All closest neighbors are proper Delaunay edges generalized, and its application to parallel algorithms

Arne Maus Jon Moen Drange

{arnem,jonmd@ifi.uio.no}

Abstract

In this paper we first prove that for any set points P in the plane, the closest neighbor b to any point p in P is a proper triangle edge bp in D(P), the Delaunay triangulation of P. We also generalize this result and prove that the j'th (second, third, ..) closest neighbors b_j to p are also edges pb_j in D(P) if they satisfy simple tests on distances from a point. Even though we can find many of the edges in D(P) in this way by looking at close neighbors, we give a three point example showing that not all edges in D(P) will be found. For a random dataset we give results from test runs and a model that show that our method finds on the average 4 edges per point in D(P). Also, we prove that the Delaunay edges found in this way form a connected graph. We use these results to outline two new parallel, and potentially faster algorithms for finding D(P). We then report results from parallelizing one of these algorithms on a multicore CPU (MPU), which resulted in a significant speedup; and on a graphics card, a NVIDA GPU, where we experienced a speeddown. We explain this by discussing the NVIDA SIMT programming model, how it differs from the well-known SIMD model, and why a speeddown is obtained instead of a speedup. Finally, we comment on the k'th closest neighbors problem.

Keywords: Delaunay triangulation, closest neighbor, graph connectivity, parallel triangulation, k closest neighbors problem, Gabriel graph.

1 Introduction

Given a set of distinct points P=$p_1, p_2, .., p_n$ in the plane, not all on a single straight line. A Delaunay triangulation D(P) of P is by definition such that the circumscribed circle C_{ijk} of any triangle $p_i p_j p_k$ in D(P) contains no point from P in its interior and the points p_i, p_j and p_k from P on its circumference[1, 5, 8]. This is the empty circumscribing circle property. If we use the procedure described in [4], D(P) is unique even if we have more than three points on such circumscribing circles for the triangles in D(P).

It is also well known [5] that all all combinations of two points a and b from P that can be circumscribed by a circle with no points from P in its interior, is an

This paper was presented at the NIK-2010 conference; see http://www.nik.no/.

edge ab in D(P). In this paper we will, when using this property, always consider such circles S_{ab} that contains a and b on its boundary and have ab as its diameter - i.e. the smallest circumscribing circle for the points a and b. The set of all such edges in P is called the Gabriel graph [12]. It has been proved to be a subgraph of the Delaunay triangulation and to be connected [14]. In this paper we will work on a subgraph of the Gabriel graph, constructed by theorem T2.

Triangulated models, preferably using a Delaunay triangulation, are important in many industrial applications; from interactive games to map-making and estimating the volume of gas and oil underground deposits. One of us have made a Delaunay triangulated model for the seabed to calculate the waves for an experimental wave power plant [3]. The overwhelming use of triangulated models today is in computer graphics, where most models are represented by a mesh of triangles [15]

The theorems T0 and T1 stated below are new in the sense that they had to the best of our knowledge, never before been explicitly stated or published in any textbook or paper[5], and was also not mentioned in the Delaunay article on Wikipedia [13] before we posted it on the 16th of June 2010. After acceptance of this paper we found theorem T0 as exercise 2, ch. 4 in [8]. Theorems T2 and T3 are indisputably new.

2 The closest neighbor Theorems

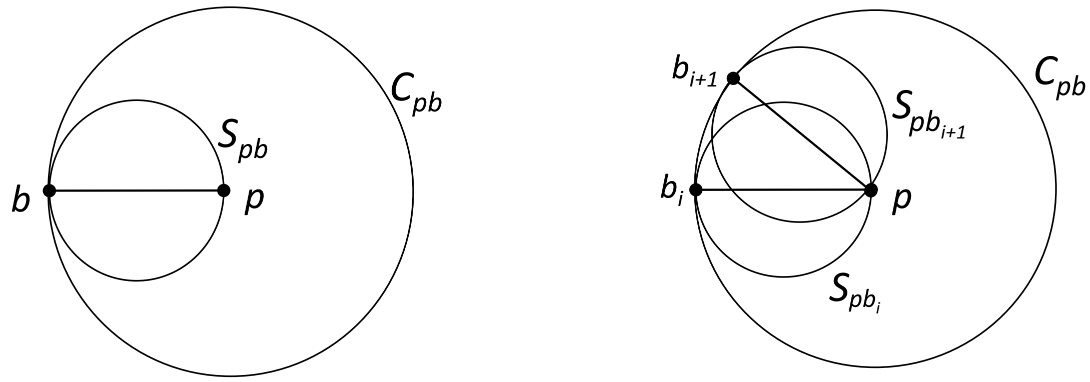
We will first prove Theorem T0 where each point p has a unique closest neighbor b. Then we will remove that assumption and formulate a more general Theorem T1 where we allow more than one closest neighbor to a point. Finally we formulate and prove a Theorem T2 which considers the k'th closest neighbors to a point a, and has T0 and T1 as special cases.

Theorem T0. *Let P be a set of points* $p_1, p_2, .., p_n$ *in the plane. Then, for all p in P, where b in P is the unique closest neighbor to p, the line segment pb is an edge in the Delaunay triangulation of P.*

Proof. The closest point b to p defines a circle C_{pb} with p as its center, pb as its radius, b on its perimeter and no point from P in its interior. By definition, the circle S_{pb} with pb as its diameter is contained in that circle C_{pb} and since S_{pb} contains no other point from P in its interior, then pb is an edge of a triangle in D(P) (fig. 1a on the following page) □

Theorem T1. *Let P be a set of points* $p_1, p_2, .., p_n$ *in the plane. Then, for all p in P, the line segments* pb_i*, where* $b_1, b_2, .., b_k$ *in P are the equal closest neighbors to p in P, are all edges in the Delaunay triangulation of P.*

Proof. The distance to one of the equal closest points b_i defines a circle C_{pb} with p as its center and $pb_1 (= pb_2, ..., = pb_k)$ as its radius. This circle has no point from P in its interior. By definition the circles S_{pb_i} with $pb_1, pb_2, ..., pb_k$ as their diameters, have all p on their boundaries. These circles are all fully contained in the circle C_{pb}. Since the points b_i are all distinct, they are all different points on the circumference of circle C_{pb}. The circles S_{pb_i} then contain no points from P, nor any of the other points b_i in their interior. Therefore all the line segments $pb_1, pb_2, ..., pb_k$ are edges in the Delaunay triangulation of P (fig. 1b on the next page). □



(a) The circle C_{pb} with no interior points and pb as radius fully contains the circle S_{pb} with pb as its diameter.

(b) The circle C_{pb} with no interior points and $pb_i = pb_{i+1}$ as radius fully including the circles S_{pb} with pb_i and pb_{i+1} as their diameters.

Figure 1: Finding circles S with no interior points from P.

We will now generalize this result to all the k closest neighbors $b_i(i = 1, 2, .., k)$ of any point p and find a sufficient condition for any of these k closest neighbors to p forming a Delaunay edge pb_i.

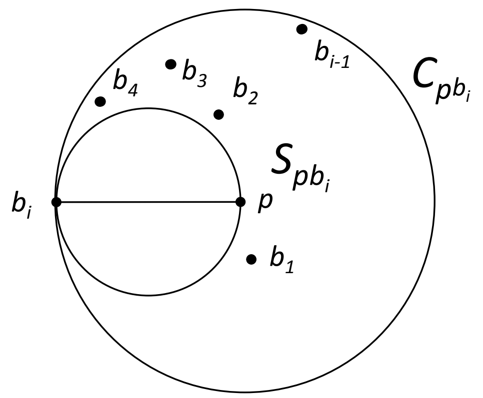
Theorem T2. *Let P be a set of points $p_1, p_2, .., p_n$ in the plane. Let $b_i(i = 1, 2, ..., k)$ in P be the k closest neighbors to a point p in P. Then pb_i is an edge in the Delaunay triangulation of P if none of the closer neighboring points b_j $(j = 1, .., i - 1)$ are included in the circle S_{pb_i} with pb_i as its diameter.*

Proof. The circle S_{pb_i} with p_{bi} as its diameter is contained in the circle C_{pb_i} with pb_i as its radius Then the points b_j $(j = i - 1, i - 2, .., 1)$ are the only points from P included in C_{pb_i} because they are closer to p than b_i. They are then the only candidates from P to be included in S_{pb_i}. If we find that none of them are included in Spb_i, we have found an empty circle S_{pb_i} that contains p and b_i on its circumference. Hence pb_i is en edge in D(P) (fig 2a on the following page) □

This is a fast test because a point is included in a circle if its distance from the circle center is smaller than the circle radius (fig 2a on the next page); or computationally faster: if the squared distance is larger then the squared circle radius. We also note that T2 with $k = 1$ is a generalization of T1 because the closest point b_1 has no closer points to test for if they are included in circle Spb_1 with b_1p as its diameter.

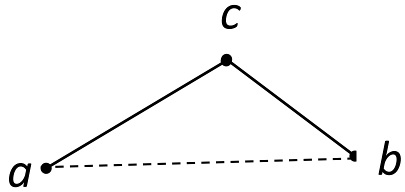
Theorem T2 does not find all Delaunay edges as demonstrated by (fig. 3a on the following page). The reason for this is that we do not use generalized expanding search circles, but only a fixed circle with the two points as its diameter. In the next section we will however give experimental and theoretical results showing that 2/3 of all edges can be found by applying T2.

We also note that all edges are usually found twice, once from each side - from a to b and from b to a. But of course, even if a is one of the closest neighbors to b, then b need not be one of the the closest neighbors to a.



(a)

Figure 2: *A sufficient test for pb_i is a Delaunay edge, with b_i as the i'th closest neighbor to p, is that the circle with pb_i as its diameter contains none of the i-1 closer neighbors to p.*



(a)

Figure 3: *An example that not all Delaunay edges are found by T2. In the triangle abc (with $\angle c > 90°$), theorem T2 will from a find ac, from c find ca and cb,and finally from b find bc. The edge ab is not found.*

Some experimental and theoretical results

Theorem T2 was implemented in two short Java programs to investigate three properties; 1) How many true Delaunay edges did it discover, 2) Was it only the closest neighbors that were used to find Delaunay edges, or did further away neighbors also contribute; and, finally: 3) How was the frequency of number of the edges found per point.

We tested these three properties with random uniform distribution of x-values and y-values independently for $n = 100, 1000, 10000, 100000$ and 500000. In the first program we tested against all the other $n - 1$ points in P, an $O(n^2)$ algorithm. In the second program we (only) tested against the k closest neighbors for each point, an $O(n)$ algorithm ($k = 5, 10, 20$ and 40). From table 1 we see that T2 determines 4 out of the average 6 neighbors per point[5].

	Avg. edges found per point				
n =	100	1000	10 000	100 000	500 000
All neighbors tested:	3,50	3,86	3,96	3,99	4.00
The 5 closest tested:	2,68	2,72	2,71	2,72	2,72
The 10 closest tested:	3,40	3,61	3,66	3,67	3,68
The 20 closest tested:	3,50	3,84	3,93	3,95	3,96
The 40 closest tested:	3,50	3,86	3,96	3,99	4,00

Table 1. *Average number of Delaunay edges found by theorem T2 per point as a function of n, the number points in P on randomly drawn datasets; and testing all, or only the* $5, 10, 20$ *or* 40 *closest neighbors.*

The reason that the number of edges found increases with n we interpret as a consequence of the ratio of interior points to points on the convex hull increases with n. Points on the convex hull have on the average only 4 edges in a full D(P) but interior points have 6 edges. It seems like a good idea to test at least the 40 closest neighbors, but that comes at steep execution time penalty, testing (on a Dell E2400 laptop with a Intel Core 2 Duo 1,4 GHz CPU) the 5 closest neighbors does 105000 points per second, 10 closest does 65000, and testing the 40 closest neighbors manages only 14000 points per second. Testing all distances took approx.8 hours for 500000 points, or 17 points per second.

In Table 2 we plotted the frequency by which the i'th closest neighbor b_i were found as a Delaunay edge pb_i , while we in Table 3 give the distribution of how many neighbors T2 found per point.

We can make a model for the results in Tables 1 and 2. If we inspect Figur 2, we observe that the area of circle C is 4 times as large as circle S since C has twice its radius. The number of edges N_k we get if we inspect all k nearest neighbors is then the sum of the probabilities that for $i = 1, 2, .., k$ that the $i - 1$ nearest neighbors are outside the circle Spb_i, which has the probability $(\frac{3}{4})^{i-1}$:

$$N_k = \sum_{i=1}^{k} (\frac{3}{4})^{i-1} = \frac{1 - (\frac{3}{4})^k}{1 - \frac{3}{4}} = 4, \; k \to \infty \qquad (1)$$

The test for *which* of the closest neighbors are found by T2 as Delaunay edges, we only give results for $n = 100, 1000$ and 500000 from program 1 where we tested all distances. The results from Program 2, testing only the 40 closest distances and for other values of n, are very similar.

% of the i'th nearest neighbor found as edge by T2				
i	$n = 100$	$n = 1000$	$n = 500000$	Predicted by: $(\frac{3}{4})^{(i-1)}$
1	100.00	100.00	100.00	100.00
2	72.00	75.50	75.00	75.00
3	54.00	55.30	56.13	56.25
4	43.00	43.20	42.29	42.19
5	27.00	30.80	31.71	31.64
6	16.00	23.20	23.85	23.73
7	8.00	15.60	17.72	17.80
8	10.00	11.10	13.29	13.35
9	7.00	8.30	9.91	10.01
10	4.00	6.20	7.50	7.51
11	3.00	5.00	5.55	5.63
..	..	..	..	..
25		0.10	0.10	0.10
26			0.07	0.08
27			0.05	0.06
28			0.04	0.04
29			0.03	0.03
30			0.02	0.02
31			0.02	0.02

Table 2. *The percentage of the i'th closest neighbor b_i from a point p found as a Delaunay edge b_ip by theorem T2 empirically for $n = 100, 1000$ and 500000, and compared with the model prediction of $(\frac{3}{4})^{i-1}$. Some lines are left out for obvious reasons.*

The reason that more neighbors are found for $n = 500000$ than for $n = 100$ and $n = 1000$, we also here interpret by a higher proportion of interior point as compared with points on the convex hull.

If we compare and contrast Tables 2 and 3, we see that many of the closer points are not found as Delaunay edges by Theorem T2, but that some far away points can be included as edges. We also see in table 3 that for a few points, only one edge is found by T2, i.e. its closest neighbor . We find no easy model for the results in table 3. The result of finding t edges to a point, $t > 1$, can be the result of many different selections of $t-1$ of these edges from the k'th nearest neighbors, each with different probabilities.

3 The connected graph theorem

We are now able to prove a theorem on the graph formed by T2.

Theorem T3. *The undirected graph G constructed by T2 is connected if we in T2 consider the at least $n/2$ closest neighbors to all points.*

Proof. By contradiction, assume T3 false. Then for some dataset P, G can be partitioned in two (or more) unconnected sub graphs A, B,... Let A be the sub

Number of neighbors found	%-age distribution of edges found per point by T2		
	$n = 100$	$n = 1000$	$n = 500000$
1	1.0006	0.400	0.027
2	16.0000	7.500	4.488
3	32.0000	29.300	28.137
4	35.0000	38.500	39.095
5	15.0000	18.400	21.003
6	1.0000	4.700	6.012
7		1.100	1.085
8		0.100	0.138
9			0.014
10			0.001

Table 3. *The distribution of the number of edges found per point in P by theorem T2 by testing all distances.*

graph with the fewest members, say m. Measure the distances from all points in A to the rest of G (G-A). Name a the point in A with the shortest distance to a point in G-A. Call that point b. Consider the circle S_{ab} with ab as diameter. S_{ab} can contain no other point; not c from A since then cb would be an even shorter distance; and likewise not a point d from G-A, since then ad would also be shorter than ab. Let A have t members, at most $n/2$ by assumption. Then in theorem T2, when looking at the k closest neighbors from a, then at least as the t'th closest point to a would have discovered ab as a new Delaunay edge since $t < n/2$. This is a contradiction since A and some part of G-A would then be connected. □

A note on the demand of checking the $n/2$, closest neighbors. It easy to construct examples where that is necessary, say two clusters, each with $n/2$ points, far apart in the plane. In practice, however, a constant of inspecting the 20 to 40 closest neighbors should suffice.

4 Two almost full parallel algorithms for Delaunay triangulation.

The work that inspired these results were attempts to parallelize Delaunay triangulation on a computer with a multicore CPU (Intel Core 2 Quad Q9550) with 4 cores and a massive parallel GPU of the SIMT(Single Instruction Multiple Thread)–type with 240 processing elements (NVIDA GTX285) [16].

The starting point was an efficient Delaunay triangulation algorithm [3] from 1984 for a single CPU. The first step in that algorithm was to find the convex hull. This can be parallelized, but is so fast compared with the rest of the triangulation, that we didn't consider it in a first version. As a second step, all data points were sorted (bucket-sort) into a grid of boxes with an expected constant, say on the average 4 data points per box (Fig. 4a on page 9). This speeds up access to all points close to a given point, in the sense that the number of points we have to test for a certain property, say included in a circle, is reduced dramatically from n points down to a small, constant number. This improved the rest of the algorithm from a $O(n^2)$ algorithm to an expected $O(n)$ algorithm. Cached memory has in the last 25 years or so also increased the need for localized access [10, 17].

The sorting phase is also kept as a sequential step and is not parallelized, also here because it is much faster than the triangulation. In the current implementations it is one or two walk through of the dataset (only one if we have the maximum and minimum x- and y-values before we start).

The rest of the algorithm, the triangulation, is definitely the most time consuming stage, We made first parallel algorithms for the multicore CPU. It was a straight forward exercise on 4 cores with a speedup of 3.5 on the triangle finding stage, and a speedup of 2.6 for the algorithm as a whole. Our second attempt, to parallelize the algorithm on the GPU was, however, not successfull. The triangulation phase was totally rewritten in C and run on the CUDA platform with assumed efficient use of the different memories (shared, global and constant) of the GPU. The C-code then ended up 3 times longer than the original Java code when we made our best effort to use the SIMT programming model.

The SIMT architecture is similar to SIMD (Single Instruction Multiple Data) vector architectures; each instruction is performed on multiple data elements. While SIMD instructions does not allow branching and expose the argument vector width to the programmer, SIMT instructions allow branching and specify the execution and branching behaviour of a single thread. For the SIMT architecture each thread executes one instruction on precisely one data element.

The SIMT-threads are grouped into batches of threads. Each batch contains a constant number of threads; CUDA uses batch sizes of 32. A SIMT function/module is executed across a set of such batches, each batch assigned to a subset of the input data elements. Every thread in the batch does execute the same instruction in parallell. When distinct threads in a batch cannot execute the same instructions (due to branching), those instructions must be executed sequentially in between the threads in the batch, speeding down the overall execution performance.

Actually, we got a significant *speeddown* compared to the sequential CPU implementation. Our analysis is that this is the result of the many branching points in the code. Allthough we were able to speed up searching and angle-calculation by doing it in parallell, most parts of the algorithm had to be performed sequentially. The result was that only 1 of 128 threads was running on every core most of the time (we used 128 threads per core), while the remaining 127 threads were waiting on various barriers. And since each processing element on a CUDA-card has a 4–6 times lower frequency (approximately 500–700 MHz) than a CPU core, the speeddown was in afterthought not surprising — we were not able to follow the SIMT programming paradigm

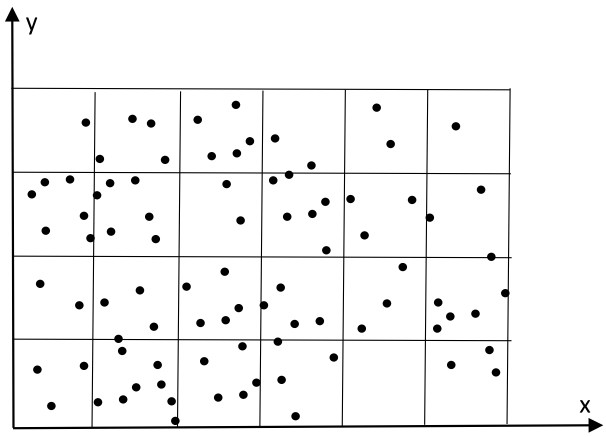
We will now, based on T1 and T2, sketch two new and efficient parallel algorithms for multicore CPUs. This would be unique Delaunay triangulations now that the co-circular case has been given a solution [4].

Algorithm 1 based on T1 (steps 3 and 4 performed in parallel):

1. Sort data points into a grid of boxes as in $fig.4$.

2. Find the convex hull of P.

3. For each point p_i, find a closest neighbor b_i, and hence a Delaunay edge p_ib_i.



(a)

Figure 4: *Bucket sorting of the dataset into boxes such that there is an expected small, constant number of points per box. This makes search for the k closest neighbors and a full Delaunay search for neighbors for each point local, an expected constant time operation - i.e. O(1) per point.*

4. With p_ib_i as a starting edge, find the other edges (and hence triangles) for point p_i counter clockwise round p_i, using an 'ordinary' Delaunay triangulation algorithm with expanding search circles.

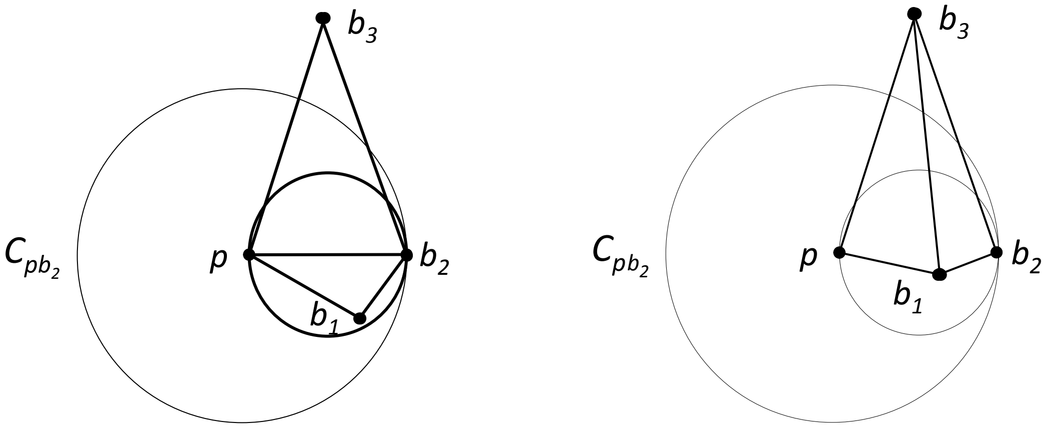
Algorithm 2 based on T2 (steps 3 and 4 performed in parallel):

1. Sort data points P into a grid of boxes as in $fig.4$.
2. Find the convex hull of P.
3. For each point p_i, find as many edges from p_i using Theorem T2.
4. Use a Delaunay triangulation algorithm for constraint Delaunay triangulation [6, 7, 8] to find the (on the average two) edges not found by T2,
5. Sort the points found counter clockwise round p_i.

5 A comment on theorem T2 and the k'th nearest neighbor problem

Theorem T2 gives a sufficient condition for determining if the i'th closest neighbor to a point p is a Delaunay edge pb_i - i.e. that all closer neighbors to p are outside the circle S_{pb_i} with pb_i as its diameter. What happens if one of the points is inside that circle, can that line pb_i still be a Delaunay edge? In Figures 5a and 5b, we look at the simple case of two sets of four points where the only difference is that we have moved b_1 closer to the line pb_2 in $fig5b$. We see that in one of the cases, pb_2 is a Delaunay edge, and in the other case it is not. We conclude then that the test in T2 is a sufficient, but not a necessary condition for pb_i to be a Delaunay edge,

Delaunay triangulations have been used for solving the k'th closest neighbor problem [2, 9, 11] to all or a subset of the points in P, while we here are doing the opposite. We can now see that all points are connected to their closest neighbor by



(a) The Circle C_{pb_2} with p_1 as an interior point in S_{pb_2}, well removed from pb_2. Then pb_2 is a Delaunay edge.

(b) The Circle C_{pb_2} with p_1 as an interior point in S_{pb_2}, close to pb_2. Then pb_2 is not a Delaunay edge.

Figure 5: *The Delaunay triangulation of the four points p, b_1, b_2, b_3 - testing if the second closest neighbor b_2 to p forms a Delaunay edge pb_2.*

Theorem T1, but that $fig5b$ illustrates that any of the k closer neighbors, $k > 1$, are not necessarily represented by an edge in D(P).

6 Conclusions and futher work

This paper introduces new properties of Delaunay triangulations and then a simpler and faster way of finding many of the Delaunay edges. We have identified a legal starting point for a parallel triangulation at any point in a dataset by finding one or more of its closest neighbors.

With Theorem T2 we have identified on the average 2/3 of all Delaunay based on distances. Is there another fast algorithm to find the remaining edges, perhaps based on the fact of how we found the first 2/3 of the edges — a challanging question.

We will work further with the multicore CPU algorithm to parallelize all its parts - it is currently doing 1 million triangles per second on a Quad core CPU. We will also like to prove these results in 3D space, which at the moment seems a straight forward exercise.

7 Acknowledgment

We would like to thank Stein Krogdahl and Morten Dæhlen for their comments on an earlier version of this manuscript.

References

[1] B. Delaunay, *Sur la sphère vide*, Izvestia Akademia Nauk SSSR, VII Seria, Otdelenie Matematicheskii i Estestvennyka Nauk, vol. 7, 1934, p.793–800.

[2] Mourad Khayati and Jalel Akaichi, *Incremental approach for Continuous k-Nearest Neighbours queries on road*, Iternational Journal of Intelligent

Information and Database Systems, Volume 2, Number 2, Pages: 204 - 221 , 2008

[3] Arne Maus, *Delaunay Triangulation and the Convex Hull of N Points in Expected Linear Time*, 1984, BIT, vol.24, p.151–163

[4] Dyken and Floater, *Preferred Directions for Resolving the Non-uniqueness of Delaunay Triangulations*, CGTA: Computational Geometry: Theory and Applications, vol.34, n. 2, p.96–101, 2006

[5] de Berg, Mark; Otfried Cheong, Marc van Kreveld, Mark Overmars (2008). *Computational Geometry: Algorithms and Applications*. Springer-Verlag. ISBN 978-3-540-77973-5., http://www.cs.uu.nl/geobook/interpolation.pdf.

[6] Wu, LiXin(2008), *Integral ear elimination and virtual point-based updating algorithms for constrained Delaunay TIN*. Science in China Series E Technological Sciences 51,Pages 135-144 Issue Volume 51, Supplement 1 / April, 2008

[7] Edelsbrunner and Shah, *Incremental Topological Flipping Works for Regular Triangulations*, ALGRTHMICA: Algorithmica, vol.15, 1996

[8] Øyvind Hjelle and Morten Dæhlen, *Triangulations and Applications*, Springer, 2006, Mathematics and Visualization, ISBN:978-3-540-33260-2

[9] Kieran F. Mulchrone, *Application of Delaunay triangulation to the nearest neighbour method of strain analysis* Journal of Structural Geology, Volume 25, Issue 5, Pages 689-702, May 2003,

[10] Arne Maus and Stein Gjessing, *A Model for the Effect of Caching on Algorithmic Efficiency in Radix based Sorting,* The Second International Conference on Software Engineering Advances, ICSEA 25. France, 2007

[11] Leonidas J. Guibas, Donald E. Knuth and Micha Sharir *Randomized incremental construction of Delaunay and Voronoi diagrams*

Journal Algorithmica Publisher Springer New York ISSN 0178-4617 (Print) 1432-0541 (Online) Pages 381-413 Issue Volume 7, Numbers 1-6, June, 1992

[12] Gabriel, K.R., and R.R. Sokal *A new statistical approach to geographic variation analysis.* Systematic Zoology 18:259-278. 1969

[13] *Wikipedia: Delaunay triangulation*

http://en.wikipedia.org/wiki/Delaunay_triangulation

[14] *Wikipedia: Gabliel Graph*

http://en.wikipedia.org/wiki/Gabriel_graph

[15] *Wikipedia: Polygon mesh*

http://en.wikipedia.org/wiki/Polygon_mesh

[16] Jon Moen Drange, *Parallell Delaunay-triangulering i Java og CUDA*. Master Thesis (in Norwegian), Dept. of Informatics, Univ. Of Oslo, May 2010

[17] *NVIDIA CUDA C Programming Guide Version 3.2, chapter 4.1, 4.2*
http://www.nvidia.com