In-Core and Out-Core Fast Parallel Triangulation Algorithm for Large Data Sets in E2 and E3, Poster, SIGGRAPH 2014, ISBN 978-1-4503-2958-3, Canada, 2014

UNIVERSITY OF WEST BOHEMIA

In-Core and Out-Core Fast Parallel Triangulation Algorithm for Large Data Sets in E^2 and E^3

Michal Smolik
University of West Bohemia
Plzen, Czech Republic
smolik@kiv.zcu.cz

Vaclav Skala
University of West Bohemia
Plzen, Czech Republic
www.vaclavskala.eu

1. Introduction

Today's applications need to process large data sets using several processors with a shared memory, i.e. in parallel processing, or/and on systems using distributed processing. In this paper we describe an approach applicable for effective triangulation in E^2 and E^3 (tetrahedralization) for large data sets using CPU and/or GPU parallel or distributed systems, e.g. on computational clusters.

In many cases we do not need exact Delaunay triangulation or another specific triangulation. Triangulation as "close enough" to the required type of triangulation is acceptable. Weakening this strict requirement enables us to formulate a simple algorithm based on "Divide & Conquer" strategy and the approach is independent from the triangulation property requirements.

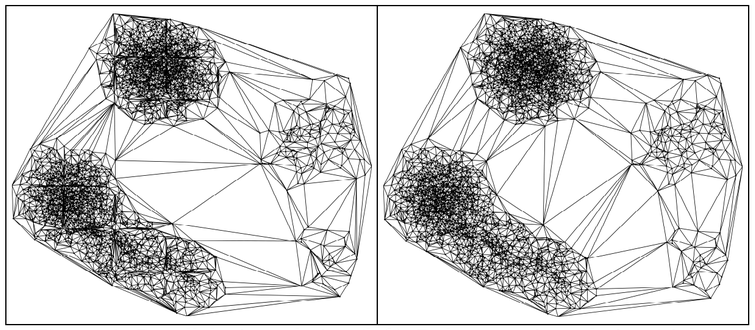


Figure 1: *Parallel triangulation (left), Delaunay triangulation (right).*

2. Principle of the Proposed Algorithm

The given data set can be split to several subsets, not necessarily rectangular, i.e. to $n \times m$ domains in E^2, resp. $n \times m \times p$ in E^3. Each data subset contains the original points plus additional corner points of the appropriate domain. Every domain is triangulated using any triangulation library.

Now, joining two triangulated domains is simple as those two domains share the same corner points. We only have to replace the common edge EF by the edge AB (see Fig. 2). It can be seen that the connection of triangulated subsets is extremely simple in the E^2 case. In the E^3 case the situation is similar and simple too.

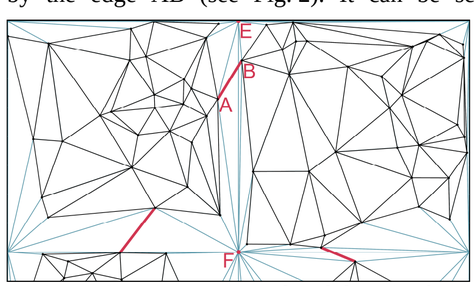


Figure 2: *Joining triangulated domains by edge EF→AB swapping.*

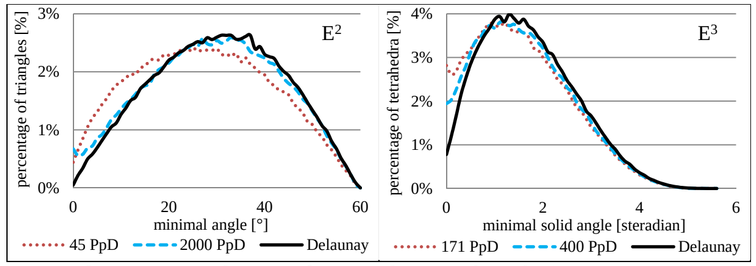
The corner points can be retained in the triangulation, or can be removed and "star-shape" holes have to be re-triangulated.

It should be noted that *domain triangulations are totally independent*. If the domains corner points have to be removed, the created holes have to be tessellated. The processes are independent again and can be made totally in parallel.

In the case of large data, we do not have to process all domains at once, but can process them in some parts considering the size of available memory. As domain triangulation and their joining are independent. Therefore there is no change of the proposed algorithm and can be implemented easily.

3. Experimental Results

The quality of triangulation has been tested on data sets with uniform distribution. The Delaunay triangulation maximizes the minimum angle therefore it is appropriate to test the distribution of minimal internal angle in triangles, resp. tetrahedra (see Graph 1&2). The more points per domain are used, the closer our triangulation is to the Delaunay triangulation. This is due to that the inner parts of all domains are Delaunay's.



Graph 1&2: *Distribution of minimal internal angle (PpD = Points per Domain).*

The Delaunay triangulation maximizes the mean inradius. It can be seen that there is only 5% difference to the Delaunay triangulation, in the case of removing the corner points.

The proposed approach has been tested on synthetic & real data sets, e.g. South Americas GIS data set, Fig. 3. Running time for $1.1 \cdot 10^6$ points was 0.42 $[s]$ and on uniform data set in E^2 the running time for 10^8 points was 28.2 $[s]$. In E^3 running time was 25.7 $[s]$ for 10^7 points, see Tab. 1 & 2, on PC Core i7 (4×2.67GHz), 12 GB RAM.

Number of points	8 threads (4 cores)	1 thread		
	Parallel triangulation	Parallel triangulation	Fade library	
	Time [s]	Time [s]	Time [s]	
316 227	0.06	0.20	0.27	
1 000 000	0.18	0.67	0.88	
3 162 277	0.65	2.23	2.96	
10 000 000	2.16	7.33	9.58	
31 622 776	7.99	24.88	35.66	
100 000 000	28.21	81.94		

Table 1: *Running times of triangulations in E^2 (uniform data sets).*

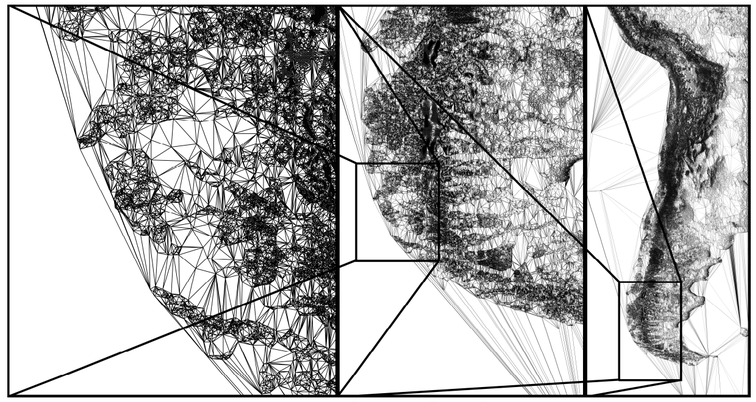


Figure 3: *Triangulation of South America.*

Number of points	8 threads (4 cores)	1 thread		
	Parallel triangulation	Parallel triangulation	TetGen library	
	Time [s]	Time [s]	Time [s]	
100 000	0.29	0.97	1.78	
316 227	0.85	2.92	5.75	
1 000 000	2.52	8.77	18.97	
3 162 277	8.11	28.35	60.60	
10 000 000	25.72	88.69	196.00	
31 622 776	81.70	278.45		

Table 2: *Running times of tetrahedralizations in E^3 (uniform data sets).*

4. Conclusions

A new fast parallel triangulation algorithm in E^2 and E^3 has been presented. It can be easily implemented on parallel environments with shared and/or distributed memory using both CPU and GPU. As it is scalable, the proposed algorithm is especially convenient for large data sets processing. The proposed approach has been implemented and tested using CPU and GPU as well.

References

CHEN,M.-B. 2011. A Parallel 3D Triangulation Method, *9th ISPA 2011*, IEEE, pp.52-56.

CIGNONI,P., MONTANI,C., SCOPIGNO,R. 1998. DeWall: A Fast Divide & Conquer Delaunay Triangulation Algorithm in Ed, *Computer Aided Design*, Vol.30, No.5, pp.333-341.

LIU,Y.-X., SNOEYING,J. 2005. A Comparison of Five Implementations 3D Delaunay Tessellations, *Combinatorial and Computational Geometry*, MSRI publ., Vol.52, pp.439-458.

SCHALLER,G., MEYER-HERMANN,M. 2004. Kinetic and Dynamic Delaunay Tetrahedralization in Three Dimensions, *Computer Physics Communications*, Vol.162, No.1, pp.9-23.

Project was supported by MSMT CR projects LG13047 and LH12181.
Printed: 7.2.2014