Discrete Applied Mathematics 160 (2012) 648–656

Contents lists available at SciVerse ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

Computing convex quadrangulations☆

T. Schiffer[a], F. Aurenhammer[b,*], M. Demuth[b]

[a] *Institute of Computer Graphics and Knowledge Visualization, University of Technology, Graz, Austria*

[b] *Institute for Theoretical Computer Science, University of Technology, Graz, Austria*

ARTICLE INFO

Article history:
Received 28 July 2010
Received in revised form 28 October 2011
Accepted 2 November 2011
Available online 29 November 2011

Keywords:
Irregular quadrilateral mesh
Convex quadrangles
Shape quality
Delaunay tetrahedra
Maximum independent set

ABSTRACT

We use projected Delaunay tetrahedra and a maximum independent set approach to compute large subsets of convex quadrangulations on a given set of points in the plane. The new method improves over the popular pairing method based on triangulating the point set.

1. Introduction

Quadrangulations. Generating a mesh in a geometric domain is a step prior to many computational tasks, in finite-element-based applications as well as in computer graphics, geographic information systems, computational geometry, and other areas. Irregular meshes (i.e., meshes with varying vertex degree) are preferred in many cases, due to their flexibility and adaptability. Again, meshes that are homogenous (having a fixed number of vertices per element) are desired, because their elements can be treated in a uniform way. The most popular of such homogenous irregular meshes are triangulations and quadrangulations or, by another name, quadrilateral meshes. The literature on triangulations is vast; we make no attempt to survey even parts of it here, but rather refer to two quite complete introductory surveys on quadrangulations [24,5], which are the topic of the present paper.

Compared to triangulations, quadrangulations are not very well understood as far as their structural properties are concerned. This may be partially due to the fact that quadrangulations for *prespecified* sets of vertices need not exist. Algorithmic properties of quadrangulations appear to have been explored even less. For example, the basic question of deciding whether a given vertex set admits a *convex* quadrangulation is still unsettled, though a decision is easy from a parity argument when nonconvex quadrangles are allowed. On the other hand, quadrangulations may be the mesh of choice in certain applications, including finite-element generation [14,18] and scattered bivariate data analysis [15].

Quadrangulation methods. When it comes to computing quadrangulations in practice, (strict) convexity of elements is mandatory in most applications. Basically two philosophies have been followed: introducing extraneous vertices (so-called Steiner points) to the original set of vertices and, alternatively, relaxing the mesh so as to contain triangles as well. In fact, there is an interrelation, as any triangulation of a given point set can be refined to a convex quadrangulation by adding one Steiner point per triangle; see, e.g., [24]. This, however, results in the undesirable effect of tripling the number of elements.

☆ Supported by the Austrian FWF Projects DK ‘Confluence of Vision and Graphics’ and NRN ‘Industrial Geometry’ S9205-N12.

* Corresponding author.
E-mail addresses: t.schiffer@cgv.tugraz.at (T. Schiffer), auren@igi.tugraz.at (F. Aurenhammer), markus.demuth@tugraz.at (M. Demuth).

0166-218X
doi:10.1016/j.dam.2011.11.002

Effort has been put into theoretical investigations on the minimal number of Steiner points needed, on the one hand, and on finding clever ways of converting triangulations into convex quadrangulations, on the other. It is known that $\lceil (n-3)/2 \rceil$ Steiner points may be necessary, and $3\lfloor n/2 \rfloor$ are always sufficient, for enabling a strictly convex quadrangulation on a given n-point set in the plane [6]. Fewer Steiner points will suffice if the domain to be decomposed into convex quadrangles is a simple polygon [23]. The special case of convex polygons is addressed in [20]. Deciding whether a polygon with holes can be quadrangulated without using Steiner points is NP-complete, even when quadrangles of any form are allowed [17].

Virtually all proposed methods for computing quadrangulations use a conversion from triangulations, including those where deformation of the vertex set is involved [21,16]. The standard idea is to pair adjacent triangles to obtain as many convex quadrangles as possible, and to use Steiner points for the leftover triangles [23,5,13]. Triangle pairing is accomplished by using appropriate matching algorithms. The quality of such approaches, concerning both the shape of the quadrangles and the number of Steiner points, heavily depends on the underlying triangulation. Shape guarantee comes, for example, when using nonobtuse triangulations [4]. To gain more flexibility, restructuring larger groups of adjacent triangles [24] or subsequent edge flipping [16] has been considered.

A new approach. In the present paper, we propose a novel and simple method for computing strictly convex quadrangulations. Instead of pairing triangles by constructing matchings in the dual graph of some triangulation [23,5], we generate a candidate set of possibly overlapping convex quadrangles by projecting three-dimensional (3D) Delaunay tetrahedra [1,9], and then construct an independent set [12] in their intersection graph. Naturally, this approach is more flexible, as it has the freedom of choosing a third coordinate for the input vertices. It leads to a substantial candidate set of strictly convex quadrangles living on the input data. As a consequence, the number of Steiner points needed to complete the quadrangulation is smaller than in the triangle pairing approach. Moreover, the quadrangles produced automatically come with a certain quality (at least experimentally), because they are projections of Delaunay tetrahedra and thus tend to be of 'squarish' shape.

Though our algorithm might be slow in the worst case, it typically shows an $O(n \log n)$ behavior. In contrast, triangle pairing based on maximum *weight* matching, which is of comparable quality [5], takes $\Theta(n^2 \log n)$ time for any input [10]. The computationally less expensive unweighted variant, maximum *cardinality* matching [19], leads to unsatisfactory results, as it does not cover the distinction between convex and nonconvex quadrangles.

The efficiency of the new method is partially due to the widely observed (and theoretically well-founded) fact that Delaunay tetrahedralizations in three dimensions tend to show a complexity that is linear in the size of their defining point cloud; see, e.g., [9]. Moreover, there exists a simple linear-time greedy heuristic for approximating maximum independent sets [12] (an NP-complete graph problem). The heuristic comes with a theoretical guarantee, and turns out to perform extremely well on the quadrangle intersection graphs in question. Observe that, when solved optimally, the maximum independent set applied to the convex pairing quadrangles in a given triangulation just constructs a maximum weight matching. Our method, due to its ability of generating different Delaunay tetrahedralizations 'on top of' the given planar vertex set, is superior to the matching approach in most cases, even though the maximum independent set may not be found. Experiments show that a larger number of quadrangles is produced, and that their shape quality is comparable. The low observed runtime makes our algorithm applicable to sets with tens of thousands of points, and thus it is a promising candidate in practice.

A preliminary version of the present paper appeared in the conference proceedings [2].

2. The algorithm

We start with a formal definition of quadrangulations. Let a k-gon be a polygon with exactly k vertices, and consider a set S of n points in the xy-plane. The convex hull of S is the smallest convex subset of the plane that contains S. A k-angulation of S is a partition of the convex hull of S into k-gons which use all and only the points in S as vertices. We will talk of a (strictly) convex k-angulation if all its k-gons are (strictly) convex. The most prominent representatives are triangulations, which of course are strictly convex, and quadrangulations, where strict convexity is a requirement in many applications. By standard graph-theoretical counting arguments, there exists *some* quadrangulation for a given vertex set S if and only if the number of vertices of the convex hull of S is even. Criteria ensuring the existence of *convex* quadrangulations are, surprisingly, not available.

It is thus, in general, not possible to construct a complete convex quadrangulation for a given vertex set S. Instead, we aim at constructing a large set of mutually disjoint quadrangles. Each such quadrangle Q has to be *valid* (with respect to S); that is, Q is strictly convex, the vertices of Q are from S, and no points of S lie in the interior of Q. We propose a simple heuristic for computing such sets of quadrangles, which can be outlined as follows.

Algorithm QUADs

Step 1 (Candidate quadrangles) Assign some z-coordinate (height) to each point in S. Compute the Delaunay tetrahedralization, $\mathrm{DT}(S')$, of the resulting point cloud S'. Collect, in a set M, all quadrangles that can be obtained by projecting the tetrahedra of $\mathrm{DT}(S')$ onto the xy-plane.

Step 2 (Intersection graph) Remove from M all invalid quadrangles, that is, those which enclose points from S. Compute the intersection graph, G, for the valid quadrangles.

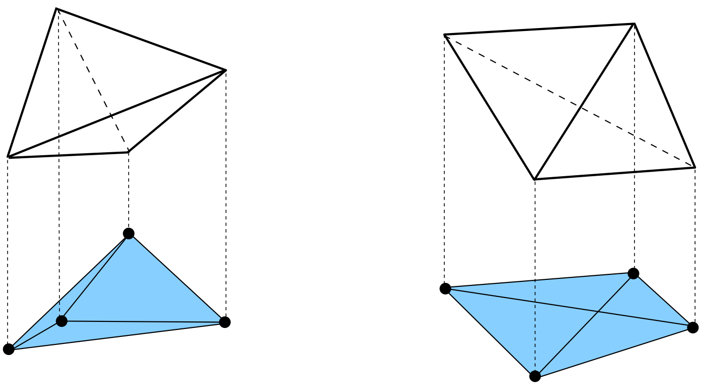


Fig. 1. Projecting Delaunay tetrahedra.

Step 3 (Maximum independent set) Approximate the maximum independent set in G. Output the corresponding set of quadrangles.

At this point, it is not clear that a useful number of quadrangles will be generated by Algorithm QUADs. Interestingly, as will be explained later, the output is quite large, almost a full quadrangulation in many cases. Observe that additional candidate quadrangles may be gained when iterating Step 1 for different choices of heights before advancing to Step 2.

When using the idea of lifting the point set S to be quadrangulated, one might ask why it is not better (and easier) to generate various triangulations of S by computing and projecting lower convex hulls of appropriately lifted point clouds S' in 3-space, and to use pairings of triangles there. We did not choose this approach, however, for two main reasons. First of all, the obtained triangulations will miss out many of the given points in S, namely, those that will not be extremal in the respective lifting S'. Thus, the lifting has to be done carefully, to ensure that all lifted points will appear on the lower convex hull—a fact complicating the use of random liftings. Second, the obtained triangulations will all belong to the class of *regular* triangulations (i.e., those obtainable by projecting the boundary of some convex polyhedron; see, e.g., [8]), which might additionally restrict the choice.

In the remainder of this section, we describe each step of algorithm QUADs in more detail, and comment on its correctness and runtime.

2.1. Candidate quadrangles

Given a set S of n points in the xy-plane, the Delaunay triangulation of S contains a triangle for each triple of points whose circumcircle does not enclose points from S. Likewise, the Delaunay tetrahedralization for a point set S' in 3-space contains a tetrahedron for each quadruple of points whose circumsphere is empty of points from S. See, for example, [1] for a survey article on this topic. A recent overview of results on Delaunay structures in various dimensions can be found in [9]. In the following, let S' be the point set obtained from lifting each point in S by an individual z-coordinate (called height in the following), and denote with $\mathrm{DT}(S')$ its Delaunay tetrahedralization.

A tetrahedron in $\mathrm{DT}(S')$ may project to one of two possible figures in the xy-plane: a triangle or – the case we are interested in – a convex quadrangle, say Q. See Fig. 1 left-hand side and right-hand side, respectively. Clearly, the vertices of Q belong to the set S. Moreover, if S is in general position,[1] then Q is strictly convex. When collecting all quadrangles that can be projected from $\mathrm{DT}(S')$ in this way, a set M of (possibly highly overlapping) convex quadrangles that live on the input vertex set S is obtained.

A well-known counting argument based on Euler's formula shows that a tetrahedralization on n points and with e edges contains at most $2e-2n$ triangles and $e-n$ tetrahedra. From $e \leq \binom{n}{2}$ we know that the size of a Delaunay tetrahedralization is $O(n^2)$. Though there exist examples where $\mathrm{DT}(S')$ attains the maximal size $\Theta(n^2)$ (at least for bad choices of heights for S, see Section 4), its observed size was $O(n)$ in all our tests. There are many efficient and stable implementations of Delaunay tetrahedralization algorithms available nowadays. In particular, a relationship to convex hulls in four dimensions can be exploited, and it enables us to use stable code for the latter problem [3]. In conclusion, the candidate set M in Step 1 typically contains $O(n)$ quadrangles and is computed in $O(n \log n)$ time.

2.2. Intersection graph

Given a set of geometric objects, their intersection graph contains a node for each object, and connects two nodes if and only if the corresponding objects have interior points in common. We are interested in the intersection graph, G, of those among the quadrangles in the candidate set M which are valid. That is, the set M first has to be relieved from quadrangles

[1] A finite set of points is said to be in general position if no three of its points lie on a common straight line.

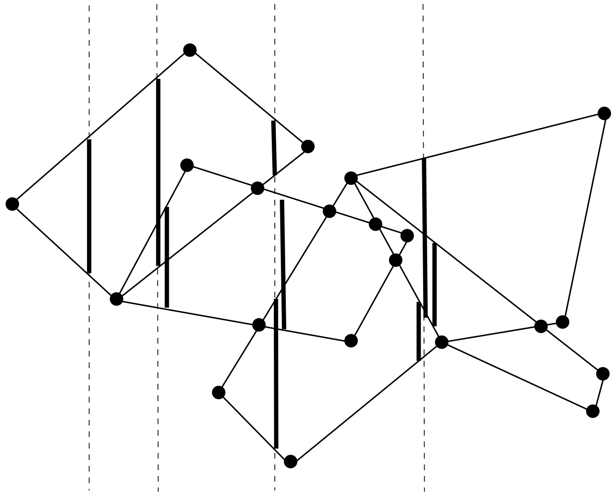


Fig. 2. Sweep line (dashed) and stabbing intervals (solid).

whose interiors contain points from S. This pruning of M can be accomplished 'on the fly' while constructing the graph G with the method below.

Computing the intersection graph of planar objects is a well-studied problem in computational geometry, efficiently solved with the plane-sweep technique; see, e.g., [22]. Conceptually, a vertical line L is swept across the plane, and at each point in time the subset of objects stabbed by L is considered. Intersections between objects are found by exploiting that the stabbing intervals they define on L have to overlap sometime. Fig. 2 gives an illustration.

Some comments are in order to adapt the general plane-sweep paradigm to our situation. To keep track of the overlap scenario on the sweep line L, we use a dynamic data structure, D, to represent stabbing intervals. A new interval is inserted into D when L hits the leftmost vertex of a quadrangle, and when the rightmost vertex of the quadrangle is reached the interval is removed from D again. This correctly updates the stabbing intervals, because each such quadrangle is convex, and thus its intersection with a straight line consists of at most one component. The overlap information among the current intervals on L is maintained, and the intersection graph G is updated, at events of the two types just mentioned (leftmost/rightmost vertex), and at points where quadrangle edges intersect. All these event points are marked with $\bullet$ in Fig. 2. In addition, for each quadrangle vertex v reached by L, all intervals containing v have to be removed from D (and their corresponding nodes from G) because such intervals correspond to quadrangles which contain input points and thus are invalid.

It is well known how to implement the data structure D efficiently using interval trees [7]. This allows us to compute the graph G in $O(k+m \log m)$ time, where m and k, respectively, are the size and the number of pairwise quadrangle intersections, for the candidate set M. An $O(n \log n)$ runtime is achieved if there are only $O(n)$ tetrahedra in Step 1, and their overlap density is constant. Our experiments in Section 4 reveal that both requirements seem to be met if random heights are chosen.

2.3. Independent set

An independent set in a graph is a subset of its vertices such that no two of them are connected by a graph edge. Finding an independent set of maximum cardinality is an NP-complete problem. In practice, several efficient algorithms that approximate the maximum independent set are available. We use a simple greedy heuristic [12] which performs particularly well for sparse and bounded-degree graphs, the situation we will face most likely with the quadrangle intersection graph. The heuristic repeatedly chooses some (allowed) vertex with smallest degree, and marks its neighbors as forbidden. Its runtime is $O(k)$, where k is the number of edges of the graph, and its quality in our experiments is much better than guaranteed in theoretical bounds.

3. Choice of heights

In Step 1 of Algorithm QUADs we assign individual heights to the vertices in the set S to be quadrangulated, and then use the Delaunay tetrahedralization $\mathrm{DT}(S')$ of the resulting 3D point cloud S' to extract convex quadrangles. The choice of heights influences which tetrahedra will appear in $\mathrm{DT}(S')$, and thus which quadrangles will end up in the candidate set M.

For each quadruple $T \subset S$ (and, in particular, for each one in convex position) there exists a height assignment for S such that T defines a tetrahedron in $\mathrm{DT}(S')$. Choose the circumsphere above the xy-plane first, project T onto it, and take heights zero for $S \setminus T$. To some extent, this shows the inherent flexibility of our approach. On the other hand, the large differences in height which are in general needed to ensure the existence of particular tetrahedra favor quadrangles that enclose vertices of S, and thus are invalid.

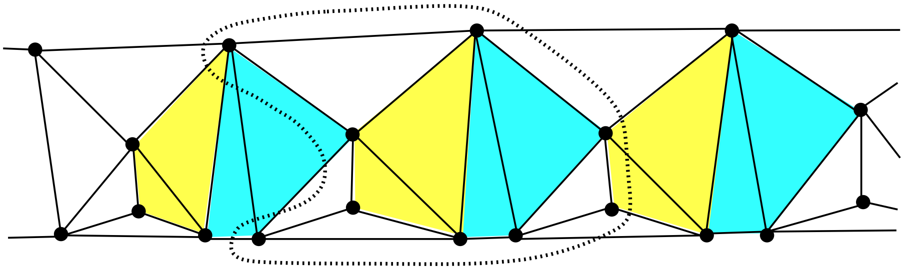


Fig. 3. Quadrangles from pairing Delaunay triangles.

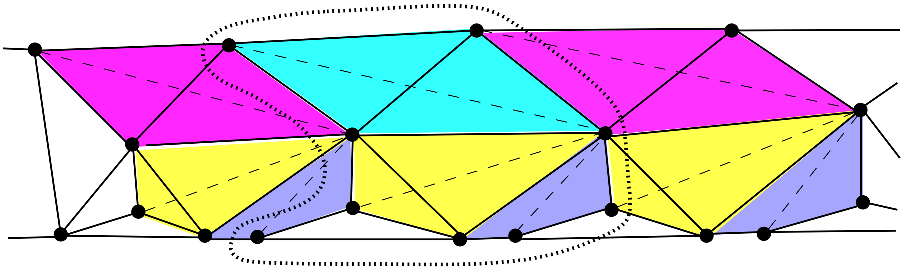


Fig. 4. Delaunay tetrahedra give more quadrangles.

Another simple observation is that all the edges of the (two-dimensional) Delaunay triangulation DT(S) can be generated by projecting DT(S'), namely, if height $h(v) = |v|^2$ is taken for each vertex $v \in S$. In this case, the lower boundary of the convex hull of S' (the boundary part being visible from $z = -\infty$), whose edges are of course edges of DT(S'), projects to DT(S); see, e.g., [1].

A certain relationship of the candidate set M to DT(S) is desired in view of the expected well-shapedness of the quadrangles. As a simple rule, if the heights are large compared to the interpoint distances in S, then the 'discrepancy' with DT(S) will be large, too. For two vertices $v, w \in S$, if $h(v)$ and $h(w)$ are both large, then the counterpart of edge vw is likely to be included in DT(S') even if v and w are far apart. Conversely, if $|h(v) - h(w)|$ is large, then vw's counterpart is unlikely to be included in DT(S') even if vw is an edge of DT(S). This suggests choosing heights from the interval $(0, c)$, where c denotes the distance between the two closest points in S. Tests with random heights from this interval substantiate this choice. However, a counterexample shows that the appearance of all the edges of DT(S') as projected edges of DT(S') cannot be forced by imposing an upper bound on the heights.

One of our objectives is to beat the popular triangle pairing approach, see, e.g., [23,5,13], which has been shown to work best if the underlying triangulation is DT(S). We will address this issue experimentally in Section 4. Here, let us give an example in which a linear number of quadrangles is gained when using our tetrahedra approach.

The underlying set S follows the repetition pattern in Fig. 3. Full lines delineate the Delaunay triangulation. Within the repeated group, which is composed of 7 triangles (and is marked by a dashed curve), pairing of triangles yields only 2 quadrangles (shaded). The same point set is shown in Fig. 4, where the shaded quadrangles come from projecting Delaunay tetrahedra when appropriate heights are chosen. In this case, 3 quadrangles are obtained for each group. Choosing random heights yields an average number of 2.5 tetrahedra per group, when a few iterations are granted in Step 1 of the algorithm; see Section 4.

We raise the question on an upper bound for the expected size of a 3D Delaunay tetrahedralization, when the z-coordinates of the points are chosen uniformly at random, but their x- and y-coordinates can be taken arbitrarily.

4. Experimental results

Assuming that the number h of points on the convex hull of our n-vertex set S is even, there are exactly

$$q(n, h) = n - \frac{h}{2} - 1$$

elements in any quadrangulation of S (if it exists). We measure the success of a quadrangulation algorithm, that is, the rate of quadrangles produced, with respect to the number $q(n, h)$. Note that this count is pessimistic, because $q(n, h)$ may by far not be reachable for special point sets [6].

Table 1 displays an extract from our experiences with the Algorithm QUADs, on the left-hand side, in comparison to the triangle pairing method in the two-dimensional (2D) Delaunay triangulation, on the right-hand side. For each run of Algorithm QUADs, random heights in the interval $(0, c)$ (see Section 3) have been used, and a fixed number of 100 iterations

Table 1
Success and quality for two quadrangulation methods.

Point set	n	Rate	f	$\alpha/\beta/e$	Rate	f	$\alpha/\beta/e$
`Random`	700	97.08	0.31	142/44/3.4	95.91	0.32	140/44/3.3
`Blocks`	255	94.67	0.28	149/37/2.8	93.03	0.32	142/43/2.6
`BlocksFilled`	280	94.81	0.31	146/39/2.8	94.07	0.34	142/44/2.5
`Graph`	380	95.62	0.21	148/39/4.3	92.60	0.24	140/45/4.3
`Extreme`	100	75.00	0.14	166/16/3.8	50.00	0.54	130/43/2.9
`DoubleCircle`	40	51.72	0.10	162/14/5.6	48.28	0.06	166/11/3.8
`Cross`	100	96.88	0.04	156/28/20.0	97.91	0.05	142/40/20.0

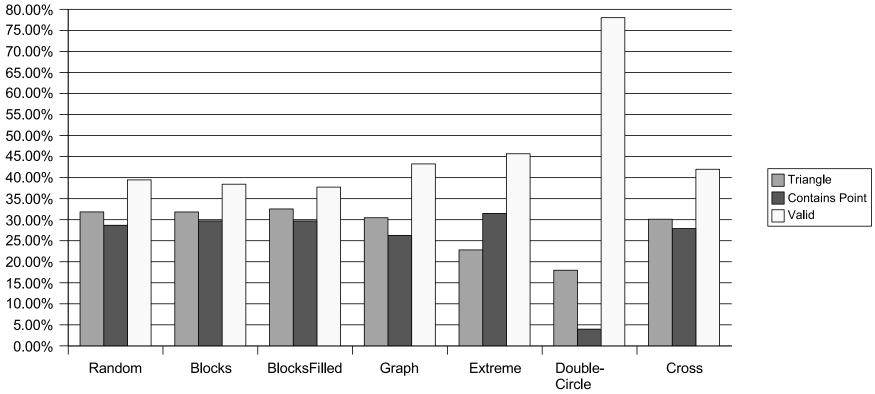


Fig. 5. Behavior of projected Delaunay tetrahedra for various point sets.

of Step 1 have been performed to collect candidate quadrangles. To get a more reliable impression, we have taken the median (in rate) of five runs per point set.

The line `Random` refers to a uniformly distributed point set. (We have averaged over 10 sets in this case.) We achieve a quadrangulation rate of over 97%, which slightly beats the pairing approach, and basically is in accordance with the results reported in [5] on random point sets.

Our method is adaptive to point configurations of varying density and shape. We included a heavily clustered point set, `Blocks` (Fig. 11, left), a slightly diluted version thereof, `BlocksFilled` (Fig. 11, right), and a 'curve-like' point set, `Graph` (Fig. 10, right). In all cases, we reached a high quadrangulation rate, and compare favorably to the pairing approach which, admittedly, turns out to be quite flexible, too.

There are point sets where we perform well, even with a single iteration of Step 1, and pairing is bad. Point set `Extreme` is an example; see Section 3 and Figs. 3 and 4. On the other hand, by simply including into our candidate set all the convex quadrangles obtainable from the 2D Delaunay triangulation, we could, for any point set, be at least as good as pairing. To be able to clearly distinguish between both algorithms, we did not exploit this advantage in our tests, however.

It is known what point sets that admit only few convex quadrangles look like, for instance, the set `DoubleCircle` from [6] (Fig. 12, left). This set realizes only a fraction (at most $\frac{2}{3}$) of the theoretical upper bound, which is $q(n, h) = n - \frac{n}{4} - 1$ in this case.[2] We get a rate of about 52%, which is not exciting but still outperforms pairing.

Point set `Cross` (Fig. 12, right), when augmented with appropriate heights, is the well-known example where the Delaunay tetrahedralization is of size $\Theta(n^2)$; see, e.g., [25]. Here, pairing is superior. Interestingly, the number of tetrahedra produced is not larger than in our other examples (roughly $6n$ in the first iteration); that is, our method retains its speed.

What can be said about the shape quality of our quadrangles? As one possibility [5], we took as a quality measure the mean maximum angle α, the mean minimum angle β, and the median vertex distance ratio e. Together, they describe the overall squarishness of the quadrangulation produced. A perhaps more descriptive alternative [13] is the mean of the values $f(Q_i) = \sqrt{2} \cdot \frac{r_i}{R_i}$, where r_i is the minimum inradius of the quadrangle Q_i, and R_i is the maximum circumradius of Q_i. We have $f(Q_i) = 1$ if and only if Q_i is a square. Table 1 reveals that pairing is better in quality. This has to be read with care, however, as we mostly produce more quadrangles (typically near the boundary), and thus may necessarily have to include less well-shaped ones. This effect becomes obvious for point set `Extreme`.

Concluding so far, it can be said that our method performs quantitatively better than pairing, which, in turn, may be qualitatively slightly better.

An advantage of Algorithm QUADs is its low observed runtime. It compares favorably to the triangle pairing method when the input becomes large; see Fig. 6. Point sets have been generated uniformly at random in a rectangle, and 40 and 100 iterations, respectively, of Step 1 for Algorithm QUADs have been executed.

2 This can even be reduced to $\frac{1}{2}$, as pointed out in [11].

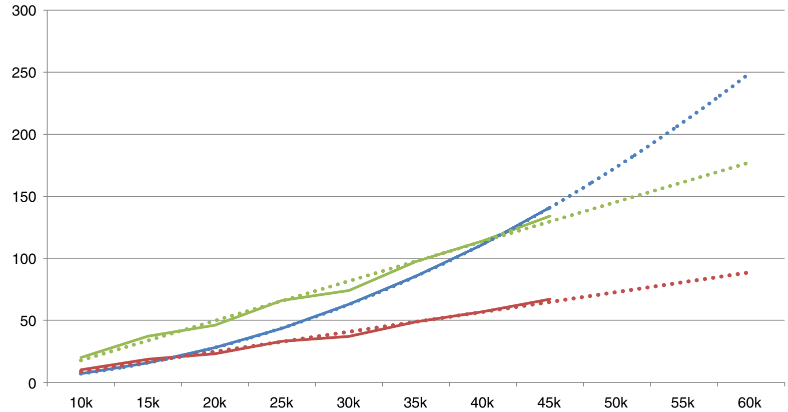


Fig. 6. Runtime comparison of Algorithm QUADs (almost linear curves) and the pairing method (quadratic curve). Input size over seconds is scaled.

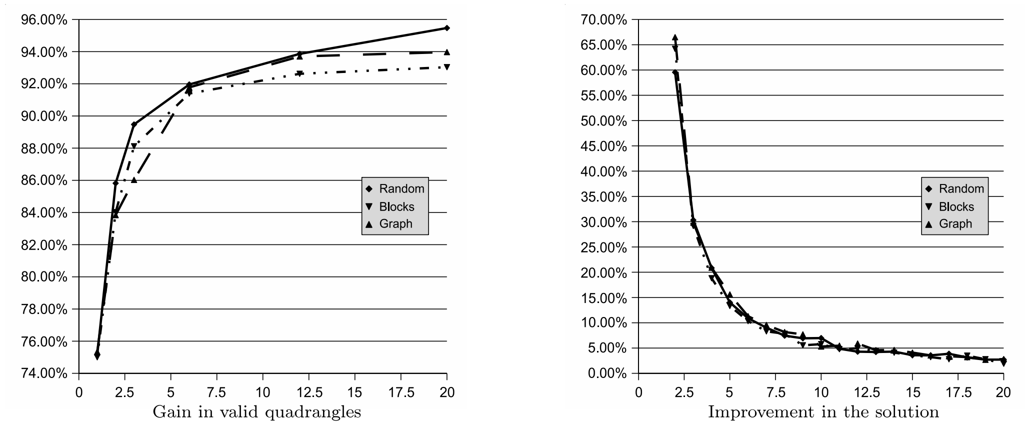


Fig. 7. The effect of iterating Step 1 of Algorithm QUADs.

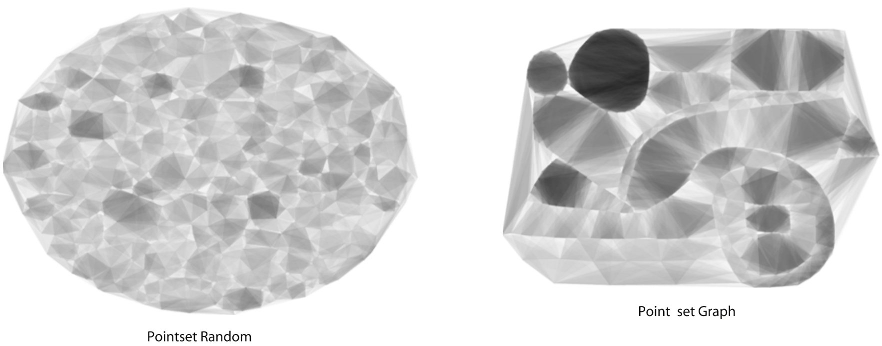


Fig. 8. Overlap densities for valid quadrangles.

In this context, let us describe some details about the implicit behavior of Algorithm QUADs.

To get an impression what happens when projecting Delaunay tetrahedra, let us have a look at Fig. 5. A certain percentage of tetrahedra project to triangles (light grey), or quadrangles that contain input points (dark grey), hence being useless for our purposes. However, the percentage of valid quadrangles (white) is always high. The data shown are for a single iteration of Step 1, averaged over 20 runs. They support the claim that our overall approach via Delaunay tetrahedralizations is well suited for generating candidate sets of valid triangles.

Fig. 7 documents the effect of iterating Step 1, for three representative cases: a uniformly distributed point set `Random`, and the point sets `Blocks` and `Graph`. (Illustrations of these point sets are given in the Appendix.) On the left-hand side, a plot of the gain in valid quadrangles over increasing number of iterations is shown. The different point sets behave quite similarly. A moderate number of iterations suffices to produce almost all valid triangles. The right-hand side gives a plot for

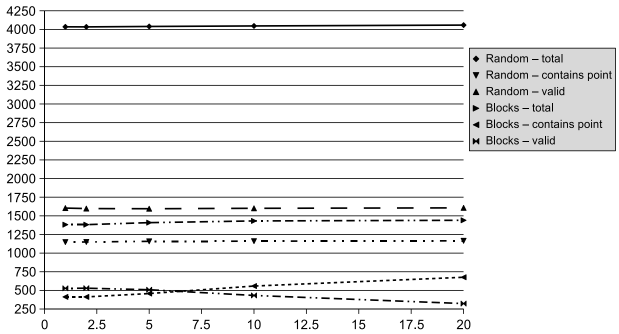
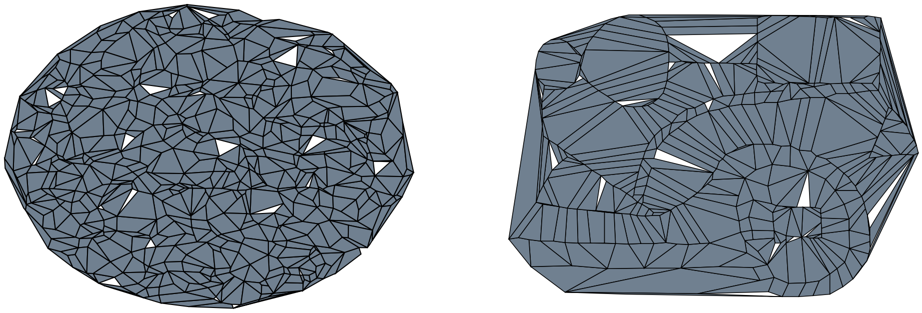


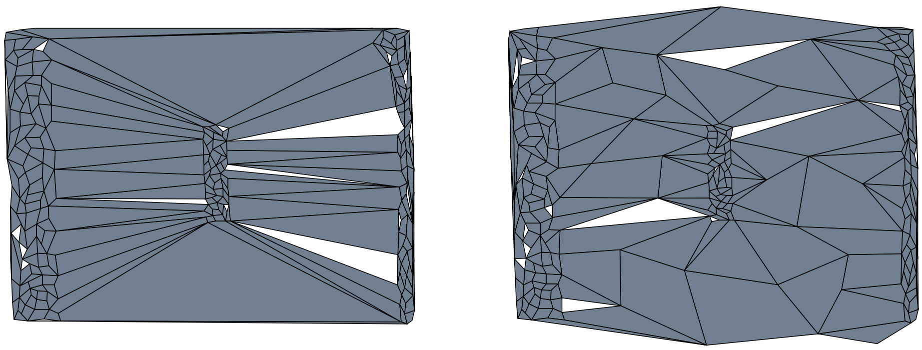
Fig. 9. Impact of changing the height interval.



Point set Random (700)

Point set Graph (380)

Fig. 10. Uniformly distributed point set, and point configuration on curves.



Point set Blocks (255)

Point set Blocks Filled (280)

Fig. 11. Heavily clustered point sets.

the improvement of the solution, i.e., of the number of elements in the (partial) quadrangulation produced. Again, most of the quadrangles are produced in the first 20 iterations.

The number of valid quadrangles collected in these iterations is less than 14 per point on average for point set `Random`, and about 17 for point set `Graph`. For both sets, the overlap density of the quadrangles is visualized in Fig. 8. This value is typically around 20 (median grey), and maximally 62 (dark grey) for the former set, and typically around 30, and maximally 132, for the latter. It can be observed that increased density arises for subsets of data points in convex positions.

Finally, we study the impact of varying the interval where the heights of the data points are (randomly) chosen from. We consider a single execution of Step 1 of the algorithm, and average over 20 runs. The plot in Fig. 9 displays, for two selected point sets, the number of computed elements (tetrahedra, non-empty quadrangles, and valid quadrangles) over the multiple of the interval length c. Recall that c is the closest-point distance realized by the respective set. There is no visible effect for point set `Random`, as well as for the total number of tetrahedra for point set `Blocks`. When expanding the height interval, the number of valid quadrangles decreases for point set `Blocks`, in spite of the fact that the number of non-empty quadrangles increases. Overall, taking c as the interval length proved to be a good choice, as reducing its length led to a clear loss in the solution. Improvements seem plausible, however, when exploiting information from earlier iterations in the subsequent choice of heights. This could be a subject for future research.

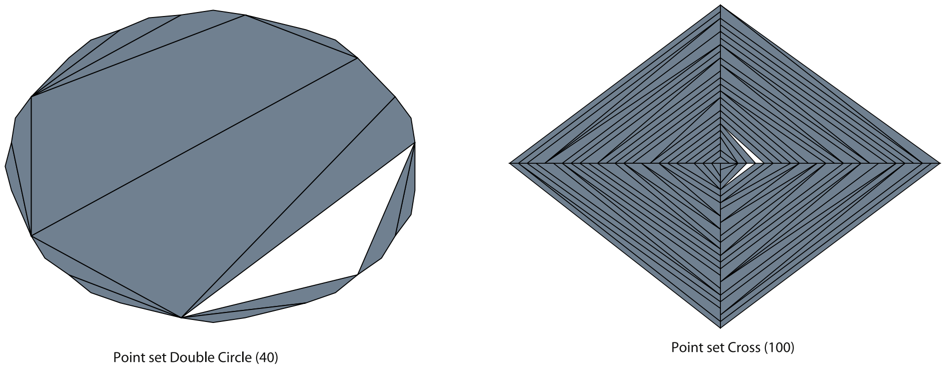


Fig. 12. Circle-like and cross-like point configurations.

Appendix. Test point sets

Figs. 10–12 illustrate the point sets we selected to test the behavior of Algorithm QUADs. For each set, its number of points is written in parentheses. The partial quadrangulations depicted were obtained after 100 iterations of Step 1. Very few non-quadrangulated parts remain, which is made quantitative in Table 1.

References

[1] F. Aurenhammer, Voronoi diagrams—a survey of a fundamental geometric data structure, ACM Comput. Surv. 23 (1991) 345–405.
[2] F. Aurenhammer, M. Demuth, T. Schiffer, Computing convex quadrangulations, in: D.-S. Kim, K. Sugihara (Eds.), in: Proc. 5th Ann. Int. Symp. Voronoi Diagrams in Science and Engineering, Voronoi's Impact on Modern Science, vol. 4, Kyiv University, Ukraine, 2008, pp. 32–43.
[3] C.B. Barber, D.P. Dobkin, H.T. Huhdanpaa, The Quickhull algorithm for convex hulls, ACM Trans. Math. Software 22 (1996) 469–483. http://www.qhull.org.
[4] M. Bern, D. Eppstein, Quadrilateral meshing by circle packing, Internat. J. Comput. Geom. Appl. 10 (2000) 347–360.
[5] P. Bose, S. Ramaswami, G. Toussaint, A. Turki, Experimental results on quadrangulations of sets of fixed points, Comput. Aided Geom. Des. 19 (2002) 533–552.
[6] D. Bremner, F. Hurtado, S. Ramaswami, V. Sacristan, Small strictly convex quadrilateral meshes of point sets, Algorithmica 38 (2004) 317–339.
[7] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, second ed., MIT Press and McGraw-Hill, 2001.
[8] H. Edelsbrunner, N.R. Shah, Incremental topological flipping works for regular triangulations, Algorithmica 15 (1996) 223–241.
[9] J. Erickson, Dense point sets have sparse Delaunay triangulations or "... but not too nasty", Discrete Comput. Geom. 33 (2005) 83–115.
[10] H.N. Gabow, Data structures for weighted matching and nearest common ancestors with linking, in: Proc. 1st Ann. ACM–SIAM Symposium on Discrete Algorithms, 1990, pp. 434–443.
[11] T. Hackl, Personal Communication, 2010.
[12] M.M. Halldòrsson, J. Radhakrishnan, Greed is good: approximating independent sets in sparse and bounded-degree graphs, Algorithmica 18 (1997) 145–163.
[13] T. Itoh, K. Shimada, Automatic conversion of triangular meshes into quadrilateral meshes with directionality, Int. J. CAD/CAM 1 (2001) 20–38.
[14] B. Joe, Quadrilateral mesh generation in polygonal regions, Comput. Aided Des. 27 (1995) 209–222.
[15] M.-J. Lai, L.L. Schumaker, Scattered data interpolation using C^2 supersplines of degree six, SIAM J. Numer. Anal. 34 (1997) 905–921.
[16] Y.K. Lee, C.K. Lee, A new indirect anisotropic quadrilateral mesh generation scheme with enhanced local mesh smoothing procedures, Int. J. Numer. Methods Eng. 58 (2003) 277–300.
[17] A. Lubiw, Decomposing polygonal regions into convex quadrilaterals, in: Proc. 1st Ann. ACM Symp. Computational Geometry, 1985, pp. 97–106.
[18] A. Malanthara, W. Gerstle, Comparative study of unstructured meshes made of triangles and quadrilaterals, in: Proc. 6th Intl. Meshing Roundtable, 1997, pp. 437–447.
[19] S. Micali, V.V. Vazirani, An $O(\sqrt{|V|}\,|E|)$ algorithm for finding a maximum matching in general graphs, in: Proc. 21st Ann. IEEE Symp. Foundations of Computer Science, 1980, pp. 17–27.
[20] M. Müller-Hannemann, K. Weihe, Minimum strictly convex quadrangulations of convex polygons, in: Proc. 13th Ann. ACM Symp. Computational Geometry, 1997, pp. 193–202.
[21] S.J. Owen, M.L. Staten, S.A. Canann, S. Saigal, Q-Morph: an indirect approach to advancing front quad meshing, Int. J. Numer. Methods Eng. 44 (1999) 1317–1340.
[22] F.P. Preparata, M.I. Shamos, Computational Geometry: An Introduction, Springer-Verlag, New York, NY, 1985.
[23] S. Ramaswami, P. Ramos, G. Toussaint, Converting triangulations to quadrangulations, Comput. Geom. Theory Appl. 9 (1998) 257–276.
[24] S. Ramaswami, M. Siqueira, T.A. Sundaram, J.H. Gallier, J.C. Gee, Constrained quadrilateral meshes of bounded size, Internat. J. Comput. Geom. Appl. 15 (2005) 55–98.
[25] J.R. Shewchuk, Stabbing Delaunay tetrahedralizations, Discrete Comput. Geom. 32 (2004) 339–343.