EXACT AND PARALLEL INTERSECTION OF 3D TRIANGULAR MESHES

Salles Viana Gomes de Magalhães, PhD. Student
Prof. W Randolph Franklin, Supervisor

Rensselaer Polytechnic Institute, Troy NY USA
Federal University of Viçosa, MG, Brasil

ACM SIGSPATIAL, Redondo Beach, 2017-11-10
Map overlay

- Important in GIS/CAD/CAM
- Two vector maps are superimposed
- The intersection between polygons from the two maps is computed
- Several applications. Ex: counties and watersheds

- This problem extends to 3D objects (triangulations)
- Example: intersection of CAD models, soil layers, etc
Challenge

- Finite precision of floating point $\Rightarrow$ roundoff errors
  
  $1000000000.0 + 1.0 - 10000000000.0 = 0.0$ (wrong)

- Common techniques (snap rounding, epsilon tweaking, etc): no guarantee

- More data & 3D $\Rightarrow$ bigger problem

- Exactness and performance: very important – this function may be a small piece of a larger program

Source: Kettner et al., Classroom examples of robustness problems in geometric computations
Our fast algorithms for large datasets

- ParCube – GPU parallel detection of cube-cube intersections
- 3D-EPUG-OVERLAY – 3D parallel map overlay
- NearptD – parallel nearest neighbor algorithm
- TiledVS – external memory viewshed computation.
- PinMesh – 3D point location
- UPLAN – path planning on road networks with polygonal constraints.
- Emflow – hydrography on massive external terrain
- EPU G-OVERLAY – 2D map overlay
- Grid-Gen – map simplification preserving topological relationships
- Parallel Multiple Observer Siting on Terrain
- RWFLOOD – hydrography on massive internal terrain
- UNION3 – volume of union of many cubes
- Connect – connected components of $1000^3$ 3D box of binary voxels
- TIN – incrementally triangulate $10000^2$ terrain (update of (Franklin, 1973)).
We often combine 5 techniques

- Arbitrary precision rational numbers: no roundoff errors
- Simulation of Simplicity: handle special cases properly
- Minimize explicit topology: compact, parallelizable.
- Parallel programming: exploit current hardware
- Uniform grid: filter for probable intersections in parallel
EPUG-OVERLAY – 2D map overlay

- Exact
- Parallel
- Uniform Grid

- Developed to evaluate our ideas
- Efficient: 20x speedup if compared against GRASS GIS
PinMesh – 3D point location

- Preprocess 3D mesh to perform point queries
- Exact and efficient (up 27 times faster than RCT, an inexact competing method) point location
- Subproblem of the mesh overlay
3D-EPUG-OVERLAY

Current work

• 3D mesh intersection
• Techniques + experience from PinMesh and EPUG-OVERLAY → 3D-EPUG-OVERLAY

source: Autodesk
Related work

• Approximate algorithms:
  • Example: voxelization

• Nef Polyhedra/CGAL:
  • Exact, sequential, slow
  • For Nef Polyhedra
    • Polyhedron: sequence of complement and intersection of half-spaces
    • Challenge: convert data
Related work - QuickCSG

• QuickCSG:
  • Recent
  • Designed to be very fast: no special cases, floating-point, parallel
  • User can try to avoid special cases: numeric perturbation
  • Error-prone
Related work - LibiGL

• Zhou's algorithm (LibiGL):
  • Very recent
  • Parallel and relatively fast
  • Uses CGAL (example: bounding-box for triangle-triangle intersection)
  • Key idea: use of winding number in mesh representation
  • Merge meshes + resolve self-intersections

Winding numbers (source: Zhou et al. [77])
Our data representation

- Intersection: pair of meshes
- Each mesh: set of polyhedra (usually one polyhedron) that partition space.

Mesh representation
- Set of triangles, plus
- Information about positive and negative sides
- No explicit global info.

ABC:
- Positive: blue
- Negative: red

ABD:
- Positive: red
- Negative: outside

source: Autodesk
Data representation

- Mesh restriction: should be “valid”
  - watertight
  - consistent
Indexing the data

- We employ a 2-level 3D uniform grid.
- Employed for detecting intersections and point location.
- Coding shortcut: Insert a 3D triangle into the cells that *its bounding box* intersects. That is many more cells than necessary (asymptotically superlinear).
- That shortcut motivates the 2 levels.

Example: detecting black-blue intersections (2D)
Algorithm summary

- Detect intersections between the two meshes
- Retesselate intersecting triangles
- Classify the triangles, both non-intersecting and retesselated.
**Rational numbers**

- Motivation: no roundoff errors.
- Each number is stored as a ratio of two integers
- E.g., \( \frac{1}{3} + \frac{2}{5} = \frac{11}{15} \)
- C++ operators are overloaded to do this
- Each operation doubles the number of digits
- Numerator and denominator are arrays of groups of digits
- Doubling is acceptable if depth of computation tree is small
- Packages like gmp++ mostly work
- Big problem: frequent allocations on global heap
- That’s slow for many objects and for multithreading.
- Solution: code to minimize allocations and use a better allocator.
- Execution time penalty: small integer factor
- Combine with interval arithmetic ([lo,hi]) for speed
- \([.30,.35] + [.48,.52] = [.78..87]\)
Simulation of Simplicity (SoS)

- Reduces the number of special cases.
- Point vs line? *Above, on, or below.*
- Combine *on* case into *above?*
- Solution must handle higher level functions correctly
- e.g., Pnpoly (Franklin, 1970) : test point inclusion in polygon by running ray up from point and counting intersections with edges.
- How many intersections when vertex is on ray?
- Much worse: ray vs polyhedron
- Sos: move ray slightly to right.
- Then no ray—vertex intersections.
Special cases

- $p(x,y,z) \rightarrow p_{\varepsilon}(x+i\varepsilon, y+i\varepsilon^2, z+i\varepsilon^3) \rightarrow$ coincidences eliminated
- $i=0$ or $1$ (which input dataset is this?)
- A vertex of one mesh is never on the plane of a triangle of the other mesh ($\rightarrow$ intersection of triangles is never a point)
- Edges from different meshes do not intersect $\rightarrow$ edges will only intersect interior of triangles
- Triangles from different meshes are never coplanar
- Etc

- Example of consequence: intersection of two 3D triangles is always an edge
Implementing SoS

- Don’t actually implement infinitesimal math.
- Instead: rewrite geometric predicates to have that effect.
  
  \[(a + \varepsilon^i < b + \varepsilon^j) \rightarrow ((a < b \mid (a == b) \& (i > j))\]

- Leads to incrutable source code.
- Computation can be initially done with the rational coordinates. If coincidence is detected → consider the infinitesimals → good performance

- Challenge: too many predicates!
- Solution → use a small set of predicates
Orientation predicates

• The algorithm was completely implemented using orientation predicates (except for the indexing) → SoS only in the orientation predicate.

• Example: detect intersection of two triangles
  • → detect intersections between edges and a triangle
  • → 5 orientations for each edge-triangle test (Segura and Feito, 2001)
Experiments

- Algorithm designed to be parallel:
  - Little data dependency, simple representation
- Implemented using OpenMP
- Compiled with g++ -O3, using Tcmalloc
- All times in seconds

Machine:
- 16-Core workstation (Dual Xeon E5-2687)
- 128 GB of RAM
- Ubuntu Linux
Datasets

- Datasets from 4 sources
- Meshes with up to 4 million triangles
- Tetra meshes with up to 8 million triangles/4 million tetrahedra
EXACT AND PARALLEL INTERSECTION OF 3D TRIANGULAR MESHES

R - Renaissance Polytechnic Institute

\[ \cap = \]
EXACT AND PARALLEL INTERSECTION OF 3D TRIANGULAR MESHES

\[ n \cap \text{intersect} = \]
EXACT AND PARALLEL INTERSECTION OF 3D TRIANGULAR MESHES

\[ \bigcap \]
Experiments

• First set of experiments: two key techniques for performance:
  • Arithmetic filtering: accelerate rationals
  • Uniform grid: easily parallelizable
• This also shows that the uniform grid can efficiently process data that much worse than uniform random, which would have coincidences only with probability 0.
• We also experimented with various grid concrete realizations.
  • Very bad: linked list or STL vector for each cell.
  • Ragged array is much better.
    • String together in one array all the cells’ contents.
    • A dope vector points to start of each cell’s contents.
Arithmetic filtering

• Makes using rationals faster.

• Arithmetic filtering → rationals: not always necessary
  • Basic idea: associate floating-point approximations to each number
  • Evaluate predicates (determinants) with the approximation
  • If signal can be trusted → use it
  • Otherwise, recompute exactly
# EXACT AND PARALLEL INTERSECTION OF 3D TRIANGULAR MESHES

# Uniform grid much faster than CGAL

<table>
<thead>
<tr>
<th>Mesh 0</th>
<th>Mesh 1</th>
<th>Mesh 0</th>
<th>Mesh 1</th>
<th># faces ($\times 10^3$)</th>
<th># int.$^a$</th>
<th>Int.tests$^b$ ($\times 10^3$)</th>
<th>Time (s)</th>
<th>Pre.proc.$^c$</th>
<th>Inter.$^d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camel</td>
<td>Armadillo</td>
<td>69</td>
<td>331</td>
<td>3</td>
<td>14</td>
<td></td>
<td>0.32</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>Armadillo</td>
<td>Armadillo</td>
<td>331</td>
<td>331</td>
<td>4,611</td>
<td>5,043</td>
<td></td>
<td>1.27</td>
<td>259.23</td>
<td></td>
</tr>
<tr>
<td>Kitten</td>
<td>RedC.Box$^e$</td>
<td>274</td>
<td>1,402</td>
<td>3</td>
<td>13</td>
<td></td>
<td>2.33</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>226633</td>
<td>461112</td>
<td>2,452</td>
<td>805</td>
<td>23</td>
<td>128</td>
<td></td>
<td>7.18</td>
<td>0.08</td>
<td></td>
</tr>
<tr>
<td>Ramesses</td>
<td>Ram.Tran.$^f$</td>
<td>1,653</td>
<td>1,653</td>
<td>36</td>
<td>237</td>
<td></td>
<td>12.38</td>
<td>0.17</td>
<td></td>
</tr>
<tr>
<td>Neptune</td>
<td>Nept.Tran.$^g$</td>
<td>4,008</td>
<td>4,008</td>
<td>78</td>
<td>647</td>
<td></td>
<td>36.24</td>
<td>0.47</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mesh 0</th>
<th>Mesh 1</th>
<th>Mesh 0</th>
<th>Mesh 1</th>
<th># faces ($\times 10^3$)</th>
<th># int.$^a$</th>
<th>Int.tests$^b$ ($\times 10^3$)</th>
<th>Time (s)</th>
<th>Pre.proc.$^c$</th>
<th>Inter.$^d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camel</td>
<td>Armadillo</td>
<td>69</td>
<td>331</td>
<td>3</td>
<td>33</td>
<td></td>
<td>0.06</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>Armadillo</td>
<td>Armadillo</td>
<td>331</td>
<td>331</td>
<td>50</td>
<td>5,351</td>
<td></td>
<td>0.25</td>
<td>63.80</td>
<td></td>
</tr>
<tr>
<td>Kitten</td>
<td>RedC.Box$^e$</td>
<td>274</td>
<td>1,402</td>
<td>3</td>
<td>27</td>
<td></td>
<td>0.08</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>226633</td>
<td>461112</td>
<td>2,452</td>
<td>805</td>
<td>23</td>
<td>307</td>
<td></td>
<td>0.16</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>Ramesses</td>
<td>Ram.Tran.$^f$</td>
<td>1,653</td>
<td>1,653</td>
<td>36</td>
<td>866</td>
<td></td>
<td>0.16</td>
<td>0.10</td>
<td></td>
</tr>
<tr>
<td>Neptune</td>
<td>Nept.Tran.$^g$</td>
<td>4,008</td>
<td>4,008</td>
<td>78</td>
<td>5,087</td>
<td></td>
<td>0.27</td>
<td>0.35</td>
<td></td>
</tr>
</tbody>
</table>

$^a$ Number of intersection tests
$^b$ Number of intersection events
$^c$ Preprocessing time
$^d$ Intersection time
Choosing the grid resolution

- Parameter: grid resolution
- Number of expected pairs of triangles: \( np \)

\[
np = \frac{n_0 \times n_1}{G_1^3 \times G_2^3}
\]

\[
G_1 \times G_2 = \sqrt[6]{\frac{n_0 \times n_1}{np}}
\]

- Experiments: \( np \) to a small constant:
  - 0.00001 (regular meshes) or 0.1 (internal structure)
  - Good performance (broad optimum)
## Choosing the grid resolution

<table>
<thead>
<tr>
<th>Grid Size</th>
<th>Grid Resolution</th>
<th>Number of Triangles</th>
<th>Intersection</th>
<th>Memory (GB)</th>
<th>Grid Time (s)</th>
<th>Intersection Time (s)</th>
<th>Class Time (s)</th>
<th>Total Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16,8</td>
<td>94,302</td>
<td>90,616</td>
<td>60</td>
<td>4.59</td>
<td>0.11</td>
<td>6.20</td>
<td>0.72</td>
<td>7.72</td>
</tr>
<tr>
<td>16,16</td>
<td>22,585</td>
<td>19,852</td>
<td>60</td>
<td>2.31</td>
<td>0.11</td>
<td>1.52</td>
<td>0.59</td>
<td>2.88</td>
</tr>
<tr>
<td>32,8</td>
<td>22,585</td>
<td>19,852</td>
<td>60</td>
<td>2.27</td>
<td>0.12</td>
<td>1.41</td>
<td>0.58</td>
<td>2.79</td>
</tr>
<tr>
<td>16,32</td>
<td>8,287</td>
<td>5,748</td>
<td>60</td>
<td>1.92</td>
<td>0.13</td>
<td>0.76</td>
<td>0.53</td>
<td>2.13</td>
</tr>
<tr>
<td>32,16</td>
<td>8,287</td>
<td>5,748</td>
<td>60</td>
<td>1.84</td>
<td>0.13</td>
<td>0.74</td>
<td>0.57</td>
<td>2.10</td>
</tr>
<tr>
<td>64,8</td>
<td>5,486</td>
<td>2,275</td>
<td>60</td>
<td>3.08</td>
<td>0.27</td>
<td>0.67</td>
<td>0.57</td>
<td>2.21</td>
</tr>
<tr>
<td>32,32</td>
<td>5,486</td>
<td>2,275</td>
<td>60</td>
<td>2.44</td>
<td>0.19</td>
<td>0.59</td>
<td>0.57</td>
<td>1.98</td>
</tr>
<tr>
<td>64,16</td>
<td><strong>5,486</strong></td>
<td><strong>2,275</strong></td>
<td><strong>60</strong></td>
<td><strong>2.08</strong></td>
<td><strong>0.18</strong></td>
<td><strong>0.61</strong></td>
<td><strong>0.60</strong></td>
<td><strong>2.11</strong></td>
</tr>
<tr>
<td>32,64</td>
<td>7,365</td>
<td>1,240</td>
<td>60</td>
<td>7.00</td>
<td>0.74</td>
<td>0.90</td>
<td>0.55</td>
<td>2.90</td>
</tr>
<tr>
<td>64,32</td>
<td>7,365</td>
<td>1,240</td>
<td>60</td>
<td>4.08</td>
<td>0.42</td>
<td>0.70</td>
<td>0.60</td>
<td>2.42</td>
</tr>
<tr>
<td>64,64</td>
<td>18,899</td>
<td>865</td>
<td>60</td>
<td>19.26</td>
<td>2.31</td>
<td>1.80</td>
<td>0.52</td>
<td>5.29</td>
</tr>
</tbody>
</table>

---

Mash 0: Ramesses (2M triangles), Mesh 1: Ramesses.rot.h(2M triangles)
## Comparing against other methods

<table>
<thead>
<tr>
<th>Mesh 0</th>
<th>Mesh 1</th>
<th>3D-Epug</th>
<th>LibiGL</th>
<th>CGAL Convert&lt;sup&gt;a&lt;/sup&gt;</th>
<th>CGAL Intersect&lt;sup&gt;b&lt;/sup&gt;</th>
<th>QuickCSG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Casting10kf</td>
<td>Clutch2kf</td>
<td>0.2</td>
<td>1.3</td>
<td>4.2</td>
<td>1.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Armadillo52kf</td>
<td>Dinosaurus40kf</td>
<td>0.1</td>
<td>3.0</td>
<td>38.0</td>
<td>21.5</td>
<td>0.1</td>
</tr>
<tr>
<td>Horse40kf</td>
<td>Cow76kf</td>
<td>0.1</td>
<td>3.2</td>
<td>51.1</td>
<td>24.2</td>
<td>0.1</td>
</tr>
<tr>
<td>Camel69kf</td>
<td>Armadillo52kf</td>
<td>0.1</td>
<td>3.2</td>
<td>54.3</td>
<td>25.7</td>
<td>0.1</td>
</tr>
<tr>
<td>Camel</td>
<td>Camel</td>
<td>13.9</td>
<td>18.0</td>
<td>62.7</td>
<td>230.6</td>
<td>0.9</td>
</tr>
<tr>
<td>Camel</td>
<td>Armadillo</td>
<td>0.2</td>
<td>11.7</td>
<td>189.9</td>
<td>80.0</td>
<td>0.3</td>
</tr>
<tr>
<td>Armadillo</td>
<td>Armadillo</td>
<td>67.0</td>
<td>88.1</td>
<td>339.7</td>
<td>1,198.2</td>
<td>4.1</td>
</tr>
<tr>
<td>461112</td>
<td>461115</td>
<td>0.8</td>
<td>58.9</td>
<td>753.2</td>
<td>473.2</td>
<td>1.1</td>
</tr>
<tr>
<td>Kitten</td>
<td>RedCircBox</td>
<td>0.3</td>
<td>28.6</td>
<td>819.8</td>
<td>329.6</td>
<td>1.1</td>
</tr>
<tr>
<td>Bimba</td>
<td>Vase</td>
<td>0.6</td>
<td>58.0</td>
<td>971.7</td>
<td>455.7</td>
<td>1.1</td>
</tr>
<tr>
<td>226633</td>
<td>461112</td>
<td>0.9</td>
<td>96.0</td>
<td>1,723.7</td>
<td>905.5</td>
<td>2.2</td>
</tr>
<tr>
<td>Ramesses</td>
<td>Ram. Transl.&lt;sup&gt;c&lt;/sup&gt;</td>
<td>1.3</td>
<td>93.0</td>
<td>1,558.8</td>
<td>946.1</td>
<td>2.4</td>
</tr>
<tr>
<td>Ramesses</td>
<td>Ram. Rot.&lt;sup&gt;d&lt;/sup&gt;</td>
<td>2.1</td>
<td>122.0</td>
<td>1,577.3</td>
<td>989.8</td>
<td>2.4</td>
</tr>
<tr>
<td>Neptune</td>
<td>Ramesses</td>
<td>1.2</td>
<td>118.1</td>
<td>3,535.5</td>
<td>1,535.6</td>
<td>4.1</td>
</tr>
<tr>
<td>Neptune</td>
<td>Nept. Tran.&lt;sup&gt;e&lt;/sup&gt;</td>
<td>2.7</td>
<td>220.2</td>
<td>5,390.7</td>
<td>2,726.2</td>
<td>6.1</td>
</tr>
<tr>
<td>68380Tet.&lt;sup&gt;f&lt;/sup&gt;</td>
<td>914686Tet.&lt;sup&gt;g&lt;/sup&gt;</td>
<td>51.3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Armad.Tet.&lt;sup&gt;h&lt;/sup&gt;</td>
<td>Arm.Tet.Trans.&lt;sup&gt;i&lt;/sup&gt;</td>
<td>263.3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>518092Tetra</td>
<td>461112Tetra</td>
<td>136.6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

- Exact, parallel
- Exact, sequential
- Inexact, parallel
## Comparing against other methods

<table>
<thead>
<tr>
<th>Mesh 0</th>
<th>Mesh 1</th>
<th>3D-Epug</th>
<th>LibiGL</th>
<th>Convert&lt;sup&gt;a&lt;/sup&gt;</th>
<th>Intersect&lt;sup&gt;b&lt;/sup&gt;</th>
<th>QuickCSG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Casting10kf</td>
<td>Clutch2kf</td>
<td>0.2</td>
<td>1.3</td>
<td>4.2</td>
<td>1.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Armadillo52kf</td>
<td>Dinosaur40kf</td>
<td>0.1</td>
<td>3.0</td>
<td>38.0</td>
<td>21.5</td>
<td>0.1</td>
</tr>
<tr>
<td>Horse40kf</td>
<td>Cow76kf</td>
<td>0.1</td>
<td>3.2</td>
<td>51.1</td>
<td>24.2</td>
<td>0.1</td>
</tr>
<tr>
<td>Camel69kf</td>
<td>Armadillo52kf</td>
<td>0.1</td>
<td>3.2</td>
<td>54.3</td>
<td>25.7</td>
<td>0.1</td>
</tr>
<tr>
<td>Camel</td>
<td>Camel</td>
<td>13.9</td>
<td>18.0</td>
<td>62.7</td>
<td>230.6</td>
<td><strong>0.9</strong></td>
</tr>
<tr>
<td>Camel</td>
<td>Armadillo</td>
<td>0.2</td>
<td>11.7</td>
<td>189.9</td>
<td>80.0</td>
<td>0.3</td>
</tr>
<tr>
<td>Armadillo</td>
<td>Armadillo</td>
<td>67.0</td>
<td>88.1</td>
<td>339.7</td>
<td>1,198.2</td>
<td><strong>4.1</strong></td>
</tr>
<tr>
<td>Armadillo</td>
<td>Armadillo</td>
<td>0.8</td>
<td>58.9</td>
<td>753.2</td>
<td>473.2</td>
<td>1.1</td>
</tr>
<tr>
<td>461112</td>
<td>461115</td>
<td>0.3</td>
<td>28.6</td>
<td>819.8</td>
<td>329.6</td>
<td>1.1</td>
</tr>
<tr>
<td>Kitten</td>
<td>RedCircBox</td>
<td>0.6</td>
<td>58.0</td>
<td>971.7</td>
<td>455.7</td>
<td>1.1</td>
</tr>
<tr>
<td>Bimba</td>
<td>Vase</td>
<td>2.1</td>
<td>96.0</td>
<td>1,723.7</td>
<td>905.5</td>
<td><strong>2.2</strong></td>
</tr>
<tr>
<td>226633</td>
<td>461112</td>
<td>1.2</td>
<td>118.1</td>
<td>3,535.5</td>
<td>1,535.6</td>
<td>4.1</td>
</tr>
<tr>
<td>Ramesses</td>
<td>Ram.Transl.</td>
<td>2.7</td>
<td>220.2</td>
<td>5,390.7</td>
<td>2,726.2</td>
<td>6.1</td>
</tr>
<tr>
<td>Ramesses</td>
<td>Ram.Rot.</td>
<td>914686Tet.</td>
<td>51.3</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Neptune</td>
<td>Nept.Tran.</td>
<td>68380Tet.</td>
<td>94131</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Neptune</td>
<td>Nept.Tran.</td>
<td>461112Tet.</td>
<td>136.6</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

*Meshes with many polyhedra: natural for our method*
Correctness evaluation

• **3D-EPUG-OVERLAY**
  • Solid foundation: SoS + rationals
  • We showed: special cases
  • Correct algorithm → Bug-free implementation?

• Evaluation:
  • Metro: Hausdorff distance
    • $\max(E(S_1, S_2), E(S_2, S_1))$
  • Evidence of correctness: I/O, FP errors in Metro
  • Compared against LibiGL
  • Visual inspection
  • Rotation experiments: mesh $\cap$ rotated mesh, rotated mesh $\cap$ rotated mesh
<table>
<thead>
<tr>
<th>Mesh 0</th>
<th>Mesh 1</th>
<th>Difference (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Casting10kf</td>
<td>Clutch2kf</td>
<td>0.0000</td>
</tr>
<tr>
<td>Armadillo52kf</td>
<td>Dinosaur40kf</td>
<td>0.0000</td>
</tr>
<tr>
<td>Horse40kf</td>
<td>Cow76kf</td>
<td>0.0000</td>
</tr>
<tr>
<td>Camel69kf</td>
<td>Armadillo52kf</td>
<td>0.0000</td>
</tr>
<tr>
<td>Camel</td>
<td>Armadillo</td>
<td>0.0000</td>
</tr>
<tr>
<td>Armadillo</td>
<td>Armadillo</td>
<td>0.0000</td>
</tr>
<tr>
<td>461112</td>
<td>461115</td>
<td>0.0000</td>
</tr>
<tr>
<td>Kitten</td>
<td>RedCircle</td>
<td>0.0000</td>
</tr>
<tr>
<td>Bimba</td>
<td>Vase</td>
<td>0.0000</td>
</tr>
<tr>
<td>226633</td>
<td>461112</td>
<td>0.0000</td>
</tr>
<tr>
<td>Ramesses</td>
<td>Ram.Transl. a</td>
<td>0.0000</td>
</tr>
<tr>
<td>Ramesses</td>
<td>Ram.Rot. b</td>
<td>0.0000</td>
</tr>
<tr>
<td>Neptune</td>
<td>Ramesses</td>
<td>0.0000</td>
</tr>
<tr>
<td>Neptune</td>
<td>Nept.Transl. c</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

Hausdorff distances vs LibiGL
Visual inspection – 3D-EPUG-OVERLAY
Visual inspection – QuickCSG
Visual inspection – QuickCSG
Conclusions

• Careful implementation → 3 exact and efficient algorithms
• Two preliminary algorithms
• EPUG-OVERLAY:
  • Faster than GRASS GIS inexact method
  • Exact
• PinMesh:
  • Up to 27x faster than RCT
  • Exact
Conclusions

- Main result: 3D-EPUG-OVERLAY
- Exact: rationals and SoS
  - Results matched reference solution
- Fast: uniform grid, parallel, simple representation, intervalU
  - Up to 101x times faster than LibiGL (also parallel)
  - Up to 1.284x/4,241x times faster than CGAL
  - Faster than QuickCSG (parallel/inexact/no special cases) in most of test cases
- Parallel → better usage of computers
- Fast and exact → good for applications like CAD/GIS (interactivity & exactness)
Future work

- Algorithms developed in sequence → use 3D-EPUG-OVERLAY improvements in other methods
- Implement other CSG operations (easy)
- Create a CGAL kernel with SoS → use CGAL algorithms (example: Delaunay)
- Improve performance of SoS predicates
- Develop strategies for choosing the grid resolution (ex: recreate grid until good resolution)
- Strategies for removing the perturbation from the output