Establishing Order in Planar Subdivisions†

David G. Kirkpatrick‡

Technical Report 87-12

May 1987

Abstract

A planar subdivision is the partition of the plane induced by an embedded planar graph. A representation of such a subdivision is *ordered* if, for each vertex v of the associated graph G, the (say) clockwise sequence of edges in the embedding of G incident with v appears explicitly.

The worst-case complexity of establishing order in a planar subdivision - i.e. converting an unordered representation into an ordered one - is shown to be $\Theta(n+\log \lambda(G))$, where n is the size (number of vertices) of the underlying graph G and $\lambda(G)$ is the number of topologically distinct embeddings of G in the plane.

† A preliminary version of this paper will appear in the Proceedings of the Third Annual ACM Symposium on Computational Geometry.

‡ Department of Computer Science, University of British Columbia, Vancouver, British Columbia, Canada V6T 1W5.

1. Introduction

A graph $G=(V,E)$ is said to be *planar* if it has an embedding in the plane (that is, a drawing with each vertex $v \in V$ represented as a point and each edge $e \in E$ represented as a simple continuous curve connecting its endpoints) such that no two edges intersect except at their endpoints. Such a drawing $\hat{G}$ is called a *planar embedding* of G. A planar embedding $\hat{G}$ induces a partition of the plane into regions or faces bounded by the edges of $\hat{G}$. We call this the *planar subdivision* associated with $\hat{G}$. If each of the edges of $\hat{G}$ is a straight line segment (respectively, a sequence of line segments) the associated subdivision is called a *straight-edge subdivision* (respectively, a *linear subdivision).* It is well known that all planar graphs can be realized as straight-edge subdivisions [F].

Since planar graphs and their associated subdivisions arise naturally in the description of polyhedra and planar maps and, either directly or indirectly, in the formulation or solution of numerous problems in computational geometry [PSh], it is important to establish standard representations for embedded planar graphs. Among the standard representations that have been proposed - and employed in solid modeling and the formulation of fundamental geometric algorithms - are the "winged edge" representation [B], the doubly-connected-edge-list (DCEL) representation [MP,PSh], and the quad-edge data structure [GS,EGS]. Each of these is a variant of the same basic structure in which the edges are given in sorted order about each of the vertices. They have the desirable properties that:

i) other natural representations are easily shown to be linear time equivalent (i.e. transformations between representations can be carried out in time linear in the size of the representation) [MP,GS];

ii) the representations extend to more general settings, for example, the modeling of

general polyhedral surfaces [B] and embeddings of undirected graphs in arbitrary two-dimensional manifolds [GS]; and

iii) algorithms which take as their input an embedded planar graph can be described in a natural and efficient manner in terms of these representations. Examples include algorithms for regularizing, triangulating, and dualizing embedded planar graphs and preprocessing planar subdivisions for fast point location [EGS,GS,LP,K,PS,TVW].

In certain applications the construction of one of these standard subdivision representations may require more than linear time and, as a consequence, this initialization may dominate the complexity of the associated algorithm. Such a situation can arise, for example, when the input is a straight-edge subdivision presented as a list of vertices, together with their location in the plane, and an *unordered* list of vertex pairs describing the edge set. More generally, each (not necessarily straight) edge of a subdivision may have its embedding described by the angles at which it meets its two endpoints. This leads us to ask what the cost is, in general, of converting such primitive (unordered) subdivision representations into one of the standard (ordered) representations. Since it suffices to order all of the edges (by angle) about each vertex, we call this the problem of *establishing order in a planar subdivision.*

Note that we are not concerned with checking that the given embedding is indeed planar (i.e. none of the edges have forbidden intersections). We know of no general method for checking the planarity of straight-edge embeddings that is $o(n \log n)$. Rather, we are asking how expensive it is to recover the (implicit) topological structure of a planar subdivision given only the combinatorial structure of the underlying graph and the basic geometrical structure of its embedding.

First, consider the naive algorithm which explicitly sorts (by angle) all of the edges incident with each vertex. This takes $\Theta(\sum_{v_i \in V} d_i \log d_i)$ time, where d_i denotes the degree (number of incident edges) of vertex v_i. For graphs of bounded degree this is $\Theta(n)$, but in general it can be as bad as $\Theta(n \log n)$. That $\Theta(n \log n)$ time might be required for certain graphs is easy to see by considering, for example, the graph $K_{1,n-1}$, the star with n–1 rays. It is straightforward to reduce the problem of sorting to the problem of establishing order in a subdivision with underlying graph $K_{1,n-1}$. Hence, an $\Omega(n \log n)$ lower bound follows in rather general computational settings [Kn,PSi].

The example of $K_{1,n-1}$ might lead us to conjecture that the naive algorithm is optimal. However, a moment's reflection reveals that linear algorithms exist even for certain cases when the underlying graph does not have bounded degree. It is well known, for example, that three-connected planar graphs have (essentially) a unique planar embedding [E]. Thus, linear time planarity algorithms [HT1,LEC] which promise only to demonstrate *an* embedding will, in fact, construct *the* embedding, in this case. In other words, the topological structure is implied by the combinatorial structure alone.

We show that it is the number of distinct embeddings of the underlying graph, rather than the magnitude of its vertex degrees, that contributes to the complexity of establishing order in planar subdivisions. Let S be a planar subdivision associated with a planar embedding $\hat{G}$ of the planar graph G. We denote by $\lambda(G)$ the number of topologically distinct embeddings of G in the plane. In section 2, we present and analyze an algorithm for establishing order in S in $O(n+log(\lambda(G)))$ time. Section 3 shows that $\Omega(n+log(\lambda(G)))$ time is required for an associated decision problem on arbitrary d-th order algebraic decision trees.

2. Establishing order by repeated face contraction

Algorithm overview

Our algorithm for establishing order in linear planar subdivisions is based on three simple observations:

i) establishing order at low degree vertices is cheap;

ii) the average degree in a planar graph is low; and

iii) knowing the order at certain vertices and the combinatorial structure of the graph, constrains the order at other vertices.

It is the detection, representation and utilization of the constraints mentioned in observation (iii) that makes the algorithm non-trivial.

The algorithm proceeds by a sequence of removals of vertices of low effective degree. The *effective degree* of a vertex v in a multigraph G is just the number of distinct endpoints of edges leaving vertex v. (Thus, parallel edges contribute only one to the effective degree.) The removal of a vertex v is a two stage process. First, the ordering of edges incident with v (which may be partially known) is completed. Second, every face which is incident on v is "shrunk" by replacing each successive pair of edges uv, vw incident on v by the single edge uw. These replacement edges are the source of possible parallel edges. However, since no new faces are introduced, edges of multiplicity m reveal pairs of vertices whose removal leads to m connected components. A face which has been shrunk to a simple loop on a vertex v either reveals a cutpoint of the graph (namely the vertex v) or holds information concerning adjacent edges in the ordering about v. Once this information has been recorded such a loop can be eliminated. The planarity of the underlying graph

guarantees that the average effective degree of vertices is always constant. Of course, since faces are removed only when they have shrunk to a single edge, the actual degree of vertices with low effective degree may be large. The crucial step in the analysis is to show that this degree descrepancy is related to the number of embeddings of the original graph.

To simplify the description we will assume that the input subdivision $\hat{G}$ is a straight-edge subdivision. The generalization of our algorithm to linear subdivisions (or even appropriately represented non-linear subdivisions) is straightforward.

The data structure

The data structure has two kinds of edges, *primitive edges* and *general edges*. We denote a typical primitive edge by e and a typical general edge by $\tilde{e}$. All of the edges of the initial subdivision are represented by symmetric pairs of primitive edges; e.sym points to the symmetric companion of primitive edge e. The field e.next, when non-null, points to the clockwise next edge following e out of the same vertex. Initially all e.next pointers are null. Indeed, the objective of the entire algorithm is to correctly set these pointers.

General edges represent sequences of primitive edges that enclose, in whole or in part, some sequence of faces incident with a fixed vertex. General edges are either *active* or *removed*. At the start, one general edge is constructed for each primitive edge. Edge $\tilde{e}$ is directed from vertex $\tilde{e}$.foot to vertex $\tilde{e}$.head. The fields $\tilde{e}$.hind_leg and $\tilde{e}$.neck point to the first and last primitive edge of $\tilde{e}$ respectively. $\tilde{e}$.front_leg points to the last primitive edge of $\tilde{e}$ that emerges from $\tilde{e}$.foot. Initially, $\tilde{e}$.hind_leg $=\tilde{e}$.front_leg $=\tilde{e}$.neck $=e$ if $\tilde{e}$ is associated with primitive edge e. General edges have four other fields which point to related general edges. $\tilde{e}$.next_in points to the counterclockwise next (active) edge entering

$\tilde{e}$.foot (This is initialized to be the general edge associated with ($\tilde{e}$.foot).sym). Similarly, $\tilde{e}$.next_out points to the clockwise next (active) edge out of $\tilde{e}$.head (initially, the general edge associated with ($\tilde{e}$.head).sym) (see Figure 1). $\tilde{e}$.active_succ is a pointer (initially null) that is ultimately used to record the clockwise ordered list of active edges around $\tilde{e}$.foot. Finally, for each edge $\tilde{e}$ we associate an angle in $[-2\pi,2\pi]$ that records the cummulative angle of displacement of successive primitive edges on $\tilde{e}$ starting at $\tilde{e}$.front_leg. This angle is used to determine, when a general edge closes to form a loop, whether the loop has clockwise or counterclockwise orientation.

< Figure 1 >

High level description

The following algorithm maintains the invariant that for each vertex v and each active edge $\tilde{e}$ out of v, all primitive edges out of v whose angles of incidence with v are clockwise greater than or equal to that of $\tilde{e}$.hind_leg and less that that of $\tilde{e}$.front_leg have their .next pointers correctly set.

```
1. (* initialize *)
   initialize general edges from primitive edges
   sort all of the lists of active edges on the ẽ.head field
   A ← V

2. while |A|>0 do

   2.1. for each v∈A
          if effective_degree(v)<12
            then choose v
```

2.2. **for** each chosen vertex v

2.2.1 sort the active edges $\tilde{e}$ of v on their $\tilde{e}$.hind_leg angles
assign $\tilde{e}$.active_succ links for each such edge $\tilde{e}$.
2.2.2 (* shrink faces incident with v *)
for each active edge $\tilde{e}$ out of v
$\hat{e} \leftarrow (\tilde{e}.\text{active\_succ}).\text{next\_in}$
$\hat{e}.\text{neck} \leftarrow \tilde{e}.\text{neck}$
$\hat{e}.\text{head} \leftarrow \tilde{e}.\text{head}$
update $\hat{e}$.angle
remove $\tilde{e}$
move $\hat{e}$ to end of active list for $\hat{e}$.foot

2.2.3 (* complete the .next ring about v *)
for each active edge $\tilde{e}$ out of v
$(\tilde{e}.\text{front\_leg}).\text{next} \leftarrow (\tilde{e}.\text{active\_succ}).\text{hind\_leg}$

2.2.4 (* deactivate v *)
remove v from A

2.3. **for** each $w \in A$

(* remove collapsed faces incident on w *)
for each active edge $\tilde{e}$ out of w with $\tilde{e}.\text{head} = w$ and $\tilde{e}.\text{angle} > 0$
$(\tilde{e}.\text{front\_leg}).\text{next} \leftarrow (\tilde{e}.\text{next\_out}).\text{hind\_leg}$
$\tilde{e}.\text{front\_leg} \leftarrow (\tilde{e}.\text{next\_out}).\text{front\_leg}$
$\tilde{e}.\text{neck} \leftarrow (\tilde{e}.\text{next\_out}).\text{neck}$
$\tilde{e}.\text{head} \leftarrow (\tilde{e}.\text{next\_out}).\text{head}$
$\tilde{e}.\text{next\_out} \leftarrow (\tilde{e}.\text{next\_out}).\text{next\_out}$
$((\tilde{e}.\text{next\_out}).\text{next\_out}).\text{next\_in} \leftarrow \tilde{e}$
remove $\tilde{e}$.next_out

2.4. resort each active edge list on the $\tilde{e}$.head field

Correctness and complexity analysis

The correctness of the algorithm follows by analyzing the cases where e.next pointers are set in steps 2.3 (when a face has collapsed) and 2.2.3 (when all active edges with the same foot as e have been ordered). In each case it is straightforward to confirm that the

invariant is preserved. It follows that when a vertex v becomes inactive all of its incident primitive edges have been ordered.

The algorithm maintains a set A of active vertices and, for each vertex $v \in A$, a list of active (general) edges. Each of these lists is maintained in sorted order on the $\tilde{e}$.head field (i.e. on the names w of neighbouring vertices). So that these lists can be processed (in steps 2.1 and 2.4) in time proportional to the effective degree of the associated vertex, they must be augmented with pointers permitting blocks of consecutive edges with identical $\tilde{e}$.head values - what we have called parallel edges - to be skipped in constant time. The initial sorting of these edge lists (in step 1) can be carried out in $O(n)$ steps by a straightforward two-phase bucket sort (here we assume that the vertices have labels $1,\ldots,n$). Similarly, the resorting in step 2.4 is $O(|A| + \sum_{v \text{ chosen}} deg_G(v))$. But, since $|A|$ decreases geometrically with each execution of the step 2, and since a vertex v is chosen only once, the total cost of step 2.4 over all executions of step 2 is $O(n)$.

The remainder of the algorithm is analyzed in a step-by-step fashion.

step 2.1: this takes $O(1)$ time for each active vertex ($O(|A|)$ in total) by scanning the edge lists (using the auxiliary pointers).

step 2.2.1: this takes $O(d(v) log\ d(v))$ time where $d(v)$ denotes the number of active edges incident with vertex v.

step 2.2.2: cost is $O(d(v))$

step 2.2.3: cost is $O(d(v))$

step 2.2.4: cost is $O(1)$

step 2.2: total cost is $O(d(v) log\ d(v))$ by the above.

step 2.3: cost is $O(|A|+f_v)$ where f_v denotes the number of discarded faces incident on v.

By the earlier arguments this cost totals $O(n)$ over all executions of step 2.

In summary,

Lemma 2.1. The algorithm establishes order in $\hat{G}$ in time $O(n+\sum_{n\in V}\delta(v)\log\delta(v))$, where $\delta(v)$ denotes the number of active edges incident with v when it is chosen for removal.

It remains to relate the expression $\sum_{n\in V}\delta(v)\log\delta(v)$ to $\lambda(G)$. To this end, it is helpful to consider the number of embeddings $\lambda(\tilde{G})$ of the structure $\tilde{G}$ given by an instantaneous description of our basic data structure. $\tilde{G}$ can be viewed as a *partial embedding* of G.

Lemma 2.2. If $\tilde{G}_2$ is formed from $\tilde{G}_1$ by the deletion of a single vertex v with effective degree $\leq d_0$ (step 2.2) then $\lambda(\tilde{G}_1)\geq\lambda(\tilde{G}_2)\cdot(\frac{d(v)}{d_0}-1)!$.

Proof. Note that some edge incident with v must have multiplicity m at least $d(v)/d_0$. There are two cases. If this multiedge is a loop then v must be a cutpoint of the original graph incident with at least m biconnected components. These components can be embedded in $(m-1)!$ ways around v. Alternatively, if the multiedge has an endpoint w different from v then v and w must lie on m distinct faces of $\hat{G}$. These faces (and the components of G that they separate) can be arranged in $(m-1)!$ ways between v and w.

Corollary 2.3.

$$\log(\lambda(G)) \geq \Omega(\sum_{v\in V}\delta(v)\log\delta(v))$$

Proof. It suffices to observe that the initial data structure $\tilde{G}$ satisfies $\lambda(\tilde{G})=\lambda(G)$. Furthermore, the algorithm modifies the data structure by a sequence of deletions of vertices with low effective degree.

Combining the results above, we have

Theorem 2.4. Order can be established in an arbitrary n vertex linear planar subdivision with underlying graph G in $O(n+log\ \lambda(G))$ time.

3. A lower bound

The notion of a planar embedding of a graph can be relaxed slightly to produce what we call a coherent embedding. An embedding of a graph G in the plane is said to be *coherent* if no two of its edges cross. Curve e_1 is said to *cross* curve e_2 if $e_1-(e_1\cap e_2)$ is disconnected. A coherent embedding is said to be *discrete* if it is a planar embedding. (A coherent embedding is not discrete if two or more of its edges coincide along some interval incident with their common endpoint.) A straightforward byproduct of any algorithm that establishes order in a linear subdivision is an algorithm that determines whether or not a given coherent linear embedding of a graph G is discrete. Thus any lower bound on the complexity of determining the discreteness of coherent linear embeddings translates to a lower bound on establishing order in linear subdivisions. The problem of determining the discreteness of a coherent linear embedding of a graph G - what we call the *discrete embedding problem for* G - is a (somewhat artificial) generalization of the *element uniqueness problem* [DL] - given n real numbers, determine if they are all distinct. The latter is well known to require $\Omega(n \log n)$ time on fixed-order algebraic decision trees [B-O]. This is an

immediate corollary of the following general result due to Ben-Or.

Theorem 3.1. [B-O, Thm. 8] Let $W \subset \mathbf{R}^n$ be any set and let T be any dth order algebraic decision tree that solves the membership problem for W. If W has N disjoint connected components, then T must have height $\Omega(log\ N - n)$.

Theorem 3.1 allows us to prove lower bounds on a powerful model of computation by counting distinct configurations. For our application the relevant counting argument is summarized in the following:

Lemma 3.2. The space of all encodings of planar linear embeddings of a planar graph G (i.e. coherent embeddings that are also discrete) has at least $\lambda(G)$ disjoint connected components.

Proof. It is impossible to move continuously between two distinct planar embeddings of G without at some point violating the discreteness condition.

Q.E.D.

We conclude that,

Theorem 3.3. For any given planar graph G with n vertices the discrete embedding problem for G requires $\Omega(n+log(\lambda(G)))$ steps on any fixed order algebraic decision tree.

4. Conclusion

An alternative approach to establishing order in planar subdivisions might be to first find the triconnected components (which can be done in $O(n)$ time [HT2]) and then to

establish the order among these components using the available geometric information. What we have demonstrated is that we do not need to use all of the machinery that seems necessary to identify triconnected components in arbitrary graphs, if we are given as part of the input a description of a planar embedding - a reasonable assumption in many geometric applications.

The reader may (with good reason) object that our model for constructing lower bounds - dth order algebraic decision trees - differs from that assumed in the description of our algorithm. This is only partly accounted for by the fact that our lower bound addresses a related decision problem. A more complete explanation for the model differences follows by considering the constraint of *uniformity*. Our algorithm *uniformly* establishes order in planar subdivisions - that is, the same algorithm can be used for all underlying planar graphs. The lower bound holds even if we fix the underlying graph. If we chose to fix the graph G then our algorithm could - at significant cost in clarity - be formulated as a fixed order - in fact quadratic - algebraic decision tree solving the discrete embedding problem for G. It remains an open question what the inherent cost is of establishing order in planar subdivisions on models of computation that permit unconstrained use of all of the operations employed in our algorithm.

Acknowledgement

This work was supported by the National Science and Engineering Research Council of Canada under grant A3583.

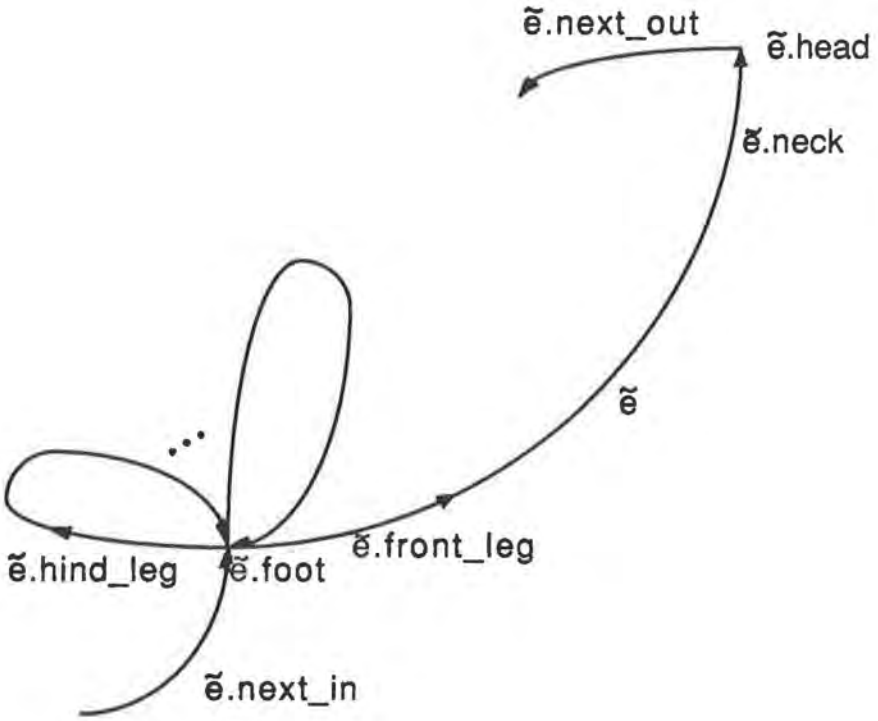

Figure 1
A general edge $\tilde{e}$ and related information

References

[AHU] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley, Reading, MA, 1974.

[B] B.G. Baumgart, A polyhedron representation for computer vision, *AFIPS Conference Proceedings,* Vol. 44, 1975 National Computer Conference, pp. 589-596.

[B-O] M. Ben-Or, Lower bounds for algebraic computation trees, *Proc. 15th Annual ACM Symposium on Theory of Computing,* 1983, pp. 80-86.

[DL] D.P. Dobkin and R. Lipton, On the complexity of computations under varying sets of primitives, *Journal of Computer and System Sciences* 18 (1979), 86-91.

[EGS] H. Edelsbrunner, L.J. Guibas and J. Stolfi, Optimal point location in a monotone subdivision, *SIAM J. Comput.* 15, 2 (1986), pp. 317-340.

[E] S. Even, *Graph Algorithms,* Computer Science Press, Potomac, MD, 1979.

[F] I. Fary, On straight line representation of planar graphs, *Acta Sci. Math. Szeged.* 11 (1948), pp. 229-233.

[GRS] L. Guibas, L. Ramshaw and J. Stolfi, A kinetic framework for computational geometry, *Proc. 24th Annual IEEE Symposium on Foundations of Computer Science,* 1983, pp. 100-111.

[GS] L.Guibas and J. Stolfi, Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams, *Proc. 15th Annual ACM Symposium on Theory of Computing,* 1983, pp. 221-234.

[HT1] J. Hopcroft and R. Tarjan, Efficient planarity testing, *JACM* 21, 4 (1974), pp. 549-568.

[HT2] J. Hopcroft and R. Tarjan, Dividing a graph into triconnected components, *SIAM J. Comput.* 2, 3 (1973), pp. 135-158.

[K] D.G. Kirkpatrick, Optimal search in planar subdivisions, *SIAM J. Comput.* 12, 1 (1983), pp. 28-35.

[Kn] D.E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching,* Addison-Wesley, Reading, MA, 1973.

[LP] D.T. Lee and F.P. Preparata, Location of a point in a planar subdivision and its

applications, *SIAM J. Comput.* 6, 1 (1977), pp. 594-606.

[LEC] A Lempel, S. Even and I. Cederbaum, An algorithm for planarity testing of graphs, *Theory of Graphs, International Symposium,* Rome, July 1966, P. Rosenstiehl (ed.), Gordon and Breach, N.Y. 1967, pp. 215-232.

[MP] D.E. Muller and F.P. Preparata, Finding the intersection of two convex polyhedra, Theoretical Computer Science 7, 2 (1978), pp. 217-236.

[PSi] W.J. Paul and J. Simon, Decision trees and random access machines, *Symp. über Logic und Algorithmik,* Zürich, 1980.

[PSh] F.P. Preparata and M.I. Shamos, *Computational Geometry,* Springer-Verlag, New York, N.Y. 1985.

[T] R.E. Tarjan, Depth-first search and linear graph algorithms, *SIAM J. Computing* 1 (1972), pp. 146-160.

[TvW] R.E. Tarjan and C.J. Van Wyk, An $O(n \log \log n)$-time algorithm for triangulating simple polygons, unpublished manuscript, 1986.