Encoding a triangulation as a permutation of its point set

Markus O. Denny
FB 14 Informatik
Universität des Saarlandes
D-66123 Saarbrücken
FAX:+49-681-302-5576
mdenny@cs.uni-saarland.de

Christian A. Sohler
FB 14 Informatik
Universität des Saarlandes
D-66123 Saarbrücken
FAX:+49-681-302-5576
csohler@cs.uni-saarland.de

Abstract

We present algorithms that given a straight edge triangulation of n points in the plane encode a triangulation as a permutation of the points. A first algorithm, of rather theoretical interest, realizes the encoding and decoding in $O(n)$ time. We also present a more practical algorithm consuming $O(n \log n)$ time. As a byproduct of this work we get a new upper bound on the number of triangulations of planar point sets of at most $2^{8.2n+O(\log n)}$.

1 Introduction

It is well known that the number of triangulations of n points in the plane is bounded from above by $2^{O(n)}$ [1]. Since there are approximately $2^{n \ln n}$ different permutations of a point set of size n, it is obvious that it is possible to encode a triangulation as a permutation of the input data.

In the case of *canonical* geometric structures related to Delaunay triangulations Snoeyink and van Kreveld [2] already introduced a linear time algorithm and listed a number of advantages. The graphs are still readable by many other applications and transmission overhead is reduced.

However our approach is a more general one. We show that it is possible to efficiently encode *any* straight edge triangulation as a permutation of its point set. Inspired by the planar point location method of Kirkpatrick [3] we present an algorithm that encodes and decodes every triangulation of sufficiently large size in $O(n)$ time. Since this algorithm only works for relatively large triangulations we also present a more practical approach based on the sweep line paradigm which encodes and decodes a triangulation in $O(n \log n)$ time. From this algorithm we also obtain a new upper bound on the number of straight line triangulations.

Let T denote the triangulation under consideration given as a set V of n points together with the set E of the corresponding edges. We assume w.l.o.g. that it is always possible to add two points to the triangulation such that the new convex hull forms a triangle around the original triangulation (for instance the virtual points $(x_{max}+1, \infty)$ and $(x_{max}+1, -\infty)$ will suffice our conditions where x_{max} denotes the maximum x-coordinate of all points in V).

2 Encoding a triangulation as a permutation

At first we provide generic algorithms for encoding and decoding.

Each generic algorithm consists of $O(\log n)$ phases. In the first part of each phase of the encoding algorithm a constant fraction of points with low degree is removed from the current triangulation and put into an initially empty set W. (For the sake of simplicity we never remove any of the 3 points of the convex hull of V.) Next we retriangulate the created holes retrieving a new triangulation T'. In the second part we encode the information needed to reconstruct the original triangulation from T' into a permutation of the points of W.

The decoding algorithm works in a reverse manner. Having read a bunch of points we insert them in the right place into the current triangulation T' meanwhile computing their permutation. With the corresponding information we can now determine the appearance of the former triangulation and finally restore the original triangulation.

Part 1 of *generic* Encoding:

- pick a constant fraction W of n points with low degree
- remove W from the triangulation
- retriangulate

PART 2:

- compute and output the permutation of W

PART 1 OF *generic* DECODING:

- read the next permutation of points
- find the right places to insert them
- regain the information from the permutation

PART 2:

- delete the superfluous edges and retriangulate

3 A linear time algorithm

3.1 The algorithm

We now specify how a phase of the linear time algorithm looks like:

PART 1 OF ENCODING:

- compute a maximal independent set I of vertices with degree at most 6
- remove I from the triangulation
- retriangulate

PART 2:

- visit all triangles of the new triangulation in a canonical way (DFS, BFS) to order I (cf. [2])
- partition I into sets A_j of constant size
- store the information needed for the reconstruction by permutating each of these A_j
- Output A_j

PART 1 OF DECODING:

- read I from the input
- visit all triangles in the same manner as in part 2 of ENCODING to order I and to determine the enclosing triangle for each $v \in I$ (see figure 1.a)
- obtain the information needed to reconstruct the triangulation from the permutation

PART 2:

- use the information gained in part 1 to restore the triangulation (see figure 1.b-d)

3.2 Correctness

First of all we show that we can compute the size of I in each phase of DECODING and also the number of points that have been removed in part 2 of ENCODING. But this proves to be easy since we know n and we can simply restrict the size of the maximal independent set I in ENCODING to a constant fraction of V and thus immediately determine the sizes of I and the A_j's. W.l.o.g. we can assume an order on the input to ensure the same enumeration of the triangles for both ENCODING and DECODING induced by DFS or BFS respectively (e.g. ccw-ordering of the edges).

It is easy to verify that the algorithm will always terminate because in each phase the number of points is always reduced by a constant fraction.

3.3 Running time

To obtain a maximal independent set I we use the greedy algorithm presented by Snoeyink and van Kreveld [2] with the difference that our maximal allowed degree d is 6. The linear time algorithm removes at first all points with degree at least $d+1$. Then the point with lowest degree is put into the set I and its neighbours are deleted. This procedure is repeated until no point remains. Analyzing the behaviour of the algorithm leads to a linear program the optimum of which turns out to be $\frac{n}{10}$ for the minimal size of I. The deletion of the points and the retriangulation can be done on the fly. This gives a running time of $O(n)$ for the first phase of ENCODING.

In the second phase we can establish an ordering of the points in linear time since we know which new triangle contains the deleted point (cf. [2]). Since each A_j has constant size the running time for both phases is $O(n)$. Thus the overall running time for ENCODING as well as DECODING is $O(n)$ since the DECODING is essentially the reverse of the ENCODING.

3.4 How reconstruction works

Consider a triangulation T. If a point p with degree d is removed the originated hole forms a polygon. The triangulation of this polygon is dual to a tree b with the following properties (see figure 2). The root of b (representing the p-enclosing triangle) has degree at most 3, the inner nodes have degree at most 2 and the sum of all edges is $d-3$. The number $h(d)$ of such trees is given by:
$h(d) = \sum_{i_j} \binom{3}{i_1}\binom{2i_1}{i_2}\cdots\binom{2i_{d-4}}{i_{d-3}}$ with $\sum_j i_j = d-3$.

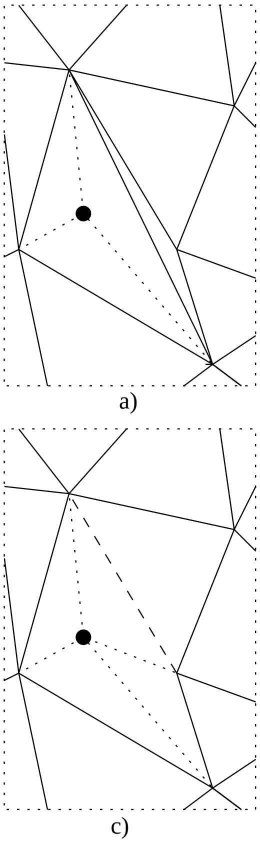

Figure 1: Reconstruction of a triangulation: a) determine the enclosing triangle and draw the first three edges (dotted lines) b)-d) According to the dual tree (see figure 2) choose the next edges to be flipped (dashed lines)

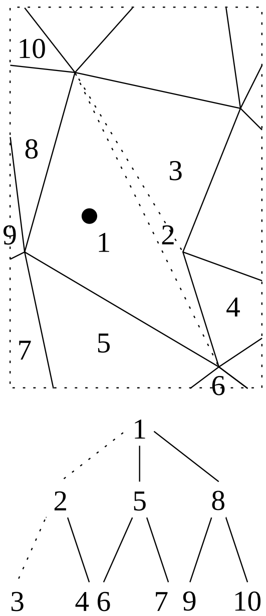

Figure 2: The dual tree (dotted lines) which describes the retriangulation to be performed

3.5 Analysis

Since we limited the degree of each removed vertex to 6, we get exactly $\Sigma_{3\leq i\leq 6} h(i) = 41$ possibilities to insert a point. Thus we need $\lceil n \log_2(41)\rceil$ bits for the encoding. The size of the independent set has to be large enough to encode this information. Since $109! > 41^{109}$ it suffices to have at least 109 points to build a block. Therefore the triangulation must have at least $10 * 109 = 1090$ points at the beginning of a phase. If during the course of the algorithm we arrive at a triangulation with fewer points, we have to use another algorithm to encode this triangulation, e.g. the one stated below.

4 An $O(n \log n)$ time algorithm

4.1 A sweep line algorithm

The algorithm we present in this section is based on the sweep line paradigm. We show that it is possible to remove a much bigger fraction of vertices with minimum degree 6 if we do not need an independent set. In a precomputing step we sort the point set of the triangulation. Now we specify how a phase of the second algorithm looks like.

Part 1 of Encoding:

- initialize the set of removed nodes I with $\oslash$
- sweep over the triangulation from left to right and stop at each vertex v
- if $deg(v) \leq 6$ then remove v from the triangulation, retriangulate and add v to I

Part 2 of Encoding:

- store the information needed for reconstruction by permutating I
- output I

Part 1 of Decoding:

- read I from input
- sort I lexicographically
- sweep over the triangulation from right to left and stop at each $v \in I$
- search the enclosing triangle
- retrieve the information needed to reconstruct the triangulation from the permutation

Part 2 of Decoding:

- use the information gained in Part 1 to insert v

4.2 Correctness

We sweep from right to left in the Decoding algorithm. Thus Decoding does exactly the same as Encoding but in reversed direction. We use common sweep line techniques to maintain an order of the triangles that intersect the sweep line. Using this order, we can easily find the enclosing triangle. All other steps are either already discussed or trivial.

4.3 Running time

We have to spend $O(n \log n)$ time for sorting. All steps of part one and two need a constant amount of time except the search for the enclosing triangle which needs $O(\log n)$ time for each vertex and the sorting of I which needs $O(n \log n)$ time. To maintain the lexicographical order during Decoding we also need $O(\log n)$ time for each inserted point. Overall, we have a running time of $O(n \log n)$ for Encoding and Decoding.

4.4 Analysis

As one can easily see the degrees of the three vertices of the convex hull sum up to at least 10 (if T consists of more than 4 vertices). Let n denote $|V|$, n' the vertices that remain after the sweep line phase (excluding the 3 points of the convex hull), and r the number of removed vertices. Each point at the left of the sweep line must have at first a minimal degree of 7 since it was not removed. For each removed vertex the total degree of all vertices on the left of the sweep line decreases at most by 2 (see figure 3).

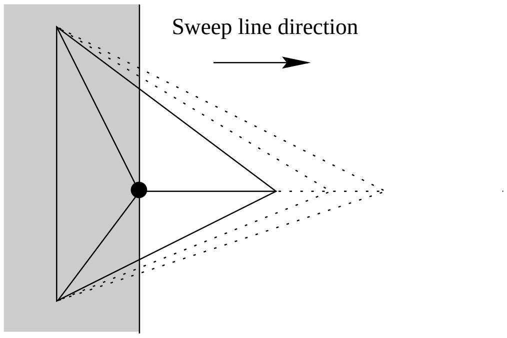

Figure 3: a worst case situation

At the end of each phase the remaining $n'+3$ points form a complete triangulation. Thus the following inequalities describing the sum of degrees hold:

$$\begin{aligned} 7n' - 2r + 10 &\leq 6(n'+3) - 12, \text{ and hence} \\ n' &\leq 2r - 4 \end{aligned}$$

By definition n' and r sum up to $n-3$. That yields the desired result of $r \geq \lfloor \frac{n}{3} \rfloor$ immediately. From the analysis of the first algorithm it follows that at least 109 points are necessary to encode the information for reconstruction. This makes it possible to execute a phase of the second algorithm for triangulations with at least 326 points. As soon as fewer than 326 points remain we cannot store enough information in the permutation. Provided a sufficiently large triangulation we encode this information in previously computed permutations. Overall, we have to store $\lceil n \log_2 41 \rceil$ bits. Thus we need at least approx. 750 points to store all the information.

4.5 A more practical point of view

If we had two more bits for each vertex we could encode any triangulation independent of the size of its point set. A practical way to get these two bits is to translate all points to reside in the upper right quadrant. The two additional bits for each point can now be stored as the sign of the coordinates. We now have to encode $\lceil n((\log_2 41) - 2) \rceil$ bits resulting in a minimal size of approx. 190 points for the triangulation.

5 New upper bounds

5.1 Idea

To derive an upper bound on the number of triangulations we slightly change the algorithm presented in the previous section. Instead of coding the information needed for reconstruction into a permutation we now encode it as a binary string $\omega(T)$. If every possible encoding string has length $f(n)$, then clearly V cannot have more than $2^{f(n)}$ different triangulations.

When encoding with a binary string, we loose the implicit information about the insertion order that was stored in the permutation. Thus we need the additional information which points were removed in each phase of the algorithm.

5.2 Analysis

For the sake of simplicity we assume that the output is ordered such that the first part of $\omega(T)$ represents the encoding for the local changes of each vertex at the time of its deletion. Let $\varsigma(E)$ denote this part of $\omega(T)$. The rest of the binary string $\sigma(V)$ contains the information which vertices were removed at each part. Thus we have:

$\omega(T) = \varsigma(E)\sigma(V)$

$\sigma(V') = \varsigma(W)\sigma(V' \setminus W)$

If $|V'| = 3$ there is only one possible triangulation and we encode it as the empty string. To encode W, we proceed as follows (cf. [4]). As a first step we use the characteristic vector representation for $\varsigma(W)$. That's a binary string s of length $|V'|$ with $s_i = 1$iff $v_i \in V'$. This can then be slightly improved by using a better encoding for $\varsigma(W)$. Since W is a subset of size exactly $\lfloor \frac{n}{3} \rfloor$ there are only $b(n) = \binom{n}{\lceil \frac{2n}{3} \rceil}$ choices for W. Hence W can be encoded by a string of length $\log_2 b(n)$. Since $\binom{n}{\alpha} \leq 2^{nH(\alpha)}$, where $H(x) = -x \log_2 x - (1-x)\log_2(1-x)$ we get that W can be encoded by a binary string of length $H(1/3)n \leq 0.9183n$ The impact of this fact will be stated in the following theorem.

Theorem

The number of triangulations is bounded from above by $2^{f(n+2)}$ *with* $f(n) \leq (\log_2(41) + 3\, H(\frac{1}{3}))n \leq 8.12\, n + O(\log n)$.

Taking into account that every crossing free subgraph G of K^n can be encoded in the same way as a triangulation in combination with a binary vector of size $3n$ coding the added edges, the following corollary is an direct consequence of the above Theorem.

Corollary 1:

The number of undirected crossing free subgraphs of the complete graph K^n *is bounded from above by* $2^{f(n+2)+3n}$.

We can also derive an upper bound on the number of directed crossing free subgraphs since every edge of the triangulation can either be deleted, replaced by a single one-directional arc or by two arcs for each direction.

Corollary 2:

The number of directed crossing free subgraphs of the complete graph K^n *is bounded from above by* $2^{f(n+2)+6n}$.

6 Acknowledgements

We would like to thank Raimund Seidel for the inspiring discussions and his support of this work.

References

[1] M. Atjai, V. Chvàtal, M. Newborn, E. Szemerédi, *Crossing-free Subgraphs,* Annals of Discrete Math. 12 (1982), 9-12

[2] J. Snoeyink, M. van Kreveld, *Good orders for incremental (re)constructions,* 13th Annual ACM Symposium on Computational Geometry 97

[3] D. Kirkpatrick, *Optimal search in planar subdivisions,* SIAM Journal on Computing, 12:28-35, 1983

[4] R. Seidel, *On the number of triangulations of planar point sets,* to appear in Combinatorica