Linear-time reconstruction of Delaunay triangulations with applications

Jack Snoeyink*
Dept. of Computer Science
University of British Columbia

Marc van Kreveld†
Dept. of Computer Science
Utrecht University

Abstract

Many of the computational geometers' favorite data structures are planar graphs, canonically determined by a set of geometric data, that take $\Theta(n \log n)$ time to compute. Examples include 2-d Delaunay triangulation, trapezoidations of segments, and constrained Voronoi diagrams, and 3-d convex hulls. Given such a structure, one can determine a permutation of the data in $O(n)$ time such that the data structure can be reconstructed from the permuted data in $O(n)$ time by a simple incremental algorithm.

As a consequence, one can permute a data file to "hide" a geometric structure, such as a terrian model based on the Delaunay triangulation of a set of sampled points, without disrupting other applications. One can even include "importance" in the ordering so the incremental reconstruction produces approximate terrain models as the data is read or received. For the Delaunay triangulation, we can also handle input in degenerate position, even though the data structures may no longer be canonically defined.

1 Introduction

Many of the favorite data structures in computational geometry can be represented as planar graphs that are canonically determined by n input data elements. Well-known examples include:

1. The Voronoi diagram of n point or line segment *sites* in the plane. This oft-reinvented data structure records proximity [1, 16]; it is the decomposition of the plane into maximally-connected regions that have the same set of sites as closest neighbors.
2. The Delaunay triangulation of n points. This dual of the Voronoi diagram joins two points with an edge if some circle touches those two sites and no other. The Delaunay is important in GIS and meshing [2, 18].

*Supported in part by grants from NSERC and Facet Decision Systems
†Supported in part by ESPRIT IV LTR project 21957 (CGAL).

3. The trapezoidation (vertical visibility map) of n line segments. Decomposing the plane into trapezoids—by extending vertical segments from each given segment endpoint to the segments above and below—imposes structure on the plane that can be exploited by intersection and other algorithms [17, 15].
4. The convex hull of n points in 3-space: The convex hull is the smallest convex set enclosing the points [17, 19]. Its surface can be represented as a planar graph.

Each of these have construction algorithms that run in optimal $\Theta(n \log n)$ time; the lower bounds are proved by reduction from sorting [14, 19]. Presorting the data in natural orders does not necessarily reduce the construction time. Seidel [22] has shown that after sorting by x-coordinate, computing a 3-d convex hull or 2-d Delaunay triangulation still requires $\Omega(n \log n)$ time. Djijev and Lingas [6] have shown the same for the Voronoi diagram.

We use ideas from Kirkpatrick's point location scheme [13] to find a permutation of the n data elements in linear time so that a simple incremental algorithm can reconstruct the desired planar graph from the permuted data in $O(n)$ time and space. We need only the following properties:

1. The graph must be canonical, depending on the data and not the order of the data.
2. There must always be a constant fraction of the data that can be deleted independently. (Kirkpatrick's "constant-degree" restriction is not necessary.)
3. It must be possible to reintroduce the deleted data by independent insertions.

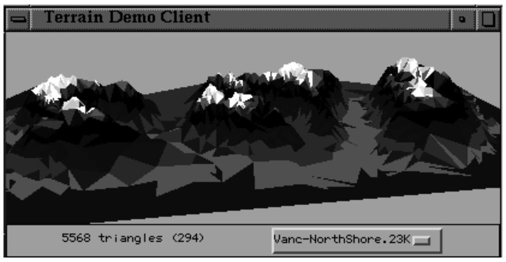

Fig. 1: TIN in Java implementation

This result is a natural combination of work in hierarchical representation of planar subdivisions [13] and incremental construction algorithms [10, 21]. It can be seen as a form of geometry compression and progressive expansion, which is of interest in computer graphics [5, 12]. We illustrate the idea with one application: compressing and transmitting terrain models. TIN (triangulated irregular network) terrain models [18] in geographic information systems (GIS) often fit a surface to points with elevations by taking the Delaunay triangulation of their 2-d projections and raising the triangles into 3-d.

Suppose that you wanted to distribute results of some GIS analysis on a terrain. You could encode and transmit maps or images, but geometric models may have more utility. Not only does rotating a terrain model gives a better 3-dimensional "feel," but a model can provide enough information to answer queries locally, such as steepest descent paths for fluid flow. However, sending all the triangles and their interconnections (say with a winged-edge data structure) requires five times more bytes than sending the point coordinates alone. Our result allows the structure to be encoded in a permutation of the points, giving three advantages:

1. The reconstruction algorithm is simple. The asymptotic constants are small for the construction of the permutation and reconstruction of the triangulation.
2. Reconstruction is incremental; after reading k data elements and spending $O(k)$ time and space, the algorithm can present partial results while the data is still arriving.
3. Storage and transmission overhead is reduced. A flat file of permuted data implicitly stores the planar graph structure, but is still readable by many other applications that are not concerned with this structure.
4. "Importance" can be incorporated into the permutation to support progressive transmission and reconstruction of geometric models by moving data that characterizes the final output early in the permutation. This can often be done without changing the asymptotic running time.

The next section outlines linear-time deconstruction and reconstruction algorithms for the Delaunay triangulation. These algorithms can choose methods for canonical traversal and computation of independent sets; Sections 3 and 4 describe our choices. Section 5 shows how to handle degenerate point sets, for which the Delaunay triangulation is not canonically defined. Section 6 compares reconstruction time to standard construction algorithms.

2 Permuting data for incremental reconstruction

The structures listed in the previous section have randomized incremental construction algorithms that run in expected $O(n \log n)$ time, where the expectation is taken over random choices in the algorithm for a worst-case input distribution. In each case, the expected cost of all changes to the data structure is linear, but the expected time to locate where the changes must occur is $\Theta(n \log n)$. Our permutation must ensure that the cost of change is linear not just in expectation, and that location is also linear in n. Fortunately, this is easily done.

In this section, we use the computation of the Delaunay triangulation of n points P in the plane to give an example of the steps of our approach. Our first algorithm produces a permutation of the data (in reverse order); our second takes this permutation and reconstructs the graph.

Permutation Construction Algorithm

P1. **Bound the problem.** Our terrain models come with bounding rectangles, so we begin with a triangulation of four points. One could instead extend the triangulation to the entire plane by adding a vertex at infinity and joining it to the convex hull vertices.

P2. **Independent set.** Find a constant fraction of vertices that form an independent set—no two vertices in the set are adjacent. Low degree is useful, but not mandatory. A simple, greedy algorithm in Section 4 guarantees an independent set of size at least 1/6 of all vertices, using vertices of degree at most nine.

P3. **Delete and locate data.** Delete the vertices of the independent set and repair the remaining triangulation, noting which new triangles now contain the deleted vertices. There will be at most one deleted vertex per triangle.

P4. **Traverse graph & output permutation.** Enumerate triangles by any canonical method: depth-first, breadth-first, etc. (Section 3 describes our favorite traversal, which requires no changes to the data structure.) Output the deleted vertices in the reverse of the order that their triangles are visited by the enumeration.

P5. **Output end of phase signal.** Our current applet inserts a special marker to signal the end of a phase. Section 3 describes other methods that don't use special markers, but use the points themselves.

P6. **Repeat from P2.** Continue until only the bounding box remains.

Lemma 1 *Given the Delaunay triangulation of n points, the Permutation Construction Algorithm can be implemented to run in $O(n)$ time.*

Proof: Step P1 executes once; either approach mentioned can be implemented to run in linear time.

Let us say that a single execution of steps P2–P5 constitutes a phase. Each phase executes in time proportional to the number of vertices involved: this is true for steps P2 and P4 by Euler's relation and simple graph traversal techniques. If we restrict the independent set to low-degree vertices, then Step P3 is linear by independent retriangulation of polygons of constant size. Otherwise, we can use the linear-time constrained-Delaunay algorithm of Wang and Chin [23].

Step P2, which is standard in hierarchical triangulations, implies that the number of vertices decreases by a constant in each phase. Using the 1/6 guaranteed in Section 4, the total time for all phases satisfies $T(n) \leq T(5n/6) + cn + O(1) \leq 6cn + O(1)$, where cn is the time for a phase with n vertices. ∎

If we now reverse the ordered list of points and reconstruct the Delaunay triangulation, our traversal can accomplish a linear amount of point location in linear time.

Reconstruction Algorithm

R1. **Initialize.** Start with the bounding box (or with an infinite vertex).

R2. **For each phase do R3–R4.** Use the signal from P5 to recognize where one phase ends and another begins.

R3. **Traverse and locate.** Visit the triangles by the same traversal as in P4 and link vertices to the triangles that contain them as they are found.

R4. **Insert entire phase.** Using Guibas and Stolfi's incremental insertion, update the Delaunay to incorporate the new vertices.

Lemma 2 *The Reconstruction Algorithm for the Delaunay triangulation can be implemented to take $O(k)$ time after reading k points.*

Proof: We first observe, by induction, that the phases of Reconstruction R2–R3 produce the triangulations of the phases of P2–P5 in reverse order.

In the base case, the initialization step R1 recreates the triangulation of the bounding box with which the Permutation Construction algorithm terminated.

Now assume, as an induction hypothesis, that R2–R3 begins its phase $i-1$ with the correct triangulation and list of vertices. The enumerations in P4 and R3 visit the triangles (and vertices) of phase $i-1$ in the same order. Thus, R3 will link all the vertices of phase $i-1$; since the algorithm identifies the ends of phases it ensures that it does not link any vertices of phase i. Insertion of all linked vertices then computes the triangulation to begin phase i. Therefore, by induction, the phases of Reconstruction compute the correct triangles.

Analyzing the running time is easy. Step R1 takes constant time. In a single phase, step R3 runs in time proportional to the number of triangles in the triangulation. Step R3 can use the incremental procedure of Guibas and Stolfi to perform each insertion in time proportional to the degree of the vertex, which is bounded by the number of edges of the triangulation in the next phase. Euler's relation ties both these quantities to the number of vertices, which increases according to the geometric series of Lemma 1. Stopping after k vertices results in a truncated series that sums to $O(k)$. ∎

3 End-of-phase signals and canonical traversal

It is important for the running time of the reconstruction algorithm to be able to detect the end of a phase. If one iteration of the loop R2 takes points from two phases, then subsequent iterations may not even complete a phase—they will be traversing different triangulations than the ones used to determine the permutation order.

On the other hand, if we avoid storing special end-of-phase markers then we may be able to "hide" terrain models without affecting other applications by simply permuting data points in a file.

Observation 3 *There are protocols to signal end-of-phase without adding markers to the permutation.*

One protocol would be to use fixed-length phases. Because the minimum phase size depends on the size of the smallest guaranteed independent set, we should use multiples of $6/5$ if we use independent sets of size at least $1/6$ of the total number of vertices. This leads to more phases and a higher constant factor in the reconstruction time.

Ideally, we could arrange for the traversal to run out of triangles at the end of a phase—that is, to ensure that the first point of the next phase in R2 does not lie in one of the triangles visited after the last point of the current phase. A simple protocol based on this idea is to move the first point of a phase to the end of the phase. Then that point will not be found during the traversal, but can be inserted separately after the traversal completes. After its insertion, the next phase can begin.

In the rest of this section, we specify our favorite canonical traversal for steps P4 and R3. Remove from a planar graph the edges of any spanning tree of vertices; the duals of the edges that remain form a spanning tree of the faces. This observation can be used to give a simple traversal of any planar graph with convex faces [4, 9]. As a bonus, we can use this traversal to do hidden surface elimination by the painter's algorithm.

Observation 4 *Any planar graph with convex faces has a canonical traversal that requires no data structure modifications and can list faces from back to front.*

Choose a direction, which we draw in Figure 2 as downward. Each vertex except the lowest rightmost selects the edge immediately below or to the right of the vertex. We can prove that this is a spanning tree: No edge is selected by both endpoints, so we have selected $v-1$ edges. The resulting graph is connected because we can start at any vertex and repeatedly follow the edge selected by the current vertex; we always move right or down, so we stop at the lowest rightmost point.

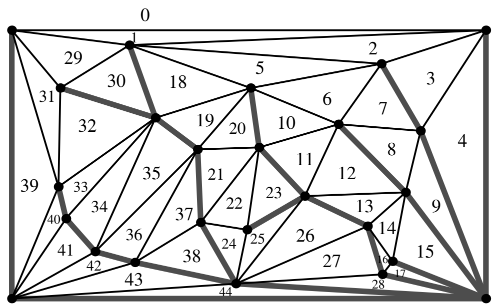

Fig. 2: Traversal from spanning tree

Now, treat spanning tree edges as the walls of a maze, and traverse the maze, keeping your left hand on a wall. Because the code is short, we include it in Figure 3. The function `wall`() recognizes selected edges by examining the coordinates of three consecutive vertices in counter-clockwise order. `drawTraverse`()

implements the traversal without making changes to the data structure. These are based on a quadedge-like data structure [11] that uses directed edges: `Sym`(e) reverses edge e, and `Lnx`(e) and `Lpr`(e) give the ccw and cw edges around the face to the left of e.

Figure 2 shows the faces numbered in the order that the traversal enters them. Along any vertical line, the numbers increase. Because the spanning tree of vertices consists of paths to the right, ending at the rightmost vertex, this traversal will draw the faces from back to front.

```
boolean wall(int e) {
  return p(e).greatereq(p(Lnx(e))) && p(Lnx(e)).greater(p(Lnx(Lnx(e))));
}

boolean entered(int e) {
  return p(e).greatereq(p(Lpr(e))) && p(Lnx(e)).greater(p(Lpr(e)));
}

void drawTraverse(Graphics g, int startedge) {
  int e = startedge, guard = 2*numedges;
  do {
    if (entered(e)) drawFace(g, e);
    if (wall(e) || wall(Sym(e))) e = Lpr(e);
    else e = Lpr(Sym(e));
  } while ((e != startedge) && (guard-- >= 0));
  if (guard < 0) throw new triangulationCorruptException();
}
```

Fig. 3: Traversing the faces of a convex subdivision

4 Bounds on maximal independent sets

Kirkpatrick [13] used a simple greedy algorithm to compute a maximal independent set (MIS) of low-degree vertices: Start by marking vertices of degree 12 or more. While there is an unmarked vertex, choose one for the MIS and mark it and its neighbors. His analysis gave an MIS with at least $1/24$ of the vertices, which can be improved to $1/18$ by initially marking vertices of degree 9 or more. Edelsbrunner [7, Problem 9.9] notes that choosing vertices of degree less than 13 in order of increasing degree gives an MIS of $1/7$ of the vertices. We show that if vertices are deleted, not just marked, then choosing vertices of degree less than 10 in order of increasing degree gives an MIS of $1/6$ of the vertices.

The algorithm is simple: Mark *high-degree* vertices, of degree 10 or more, then "delete" all marked vertices by decrementing the degrees of their neighbors. While vertices remain, choose a vertex of lowest degree for the MIS and delete it and its neighbors by marking them and decrementing the degrees of their neighbors. This can be implemented to run in linear time by storing a degree field for each vertex and maintaining ten lists (doubly-linked to support $O(1)$-time insertion and deletion of nodes) for vertices with degrees 0 through 9.

Let n_i, for $0 \le i \le 9$, be the number of vertices in the MIS that were chosen when their degree field was i. Thus, the size of the MIS is

$$\sum_{0 \le i \le 9} n_i. \tag{1}$$

Let n_h be the number of high-degree vertices. Because each vertex is deleted once,

$$n_h + \sum_{0 \le i \le 9} (i+1)n_i = n. \tag{2}$$

The deletion of high-degree vertices removes, at a minimum, $10n_h$ edges minus the number of edges in the graph induced by the high-degree vertices: zero edges when $n_h \le 1$, at most one edge when $n_h = 2$, and at most $3n_h - 6$ when $n_h > 2$. Thus, the first deletion removes at least $\min\{10n_h, 9n_h + 1, 7n_h + 6\}$ edges. Later, when a vertex of degree i is deleted, its i neighbors all have degree at least i, too. At a minimum, deletion eliminates one edge for each n_1, and $(i+1)i - (3(i+1) - 6) = i(i-2) + 3$ edges for each n_i with $2 \le i \le 9$. Since there are initially at most $3n - 6$ edges,

$$\min\{10n_h, 9n_h + 1, 7n_h + 6\} + n_1 + \sum_{2 \le i \le 9} \big(i(i-2) + 3\big)n_i \le 3n - 6. \tag{3}$$

This gives us three linear programming problems to find the minimum-size MIS according to (1), subject to (2) & (3). One minimum feasible solution has $n_4 = 3$, $n_5 = (n - 15)/6$, and the rest zero, for an independent set of more than 1/6 of the vertices. Patrice Belleville (private communication) has shown that this analysis is still pessimistic, and that the worst-case denominator is in [5.2, 5.5]. In our tests with Delaunay triangulations, the average MIS found by this algorithm contains 30% of the vertices.

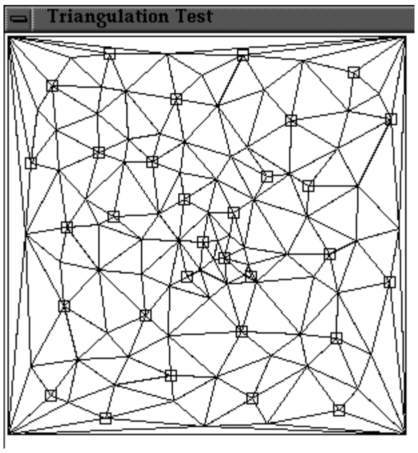

Fig. 4: MIS of 31/100 vertices

Since our algorithms do not require the MIS to have low-degree vertices, 4- or 5-colorings could be used to obtain independent sets guaranteeing 1/4 or 1/5 of the vertices. The best algorithm known for 4-coloring a planar graph takes quadratic time and must handle 633 cases [20], but 5-coloring in linear time is not especially difficult [3, 20]. The greedy algorithm above does avoid high degree vertices; it also gives us more freedom to select "important points" for our terrain model application.

5 Dealing with degenerate point sets

Thus far we have assumed that the Delaunay triangulation is determined canonically from the data, which need not be true for degenerate input: in a quadrilateral formed by four co-circular points, either of the two diagonals can be chosen to complete the triangulation. Strict adherence to the empty circle condition will choose both or neither, depending on the definition of "empty." Unfortunately, if the reconstruction chooses a different triangulation than the one used to construct the permutation, then the canonical traversal may not locate all of the points in a phase and the reconstruction will slow down.

One natural way to define a canonical triangulation in the presence of degeneracies is to enforce general position by an infinitesimal perturbation of the points. This succeeds with trapezoidations of line segments, where degeneracies are endpoints having the same x coordinate—if ties are broken by y-coordinates, then this is equivalent to perturbing the endpoints to general position by an infinitesimal skew transformation. Most perturbation methods described for the Delaunay triangulation, however, use a global index or memory address of a point. Since we permute the points, this is not an option. ("Simulation of Simplicity" [8] can be implemented by using local indices determined by lexicographic ordering.)

The *Delaunay graph* of a set S of points is a graph with vertex set S in which there is an edge between $p, q \in S$ if and only if there is a closed disc D in the plane with $D \cap S = \{p, q\}$. The Delanunay graph is canonically determined by the set S and has convex faces whose vertices are co-circular; when S is in general position, the Delaunay graph is the Delaunay triangulation.

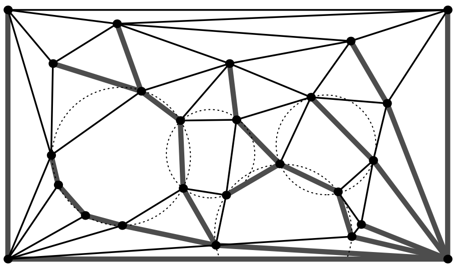

Fig. 5: Spanning tree in Delaunay graph of degenerate point set

One could obtain a canonical triangulation by triangulating each face of the Delaunay graph canonically; e.g., by connecting vertices to the highest vertex in each face. This solution works, but it does increase the degree of the highest vertex on a face, making such vertices less likely to be chosen in an independent set even though they may otherwise be unimportant.

Notice, however, that reconstruction depends on a canonical *traversal* of the faces of the Delaunay graph, and not a canonical *triangulation.* It is sufficient to triangulate each convex face such that the traversal visits all triangles within a face before it exits that face. Doing so uses fewer cases than Simulation of Simplicity.

Figure 5 illustrates the spanning tree chosen in a Delaunay graph by the selection rule of Section 3. A triangulation of this graph having the same spanning tree is called *traversable*, since our traversal visits all triangles within any face before it exits the face. We can build traversable triangulations with a standard incremental construction [11] if we carefully consider diagonal swaps for quadrilaterals with co-circular vertices.

Lemma 5 *An incremental algorithm builds a traversable triangulation for a set of points S. After point location, insertion of a point p takes time proportional to its degree.*

Proof: For this abstract, we simply sketch the changes to the standard incremental algorithm [11] for inserting a point p.

To insert p, we locate the triangle containing p and perform a standard insertion [11], creating edges joining p to the triangle vertices, and swapping diagonals to eliminate adjacent triangles whose circumcircles contain p. This forms all edges of the Delaunay graph that are incident on p. We now check that the incident faces of the Delaunay graph have traversable triangulations. In other words, that the vertices that are now neighbors of p do not select new spanning tree edges using the rule of Section 3.

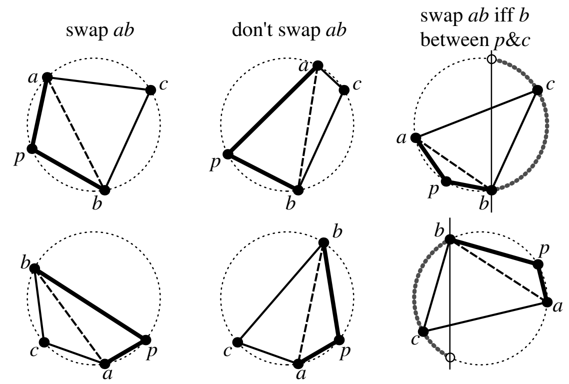

Fig. 6: Cases for swapping ab when new point p is on circumcircle of $\triangle abc$

Consider a quadrilateral $pbca$, formed from adjacent triangles, with all four points co-circular. As illustrated in Figure 6, we decide whether to swap diagonal $\overline{ab}$ for $\overline{pc}$ based on the x-coordinate order of a, p, b, and maybe c. If p is right or left of both a and b then swap $\overline{ab}$ iff a is between p and b. Otherwise, swap $\overline{ab}$ iff b is between p and c.

In the first case of figure 6, one must check whether the next edge out of a is in the Delaunay graph, and perform swaps as long as it is not. In all other cases, at most one "degenerate swap" is necessary to make the triangulation of the incident face traversable again, since both $\overline{pa}$ and $\overline{pb}$ are edges of the Delaunay graph. ■

Theorem 6 *The Permutation Computation and Reconstruction algorithms work correctly in $O(n)$ time if degeneracies are handled according to Lemma 5.*

Proof: We sketch the main difficulty, which is to show that the independent set that was deleted from one triangulation will be an independent set when it is reinserted in another triangulation.

Consider replacing the vertices of the independent set. If the interior of any circle associated with a face of the Delaunay graph contained two vertices, then they would be neighbors and the set would not be independent. Thus, each face has at most one vertex associated with one of its triangles. Inserting the vertices into the faces first ensures that the Delaunay graph is unchanged during traversal and initial insertion; we then restoring the traversability of the triangulation by swaps.

Lemma 5 then insures that the total change is proportional to vertex degrees, which sum to linear. ■

6 Experiments

With Bernd Juenger, we have been performing experiments to compare reconstruction and construction times using various simple point location schemes. Figure 7 shows timing of Java code (using the Sun beta JIT compiler on a 170Mhz ultraSparc) for the incremental construction in Guibas and Stolfi [11] in which points are located by walking through triangles from edge zero, a central edge, or the most recent edge, versus the reconstruction using the traversal and using the walk through triangles in the order given by the permutation (with explicit signals for end-of-phase.)

The good performance of the reconstruction that walks the permutation order leads to two questions related to multiple point location: In a triangulation of n points in the plane, show that there exists a path on m given vertices that has $O(n)$ crossings with triangulation edges. Then, find such a path in $O(n + m)$ time.

7 Conclusion

We have shown that simple ideas give a method to hide geometric structure in a permutation of the data so that, for example, a Delaunay triangulation can be reconstructed in $O(n)$ time from the permutation. These ideas also apply to trapezoidations of segments and constrained Voronoi diagrams in the plane, and to convex hulls in 3-d.

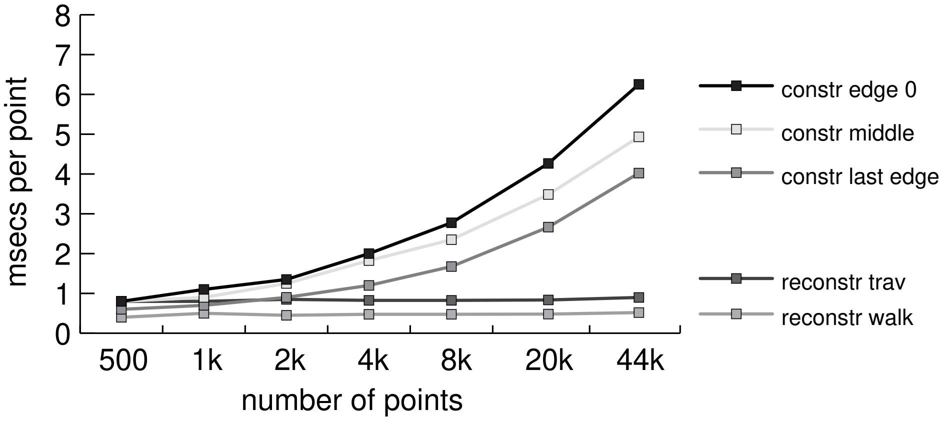

Fig. 7: Timing results for construction vs reconstruction

Acknowlegement

The data for figure 1 was provided by Facet Decision Systems. We thank them for their support of this research. Bernd Juenger performed the timing comparisons in figure 7.

References

[1] Franz Aurenhammer. Voronoi diagrams—A survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, 1991.

[2] Marshall Bern and David Eppstein. Mesh generation and optimal triangulation. In F. K. Hwang and D.-Z. Du, editors, *Computing in Euclidean Geometry*. World Scientific, March 1992.

[3] Norishige Chiba, Takao Nishizeki, and Nobuji Saito. A linear 5-coloring algorithm of planar graphs. *Journal of Algorithms*, 2:317–327, 1981.

[4] M. de Berg, M. van Kreveld, R. van Oostrum, and M. Overmars. Simple traversal of a subdivision without extra storage. *Int. J. of GIS*, 11, 1997.

[5] M. Deering. Geometry compression. *Computer Graphics*, pages 13–20, 1995. Proceedings of SIGGRAPH '95.

[6] H. Djidjev and A. Lingas. On computing the Voronoi diagram for restricted planar figures. In *WADS '91: Second Workshop on Data Structures and Algorithms*, number 519 in Lecture Notes in Computer Science, pages 54–64. Springer-Verlag, 1991.

[7] Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, Berlin, 1987.

[8] Herbert Edelsbrunner and Ernst Peter Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9(1):66–104, 1990.

[9] Michael T. Goodrich and Roberto Tamassia. Dynamic trees and dynamic point location. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 523–533, 1991. To appear in *SIAM Journal on Computing*, 1997.

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

[11] Leonidas Guibas and Jorge Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, 1985.

[12] H. Hoppe. Progressive meshes. *Computer Graphics*, pages 99–108, 1996. Proceedings of SIGGRAPH '96.

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

[14] D. T. Lee and F. P. Preparata. Computational geometry: a survey. *IEEE Trans. Comput.*, C-33:1072–1101, 1984.

[15] Ketan Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms.* Prentice-Hall, Englewood Cliffs, N.J., 1993.

[16] Atsuyuki Okabe, Barry Boots, and Kokichi Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams.* John Wiley & Sons, 1992.

[17] Joseph O'Rourke. *Computational Geometry in C.* Cambridge University Press, 1994.

[18] T. K. Peucker, R. J. Fowler, J. J. Little, and D. M. Mark. The triangulated irregular network. In *Amer. Soc. Photogrammetry Proc. Digital Terrain Models Symposium*, pages 516–532, 1978.

[19] Franco P. Preparata and Michael I. Shamos. *Computational Geometry—An Introduction.* Springer-Verlag, New York, 1985.

[20] Neil Robertson, Daniel P. Sanders, Paul Seymour, and Robin Thomas. Efficiently four-coloring planar graphs. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 571–575, 1996.

[21] R. Seidel. Backwards analysis of randomized geometric algorithms. In J. Pach, editor, *New Trends in Discrete and Computational Geometry*, volume 10 of *Algorithms and Combinatorics*, pages 37–68. Springer-Verlag, 1993.

[22] Raimund Seidel. A method for proving lower bounds for certain geometric problems. In Godfried T. Toussaint, editor, *Computational Geometry*, pages 319–334. North Holland, Amsterdam, 1985.

[23] Cao An Wang and Francis Chin. Finding the constrained Delaunay triangulation and constrained Voronoi diagram of a simple polygon in linear time. In Paul Spirakis, editor, *Algorithms–ESA '95*, number 979 in Lecture Notes in Computer Science, pages 280–294. Springer-Verlag, 1995.