Out-of-core constrained Delaunay tetrahedralizations for large scenes

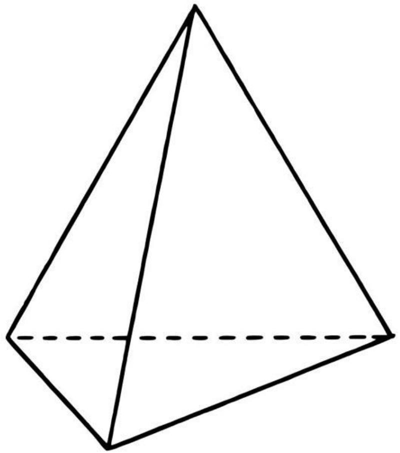
Ziya Erkoç[1] Aytek Aman[1] Uğur Güdükbay[1] Hang Si[2]

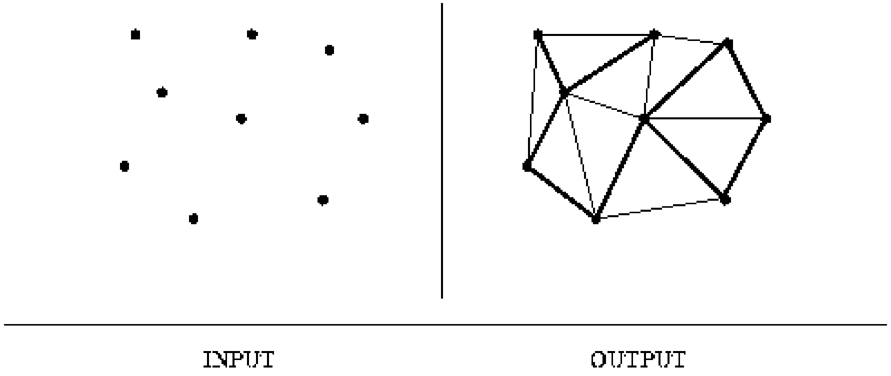
1 Department of Computer Engineering, Bilkent University, Ankara 06800 Turkey
2 Weierstrass Institute, Mohrenstrasse 39, 10117 Berlin, Germany

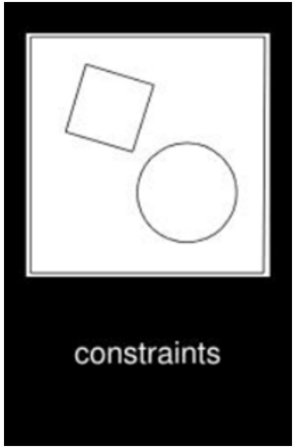
NUMGRID 2020

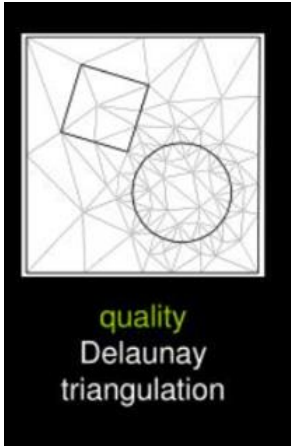
INTRODUCTION

- Delaunay Triangulation (2D)
- Delaunay Tetrahedralization (3D)
- Constrained Delaunay Triangulation (2D) [2]
- Constrained Delaunay Tetrahedralization (3D)



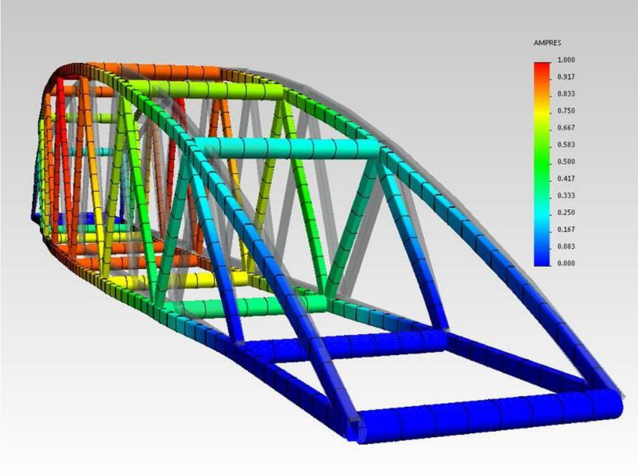


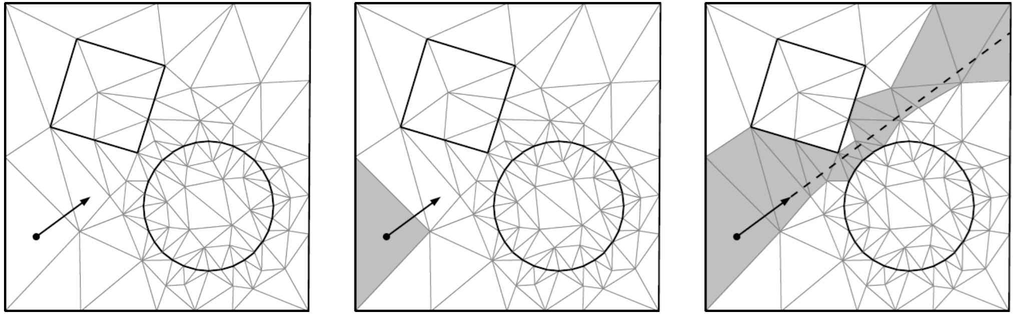




APPLICATIONS

- Finite Element Methods
- Ray Tracing accelerators [8]



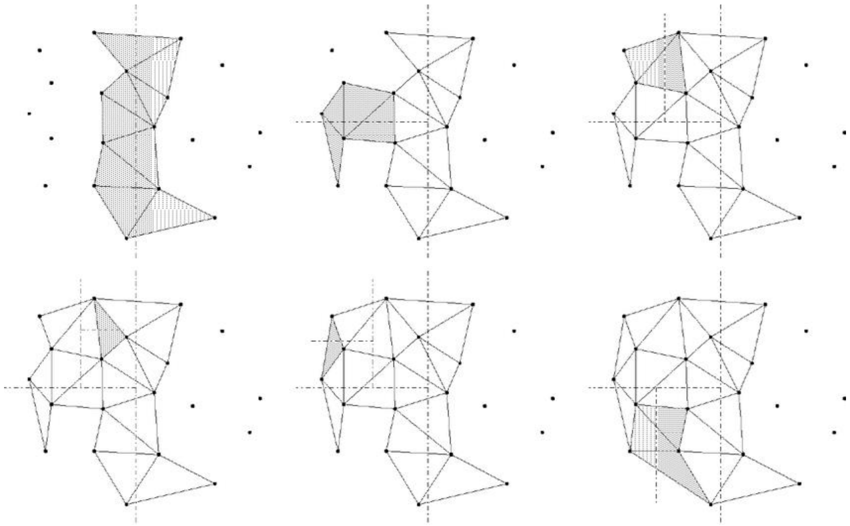


MOTIVATION

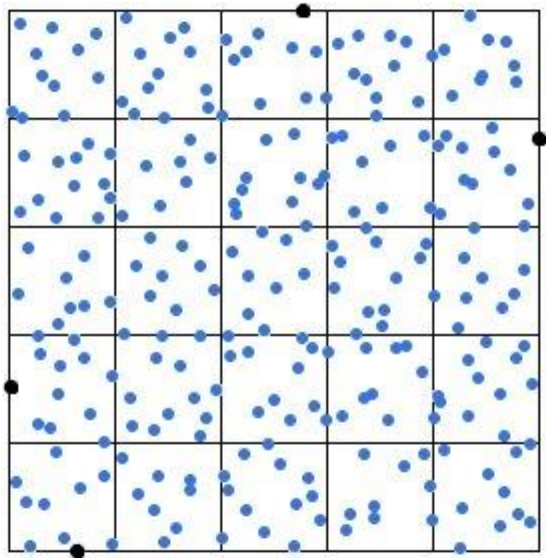
- Insufficiency of memory
- Need for memory efficient algorithm
- Developed out-of-core divide-and-conquer algorithm

RELATED WORKS

- Cignoni et al.'s Divide-and-conquer Dewall algorithm [3]
    - Not constrained

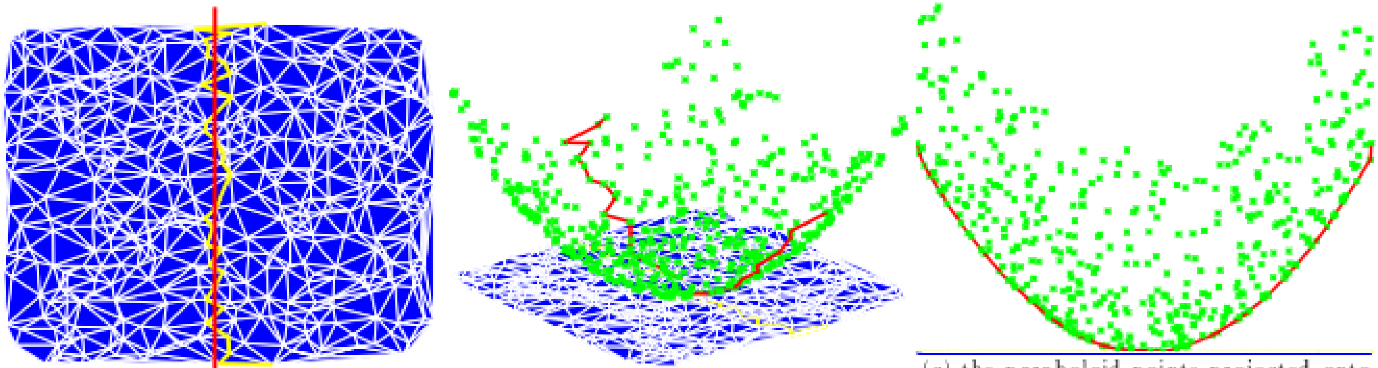


- Smolik & Skala's out-of-core algorithm [6]
    - Not constrained



RELATED WORKS

- Blelloch et al.'s parallel divide-and-conquer algorithm[1]
  - Not constrained



(a) Median line $\mathcal{L}$ and path $\mathcal{H}$.

(b) points projected onto the paraboloid.

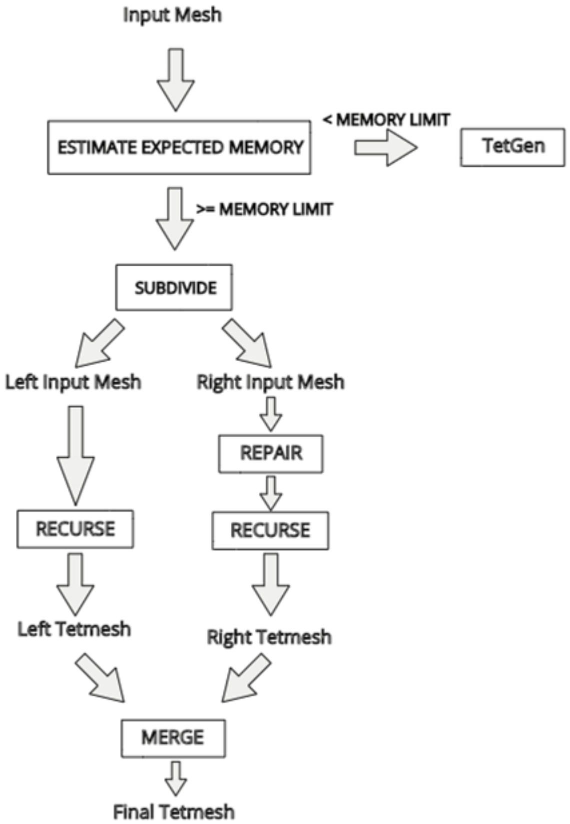
(c) the paraboloid points projected onto the vertical plane.

TETGEN

- Quality tetrahedral mesh generator [5]
- Constrained Delaunay Tetrahedralization
- Base case of our Divide and Conquer algorithm
- Need to satisfy its requirements, which we will describe shortly

OUR APPROACH

- Estimating the memory required by TetGen using Linear Regression
- Dividing the input mesh into two using CGAL's clip function [7]
- Repairing overlapping vertices and edges
- Recursively calling our algorithm to construct tetrahedralization of both sides
- Merging both tetrahedral meshes



OUR APPROACH

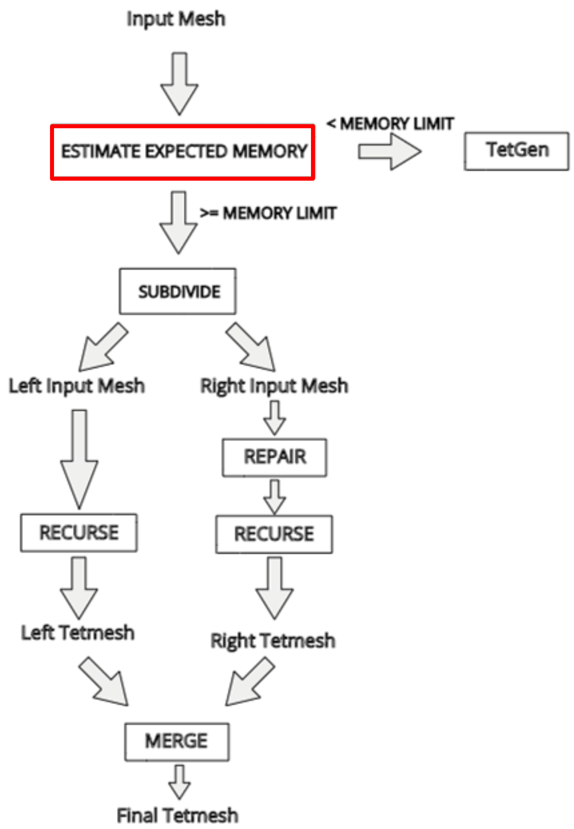
Memory Estimation

Linear regression model generated from the below data is $y = 4.57 \times 10^{-3}\ X + 3.12$

X: number of vertices

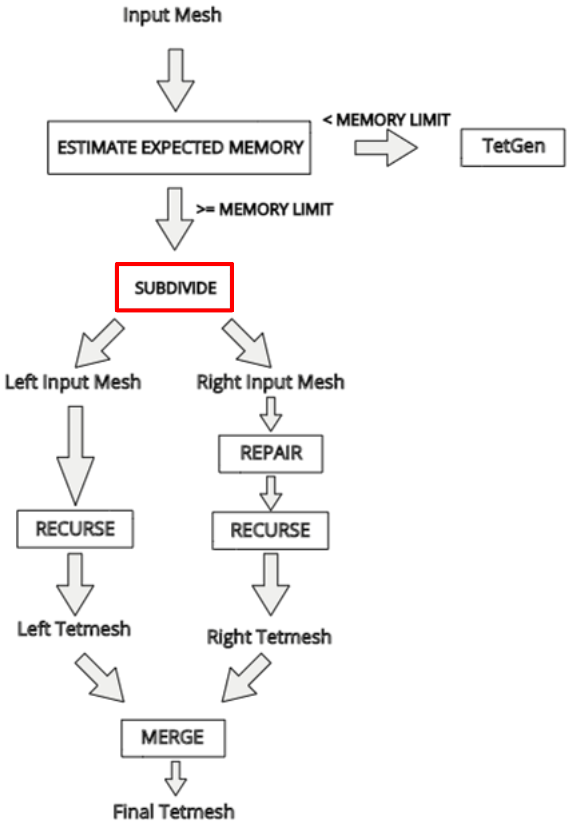
y: expected memory requirement in MB

Number of vertices	Memory Requirement (MB)	Expected Memory Requirement (MB)
1,440	7.03	9.70
2,880	13.97	16.28
34,560	167.67	161.04
112,220	514.80	515.91
172,971	792.97	793.51

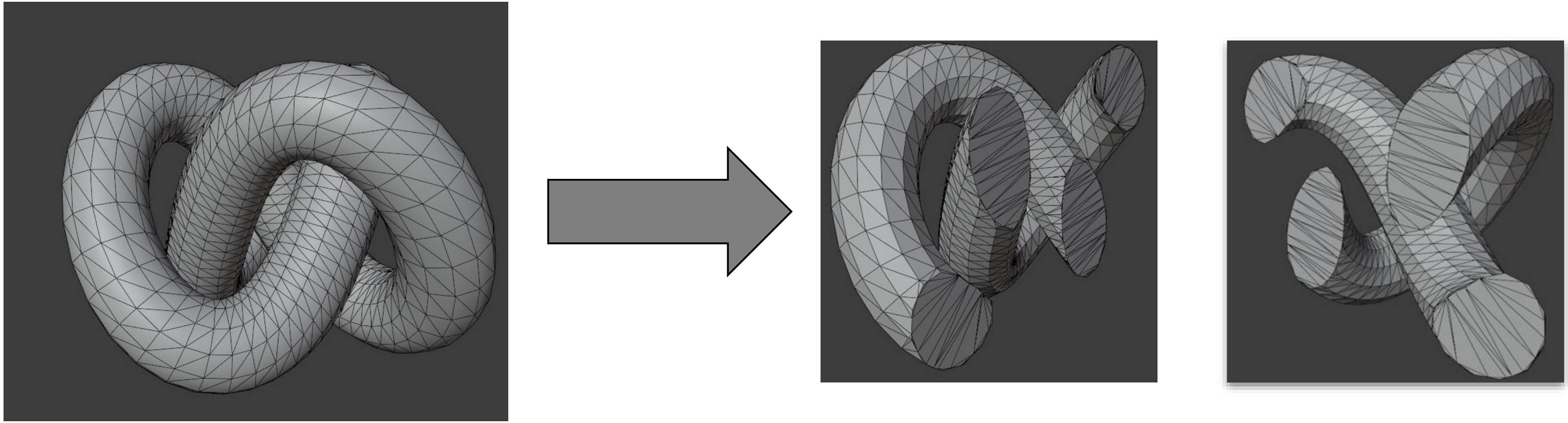


OUR APPROACH
Subdivision

- Used CGAL's clipping routine [7]
- Cut the object from the middle into two
- TetGen requires closed surface
- 2D Triangulation of one side by CGAL
- Copy triangulation to the other side



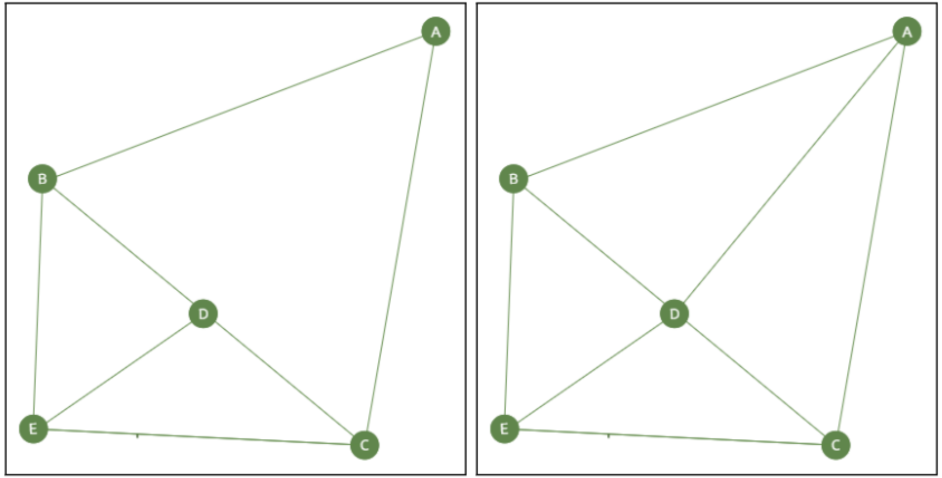
OUR APPROACH
Subdivision



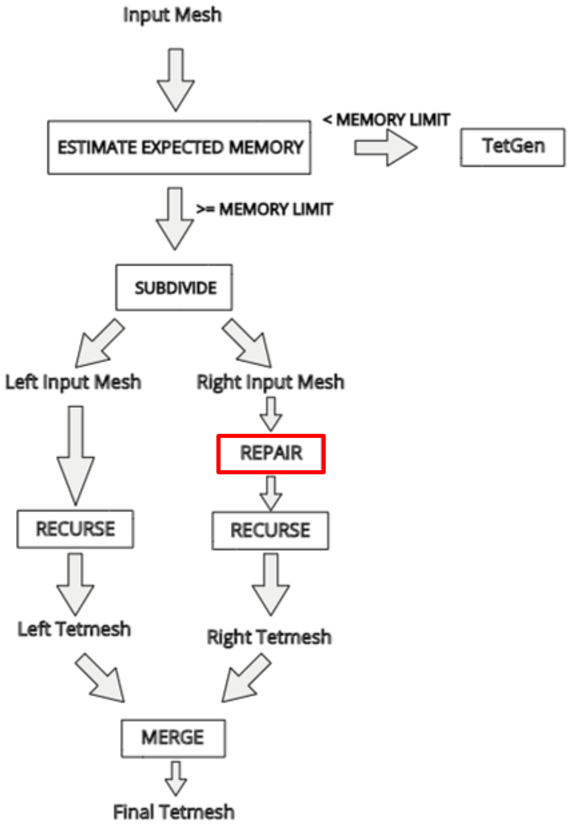
OUR APPROACH

Repairing

- Overlapping vertices
- Overlapping edges



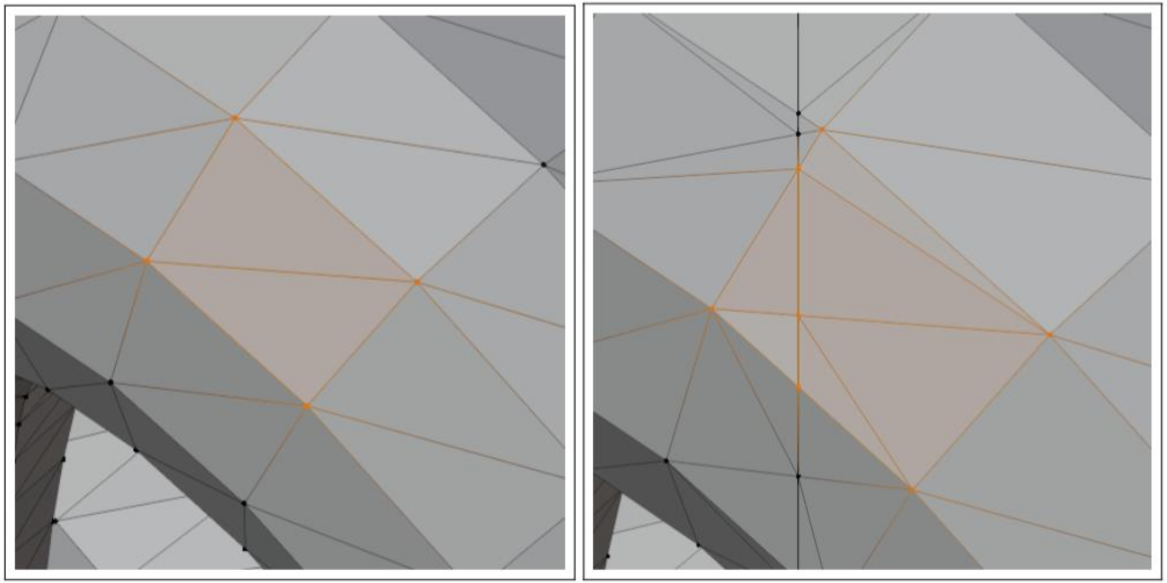
(a) (b)



OUR APPROACH

Repairing

- Overlapping edges

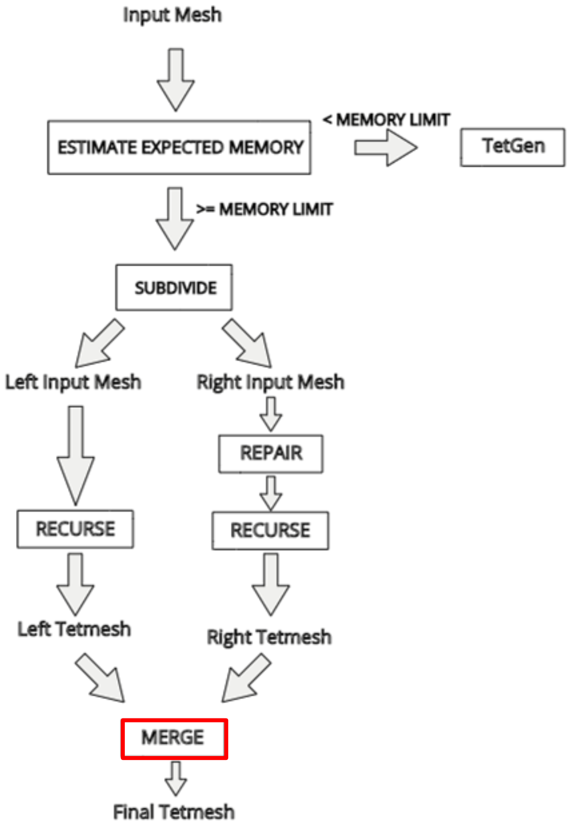


OUR APPROACH

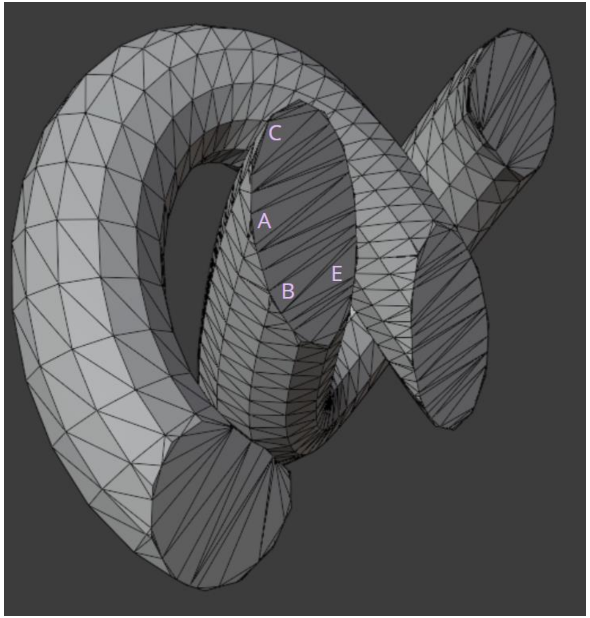
Merging

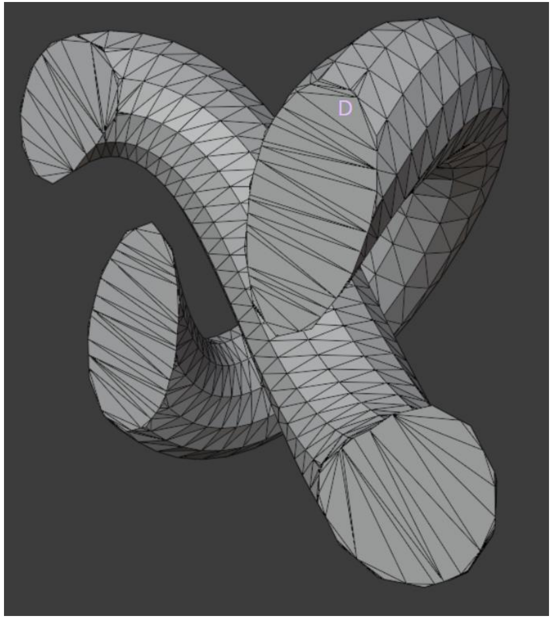
- Two mesh files are merged into one
- Missing neighbour relations around the cut plane
- Spatial hashing

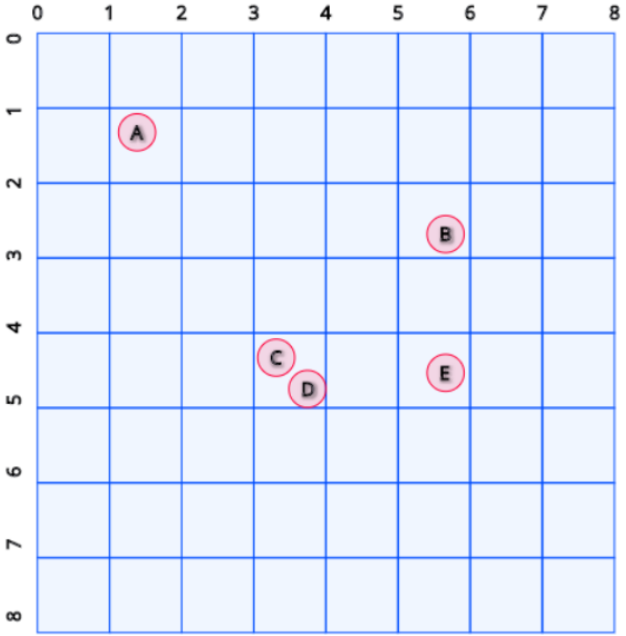
$left\_centroids$ = GET_CENTROIDS($left\_mesh\_file$)
$left\_centroids\_grid$ = GENERATE_GRID($left\_centroids$)
$right\_centroids$ = GET_CENTROIDS($right\_mesh\_file$)
for each $right\_centroid \in right\_centroids$ **do**
    if $right\_centroid \in left\_centroids\_grid$ **then**
        ADD_NEIGHBOUR($right\_tet$, $left\_tet$, $output\_mesh\_file$)
    end if
end for



OUR APPROACH
Merging







RESULTS

- Used computer with Intel Xeon E5-2620 2.10 GHz processor and 64 GB of RAM
- Tracked peak physical memory usage using Task Manager

RESULTS
Runtime and Memory

- Our algorithm used less memory but took more time
- Clipping and repairing take time

Experiment Number	Number of vertices	Number of faces	TetGen		Ours	
			Time (s)	Memory (MB)	Time (s)	Memory (MB)
1	172,969	345,938	37.264	850	429.437	483
2	172,969	345,938	66.537	947	456.67	564
3	345,938	691,876	43.958	1,700	138.199	922
4	1,346,688	2,693,376	294.925	6,605	727.893	3,584
5	1,704,146	3,408,292	380.739	8,359	642.101	4,537
6	17,682,248	35,364,496	91,101,455	53,160	7,047,264	46,614
7	27,164,160	54,328,320	N/A	N/A	22,221.27	58,964

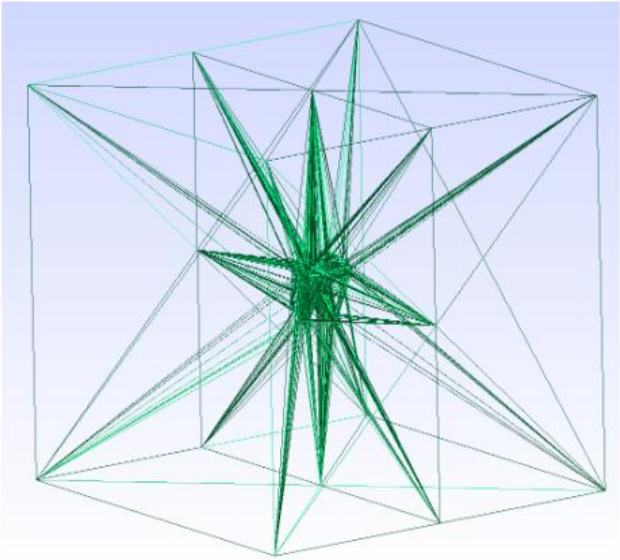
RESULTS
Runtime and Memory

TetGen fails when physical memory and virtual memory exhausted

Experiment Number	Number of vertices	Number of faces	TetGen		Ours	
			Time (s)	Memory (MB)	Time (s)	Memory (MB)
1	172,969	345,938	37.264	850	429.437	483
2	172,969	345,938	66.537	947	456.67	564
3	345,938	691,876	43.958	1,700	138.199	922
4	1,346,688	2,693,376	294.925	6,605	727.893	3,584
5	1,704,146	3,408,292	380.739	8,359	642.101	4,537
6	17,682,248	35,364,496	91,101,455	53,160	7,047,264	46,614
7	27,164,160	54,328,320	N/A	N/A	22,221.27	58,964

RESULTS

Runtime and Memory



RESULTS

Runtime and Memory

RESULTS
Quality

- Used aspect ratio measure of TetGen (longest edge / smallest height)
- Ours produced slightly worse tetrahedra

Model Name	Number of vertices	Number of faces	TetGen			Ours		
			Minimum	Maximum	Average	Minimum	Maximum	Average
Torus Knot	1,440	2,880	1.90	125.74	7.52	2.01	10,365.34	22.20
Neptune	112,224	224,448	1.30	536.49	8.72	1.28	250,088.04	12.92
Armadillo	172,969	345,938	1.3	262,232.06	7.31	1.27	260,203.69	12.84

RESULTS
Quality

- Boundary tetrahedra causing bad quality

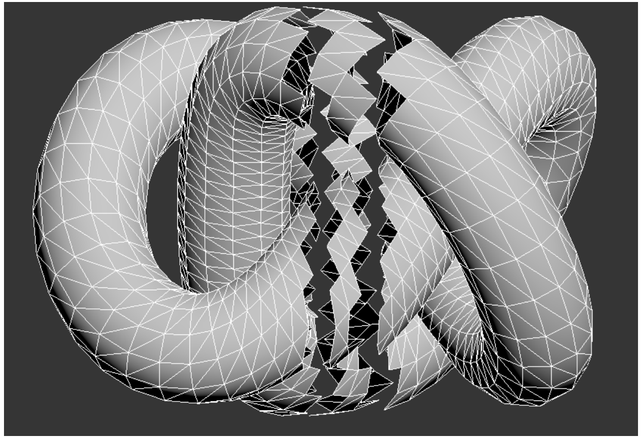
Model Name	Number of vertices	Number of faces	TetGen			Ours		
			Minimum	Maximum	Average	Minimum	Maximum	Average
Torus Knot	1,440	2,880	1.90	125.74	7.52	2.01	10,365.34	22.20
Neptune	112,224	224,448	1.30	536.49	8.72	1.28	250,088.04	12.92
Armadillo	172,969	345,938	1.3	262,232.06	7.31	1.27	260,203.69	12.84

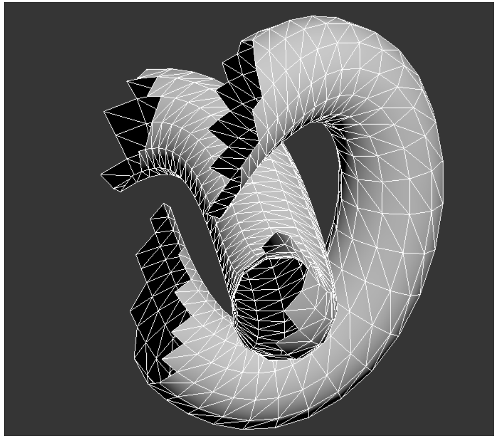
CONCLUSION

- Our method uses less memory than TetGen
- TetGen is faster
- Worst case when cutting plane coincides input
- Later, cleverer algorithm for plane selection (curved etc.) or subdivision
- Worse quality tetrahedra
- Later, refinement step to increase quality
- Not considering mesh densities
- Later, handling varying mesh densities

CURRENT WORK

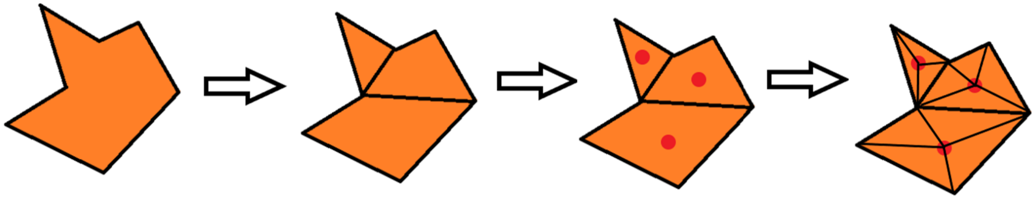
- Subdividing the mesh without inserting vertices to surface
- Closing open surfaces using concave decomposition

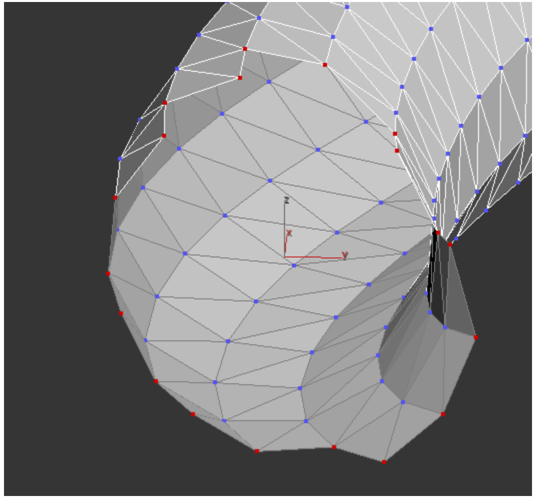


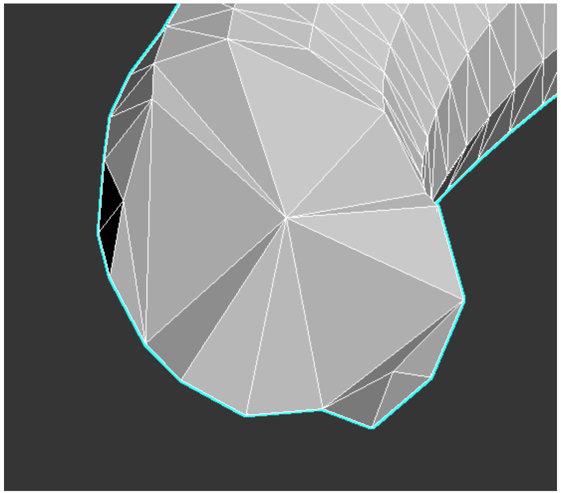


CURRENT WORK

- Moving vertices to 2D plane
- Applying convex decomposition
- Inserting vertices inside the object not on the surface
- Closing the surface
- Moving back to 3D







REFERENCES

[1] Blelloch, G.E., Miller, G.L., Talmor, D.: Developing a practical projection-based parallel Delaunay algorithm. In: Proceedings of the Twelfth Annual Symposium on Computational Geometry, SCG ’96, p. 186–195. ACM, New York, NY, USA (1996)
[2] Chew, L.P.: Constrained Delaunay triangulations. Algorithmica 4(1-4), 97–108 (1989)
[3] Cignoni, P., Montani, C., Scopigno, R.: DeWall: A fast divide and conquer Delaunay triangulation algorithm in E^d. Computer-Aided Design 30(5), 333–341 (1998)
[4] Lagae, A., Dutré, P.: Accelerating ray tracing using constrained tetrahedralizations. Computer Graphics Forum 27(4), 1303–1312 (2008)
[5] Si, H.: TetGen, a Delaunay-based quality tetrahedral mesh generator. ACM Transactions on Mathematical Software (TOMS)41(2), 1–36 (2015)
[6] Smolik, M., Skala, V.: Fast Parallel Triangulation Algorithm of Large Data Sets in E2 and E3 for In-Core and Out-Core Memory Processing. In: Proceedings of the International Conference on Computational Science and Its Applications, ICCSA ’14, pp. 301–314. Springer (2014)
[7] The CGAL Project: CGAL User and Reference Manual, 5.0.2 ed. (2020). URL https://doc.cgal.org/5.0.2/Manual/packages.html
[8] Woop, S., Schmittler, J., Slusallek, P.: RPU: a programmable ray processing unit for realtime raytracing. ACM Transactions on Graphics (TOG)24(3), 434–444 (2005)