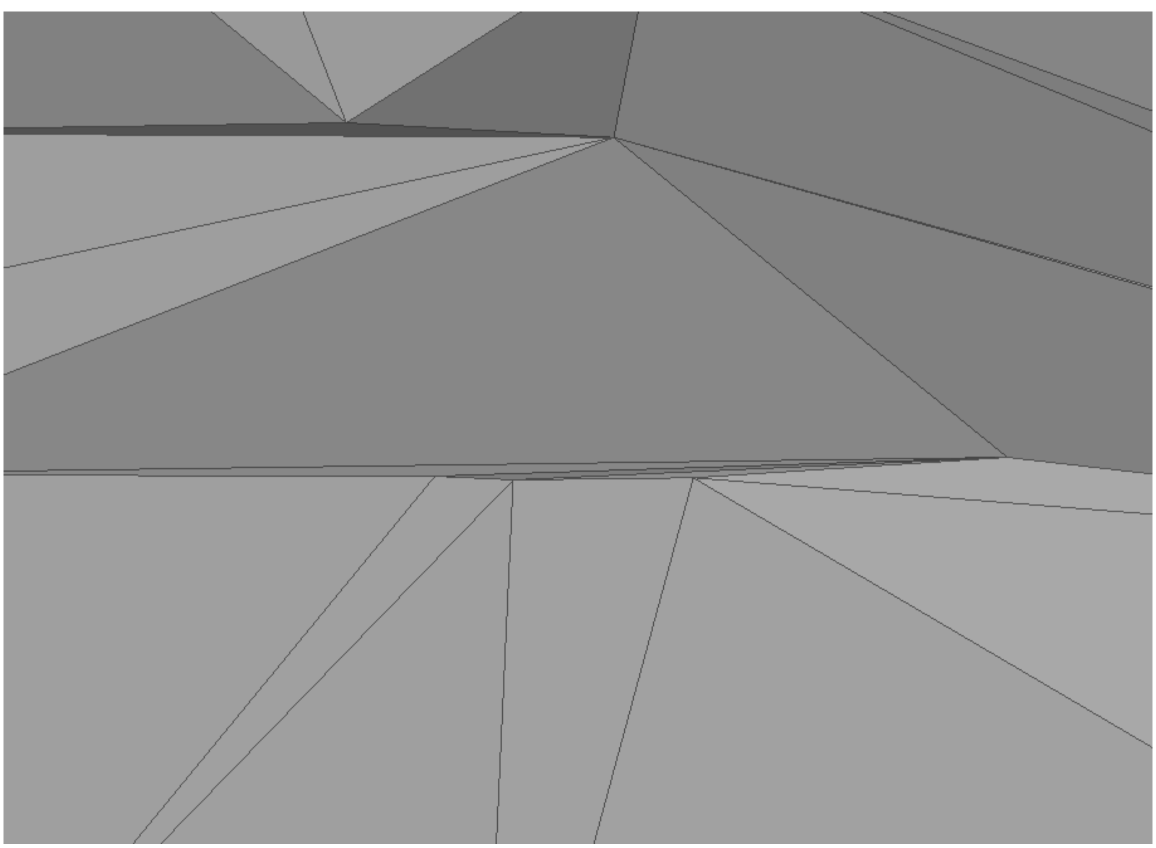
Example of result

- Hard to process triangles → roundoff errors



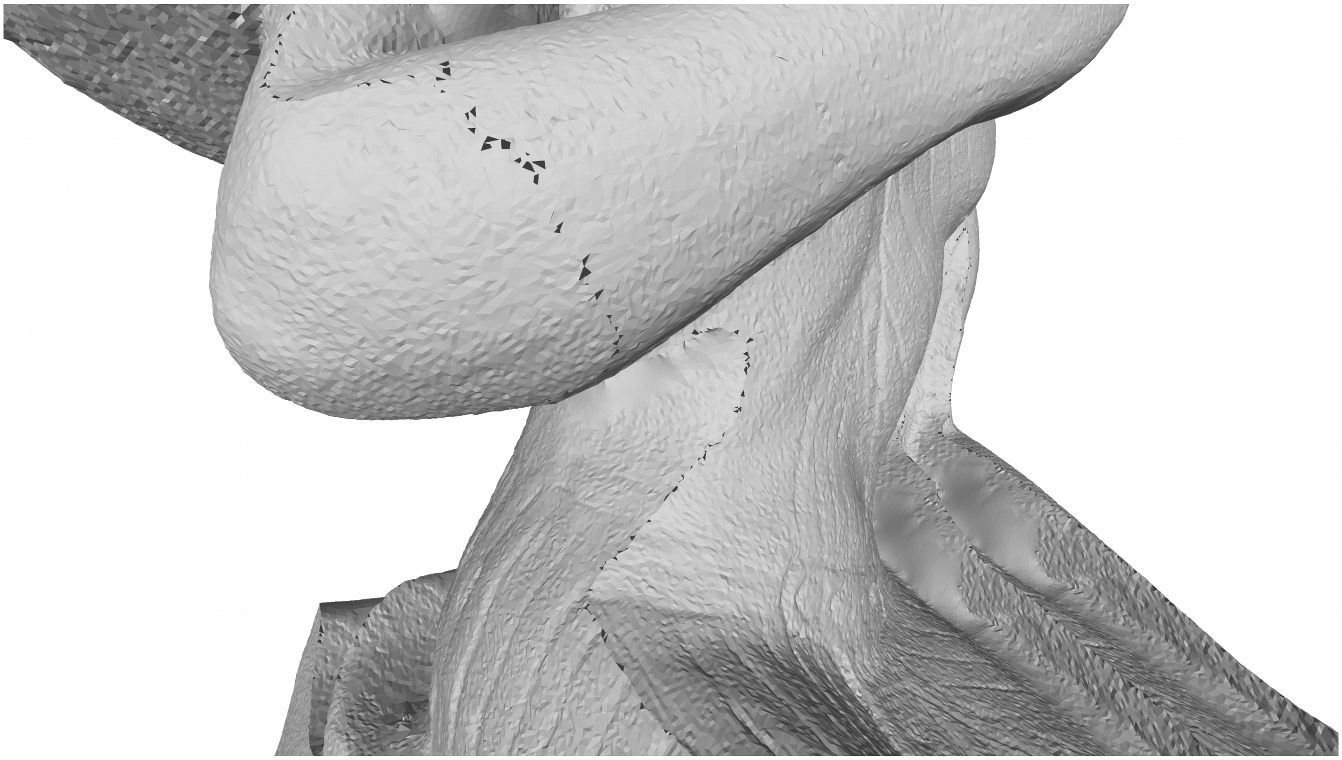
Example of result

- Hard to process triangles → roundoff errors



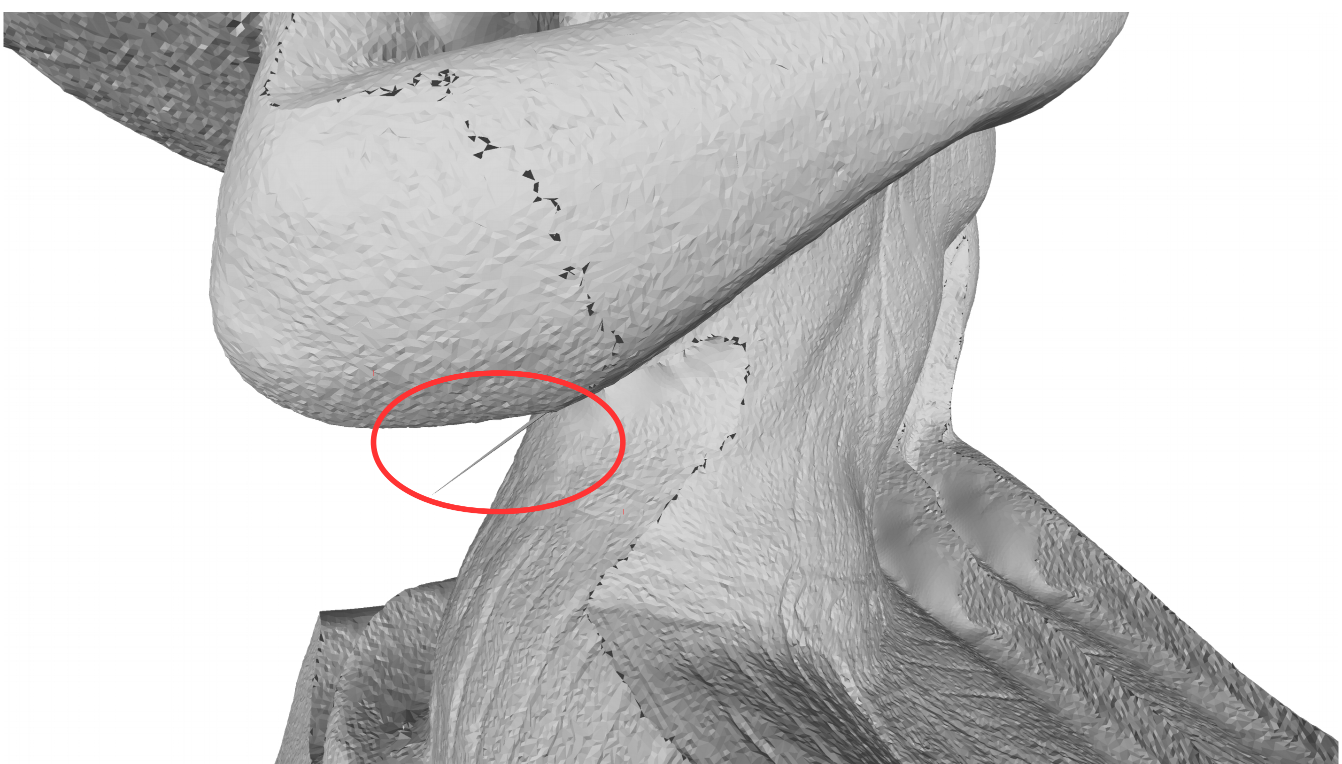
Example of result

- QuickCSG: Ramesses vs Ramesses translated.
  - No error reported
  - Several failures



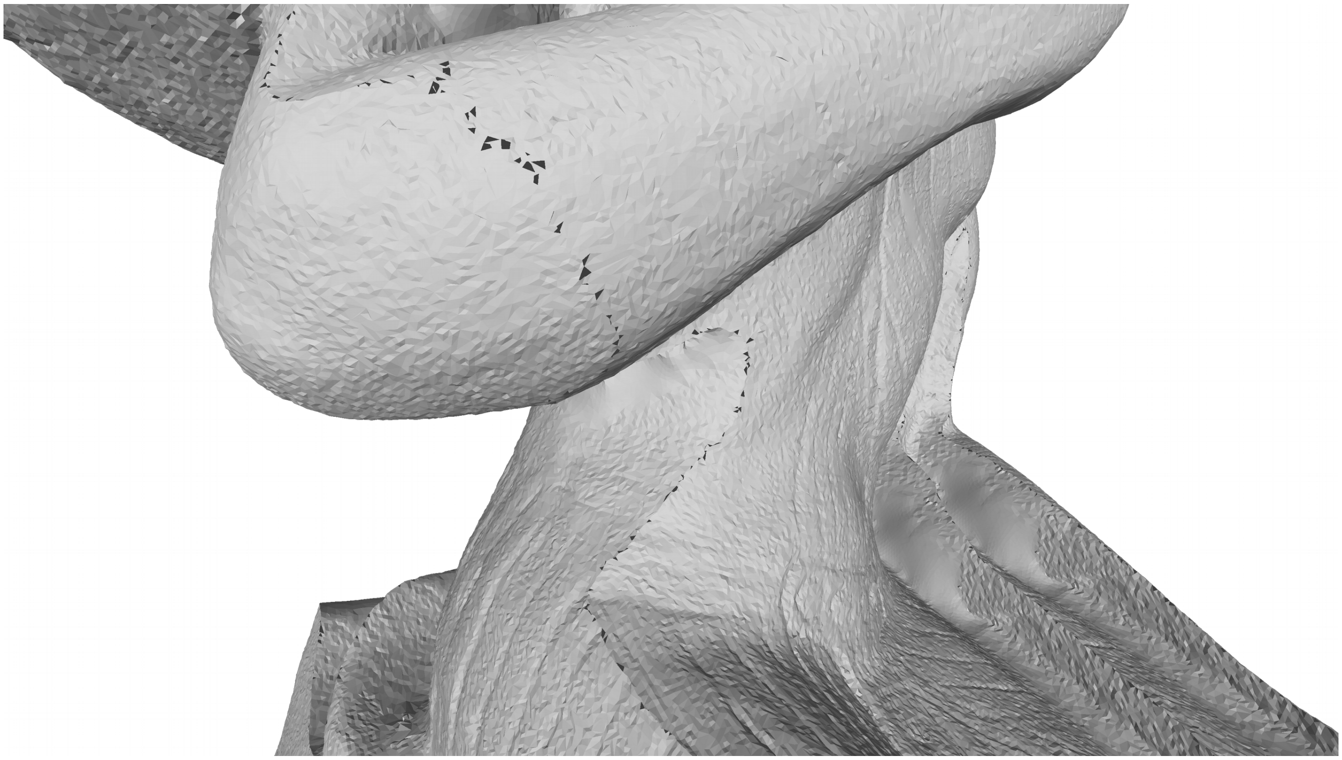
Example of result

- QuickCSG: Ramesses vs Ramesses translated.
  - To mitigate: numerical perturbation
  - Does not work always (figure: max perturbation = 10^{-1})



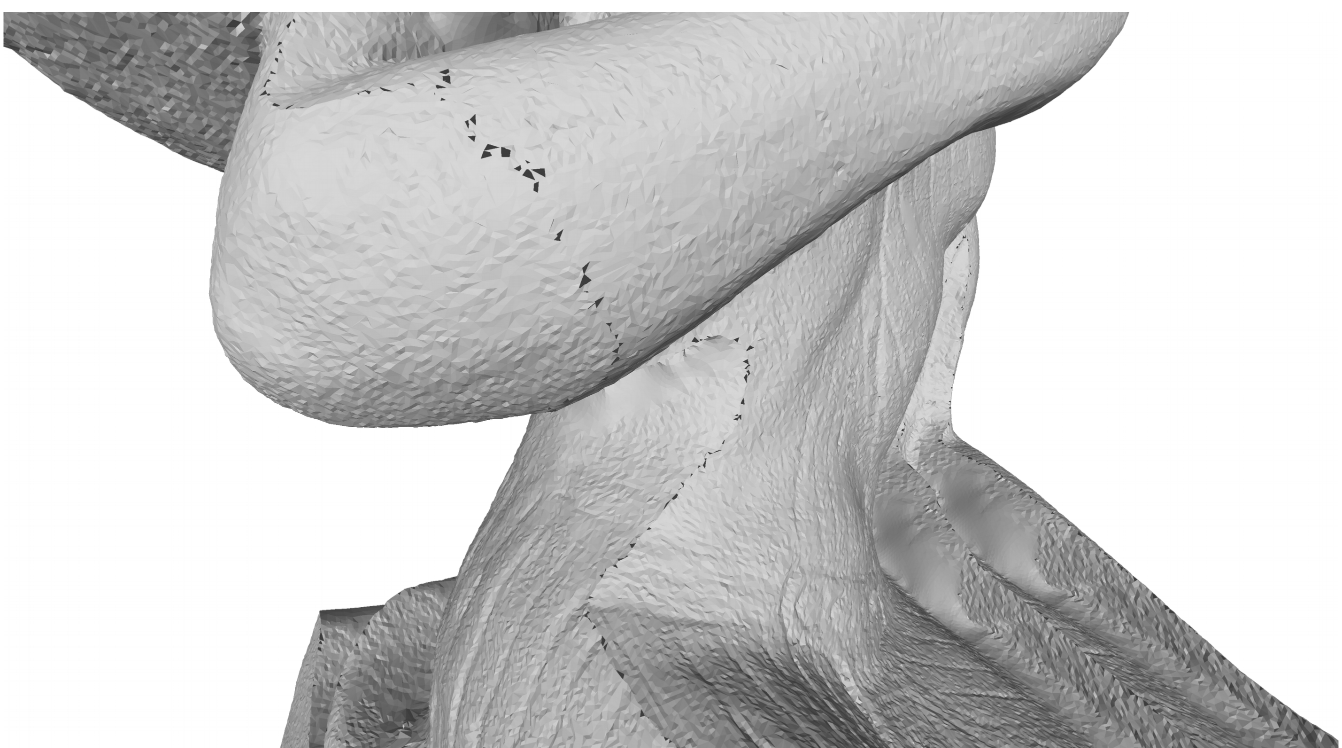
Example of result

- QuickCSG: Ramesses vs Ramesses translated.
  - To mitigate: numerical perturbation
  - Does not work always (figure: max perturbation = 10^{-3})



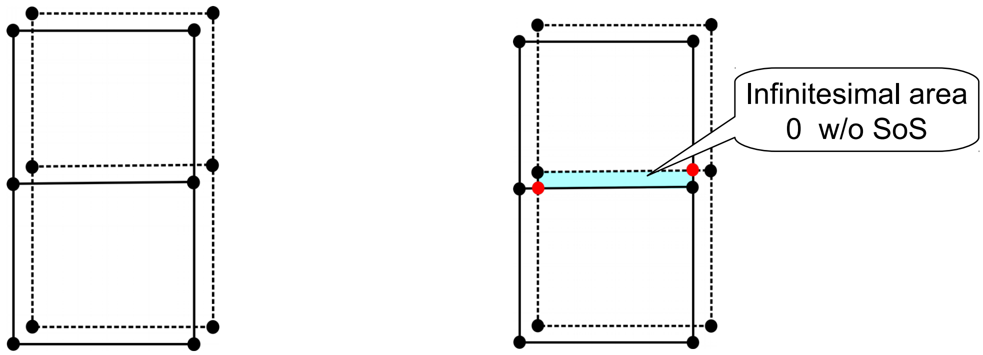
Example of result

- QuickCSG: Ramesses vs Ramesses translated.
  - To mitigate: numerical perturbation
  - Does not work always (figure: max perturbation = 10^{-12})



The perturbed result

- Result with SoS.
  - Result is valid considering the perturbed data.
  - If perturbation is removed → possible topological errors, triangles with area 0, polyhedra with volume 0, etc.
  - Solution:
    - Do not remove the perturbation (i.e., other algorithms should know how the dataset was perturbed).
    - Use regularization and other techniques to clean the results.

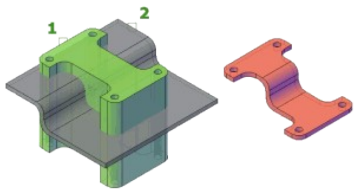


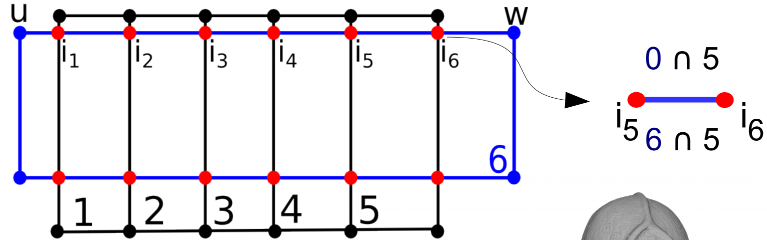
Self intersection, 2 regions/mesh

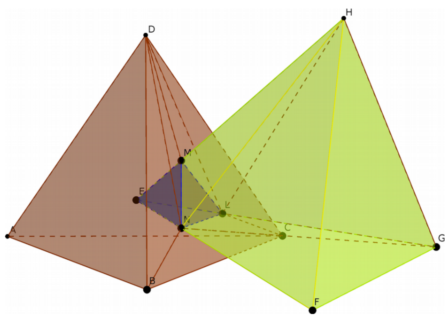
Conclusions

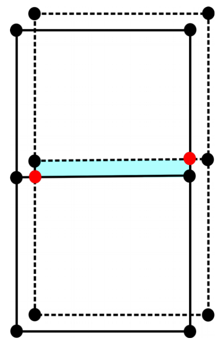
- 3D-EPUG-OVERLAY
  - Exact
  - Parallel
  - Uniform grid
- Part of a bigger project
  - Exact and parallel geometric algorithms
  - Applications in GIS, CAD and AM
- Fast <u>and</u> exact
- Future work:
  - Improve performance (mainly of SoS calls)
  - Use similar ideas for other problems

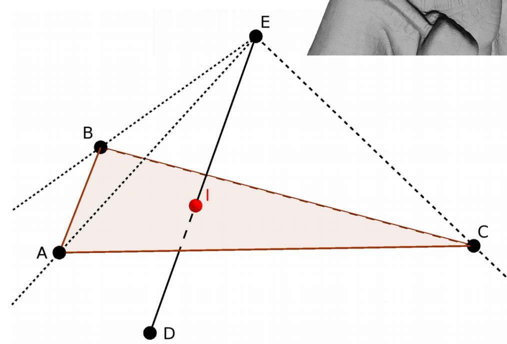
Thank you!











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' ? → barycentric coordinates → $0 < \lambda_i < 1$ for i=1,2 and 3 ?

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

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

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

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

Simulation of Simplicity

- Degeneracies: det = 0 → vertical triangle
- Point on boundary of t' ($\lambda_i = 0$ or 1).

- SoS → $q(x,y,z)$ → $q_\varepsilon(x+\varepsilon, y+\varepsilon^2, z+\varepsilon^3)$, $q'(x,y)$ → $q'_\varepsilon(x+\varepsilon, y+\varepsilon^2)$
  - q'_ε will never be on a vertex or edge of t'.
    - q' is not on vertex/edge → q'_ε is also not on vertex/edge (infinitesimal).
    - q' is on vertex/edge → q'_ε is not on vertex/edge (infinitesimal/slope).
      - Ex: q' is on an edge → q'_ε cannot be on the same edge (slope would be infinitesimal)

Simulation of Simplicity

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

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

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

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

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

Simulation of Simplicity

- SoS implementation:
  - q'_ε will never be on a vertex or edge of t' → if det=0 → false
  - Replace q' with q'_ε → λ_i with $\lambda_{\varepsilon i}$
  - E.g.: is $0 < \lambda_{\varepsilon 0}$?
    - $\lambda_0 \neq 0$ → check λ_0
    - $\lambda_0 = 0$ → check t'_{1y} - t'_{2y}
    - t'_{1y} - $t'_{2y} = 0$ → check t'_{2x} - t'_{1x}
    - Both can't be 0.

$$\lambda_{\epsilon_0} = \lambda_0 + \frac{(t'_{1y} - t'_{2y}) \times \epsilon + (t'_{2x} - t'_{1x}) \times \epsilon^2}{det}$$

$$\lambda_{\epsilon_1} = \lambda_1 + \frac{(t'_{2y} - t'_{0y}) \times \epsilon + (t'_{0x} - t'_{2x}) \times \epsilon^2}{det}$$

$$\lambda_{\epsilon_2} = 1 - \lambda_{\epsilon_0} - \lambda_{\epsilon_1}$$