Fast Reconstruction of Delaunay Triangulations

Christian Sohler
Heinz Nixdorf Institute and Dept. of Math. and Computer Science,
University of Paderborn, D-33095 Paderborn, Germany
e-mail: csohler@uni-paderborn.de

July 8, 1999

Abstract

We present a new $O(n)$ algorithm to compute good orders for the point set of a Delaunay triangulation of n points in the plane. Such a good order makes reconstruction in $O(n)$ time with a simple algorithm possible. In contrast to the algorithm of Snoeyink and van Kreveld [1], which is based on independent sets, our algorithm uses a breadth first search (BFS) to obtain these orders. Both approaches construct such orders by repeatedly removing a constant fraction of vertices from the current triangulation. The advantage of the BFS approach is that we can give significantly better bounds on the fraction of removed points in a phase of the algorithm. We can prove that a single phase of our algorithm removes at least $\frac{1}{3}$ of the points, even if we restrict the degree of the points (at the time they are removed) to 6. We implemented and compared both algorithms. Our algorithms is slightly faster and achieves about 15% better vertex data compression when using a simple variable length code to encode the differences between two consecutive vertices of the given order.

1 Introduction

The Delaunay triangulation of a planar point set is an important structure in computational geometry and its application areas. It is, for example, used in geographic information systems (GIS) to represent terrain models (using an additional coordinate for the height of a point). These models often consist of several millions of triangles. Suppose you want to transmit such models from a server to a client over a slow communication link. At first glance, there are two possibilities:

a) Transmit the point set; since a Delaunay triangulation is a canonical structure, it can be computed from the point set. However, this requires $\Theta(n \log n)$ time, in general [3] [4].

b) Transmit all adjacency information and the point set. This requires either non-trivial geometry compression algorithms [5] [6] [7] or results in a huge transmission overhead.

Recently, Snoeyink and van Kreveld [1] introduced a third possibility, which is also considered in this paper: compute a special order for the point set. If the point set arrives in this order at the client, the Delaunay triangulation can be recomputed in $O(n)$ time. Provided the server has already computed the Delaunay triangulation, it is possible to compute such an order in $O(n)$ time as shown in [1]. The order-computation-algorithm in [1] has $O(\log n)$ phases and in each phase an independent set I of $\Omega(n')$ points is removed from the current triangulation, where n' denotes the number of points in the current triangulation. Then the created holes are retriangulated and each removed point gets a reference pointer to its enclosing triangle. This is followed by a (canonical) DFS or BFS on the triangles of the current triangulation. The order of the point set is then given by the time the enclosing triangles for the points in I are visited.

The reconstruction algorithm also has $O(\log n)$ phases corresponding to the phases of the order-computation-algorithm. At the beginning of each phase, the points can be located by the same canonical DFS or BFS as in the order-computation-algorithm. This way, the point location for I can be done in $O(n')$ time.

Our method relies on a BFS on the graph of the triangulation. We do not require the points removed in a single phase to be an independent set. The advantage is, that we can remove more points from the triangulation in a single phase.

If we transmit the point set of a triangulation we also want to compress the vertex coordinates. This is often done [5] [6] [7] [8] by rounding the coordinates to k-bit integers and encoding the differences between two consecutive vertices by a variable length code. Our method ensures that two consecutive points in the computed order are usually close to eachother. We compared our algorithm to the one in [1] with respect to the compression results that were achieved with a simple variable length code. The result is that our algorithm needs about 15% less memory to store or transmit a compressed point set. We also compared the running times of both algorithms experimentally. Here our algorithms perform slightly better (about 20% for order computation and 10% for reconstruction).

The disadvantage of our approach is that it gives up parallel insertion of the points, which is possible when the points are independent.

Nevertheless, it is well-suited for progressive transmission since we can insert each point immediately when it arrives at the client. The transmitted points at the end of each phase give an approximation of the final triangulation. Within a phase the BFS-order guarantees a 'nice' reconstruction.

In section 2 and 3 we explain the algorithms to compute good orders and to reconstruct the triangulations. In section 4 we analyze the algorithms. Then we give experimental results.

2 Computing Good Orders

To compute a good order we proceed similar to [1] and [2] by repeatedly removing a constant fraction of vertices from the current triangulation and retriangulating the created holes until only a constant number of points remain. We describe in this section how a single batch is computed (this corresponds to a single phase of the algorithm). To simplify the analysis we will from now on assume that all triangulations have a convex hull defined by exactly 3 extreme points (this can be easily achieved by adding two dummy points). These points may not be removed in the algorithm of this section.

First we have to introduce a unique order for the vertices of a triangulation τ. Let τ be represented by a planar map, i. e. the edges are ordered clockwise around the vertices. Consider a breadth first search (BFS) of τ that starts at the leftmost vertex with the upper boundary edge. If a next edge is requested during the BFS we always choose the next edge clockwise around the current vertex. Then such a BFS is unique and we can order vertices and edges by the first time they are visited by that procedure.

At the beginning of a phase of the following algorithm we compute the reverse BFS order for the vertex set of the triangulation. Then we process all vertices in that order. We simply remove all vertices that have degree smaller than 7 at the time they are considered.

The computation of a batch works as follows:

```
ComputeBatch(τ)
    compute reverse BFS-order for the vertex set of τ
    for each vertex in the defined order
        if deg(v) < 7
            remove v and retriangulate
            put v on stack S
```

The order on the stack S and the algorithm in the next section make it possible to insert the computed batch in $O(n)$ time. We can compute a good order for the whole point set of the triangulation by iteratively applying the above algorithm. The explanation how this order can be used for fast reconstruction follows in the next chapter.

The running time $T(n)$ of the algorithm is $T(n) = T(n/c) + O(n) = O(n)$.

3 Reconstruction

Basically, the reconstruction algorithm works reverse to the algorithm from section two. Again we have $O(\log n)$ phases and in each phase we (sequentially) insert a batch of points into the current triangulation. In each phase we have to compute a BFS to insert the points of the batch. In contrast to the algorithm from section 2, we do *not* compute the BFS at the beginning of the algorithm and process all vertices in the computed order. We have to insert points during the BFS and these newly inserted points have influence on the BFS order. We therefore process the BFS tree level by level. For each level we reconstruct the neighborhood of the vertices of that level using the next points of the batch. Then we compute the next level of the BFS-tree. In this way, the BFS tree is adjusted at each new level. In fact, the BFS-tree we compute is the BFS-tree of the Delaunay triangulation *after* the insertion of the batch. We now give the algorithm in pseudocode. Please note that all **for each** loops must be performed in (unique) BFS order as defined in section 2, i.e. the triangles are visited in clockwise order around the current vertex v starting with the triangle adjacent to edge $e = (parent(v), v)$ of the BFS-tree. Let B the current batch and b its first vertex:

```
RestoreBatch()
    for each level l of the BFS-tree
        for each vertex v at level l of the BFS-tree
            for each triangle t incident to v
                if b is in circumcircle(t)
                    insert b
                    update triangle list
                    b = B.next()
        compute next level of the BFS-tree
```

During the execution of the above algorithm for batch B and triangulation τ we call a triangle *certified*, if it is a triangle of the Delaunay triangulation of $V \cup B$ where V is the point set of τ.

Now we can state the following lemma:

Lemma 1. After a triangle has been tested in line 5 of the above algorithm and was not destroyed (that is, b was not in $circumcircle(t)$) it is certified.

Lemma 2. After all vertices of a level of the BFS-tree have been processed, all incident triangles are certified. Thus the neighborhood of these vertices has correctly been reconstructed and we can proceed with our algorithm and compute the next level of the BFS-tree.

Proof: Follows immediately from Lemma 1.

A direct consequence of Lemma 2 is that no triangle has to be checked twice during a single phase of the algorithm.

Now we must take a closer look at the case when a vertex b is inserted. Since we know a triangle t that is destroyed by b, we can find all destroyed triangles by a traversal of the (destroyed) triangles starting at t. Since the degree of b is restricted to at most 6, we can insert the vertex in constant time. After the insertion, we have two new triangles adjacent to the current vertex. We can continue the **for each** loop with these two triangles (recall that the triangles are visited in clockwise order).

Theorem 1. Algorithm RestoreBatch() correctly inserts a batch of vertices resulting from the algorithm of section 2 into a triangulation also resulting from that algorithm.

Proof: By induction on the level of the BFS-tree. Level 0 (the root) was not removed and therefore is correctly reconstructed. Assume the triangulation has been correctly reconstructed up to level n. Then Lemma 2 implies that we can reconstruct level $n+1$ by processing all vertices of level n.

To complete this section, we will now give a proof of Lemma 1.

Proof: By contradiction. Let t be the first triangle that is tested, not destroyed, and that does not belong to the triangulation τ_{EOP} of the vertex set of the current triangulation *and* the batch (which is the triangulation at the end of the phase). Then either not the whole batch is inserted or t is destroyed by some vertex b that is still in the batch at the time t is tested. We first regard the second case.

At the time t is tested, b cannot be the first vertex of the batch, since otherwise t was destroyed. Thus there exists at least one other vertex b' on the batch that is inserted before b is inserted. Let v be the smallest vertex (in the order defined by a BFS in τ_{EOP}) adjacent to b in τ_{EOP}. Then v is also the smallest vertex of t, because t is destroyed when b is inserted and by the choice of t (if the edge between b and v is destroyed by later insertion of a vertex w, we can use the similar arguments with w to get a contradiction). Furthermore, v is the smallest vertex whose neighborhood has not been restored. Thus b' is also adjacent to v (otherwise, b would appear before b' on the batch which contradicts the choice of b').

We conclude, that b and b' are adjacent to v and that t appears before b' and b' before b in clockwise order around v.

Let u be the vertex of t that is further away from b in counter-clockwise order around v. Then the segment bu either crosses vb' or it crosses the edge $(parent(v), v)$ which belongs to a certified triangle (or it crosses some other certified triangle that is older than t) or b' is in the triangle uvb. All three cases cannot occur: if bu crosses vb', then b' is no longer adjacent to v after the insertion of b. If bu crosses another certified triangle that is older than t, we

have a contradiction to the choice of t. And if b' is in the triangle uvb, then uvb cannot be a triangle of the Delaunay triangulation. But uvb is a triangle of the triangulation, if the insertion of b destroys t. Thus we have found contradictions in all cases and t cannot be destroyed by another point b.

We still have to regard the other possibility that t is not destroyed but does not belong to τ_{EOP}, that is, not the whole batch of points is inserted. Let b be the vertex that destroys t (otherwise t appears in τ_{EOP}). Then there is another vertex b' before b in the batch because otherwise t would be destroyed when it is tested. Since b' is before b in the batch, it either destroys t or some smaller triangle (this follows using similar arguments as above). By choice of t all smaller triangles are certified, thus t is destroyed which is a contradiction. This completes the proof of lemma 1.

The running time of the algorithm is linear in the number of created triangles and thus $O(n)$.

Remark. When implementing the algorithm it is possible to substitute most of the incircle tests by 1 or 2 sideness tests. We did not use sideness tests in the pseudocode to keep the code as simple as possible.

4 Analysis

We first prove that for each triangulation τ with n vertices the algorithm from section 2 computes a batch of size $\Omega(n)$. Therefore let s be the number of vertices that remain in the triangulation after the batch has been computed and let b be the size of the batch. After the batch has been computed the overall degree of the remaining vertices is $6s - 12$ by Euler's formula. Each vertex that remains in the triangulation has an initial degree of at least 7 when it is considered by the algorithm. We regard the overall degree of all vertices that have not been removed from the triangulation. This overall degree can only be lowered when another vertex is removed. When retriangulating the created hole we observe that for two adjacent vertices always at least one of them is incident to a new edge. Thus the removal of a vertex can at most lower the overall degree of the remaining vertices by 3. The 3 boundary vertices of the convex hull are not removed and have degree at least 3. We put these observations together:

$$\begin{gathered} n = s + b \\ 7(s-3) + 3 \cdot 3 - 3b \leq 6s - 12 \end{gathered}$$

Substitution immediately yields:

$$b \geq \tfrac{1}{4}n$$

Thus each batch contains at least 25% of the vertices of the current triangulation. But we can still improve the analysis. If we consider a vertex v with degree 6 (or less) it first of all must be connected to its parent in the BFS tree. Then for every v except for the 3 boundary vertices there is at least one vertex p that is adjacent to v with p smaller than v (in the usual order). Furthermore, p has two neighbors that are adjacent to v. One of them is also smaller than v. The remaining 4 vertices adjacent to v can be larger than v. Thus the degree of the remaining vertices can only be lowered by 2 and hence:

$$b \geq \tfrac{1}{3}n.$$

The table below given the lower bounds on the batch size for different constants:

constant	6	7	8	9
batch size	$\frac{1}{3}n$	$\frac{2}{5}n$	$\frac{1}{2}n$	$\frac{1}{2}n$

Our experiments show that the batch size in practice is much bigger. If we restrict to vertices with a degree of at most 6, about 80% of the vertices (randomly distributed in the unit cube) were removed.

5 Implementation and Experimental Results

We implemented prototypes of both algorithm in C++. Geometric primitives are computed using double precision floating point arithmetic. We restricted ourselves to incircle tests to simplify the implementation. Our algorithm is basically implemented the way it is presented in this paper. The algorithm of Snoeyink and van Kreveld has been slightly changed. Let τ be the triangulation resulting from the removal of an independent set I. Then the points in I are sorted with respect to the time a BFS on τ reaches the *first triangle that is destroyed by a point* when it is inserted. This way (and using some mark bits) we avoid unnecessary incircle tests (about 15-20%) during reconstruction. All inputs are point sets randomly distributed in the unit cube. With algorithm 1 we always refer to the algorithm by Snoeyink and van Kreveld [1].

5.1 Compression of Vertex Coordinates

We compare the effectiveness of the compression of the sequence of points computed by either algorithm. Therefore we round the coordinates to 16-bit integers and encode the position of each vertex relative to the position of its predecessor in the computed order. That is, we compute the difference between these two vertices and encode this difference using variable length code. We process $x-$ and $y-$coordinates separately. Our code consists of two tag bits, a sign bit, and a variable number (depending on the tag) of bits that encode the given difference. The following table describes the code we used:

Tag	encodable difference
00	-63 to 63
01	-511 to 511
10	-4095 to 4095
11	<-4095 or >4095

Table 1: A simple variable length code

We encoded the sequences of points computed by either algorithm using this variable length code. It turns out that our algorithm needs roughly 4 bits per vertex less than the one by Snoeyink and van Kreveld. The following table gives our experimental results. The initial distribution of the point set was also uniform in the unit square.

#vertices	bits/vertex Algorithm 1 [1]	bits/vertex Algorithm 2
5000	31.85	27.49
10000	30.74	26.50
20000	29.62	25.45
40000	28.47	24.39
80000	27.34	23.42

Table 2: Compression results

5.2 Running Time

We compared our implementation of both algorithms. First of all note that both algorithms are fastest (on our inputs), if the degree of the removed vertices is as low as possible. Therefore, both remove only vertices with degree at most 6. The algorithm presented in this paper performs slightly (about 20% for construction and 10% for reconstruction) better than [1]. We also counted the number of incircle tests done by both algorithms. If we want to reduce the number of incircle tests the algorithm of Snoeyink and van Kreveld performs best, if the degree of the removed vertices is restricted to 9. Then it needs roughly 4% more incircle tests than our algorithm does. both algorithm use less than 10 incircle tests (recall that no sideness tests are used) for each inserted point on the average. In the table below we refer with 'CO' to the procedure that computes a well ordered point set from an existing Delaunay triangulation and with 'Rec' to the computation of a Delaunay triangulation from such an ordered point set. We also compared both algorithms with the $O(n^{\frac{4}{3}})$ time (on random point sets) algorithm by Mücke, Saias, and Zhu [9]. We refer with 'MSZ' to this algorithm in the table below. Time is measured in seconds on a SUN SPARCstation 4 with 110 MHz.

#vertices	MSZ [9]	Alg1-CO [1]:	Alg 2-CO:	Alg 1-Rec [1]:	Alg 2-Rec:
5000	4	2	2	2	1
10000	9	5	4	2	2
20000	22	9	7	5	5
40000	52	18	14	11	10
80000	2:04	36	30	23	19

Table 3: Comparison of the two algorithms

#vertices	#incircle-tests -Algorithm 1 [1]	#incircle-tests -Algorithm 2
5000	50K	48K
10000	100K	96K
20000	199K	192K
40000	399K	383K
80000	799K	768K

Table 4: Number of incircle-tests used by both algorithms

6 Conclusion

We presented a new method to compute orders for a Delaunay triangulation that allow reconstruction of the triangulation in linear time. We can guarantee that in a single phase of our algorithm $\frac{1}{3}$ of the points are removed (when the degree of the removed points is restricted to at most 6). This is a significant improvement to the algorithm of Snoeyink and van Kreveld (which guarantees that $\frac{1}{10}$ of the points of degree 6 can always be removed). Our algorithm is well-suited for

data compression and progressive transmission of Delaunay triangulations. It provides a useful alternative to the algorithm in [1].

References

[1] J. Snoeyink, M. van Kreveld. Linear-time reconstruction of Delaunay triangulations with applications, European Symposium on Algorithms, 1997

[2] M. Denny, C. Sohler. Encoding a triangulation as a permutation of its point set. 9th Canadian Conference on Computational Geometry, 1997.

[3] L. J. Guibas, D. E. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. Algorithmica, 7:381-413, 1992.

[4] J.-D. Boissonnat and M. Teillaud. On the randomized construction of the Delaunay tree. Theoret. Comput. Sci. , 112:339-354, 1993.

[5] J. Rossignac. Georgia Institute of Technology, Technical Report: GIT-GVU-98-35, October 10, 1998.

[6] H. Hoppe. Progressive Meshes. Proc. SIGGRAPH '96, pages 99-108. ACM SIGGRAPH, 1996.

[7] G. Taubin, J. Rossignac. Geometric Compression through topological surgery. ACM Trans. on Graphics, 17(2):84-115, 1998.

[8] M. Deering. Geometry Compression. Computer Graphics (Proc. SIGGRAPH), pages 13-20, 1995.

[9] E.P. Mücke I. Saias, and B. Zhu. Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulations. Proc. 12th Annu. ACM Sympos. Comput. Geom., pages 274-283, 1996.