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

Mesh intersection

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

- 2D intersection also extends to 3D.

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

- Our focus: 3D triangulated meshes
**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 \(\rightarrow\) 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)

Examples from CGAL mailing list (there are several other similar threads): People want exactness and performance!
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: \( \frac{2}{3} + \frac{4}{5} = \frac{22}{15} \)
  - 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”: $O(|A| \times |B|)$
- 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

![Image of triangle-triangle intersections]

Second step: retessellation

- 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.

![Image of retessellation]
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.

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 $\rightarrow$ $t$ 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: 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 $\varepsilon^i$
  - 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 conceptual → resulting rectangle is actually equal to input triangles!

- 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.

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+\epsilon, y+\epsilon^2, z+\epsilon^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)

<table>
<thead>
<tr>
<th>Mesh 0</th>
<th>Mesh 1</th>
<th>Triangles (thousands)</th>
<th>Running times (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Mesh 0</td>
<td>Mesh 1</td>
</tr>
<tr>
<td>Casting10kf</td>
<td>Clutch2kf</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>Armadillo52kf</td>
<td>Dinosaur40kf</td>
<td>52</td>
<td>40</td>
</tr>
<tr>
<td>Horse40kf</td>
<td>Cow76kf</td>
<td>40</td>
<td>76</td>
</tr>
<tr>
<td>Camel69kf</td>
<td>Armadillo52kf</td>
<td>69</td>
<td>52</td>
</tr>
<tr>
<td>Camel</td>
<td>Armadillo</td>
<td>69</td>
<td>331</td>
</tr>
<tr>
<td>Armadillo</td>
<td>Armadillo</td>
<td>331</td>
<td>331</td>
</tr>
<tr>
<td>461112</td>
<td>461115</td>
<td>805</td>
<td>822</td>
</tr>
<tr>
<td>Kitten</td>
<td>RedCircBox</td>
<td>274</td>
<td>1402</td>
</tr>
<tr>
<td>Bimba</td>
<td>Vase</td>
<td>150</td>
<td>1792</td>
</tr>
<tr>
<td>226633</td>
<td>461112</td>
<td>2452</td>
<td>805</td>
</tr>
<tr>
<td>Ramesses</td>
<td>Ramess.Trans.</td>
<td>1653</td>
<td>1653</td>
</tr>
<tr>
<td>Ramesses</td>
<td>Ramess.Rot.</td>
<td>1653</td>
<td>1653</td>
</tr>
<tr>
<td>Neptune</td>
<td>Ramesses</td>
<td>4008</td>
<td>1653</td>
</tr>
<tr>
<td>Neptune</td>
<td>Nept.Transl</td>
<td>4008</td>
<td>4008</td>
</tr>
<tr>
<td>ArmadilloTetra</td>
<td>ArmadilloTetraTransl</td>
<td>1602</td>
<td>1602</td>
</tr>
<tr>
<td>518092_tetra</td>
<td>461112_tetra</td>
<td>5938</td>
<td>8495</td>
</tr>
</tbody>
</table>
Experiments

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

<table>
<thead>
<tr>
<th>Mesh 0</th>
<th>Mesh 1</th>
<th>Triangles (thousands)</th>
<th>Running times (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Mesh 0</td>
<td>Mesh 1</td>
</tr>
<tr>
<td>Casting10kf</td>
<td>Clutch2kf</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>Armadillo52kf</td>
<td>Dinosaur40kf</td>
<td>52</td>
<td>40</td>
</tr>
<tr>
<td>Horse40kf</td>
<td>Cow76kf</td>
<td>40</td>
<td>76</td>
</tr>
<tr>
<td>Camel69kf</td>
<td>Armadillo52kf</td>
<td>69</td>
<td>52</td>
</tr>
<tr>
<td>Camel</td>
<td>Camel</td>
<td>69</td>
<td>69</td>
</tr>
<tr>
<td>Armadillo</td>
<td>Armadillo</td>
<td>69</td>
<td>331</td>
</tr>
<tr>
<td>461112</td>
<td>461112</td>
<td>805</td>
<td>822</td>
</tr>
<tr>
<td>Kitten</td>
<td>RedCircBox</td>
<td>274</td>
<td>1402</td>
</tr>
<tr>
<td>Camel</td>
<td>Vase</td>
<td>150</td>
<td>1792</td>
</tr>
<tr>
<td>226633</td>
<td>461112</td>
<td>2452</td>
<td>805</td>
</tr>
<tr>
<td>Ramesses</td>
<td>Ramess.Trans.</td>
<td>1653</td>
<td>1653</td>
</tr>
<tr>
<td>Ramesses</td>
<td>Ramess.Rot.</td>
<td>1653</td>
<td>1653</td>
</tr>
<tr>
<td>Neptune</td>
<td>Ramesses</td>
<td>4008</td>
<td>1653</td>
</tr>
<tr>
<td>Neptune</td>
<td>Nept.Transl.</td>
<td>4008</td>
<td>4008</td>
</tr>
<tr>
<td>ArmadilloTetra</td>
<td>ArmadilloTetraTransl</td>
<td>1602</td>
<td>1602</td>
</tr>
<tr>
<td>518092_tetra</td>
<td>461112_tetra</td>
<td>5938</td>
<td>8495</td>
</tr>
</tbody>
</table>

Experiments

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

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

<table>
<thead>
<tr>
<th>Mesh 0</th>
<th>Mesh 1</th>
<th>Triangles (thousands)</th>
<th>Running times (s)</th>
<th>CGAL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Mesh 0</td>
<td>Mesh 1</td>
<td>Out</td>
</tr>
<tr>
<td>Casting10kf</td>
<td>Clutch2kf</td>
<td>10</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Armadillo52kf</td>
<td>Dinosaur40kf</td>
<td>52</td>
<td>40</td>
<td>25</td>
</tr>
<tr>
<td>Horse40kf</td>
<td>Cow76kf</td>
<td>40</td>
<td>76</td>
<td>24</td>
</tr>
<tr>
<td>Camele69kf</td>
<td>Armadillo52kf</td>
<td>69</td>
<td>52</td>
<td>16</td>
</tr>
<tr>
<td>Camel</td>
<td>Camel</td>
<td>69</td>
<td>69</td>
<td>81</td>
</tr>
<tr>
<td>Camel</td>
<td>Armadillo</td>
<td>69</td>
<td>331</td>
<td>43</td>
</tr>
<tr>
<td>Armadillo</td>
<td>Armadillo</td>
<td>331</td>
<td>331</td>
<td>441</td>
</tr>
<tr>
<td>461112</td>
<td>461115</td>
<td>805</td>
<td>822</td>
<td>808</td>
</tr>
<tr>
<td>Kitten</td>
<td>RedCircBox</td>
<td>274</td>
<td>1402</td>
<td>246</td>
</tr>
<tr>
<td>Bimba</td>
<td>Vase</td>
<td>150</td>
<td>1792</td>
<td>724</td>
</tr>
<tr>
<td>226633</td>
<td>461112</td>
<td>2452</td>
<td>805</td>
<td>1437</td>
</tr>
<tr>
<td>Ramesses</td>
<td>Ramess.Trans.</td>
<td>1653</td>
<td>1653</td>
<td>1571</td>
</tr>
<tr>
<td>Ramesses</td>
<td>Ramess.Rot.</td>
<td>1653</td>
<td>1653</td>
<td>1691</td>
</tr>
<tr>
<td>Neptune</td>
<td>Ramesses</td>
<td>4008</td>
<td>1653</td>
<td>1112</td>
</tr>
<tr>
<td>Neptune</td>
<td>Nept.Transl.</td>
<td>4008</td>
<td>4008</td>
<td>3303</td>
</tr>
<tr>
<td>ArmadilloTetra</td>
<td>ArmadilloTetraTransl</td>
<td>1602</td>
<td>1602</td>
<td>61325</td>
</tr>
<tr>
<td>518092.tetra</td>
<td>461112.tetra</td>
<td>5938</td>
<td>8495</td>
<td>23181</td>
</tr>
</tbody>
</table>

Experiments

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

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

<table>
<thead>
<tr>
<th>Mesh 0</th>
<th>Mesh 1</th>
<th>Triangles (thousands)</th>
<th>Running times (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Mesh 0</td>
<td>Mesh 1</td>
</tr>
<tr>
<td>Casting10kf</td>
<td>Clutch2kf</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>Armadillo52kf</td>
<td>Dinosaur40kf</td>
<td>52</td>
<td>40</td>
</tr>
<tr>
<td>Horse40kf</td>
<td>Cowl76kf</td>
<td>40</td>
<td>76</td>
</tr>
<tr>
<td>Camel96kf</td>
<td>Armadillo52kf</td>
<td>69</td>
<td>52</td>
</tr>
<tr>
<td>Camel</td>
<td>Armadillo</td>
<td>69</td>
<td>69</td>
</tr>
<tr>
<td>Armadillo</td>
<td>Armadillo</td>
<td>331</td>
<td>331</td>
</tr>
<tr>
<td>461112</td>
<td>461115</td>
<td>805</td>
<td>822</td>
</tr>
<tr>
<td>Kitten</td>
<td>RedCircBox</td>
<td>274</td>
<td>1402</td>
</tr>
<tr>
<td>Bimba</td>
<td>Vase</td>
<td>150</td>
<td>1792</td>
</tr>
<tr>
<td>226633</td>
<td>461112</td>
<td>2452</td>
<td>805</td>
</tr>
<tr>
<td>Ramesses</td>
<td>Ramess.Trans.</td>
<td>1653</td>
<td>1653</td>
</tr>
<tr>
<td>Ramesses</td>
<td>Ramess.Rot.</td>
<td>1653</td>
<td>1653</td>
</tr>
<tr>
<td>Neptune</td>
<td>Ramesses</td>
<td>4008</td>
<td>1653</td>
</tr>
<tr>
<td>Neptune</td>
<td>Nept.Trans</td>
<td>4008</td>
<td>4008</td>
</tr>
<tr>
<td>ArmadilloTetra</td>
<td>ArmadilloTetra Transl</td>
<td>1602</td>
<td>1602</td>
</tr>
<tr>
<td>518092_tetra</td>
<td>461112_tetra</td>
<td>5938</td>
<td>8495</td>
</tr>
</tbody>
</table>

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

- 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.

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 and exact

- Future work:
  - Improve performance (mainly of SoS calls)
  - Use similar ideas for other problems
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 $3$?

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

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

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

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

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

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

Simulation of Simplicity

- SoS implementation:
  - \( q'_e \) will never be on a vertex or edge of \( t' \) → if \( \text{det}=0 \) → false
  - Replace \( q' \) with \( q'_e \)

\[
\lambda_0 = \frac{(t'_{1y} - t'_{2y}) \times (q'_x - t'_{2x}) + (t'_{2x} - t'_{1x}) \times (q'_y - t'_{2y})}{\text{det}}
\]
\[
\lambda_1 = \frac{(t'_{2y} - t'_{0y}) \times (q'_x - t'_{2x}) + (t'_{0x} - t'_{2x}) \times (q'_y - t'_{2y})}{\text{det}}
\]
\[
\lambda_2 = 1 - \lambda_0 - \lambda_1
\]
\[
\text{det} = (t'_{1y} - t'_{2y}) \times (t'_{0x} - t'_{2x}) + (t'_{2x} - t'_{1x}) \times (t'_{0y} - t'_{2y})
\]
Simulation of Simplicity

- SoS implementation:
  - \( q'_e \) will never be on a vertex or edge of \( t' \) → if \( \text{det} = 0 \) → false
  - Replace \( q' \) with \( q'_e \) → \( \lambda \) with \( \lambda_{ei} \)
  - E.g.: is \( 0 < \lambda_{e0} \)?
    - \( \lambda_0 \neq 0 \) → check \( \lambda_0 \)
    - \( \lambda_0 = 0 \) → check \( t'_1 - t'_2 \)
    - \( t'_1 - t'_2 = 0 \) → check \( t'_2 - t'_1 \)
  - Both can't be 0.

\[
\lambda_{\varepsilon_0} = \lambda_0 + \frac{(t'_{1y} - t'_{2y}) \times \varepsilon + (t'_{2x} - t'_{1x}) \times \varepsilon^2}{\text{det}}
\]

\[
\lambda_{\varepsilon_1} = \lambda_1 + \frac{(t'_{2y} - t'_{0y}) \times \varepsilon + (t'_{0x} - t'_{2x}) \times \varepsilon^2}{\text{det}}
\]

\[
\lambda_{\varepsilon_2} = 1 - \lambda_{\varepsilon_0} - \lambda_{\varepsilon_1}
\]