

## Constructing Plane Spanners of Bounded Degree and Low Weight<sup>1</sup>

Prosenjit Bose,<sup>2</sup> Joachim Gudmundsson,<sup>3</sup> and Michiel Smid<sup>2</sup>

**Abstract.** Given a set  $S$  of  $n$  points in the plane, we give an  $O(n \log n)$ -time algorithm that constructs a plane  $t$ -spanner for  $S$ , with  $t \approx 10$ , such that the degree of each point of  $S$  is bounded from above by 27, and the total edge length is proportional to the weight of a minimum spanning tree of  $S$ . Previously, no algorithms were known for constructing plane  $t$ -spanners of bounded degree.

**Key Words.** Computational geometry, Spanners, Planar graphs, Bounded degree graphs, Low weight graphs.

**1. Introduction.** Given a set  $S$  of  $n$  points in the plane and a real number  $t > 1$ , a  $t$ -spanner for  $S$  is a graph  $G$  with vertex set  $S$  such that any two vertices  $u$  and  $v$  are connected by a path in  $G$  whose length is at most  $t \cdot |uv|$ , where  $|uv|$  is the Euclidean distance between  $u$  and  $v$ . If this graph has  $O(n)$  edges, then it is a sparse approximation of the (dense) complete Euclidean graph on  $S$ .

Many algorithms are known that compute  $t$ -spanners with  $O(n)$  edges [1], [2], [8], [20], [22],[24], [26] that have additional properties such as bounded degree [4], [6], small spanner diameter [4] (i.e., any two points are connected by a  $t$ -spanner path consisting of only a small number of edges), low weight [11], [13], [19] (i.e., the total length of all edges is proportional to the weight of a minimum spanning tree of  $S$ ), planarity [3], [21], and fault-tolerance [10], [23]; see also the surveys [18] and [25]. All these algorithms compute  $t$ -spanners for any given constant  $t > 1$ . Observe that some properties are conflicting. For instance a spanner with constant degree cannot have spanner diameter  $o(\log n)$ .

In this paper we consider the construction of plane  $t$ -spanners. A graph, whose vertices are points in  $\mathbb{R}^2$  with straight-line edges, is said to be *plane*, if no two edges intersect, except possibly at their endpoints. Obviously, in order for a  $t$ -spanner to be plane,  $t$  must be at least  $\sqrt{2}$ . It is known that the Delaunay triangulation is a  $t$ -spanner for  $t = 2\pi/(3 \cos(\pi/6))$ , see [21]. Furthermore, Das and Joseph [12] showed that other plane graphs such as the minimum weight triangulation and the greedy triangulation are  $t$ -spanners, for some constant  $t$ . In 1992 Levcopoulos and Lingas [22] showed that, for

<sup>1</sup> Research supported in part by the Natural Science and Engineering Research Council of Canada and by the Netherlands Organization for Scientific Research.

<sup>2</sup> School of Computer Science, Carleton University, Ottawa, Ontario, Canada K1S 5B6. {jit,michiel}@scs.carleton.ca. Research supported in part by NSERC.

<sup>3</sup> Department of Mathematics and Computing Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands. h.j.gudmundsson@TUE.nl.

any real number  $r > 0$ , a plane graph can be constructed from the Delaunay triangulation, that is a  $t$ -spanner for  $t = (1 + 1/r)2\pi/(3 \cos(\pi/6))$  and whose total edge length is at most  $2r + 1$  times the weight of a minimum spanning tree of  $\mathcal{S}$ . Observe that all these spanners may have unbounded degree. Arikati et al. [3] considered the problem of constructing a plane spanner among polygonal obstacles. They show how to construct a  $(\sqrt{2} + \varepsilon)$ -spanner in  $O(n \log n)$  time using  $O(n)$  Steiner points, for any real constant  $\varepsilon > 0$ .

No algorithms are known that compute a plane  $t$ -spanner, for some constant  $t$ , of bounded degree. In this paper we show how to compute such a spanner that, additionally, has low weight. That is, the main result of this paper is the following theorem.

**THEOREM 1.1.** *There is an  $O(n \log n)$ -time algorithm that computes, when given a set  $\mathcal{S}$  of  $n$  points in the plane and any real constant  $\varepsilon > 0$ , a graph*

1. *that is plane,*
2. *that is a  $t$ -spanner for  $\mathcal{S}$ , with  $t = (1 + \varepsilon)(\pi + 1)2\pi/(3 \cos(\pi/6))$ ,*
3. *in which each point of  $\mathcal{S}$  has degree at most  $27$ ,*
4. *and whose weight is bounded from above by a constant times the weight of a minimum spanning tree of  $\mathcal{S}$ .*

We outline the global structure of our algorithm, see also Figure 1. In the first step the Delaunay triangulation of  $\mathcal{S}$ , denoted  $\mathcal{DT}$ , is computed. Recall that  $\mathcal{DT}$  is the triangulation of  $\mathcal{S}$  such that no point of  $\mathcal{S}$  is inside the circumcircle of any triangle of the triangulation. In the second step a degree-3 spanning subgraph  $\mathcal{SG}$  of  $\mathcal{DT}$  is computed. This graph  $\mathcal{SG}$  contains the convex hull of  $\mathcal{S}$  and is a (possibly degenerate) simple polygon with vertex set  $\mathcal{S}$ . This part of the algorithm is described in Section 2. The polygon is then transformed into a simple non-degenerate polygon  $\mathcal{P}$  in the third step of the algorithm.

Next, the algorithm processes the vertices of  $\mathcal{P}$ , in an order that is obtained from a breadth-first order of  $\mathcal{DT}$ , and carefully adds Delaunay edges to  $\mathcal{P}$ . The resulting graph, denoted  $\mathcal{G}_{\mathcal{P}}$ , is a plane spanner for the vertices of  $\mathcal{P}$ . This part of the algorithm is described in Section 3.

In the last step of the algorithm, as shown in Section 4, we run the greedy spanner algorithm by Gudmundsson et al. [19] on the graph  $\mathcal{G}_{\mathcal{P}}$ . This results in a graph  $\mathcal{G}$  for which the claims in Theorem 1.1 hold.

**Algorithm** PLANARSPANNER( $\mathcal{S}, \varepsilon$ )

1.  $\mathcal{DT} \leftarrow$  Delaunay triangulation of  $\mathcal{S}$  (see [15])
2.  $\mathcal{SG} \leftarrow$  SPANNINGGRAPH( $\mathcal{DT}$ ) (see Section 2)
3.  $\mathcal{P} \leftarrow$  TRANSFORMPOLYGON( $\mathcal{SG}$ ) (see Section 2.1)
4.  $\mathcal{G}_{\mathcal{P}} \leftarrow$  POLYGONSPANNER( $\mathcal{P}, \mathcal{DT}$ ) (see Section 3)
5.  $\mathcal{G} \leftarrow$  GREEDYSPANNER( $\mathcal{G}_{\mathcal{P}}, \varepsilon$ ) (see [19])
6. output  $\mathcal{G}$

**Fig. 1.** The construction of a plane sparse spanner.

**2. Computing a Degree-3 Plane Spanning Graph (SPANNINGGRAPH).** The aim of this section is to describe Step 2 of the algorithm in detail. In other words, given a set  $\mathcal{S}$  of  $n$  points in the plane, we show how to construct a low-degree plane spanning graph  $\mathcal{SG}$  of  $\mathcal{S}$  that contains the convex hull of  $\mathcal{S}$ .

Let  $d$  be the maximum degree of any vertex of  $\mathcal{SG}$ . The key property we exploit is that the graph  $\mathcal{SG}$  is a plane graph with exactly two faces, an interior face and an exterior face. The interior face has the property that every vertex of  $\mathcal{SG}$  is on its boundary. If  $\mathcal{SG}$  has no cut vertices, then it is a simple polygon. The presence of cut vertices in  $\mathcal{SG}$  renders it into a degenerate polygon. However, by symbolically modifying the cut vertices, we show how to convert it into a simple polygon. Essentially, this process has a cut vertex symbolically appear on the boundary of the interior face several times, so that it is no longer a cut vertex (see the next section for details). In the conversion process, a cut vertex of degree  $d$  is converted into  $d$  vertices each of degree 2. After the processing is completed on the simple polygon, these  $d$  vertices are then joined again into one vertex. Thus, the final degree of the spanner depends on  $d$ . The lower the value of  $d$ , the smaller the final degree of the spanner.

*Degree-5 plane spanning subgraph.* Let  $G$  be the union of the convex hull of  $\mathcal{S}$  and a minimum spanning tree  $T$  of  $\mathcal{S}$ . Then  $G$  is clearly a plane spanning graph. It is well known that each vertex of  $T$  has degree at most 6, and the angle between any two edges of  $T$  sharing an edge is at least  $60^\circ$ . Therefore,  $G$  has degree less than or equal to 6. This can be improved to 5 since there always exists a minimum spanning tree of degree at most 5.

Another degree-5 spanning subgraph can be obtained as follows: Consider an arbitrary triangulation  $\Delta$  of  $\mathcal{S}$ . Barnette [5] has shown that every 3-connected planar graph contains a spanning tree of maximum degree at most 3; his proof of this result implies an  $O(n \log n)$ -time algorithm for computing such a tree. If we apply this to  $\Delta$ , then we obtain a plane degree-3 spanning tree of  $\mathcal{S}$ . However, by adding the convex hull of  $\mathcal{S}$  to this tree, we again obtain a degree-5 plane spanning graph of  $\mathcal{S}$  that contains the convex hull of  $\mathcal{S}$ .

*Degree-3 plane spanning subgraph.* We now show how to reduce the degree to 3. Our construction uses the *canonical ordering* of de Fraysseix et al. [16]. Let  $\Delta$  be an arbitrary triangulation of  $\mathcal{S}$ , and let  $(v_1, v_2, \dots, v_n)$  be a numbering of the points of  $\mathcal{S}$ . For any  $i$  with  $1 \leq i \leq n$ , let  $\Delta_i$  denote the subgraph of  $\Delta$  induced by the vertices  $v_1, v_2, \dots, v_i$ , and let  $C_i$  be the outer face of  $\Delta_i$ . We say that the numbering  $(v_1, v_2, \dots, v_n)$  is a canonical ordering of  $\Delta$ , if the following conditions are satisfied:

1.  $v_1$  and  $v_2$  are adjacent in  $\Delta$  and are both on the convex hull of  $\mathcal{S}$ .
2. For all  $i$  with  $3 \leq i \leq n$ , the following hold:
  - (a) The subgraph  $\Delta_i$  is 2-connected.
  - (b) The point  $v_i$  is in the interior of the outer face  $C_{i-1}$  of  $\Delta_{i-1}$ .
  - (c) Let  $N_i$  be the set of all vertices of  $\Delta_{i-1}$  that are connected by an edge, in  $\Delta$ , to  $v_i$ . The set  $N_i$  contains at least two elements, and forms a path on the boundary of  $C_{i-1}$ .

A canonical ordering always exists and can be computed, in reverse order, as follows. Start with the triangulation  $\Delta$ , and successively remove a vertex from the boundary of

the outer face that is adjacent to exactly two vertices on the outer face. de Fraysseix et al. [16] showed that such a vertex always exists. The time to compute this canonical ordering is  $O(n)$  [9].

We apply this to an arbitrary triangulation  $\Delta$  of  $\mathcal{S}$ . Consider a canonical ordering  $(v_1, \dots, v_n)$  of  $\Delta$ . For each  $i$  with  $3 \leq i \leq n$ , we compute a subgraph  $G_i$  of  $\Delta_i$ . During this construction, we maintain the following invariant:

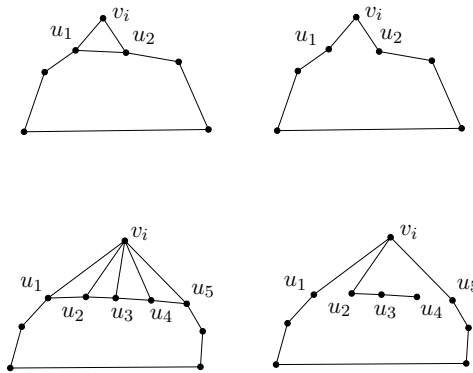
**INVARIANT.** The graph  $G_i$  is a spanning subgraph of  $\Delta_i$ , its maximum degree is at most 3, and the only cycle in  $G_i$  is the boundary of the outer face  $C_i$  of  $\Delta_i$ .

We define the graph  $G_3$  to be the triangle spanned by  $v_1, v_2$ , and  $v_3$ , where  $v_1$  and  $v_2$  lie on the convex hull of  $\mathcal{S}$ . Let  $i$  be an integer with  $4 \leq i \leq n$ , and assume we have constructed the graph  $G_{i-1}$ . Consider the set  $N_i$  of all vertices of  $\Delta_{i-1}$  that are connected by an edge, in  $\Delta$ , to  $v_i$ . Recall that  $N_i$  forms a path on the boundary of  $C_{i-1}$  containing at least two vertices. Let  $u_1, \dots, u_k$  be all vertices of  $N_i$ , in the order in which they occur on this path. We distinguish two cases:

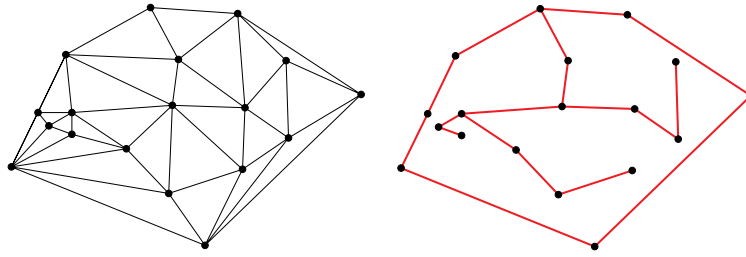
$k = 2$ . We obtain  $G_i$  from  $G_{i-1}$  by removing the edge  $(u_1, u_2)$  and adding the edges  $(v_i, u_1)$  and  $(v_i, u_2)$ , as shown in the top part of Figure 2.

$k > 2$ . We obtain  $G_i$  from  $G_{i-1}$  by removing the edges  $(u_1, u_2)$  and  $(u_{k-1}, u_k)$  and adding the edges  $(v_i, u_1)$ ,  $(v_i, u_2)$ , and  $(v_i, u_k)$ , as illustrated in the bottom part of Figure 2.

It is clear that the invariant is maintained during this construction. Hence, the graph  $\mathcal{SG} = G_n$  is a plane spanning graph of  $\mathcal{S}$  of maximum degree at most 3. Observe that  $C_n$  is equal to the outer face of  $\Delta$ . Therefore, the boundary of  $C_n$  is the convex hull of  $\mathcal{S}$  and it is the only cycle in  $\mathcal{SG}$ . Hence, the invariant implies that  $\mathcal{SG}$  contains the convex hull of  $\mathcal{S}$ . Observe that the constrained dual of  $\Delta$ , constrained by the edges of  $G_i$ , is a tree of degree at most 3. This is precisely why we will be able to symbolically transform  $\mathcal{SG}$  into a simple polygon as outlined in the next section.



**Fig. 2.** The construction of a degree-3 plane spanning graph.



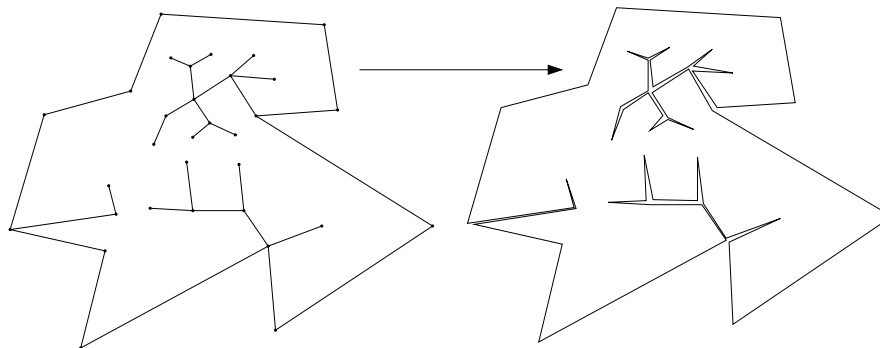
**Fig. 3.** A triangulation  $\Delta$  (on the left), and the output of algorithm  $\text{SPANNINGGRAPH}(\Delta)$  (on the right).

The algorithm for computing  $\mathcal{SG}$  from the canonical ordering of the vertices of  $\Delta$  as described above is denoted by  $\text{SPANNINGGRAPH}(\Delta)$ . An example is given in Figure 3. Observe that  $\mathcal{SG}$  contains exactly one cycle, the convex hull of  $\mathcal{S}$ . The part of  $\mathcal{SG}$  within the convex hull is a forest connected to  $\mathcal{S}$ . The following lemma summarizes the result of this section.

**LEMMA 2.1.** *Given a triangulation  $\Delta$  of a set  $\mathcal{S}$  of  $n$  points in the plane, we can compute, in  $O(n)$  time, a plane spanning subgraph  $\mathcal{SG}$  with  $n$  edges whose only cycle is the convex hull of  $\mathcal{S}$ , and in which each vertex has degree at most 3.*

**2.1. Algorithm TRANSFORMPOLYGON.** As was alluded to in the previous section, the output  $\mathcal{SG}$  of algorithm  $\text{SPANNINGGRAPH}(\Delta)$  is a polygon, however, this polygon may be degenerate, i.e., there may be vertices of degree 1 and there may be cut vertices. An easy way to handle this problem is to symbolically transform the degenerate polygon  $\mathcal{SG}$  into a non-degenerate simple polygon  $\mathcal{P}$ . This is done by doubling edges incident on degenerate vertices of  $\mathcal{SG}$  and then running an Euler tour. Figure 4 illustrates how this transformation is performed.

This transformation does not affect the complexity of the spanner algorithm if the vertices are only split symbolically. We denote the algorithm that transforms a degenerate polygon  $\mathcal{SG}$  into a non-degenerate polygon  $\mathcal{P}$  by  $\text{TRANSFORMPOLYGON}(\mathcal{SG})$ .



**Fig. 4.** Illustrating algorithm  $\text{TRANSFORMPOLYGON}$ .

**3. Computing a Spanner of a Simple Polygon (POLYGONSPANNER).** The idea behind the main algorithm is to start with the Delaunay triangulation of  $\mathcal{S}$ , denoted  $\mathcal{DT}$ , and then remove edges from it until we are left with a plane graph  $\mathcal{G}_{\mathcal{P}}$  having the following two properties:

1. every vertex in  $\mathcal{G}_{\mathcal{P}}$  has bounded degree, and
2. for every edge  $(x, y)$  in  $\mathcal{DT}$ , there is a path between  $x$  and  $y$  in  $\mathcal{G}_{\mathcal{P}}$  whose length is at most  $\pi + 1$  times  $d_{\mathcal{DT}}(x, y)$ , where  $d_{\mathcal{DT}}(x, y)$  denotes the shortest distance in  $\mathcal{DT}$  between  $x$  and  $y$ .

The two properties combined imply that  $\mathcal{G}_{\mathcal{P}}$  is a plane graph in which every vertex has bounded degree and that is a  $((\pi + 1)2\pi/(3 \cos(\pi/6)))$ -spanner of the point set  $\mathcal{S}$ . To achieve the first property, we construct the degree-3 spanning subgraph,  $\mathcal{SG}$ , of the Delaunay triangulation as described in the previous section. Of course, this subgraph is not necessarily an  $O(1)$ -spanner. Therefore, we need to add some carefully chosen edges in order to maintain the first property and achieve the second, the result then follows from the fact that  $\mathcal{DT}$  is a  $t$ -spanner of  $\mathcal{S}$  with  $t \leq 2\pi/(3 \cos(\pi/6))$  [21].

The starting point of this section is the simple polygon  $\mathcal{P}$  that is computed by algorithm TRANSFORMPOLYGON( $\mathcal{SG}$ ). Recall that the edges of  $\mathcal{P}$  are edges in  $\mathcal{DT}$ . Let  $\mathcal{E}$  be the set of edges of the Delaunay triangulation of  $\mathcal{S}$ . Now, since the edges of  $\mathcal{P}$  are also Delaunay edges, we have  $\mathcal{E} \subseteq \mathcal{DT}$  and each edge  $(x, y)$  in  $\mathcal{E}$  that is not an edge of  $\mathcal{P}$  must be a diagonal of  $\mathcal{P}$ . Thus, it is sufficient to show how to properly approximate all such diagonals within  $\mathcal{P}$ .

Let  $t > 1$  be a real number, and let  $\mathcal{V}$  be the vertex set of  $\mathcal{P}$ . A graph  $G$  defined on the vertex set  $\mathcal{V}$  respects  $\mathcal{P}$  if every edge of  $G$  is either an edge of  $\mathcal{P}$  or an internal diagonal of  $\mathcal{P}$ . We say that such a graph is a  $t$ -spanner for  $\mathcal{V}$  with respect to  $\mathcal{P}$ , if for every pair of points  $u, v \in \mathcal{V}$  that form a diagonal of  $\mathcal{P}$ , there exists a path between  $u$  and  $v$  in the graph  $G$ , whose length is at most  $t$  times the Euclidean distance  $|uv|$  between  $u$  and  $v$ . Note that by construction, every path between  $u$  and  $v$  in  $G$  is a path whose intersection with the exterior of  $\mathcal{P}$  is empty.

In this section we show how to compute, for some constant  $t > 1$ , a plane  $t$ -spanner for  $\mathcal{V}$  with respect to  $\mathcal{P}$ , whose degree is bounded by a constant. By constructing such a spanner, we will show that we obtain the desired spanner for the point set  $\mathcal{S}$ .

Let  $G = (\mathcal{V}, \mathcal{E})$  be the plane graph, where  $\mathcal{V}$  and  $\mathcal{E}$  are as defined above. Let  $v_1$  be an arbitrary vertex of  $\mathcal{V}$ , and let  $v_1, v_2, \dots, v_n$  be the vertices of  $\mathcal{V}$  according to a clockwise breadth-first ordering (bf-order) of  $G$  with start vertex  $v_1$ . A clockwise breadth-first search of a graph is simply a standard breadth-first search with the additional constraint that when the children of a vertex are processed, they are processed in clockwise order, as shown in Figure 5. For any two vertices  $u$  and  $v$  of  $\mathcal{V}$ , we write  $u <_{\text{bf}} v$ , if  $u$  strictly precedes  $v$  in the bf-ordering. We extend this ordering to  $\leq_{\text{bf}}$  in the obvious way.

We start with some properties of the bf-numbering. Let  $\mathcal{T}(G, v_1)$  be the breadth-first search tree of  $G$  rooted at vertex  $v_1$ . The level of any vertex  $v$  in  $\mathcal{T}(G, v_1)$  is denoted by  $\ell(v)$ . The root is at level zero, i.e.,  $\ell(v_1) = 0$ .

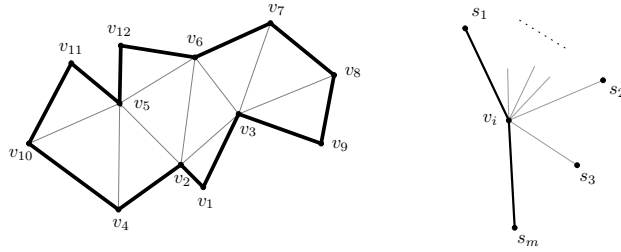


Fig. 5. An illustration of clockwise breadth-first search.

LEMMA 3.1. *Let  $v$  be any vertex in  $\mathcal{T}(G, v_1)$  with  $v \neq v_1$ .*

- (a) *Any Delaunay edge  $(v, w)$  that is not an edge of  $\mathcal{P}$  partitions  $\mathcal{P}$  into two simple polygons  $P_1$  and  $P_2$ . We may assume without loss of generality that  $v_1$  is a vertex of  $P_1$ . For every vertex  $u$  in  $P_2$  with  $u \neq v$  and  $u \neq w$ , we have  $\min(\ell(v), \ell(w)) < \ell(u)$  and  $\max(\ell(v), \ell(w)) \leq \ell(u)$ .*
- (b) *The line segment between  $v$  and its parent  $u$  in  $\mathcal{T}(G, v_1)$  partitions  $\mathcal{P}$  into two simple polygons  $P_1$  and  $P_2$  such that all children of  $u$  that precede  $v$  are in  $P_1$  and all siblings that succeed  $v$  are in  $P_2$ . The vertex  $v$  can be adjacent to at most one child of  $u$  that precedes  $v$ .*

PROOF. The first claim holds since either  $v$  or  $w$  must be an ancestor of all vertices in  $P_2$ . The second claim follows from the observation that the siblings of the parent of  $v$  are ordered in clockwise order, see Figure 5.  $\square$

The sequence of vertices adjacent to a vertex  $v$ , sorted in clockwise order around  $v$ , are denoted by  $\mathcal{N}(v)$  and are called the *ordered neighbors* of  $v$ . Finally, let  $p(v)$  denote the parent of  $v$  in  $\mathcal{T}(G, v_1)$ .

LEMMA 3.2. *The vertices in  $\mathcal{N}(v)$  preceding  $v$  in the bf-ordering are the parent of  $v$  and at most two vertices in  $\mathcal{N}(v) \cap \mathcal{N}(p(v))$ .*

PROOF. If  $v$  is the root then no vertex precedes  $v$ , otherwise since  $(v, p(v))$  is a Delaunay edge it holds that  $(v, p(v))$  must be on the boundary of one or two triangles, and hence adjacent to one vertex  $v_1$  or two vertices  $v_1$  and  $v_2$ .

In the case when  $v$  and  $p(v)$  are both adjacent to exactly one vertex  $v_1$ , the triangle  $(p(v), v, v_1)$  partitions  $\mathcal{P}$  into three simple polygons, denoted  $P_1$ ,  $P_2$ , and  $P_3$  as illustrated in Figure 6(a).  $P_2$  is bounded by  $(p(v), v_1)$  and hence cannot contain any vertices connected to  $v$ . Now consider  $P_3$  with boundary edge  $(v_1, v)$ . There are two cases:

- If  $\ell(v) = \ell(v_1)$  then every vertex  $u$  in  $P_3$ , except  $v$  and  $v_1$ , must be a successor of  $v$  since  $\ell(u) > \ell(v)$  according to Lemma 3.1(a).
- If  $\ell(v) > \ell(v_1)$  then every point  $u$  in  $P_3$  must be a successor of  $v$  since  $p(v) <_{\text{bf}} v_1$ , otherwise  $p(v)$  could not be the parent of  $v$ .

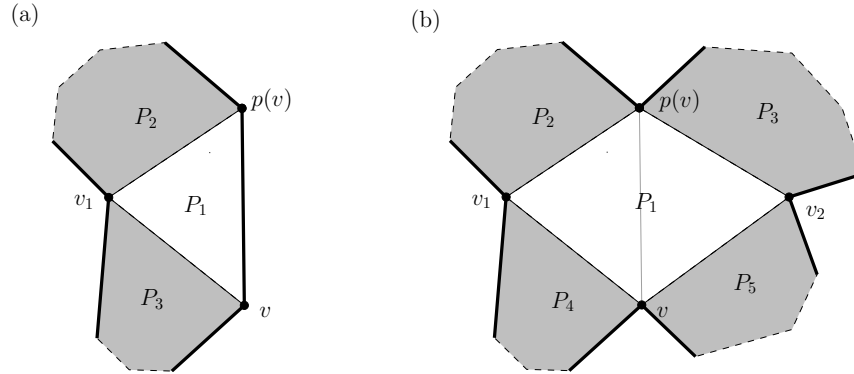


Fig. 6. Illustration for proof of Lemma 3.2.

Thus the lemma holds in the case when  $v$  and  $p(v)$  are both adjacent to exactly one vertex.

The proof for the case when  $v$  and  $p(v)$  are both adjacent to two vertices is almost identical. Let  $v_1$  and  $v_2$  be the two vertices adjacent to  $v$  and  $p(v)$ . The quadrangle  $(p(v), v_1, v, v_2)$  partitions  $\mathcal{P}$  into five simple polygons, denoted  $P_1, \dots, P_5$  as illustrated in Figure 6(b).  $P_2$  and  $P_3$  are bounded by  $(p(v), v_1)$  and  $(p(v), v_2)$ , respectively, and hence cannot contain any vertices connected to  $v$ . Now consider  $P_4$  and  $P_5$  with boundary edge  $(v_1, v)$  and  $(v_2, v)$ , respectively. The proof for both  $P_4$  and  $P_5$  are identical. We consider  $P_4$ . There are two cases:

- If  $\ell(v) = \ell(v_1)$  then every vertex  $u$  in  $P_4$ , except  $v$  and  $v_1$ , must be a successor of  $v$  since  $\ell(u) > \ell(v)$  according to Lemma 3.1(a).
- If  $\ell(v) > \ell(v_1)$  then every point  $u$  in  $P_4$  must be a successor of  $v$  since  $p(v) <_{\text{bf}} v_1$ , otherwise  $p(v)$  could not be the parent of  $v$ .

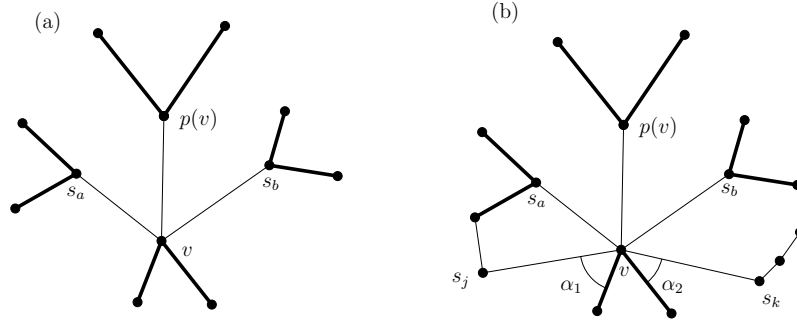
The lemma follows.  $\square$

Now we are ready to present the algorithm that computes a bounded-degree plane spanner for  $\mathcal{V}$  with respect to  $\mathcal{P}$ . Recall that  $G = (\mathcal{V}, \mathcal{E})$  is the plane graph, where  $\mathcal{E}$  is the union of the edge set of  $\mathcal{P}$  and the set of edges in the Delaunay triangulation of  $\mathcal{P}$ . Our algorithm processes all the vertices of  $\mathcal{P}$  following the bf-ordering starting with  $v_1$ .

Initialize  $\mathcal{E}_{\mathcal{P}}$  to be the set of edges of  $\mathcal{P}$ . At the end of the processing,  $\mathcal{E}_{\mathcal{P}}$  will contain the edges of the desired spanner. Let  $s_1, \dots, s_m$  be the ordered neighbors  $\mathcal{N}(v_1)$  of  $v_1$ . That is,  $s_1, \dots, s_m$  are all vertices that are connected by an edge of  $\mathcal{E}$  to  $v_1$ . These vertices occur in clockwise order around  $v_1$ , where  $(s_1, v_1)$  and  $(s_m, v_1)$  are edges of  $\mathcal{P}$ . The following two steps are performed when processing  $v_1$ .

First, the interior angle  $\angle s_1 v_1 s_m$  is divided into a minimum number of intervals of at most  $90^\circ$ . For each interval, add the shortest edge in  $\mathcal{E}$  that is connected to  $v_1$  to the edge set  $\mathcal{E}_{\mathcal{P}}$ . Second, add to  $\mathcal{E}_{\mathcal{P}}$  all edges  $(s_j, s_{j+1})$ ,  $1 \leq j < m$ . This terminates the processing of vertex  $v_1$ .

In each iteration, one vertex is processed according to its bf-ordering. One iteration is described as follows. Assume that vertex  $v_i$  of  $\mathcal{P}$  is next to be processed, see Figure 7.



**Fig. 7.** A possible situation when  $v$  is to be processed by the algorithm. (a) When  $p(v)$  was processed, at most three edges may have been added that are incident on  $v$ . (b) Additionally, at most two edges may be added during the processing of  $s_a$  and  $s_b$ .

Let  $s_1, \dots, s_m$  be the ordered neighbors of  $v_i$ , in clockwise order around  $v_i$ . We assume that  $(s_1, v_i)$  and  $(s_m, v_i)$  are edges of  $\mathcal{P}$ . It follows from Lemma 3.2 that the parent of  $v_i$  has been processed and possibly also two vertices  $s_a$  and  $s_b$  in  $\mathcal{N}(v_i) \cap \mathcal{N}(p(v_i))$ .

Let  $s_j$  and  $s_k$  be the first and last vertex in the ordered neighborhood of  $v_i$  such that  $(s_j, v_i) \in \mathcal{E}_{\mathcal{P}}$  and  $(s_k, v_i) \in \mathcal{E}_{\mathcal{P}}$ . Observe that  $s_j$  and  $s_k$  must exist. Let  $\alpha_1(v_i) = \angle s_1 v_i s_j$  and  $\alpha_2(v_i) = \angle s_k v_i s_m$ . The following two steps are performed.

First, divide the interior angle  $\angle s_1 v_i s_j$  into a minimum number of intervals of at most  $90^\circ$ . The same is done for the angle  $\angle s_k v_i s_m$ . For each interval, we add to  $\mathcal{E}_{\mathcal{P}}$ , the shortest edge in  $\mathcal{E}$  that connects  $v_i$  to a vertex that has not been processed. Secondly, we add all edges  $(s_l, s_{l+1})$ ,  $1 \leq l < j$  and  $k \leq l < m$ , to  $\mathcal{E}_{\mathcal{P}}$ . Vertex  $v_i$  has now been processed.

We continue processing vertices in this way until all vertices have been processed. Observe that after a vertex has been processed, no additional edges incident on it are ever added. The output of this algorithm is the graph  $\mathcal{G}_{\mathcal{P}} = (\mathcal{V}, \mathcal{E}_{\mathcal{P}})$ . We refer to this algorithm as  $\text{POLYGONSPANNER}(\mathcal{P}, DT)$ .

**3.1. Analysis of Algorithm POLYGONSPANNER.** In this section we show that the graph, denoted  $\mathcal{G}_{\mathcal{P}}$ , produced by  $\text{POLYGONSPANNER}$  is a spanner with respect to  $\mathcal{P}$  and that the degree of any vertex of  $\mathcal{G}_{\mathcal{P}}$  is bounded by a constant.

**LEMMA 3.3.** *Let  $v$  be any vertex of  $\mathcal{P}$ . At the moment when  $v$  is about to be processed by algorithm  $\text{POLYGONSPANNER}$ , there are at most five edges in  $\mathcal{E}_{\mathcal{P}}$  that are incident on  $v$ .*

**PROOF.** If  $v = v_1$ , then  $\mathcal{E}_{\mathcal{P}}$  is the set of edges of  $\mathcal{P}$ . The claim follows from the results in Section 2. Assume that  $v \neq v_1$ . The proof uses Lemma 3.2. First observe that the parent  $p(v)$  of  $v$  is the first vertex adjacent to  $v$  that is being processed. The edges incident on  $v$  that the algorithm may add to  $\mathcal{E}_{\mathcal{P}}$  while processing  $p(v)$  are the Delaunay edge  $(v, p(v))$  and possibly the two edges connecting  $v$  to the adjacent vertices among the ordered neighbors of  $p(v)$ ; hence a total of at most three edges. Assume that  $v$  is connected to the vertices  $v_1$  and  $v_2$  on the ordered neighbors of  $p(v)$ . These are the only two vertices adjacent to  $v$  that can be processed before  $v$  is processed. However, since

$p(v)$  is in  $\mathcal{N}(v) \cap \mathcal{N}(v_1)$  and also in  $\mathcal{N}(v) \cap \mathcal{N}(v_2)$ , at most two edges incident on  $v$  can be added to  $\mathcal{E}_{\mathcal{P}}$  while processing  $v_1$  and  $v_2$ , i.e., the two edges  $(s_j, v)$  and  $(s_k, v)$  in Figure 7. Hence, when  $v$  is being processed, there are at most five edges in  $\mathcal{E}_{\mathcal{P}}$  incident on  $v$ .  $\square$

Once a vertex has been processed, no additional edges incident on it are ever added. This simple observation, together with the above results, shows that the degree, in the graph  $\mathcal{G}_{\mathcal{P}}$ , of any vertex  $v \in \mathcal{V}$  is bounded from above by

$$5 + \lceil \alpha_1(v)/90^\circ \rceil + \lceil \alpha_2(v)/90^\circ \rceil + 2 \leq 9 + \alpha_1(v)/90^\circ + \alpha_2(v)/90^\circ < 13,$$

since  $\alpha_1(v) + \alpha_2(v) < 360^\circ$ . Hence, the degree is at most 12. This immediately gives a bound of at most 36 since each vertex can be split into three vertices by procedure TRANSFORMPOLYGON. In Section 4 we show how to reduce this bound. However, first we prove the following lemma that deals with the spanner properties of the edges of  $\mathcal{G}_{\mathcal{P}}$ .

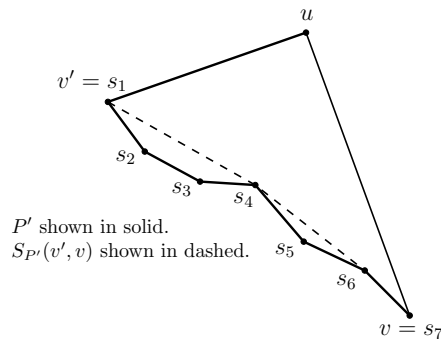
**LEMMA 3.4.** *For any edge  $(u, v) \in \mathcal{DT}$  that is in the interior of  $P$ , there exists a path in  $\mathcal{G}_{\mathcal{P}}$  between  $u$  and  $v$  whose length is at most  $\pi + 1$  times the length of edge  $(u, v)$ .*

**PROOF.** Consider an arbitrary edge  $e = (u, v)$  in the Delaunay triangulation of  $S$  that is contained in the interior of  $\mathcal{P}$ . If  $e \in \mathcal{G}_{\mathcal{P}}$  then the lemma holds. Assume that  $e \notin \mathcal{G}_{\mathcal{P}}$ .

Assume without loss of generality that  $u <_{\text{bf}} v$ . It follows from the algorithm that there exists a vertex  $v'$  in the ordered neighbors of  $u$  such that  $\angle v'uv < 90^\circ$  and  $(v', u) \in \mathcal{E}_{\mathcal{P}}$ . Let  $v' = s_1, \dots, s_k = v$  be the sequence of vertices in the ordered neighbors of  $u$  from  $v'$  to  $v$ . Since  $(v', u) \in \mathcal{E}_{\mathcal{P}}$  is shorter than  $(u, v)$  by definition, we only need to show that there is a path between  $v$  and  $v'$  in  $\mathcal{G}_{\mathcal{P}}$  that is not much longer than  $|uv|$ .

Let  $P'$  be the polygon consisting of the vertices  $u, s_1, \dots, s_k$ . Observe that  $P'$  is a subpolygon of  $\mathcal{P}$ . Consider the shortest path inside  $P'$  from  $v'$  to  $v$ , denoted  $S_{P'}(v', v)$ . If  $v$  and  $v'$  are visible in  $P'$ , then the shortest path is simply the edge  $vv'$  whose length is at most  $2|uv|$ , by the triangle inequality and the fact that  $|uv| \geq |uv'|$  by construction.

On the other hand, if  $v$  and  $v'$  are not visible in  $P'$ , then the shortest path is a convex chain consisting of diagonals of  $P'$  contained in the triangle  $uvv'$ , see Figure 8. Since



**Fig. 8.** Illustration for Lemma 3.4.

both  $v$  and  $v'$  are visible from  $u$ , the 3-vertex path  $v'uv$  is a path in polygon  $P'$  from  $v'$  to  $v$  contained in the triangle  $uvv'$ . Any path that goes outside the triangle can be shortcut, therefore, the shortest path must be contained in the triangle. The length of  $S_{P'}(v', v)$  is at most  $2|uv|$  since by convexity, any convex path from  $v'$  to  $v$  contained in the triangle  $uvv'$  must be shorter in length than the sum of the lengths  $|v'u|$  and  $|uv|$ . By construction, we have  $|uv| \geq |uv'|$ , therefore  $S_{P'}(v', v)$  is at most  $2|uv|$ .

An edge of  $S_{P'}(v', v)$  has as endpoints two vertices among the neighbors of  $u$ . Let  $(s_i, s_j)$  be an arbitrary edge in  $S_{P'}(v', v)$ . Let  $D_{P'}(s_i, s_j)$  be the sequence of edges between  $s_i$  and  $s_j$  in the ordered neighbors of  $u$ . This path is in  $\mathcal{G}_{\mathcal{P}}$ . We bound the length of  $D_{P'}(s_i, s_j)$  in terms of  $|s_i s_j|$ .

Bose and Morin [7] showed (by modifying an argument in [17]) that the length of  $D_{P'}(s_i, s_j)$  is at most  $\pi/2$  times  $|s_i s_j|$ , provided that (i) the straight-line segment between  $s_i$  and  $s_j$  lies outside the Voronoi region induced by  $u$ , and (ii) that the path lies on one side of the line through  $s_i$  and  $s_j$ .

Condition (ii) holds trivially. We now show that condition (i) also holds. The vertices  $u, s_i$ , and  $s_j$  form a triangle where  $\angle s_i u s_j < 90^\circ$ . This implies that for any point  $x$  on the segment  $[s_i, s_j]$ ,  $\min\{|x s_i|, |x s_j|\} < |x u|$ . This means that  $x$  cannot be in the Voronoi region induced by  $u$ . Therefore,  $D_{P'}(s_i, s_j)/|s_i s_j| \leq \pi/2$ .

Putting it all together, it follows that the shortest path in  $\mathcal{G}_{\mathcal{P}}$  between  $v'$  and  $v$  has length at most  $\pi|uv|$ , and hence the lemma follows, since the shortest path in  $\mathcal{G}_{\mathcal{P}}$  between  $u$  and  $v$  has length at most  $(\pi + 1)|uv|$ .  $\square$

**4. Putting It All Together.** In this section we perform the analysis of algorithm PLANARSPANNER (see Figure 1), which finally proves Theorem 1.1.

We first consider the running time of the algorithm. Computing the Delaunay triangulation in Step 1 takes  $O(n \log n)$  time, see [15]. Step 2 takes  $O(n)$  time, according to Lemma 2.1, as do Steps 3 and 4 according to Sections 2.1 and 3, respectively. By the results in [19], Step 5 takes  $O(n \log n)$  time. Hence, the time complexity of the entire algorithm is  $O(n \log n)$ .

Theorem 1.1 claims that the resulting graph has four properties which we discuss below, one by one:

1.  $\mathcal{G}$  is **plane**. Since  $\mathcal{G}$  is a subgraph of  $\mathcal{G}_{\mathcal{P}}$ , and  $\mathcal{G}_{\mathcal{P}}$  is a subgraph of the Delaunay triangulation of  $\mathcal{S}$ , it is clear that  $\mathcal{G}$  is plane, proving the first claim in Theorem 1.1.
2.  $\mathcal{G}$  is a  **$t$ -spanner for  $t \approx 10$** . The Delaunay triangulation is a  $(2\pi/(3 \cos(\pi/6)))$ -spanner for  $\mathcal{S}$ , see [21]. Consider any two distinct points  $u$  and  $v$  of  $\mathcal{S}$ , and let  $Q$  be a path in the Delaunay triangulation between  $u$  and  $v$ . Consider an arbitrary Delaunay edge  $(x, y)$  on  $Q$  that is not an edge in  $\mathcal{G}_{\mathcal{P}}$ . This edge must be a diagonal in  $\mathcal{P}$ . By Lemma 3.4, there is a path in  $\mathcal{G}_{\mathcal{P}}$  between  $x$  and  $y$  whose length is at most  $(\pi + 1)|xy|$ . Therefore, there is a path in  $\mathcal{G}_{\mathcal{P}}$  between  $u$  and  $v$  of length at most  $(\pi + 1)(2\pi/(3 \cos(\pi/6)))|uv|$ . Hence,  $\mathcal{G}_{\mathcal{P}}$  is a  $(\pi + 1)(2\pi/(3 \cos(\pi/6)))$ -spanner for  $\mathcal{S}$ .

Step 5 refers to the greedy algorithm of Gudmundsson et al. [19]. This algorithm takes as input an arbitrary  $t$ -spanner, in our case  $\mathcal{G}_{\mathcal{P}} = (\mathcal{S}, \mathcal{E}_{\mathcal{P}})$ , and an arbitrary real constant  $\varepsilon > 0$ , and computes a  $((1 + \varepsilon)t)$ -spanner  $\mathcal{G} = (\mathcal{S}, \mathcal{E})$  of  $\mathcal{G}_{\mathcal{P}}$ , such that  $\mathcal{E} \subseteq \mathcal{E}_{\mathcal{P}}$ , see also item 4 below.

3.  **$\mathcal{G}$  has degree at most 27.** As described in Section 2.1, the degenerate polygon  $\mathcal{S}\mathcal{G}$  is transformed in Step 3 to a simple polygon  $\mathcal{P}$  by doubling some vertices and edges. It is easily seen that in this process, any original point  $u$  is replaced by at most three new vertices  $u_1, u_2$ , and  $u_3$ . An analysis similar to the one just above Lemma 3.4 shows that when these three vertices are merged again, the degree of  $u$  in  $\mathcal{G}_{\mathcal{P}}$  is at most  $3 \cdot 5 + 3 + \lceil \alpha_1(u_1)/90^\circ \rceil + \lceil \alpha_2(u_1)/90^\circ \rceil + \lceil \alpha_1(u_2)/90^\circ \rceil + \lceil \alpha_2(u_2)/90^\circ \rceil + \lceil \alpha_1(u_3)/90^\circ \rceil + \lceil \alpha_2(u_3)/90^\circ \rceil$ . Since all these six interior angles sum up to  $360^\circ$ , the degree of  $u$  is at most 27.
4.  **$\mathcal{G}$  has weight  $O(wt(MST))$ .** This follows from the following result in [11], [13], [14], and [19]. Let  $G$  be an arbitrary  $t$ -spanner for a point set  $\mathcal{S}$  having  $O(n)$  edges, where  $t$  is a constant, and let  $\varepsilon > 0$  be an arbitrary real constant. The greedy algorithm computes a subgraph  $G'$  of  $G$  which is a  $(1 + \varepsilon)$ -spanner of  $G$  and that satisfies the so-called leapfrog property. The graph  $G'$  can be computed in  $O(n \log n)$  time. Das and Narasimhan [13] have shown that any set of edges that satisfies the leapfrog property has weight  $O(wt(MST))$ , where  $MST$  is a minimum spanning tree of  $\mathcal{S}$ .  
 Since  $\mathcal{G}_{\mathcal{P}}$  is a  $((\pi + 1)(2\pi/(3 \cos(\pi/6))))$ -spanner for  $\mathcal{S}$ , it follows that the graph  $\mathcal{G}$  produced by the greedy algorithm is a  $(2\pi(1 + \varepsilon)(\pi + 1)/(3 \cos(\pi/6)))$ -spanner of  $\mathcal{S}$ , whose weight is  $O(wt(MST))$ .

This concludes the proof of Theorem 1.1.

**5. The Constrained Case.** The results in this paper generalize to the *constrained case*, i.e., when we are given a set  $\mathcal{S}$  of  $n$  points in the plane together with a set  $\mathcal{C}$  of edges. The endpoints of the edges are assumed to be points in  $\mathcal{S}$  and the edges of  $\mathcal{C}$  are assumed to be non-crossing.

We will show the following theorem:

**THEOREM 5.1.** *Let  $\mathcal{S}$  be a set of  $n$  points in the plane, let  $\mathcal{C}$  be a set of non-crossing edges whose endpoints belong to  $\mathcal{S}$ , and let  $\mathcal{D}(\mathcal{S}, \mathcal{C})$  be the maximal degree of any vertex in the graph  $(\mathcal{S}, \mathcal{C})$ . There is an  $O(n \log n + n \cdot \mathcal{D}(\mathcal{S}, \mathcal{C}))$ -time algorithm that computes a graph  $\mathcal{G}$*

1. *that is plane,*
2. *that is a  $t$ -spanner of the visibility graph of  $\mathcal{S}$  and  $\mathcal{C}$ , for  $t = \pi(\pi + 1)(1 + \sqrt{5})/2$ , and*
3. *in which each point of  $\mathcal{S}$  has degree at most  $8 \cdot \mathcal{D}(\mathcal{S}, \mathcal{C}) + 27$ .*

The algorithm and the analysis has to be slightly changed for the constrained case. We denote the modified algorithm as **CONSTRAINEDPLANARSPANNER**; its pseudocode is shown in Figure 9.

The first step of algorithm **CONSTRAINEDPLANARSPANNER** is to produce the *constrained Delaunay triangulation* [8], [27] of  $\mathcal{S}$  and  $\mathcal{C}$ , denoted  $\mathcal{CDT}(\mathcal{S}, \mathcal{C})$ .

Let  $G$  denote the straight-line graph described by  $\mathcal{S}$  and  $\mathcal{C}$ . A triangulation  $T$  is a constrained Delaunay triangulation of  $G$  if each edge in  $G$  is an edge of  $T$  and for each remaining edge  $e$  of  $T$  there exists a circle  $C$  with the following properties: (i) the

**Algorithm** CONSTRAINEDPLANARSPANNER( $\mathcal{S}, \mathcal{C}$ )

1.  $\mathcal{CDT} \leftarrow$  Constrained Delaunay triangulation of  $\mathcal{S}$  and  $\mathcal{C}$
2.  $\mathcal{SG} \leftarrow$  SPANNINGGRAPH( $\mathcal{CDT}$ )
3.  $\mathcal{SG}' \leftarrow \mathcal{SG} \cup \mathcal{C}$
4.  $\mathcal{P} = \{\mathcal{P}_1, \dots, \mathcal{P}_k\} \leftarrow$  TRANSFORMPOLYGON( $\mathcal{SG}'$ )
5. **for**  $i \leftarrow 1$  **to**  $k$  **do**
6.      $\mathcal{G}_{\mathcal{P}_i} \leftarrow$  POLYGONSPANNER( $\mathcal{P}_i, \mathcal{CDT}$ )
7.  $\mathcal{G} \leftarrow \mathcal{G}_{\mathcal{P}_1} \cup \dots \cup \mathcal{G}_{\mathcal{P}_k}$
8. output  $\mathcal{G}$

**Fig. 9.** The construction of a constrained plane sparse spanner.

endpoints of  $e$  are on the boundary of  $\mathcal{C}$ , and (ii) if any vertex of  $\mathcal{G}$  is in the interior of  $\mathcal{C}$  then it cannot be “seen” from at least one of the endpoints of  $e$ .

Next a spanning graph  $\mathcal{SG}$  is computed. As noted in Section 2, the spanning graph algorithm can be run for any given triangulation, hence  $\mathcal{SG}$  is, as before, a spanning graph of degree 3 that includes the convex hull of  $\mathcal{S}$ . Let  $\mathcal{SG}'$  denote the graph obtained by adding the edges in  $\mathcal{C}$  to the spanning graph  $\mathcal{SG}$ . The degree of  $\mathcal{SG}'$  is bounded by  $3 + \mathcal{D}(\mathcal{S}, \mathcal{C})$ , where  $\mathcal{D}(\mathcal{S}, \mathcal{C})$  is the maximal degree of any vertex in the graph  $(\mathcal{S}, \mathcal{C})$ . In the basic algorithm the spanning graph contained exactly  $n$  edges and it included the convex hull of  $\mathcal{S}$ , hence  $\mathcal{SG}$  described a, possibly degenerate, hole-free polygon. This is not true for the constrained case since  $\mathcal{SG}'$  will be a graph with more than  $n$  edges although it will include the convex hull of  $\mathcal{S}$ . Instead the graph  $\mathcal{SG}'$  might partition the convex hull of  $\mathcal{S}$  into many, possibly degenerate, polygons. Again, it is not hard to see how  $\mathcal{SG}'$  can be transformed into a set of simple polygons,  $\mathcal{P}_1, \dots, \mathcal{P}_k$ , by splitting some vertices and doubling some edges as described in Section 2.1. Hence, every vertex can be replaced by at most  $3 + \mathcal{D}(\mathcal{S}, \mathcal{C})$  new vertices. This transformation does not affect the complexity of the spanner algorithm if the vertices are only split symbolically.

Next, each of the  $k$  polygons is processed by running algorithm POLYGONSPANNER described in Section 3. Hence, for each simple polygon  $\mathcal{P}_i$ ,  $1 \leq i \leq k$ , a spanner  $\mathcal{G}_{\mathcal{P}_i}$  is constructed with the following two properties:

1. every vertex in  $\mathcal{G}_{\mathcal{P}_i}$  has bounded degree, and
2. for every edge  $(x, y)$  in  $\mathcal{CDT}$  that is a diagonal in  $\mathcal{P}_i$ , there is a path between  $x$  and  $y$  in  $\mathcal{G}_{\mathcal{P}_i}$  whose length is at most  $(\pi + 1)$  times  $\mathcal{D}_{\mathcal{CDT}}(x, y)$ , where  $\mathcal{D}_{\mathcal{CDT}}(x, y)$  denotes the shortest distance in  $\mathcal{CDT}$  between  $x$  and  $y$ .

Theorem 5.1 claims that the resulting graph has three properties which we discuss below. As above we first consider the running time of the algorithm. Computing the constrained Delaunay triangulation in Step 1 takes  $O(n \log n)$  time (see [8] and [27]). Step 2 takes  $O(n)$  time, according to Lemma 2.1, as does Step 3. In Step 4 the graph is transformed into a set of simple polygons by splitting edges and vertices. Every vertex and edge is split at most  $(\mathcal{D}(\mathcal{S}, \mathcal{C}) + 3)$  times, hence the time to perform Step 4 is  $O(n \cdot \mathcal{D}(\mathcal{S}, \mathcal{C}))$ . The same bound holds for Steps 5–7.

Hence, the time complexity of the algorithm is  $O(n \log n + n \cdot \mathcal{D}(\mathcal{S}, \mathcal{C}))$ . We now turn our attention to the three claimed properties of  $\mathcal{G}$ :

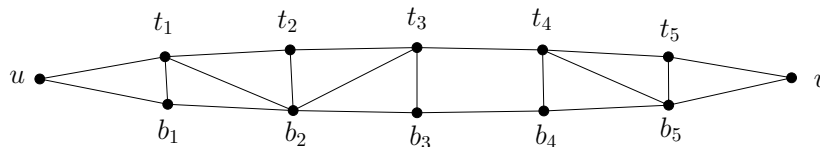
1.  $\mathcal{G}$  is plane. Since  $\mathcal{G}$  is a subgraph of the constrained Delaunay triangulation of  $\mathcal{S}$  and  $\mathcal{C}$ , it follows that  $\mathcal{G}$  is plane.
2.  $\mathcal{G}$  is a **(21.04)-spanner of the complete visibility graph on  $\mathcal{S}$  and  $\mathcal{C}$** . One cannot compare  $\mathcal{CDT}$  with the complete graph, instead we compare it with the visibility graph of  $\mathcal{S}$  and  $\mathcal{C}$ . The visibility graph  $\mathcal{VIS}(\mathcal{S}, \mathcal{C})$  of  $\mathcal{S}$  and  $\mathcal{C}$  is the undirected graph that has  $\mathcal{S}$  as a vertex set and in which two vertices are connected by an edge whenever they “see” each other, i.e., the open line segment joining them is disjoint from all segments in  $\mathcal{C}$  or is contained in a segment. It is straightforward to modify Dobkin et al.’s [17] proof to show that  $\mathcal{CDT}(\mathcal{S}, \mathcal{C})$  is a  $(\pi(1 + \sqrt{5})/2)$ -spanner of  $\mathcal{VIS}(\mathcal{S}, \mathcal{C})$ , which implies that  $\mathcal{G}$  is a  $(\pi(\pi + 1)(1 + \sqrt{5})/2)$ -spanner of  $\mathcal{VIS}(\mathcal{S}, \mathcal{C})$ , since  $\mathcal{G}$  is a  $(\pi + 1)$ -spanner of  $\mathcal{CDT}(\mathcal{S}, \mathcal{C})$ .
3.  $\mathcal{G}$  has degree at most  $8\mathcal{D}(\mathcal{S}, \mathcal{C}) + 27$ . As described in Section 2.1, a degenerate polygon  $\mathcal{SG}$  is transformed in Step 3 to a set  $\mathcal{P}$  of simple polygons by doubling some vertices and edges. In this process any original point  $u$  is replaced by at most  $q := 3 + \mathcal{D}(\mathcal{S}, \mathcal{C})$  new vertices  $u_1, \dots, u_q$ . An analysis similar to the one for the standard case shows that when these  $q$  vertices are merged again, the degree of  $u$  in  $\mathcal{G}_{\mathcal{P}}$  is at most  $q \cdot 5 + q + \lceil \alpha_1(u_1)/90^\circ \rceil + \lceil \alpha_2(u_1)/90^\circ \rceil + \dots + \lceil \alpha_1(u_q)/90^\circ \rceil + \lceil \alpha_2(u_q)/90^\circ \rceil$ . Since all these  $q$  interior angles sum up to at most  $360^\circ$ , the degree of  $u$  is at most  $8q + 3 = 8\mathcal{D}(\mathcal{S}, \mathcal{C}) + 27$ .

This concludes the proof of Theorem 5.1.

**6. Concluding Remarks.** There are additional properties that would be interesting to consider, for example minimizing the spanner diameter. Below we show that this property cannot be obtained for plane spanners.

**THEOREM 6.1.** *For any real number  $t > 1$ , there exist a set  $\mathcal{S}$  of  $n$  points and two points  $u, v \in \mathcal{S}$  such that in any plane  $t$ -spanner of  $\mathcal{S}$ , any  $t$ -spanner path between  $u$  and  $v$  contains at least  $n/2 + 1$  edges.*

**PROOF.** The construction of the point set is illustrated in Figure 10. For simplicity, we assume that  $n \geq 8$  and  $n$  is even. Let  $m = (n - 2)/2$  and define the points  $u = (0, 0)$ ,  $v = (tn/2, 0)$ , and  $t_i = (it, 1/m)$  and  $b_i = (it, -1/m)$  for  $1 \leq i \leq m$ . Define  $\mathcal{S} = \{u, v, t_1, \dots, t_m, b_1, \dots, b_m\}$ .



**Fig. 10.** The proof of Theorem 6.1.

Observe that  $0 < |t_i b_i| = 2/m < 1$  and  $|t_i t_{i+1}| = |b_i b_{i+1}| = t$  for all  $i$ . Moreover,  $|ut_1| = |ub_1| = |vt_m| = |vb_m| > t$ . Therefore, every  $t$ -spanner of  $\mathcal{S}$  must contain all edges  $(t_i, b_i)$ ,  $1 \leq i \leq m$ . Moreover, if this  $t$ -spanner is plane, then it does not contain any of the edges  $(t_i, b_j)$ ,  $(t_i, t_j)$ , and  $(b_i, b_j)$  with  $|j - i| \geq 2$ . It follows that every path in a plane  $t$ -spanner between  $u$  and  $v$  passes through either  $b_i$  or  $t_i$  for every  $1 \leq i \leq m$ . The lemma follows.  $\square$

**Acknowledgements.** The authors thank Anil Maheshwari, Pat Morin, and David Wood for fruitful discussions on this topic.

## References

- [1] I. Althöfer, G. Das, D. Dobkin, and D. Joseph. Generating sparse spanners for weighted graphs. In *Proc. 2nd Scandinavian Workshop on Algorithm Theory*, pages 26–37, 1990.
- [2] I. Althöfer, G. Das, D. P. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9:81–100, 1993.
- [3] S. Arikati, D. Z. Chen, L. P. Chew, G. Das, M. Smid, and C. D. Zaroliags. Planar spanners and approximate shortest path queries among obstacles in the plane. In *Proc. 4th European Symposium on Algorithms*, pages 514–528, 1996.
- [4] S. Arya, G. Das, D. M. Mount, J. S. Salowe, and M. Smid. Euclidean spanners: short, thin, and lanky. In *Proc. 27th Annual ACM Symposium on Theory of Computing*, pages 489–498, 1995.
- [5] D. Barnette. Trees in polyhedral graphs. *Canadian Journal of Mathematics*, 18:731–736, 1966.
- [6] P. Bose, J. Gudmundsson, and P. Morin. Ordered theta graphs. *Computational Geometry - Theory and Applications*, 28(1):11–18, 2004.
- [7] P. Bose and P. Morin. Online routing in triangulations. In *Proc. 10th Annual International Symposium on Algorithms and Computation*, pages 113–122. Volume 1741 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1999.
- [8] P. Chew. Constrained delaunay triangulations. *Algorithmica*, 4:97–108, 1989.
- [9] M. Chrobak and T. Payne. A linear-time algorithm for drawing a planar graph on a grid. *Information Processing Letters*, 54:241–246, 1995.
- [10] A. Czumaj and H. Zhao. Fault-tolerant geometric spanners. In *Proc. 19th Symposium on Computational Geometry*, pages 1–10, 2003.
- [11] G. Das, P. Heffernan, and G. Narasimhan. Optimally sparse spanners in 3-dimensional Euclidean space. In *Proc. 9th Annual ACM Symposium on Computational Geometry*, pages 53–62, 1993.
- [12] G. Das and D. Joseph. Which triangulations approximate the complete graph? In *Proc. International Symposium on Optimal Algorithms*, pages 168–192. Volume 401 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1989.
- [13] G. Das and G. Narasimhan. A fast algorithm for constructing sparse Euclidean spanners. *International Journal of Computational Geometry and Applications*, 7:297–315, 1987.
- [14] G. Das, G. Narasimhan, and J. Salowe. A new way to weigh malnourished Euclidean graphs. In *Proc. 6th ACM-SIAM Symposium on Discrete Algorithms*, pages 215–222, 1995.
- [15] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*, 2nd edition. Springer-Verlag, Berlin, 2000.
- [16] H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10:41–51, 1990.
- [17] D. P. Dobkin, S. J. Friedman, and K. J. Supowit. Delaunay graphs are almost as good as complete graphs. *Discrete & Computational Geometry*, 5:399–407, 1990.
- [18] D. Eppstein. Spanning trees and spanners. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 425–461. Elsevier Science, Amsterdam, 2000.
- [19] J. Gudmundsson, C. Levcopoulos, and G. Narasimhan. Improved greedy algorithms for constructing sparse geometric spanners. *SIAM Journal of Computing*, 31(5):1479–1500, 2002.

- [20] J. M. Keil. Approximating the complete Euclidean graph. In *Proc. 1st Scandinavian Workshop on Algorithm Theory*, pages 208–213, 1988.
- [21] J. M. Keil and C. A. Gutwin. Classes of graphs which approximate the complete Euclidean graph. *Discrete & Computational Geometry*, pages 13–28, 1992.
- [22] C. Levcopoulos and A. Lingas. There are planar graphs almost as good as the complete graphs and almost as cheap as minimum spanning trees. *Algorithmica*, 8:251–256, 1992.
- [23] C. Levcopoulos, G. Narasimhan, and M. Smid. Improved algorithms for constructing fault-tolerant spanners. *Algorithmica*, 32:144–156, 2002.
- [24] J. S. Salowe. Constructing multidimensional spanner graphs. *International Journal of Computational Geometry & Applications*, 1(2):99–107, 1991.
- [25] M. Smid. Closest point problems in computational geometry. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 877–935. Elsevier Science, Amsterdam, 2000.
- [26] P. M. Vaidya. A sparse graph almost as good as the complete graph on points in  $K$  dimensions. *Discrete & Computational Geometry*, 6:369–381, 1991.
- [27] C. A. Wang and L. K. Schubert. An optimal algorithm for constructing the delaunay triangulation of a set of line segments. In *Proc. 3rd Annual ACM Symposium on Computational Geometry*, pages 223–232, 1987.