1 Connected Planar Subdivisions

A two-dimensional planar subdivision is defined by a set of segments that do not intersect, except possibly at their endpoints. These segments, which may or may not be straight line segments, partition the plane into 2-dimensional connected regions, called *cells*. Typically, computations involving subdivisions require that we be able to traverse a cell, marching around its boundary, that we be able to walk from one cell to an adjacent cell, and that we be able to enumerate in radial order the segments touching a particular endpoint.

1.1 The Size of Planar Subdivisions

Using the language of graph theory, a subdivision is an embedded planar graph $G = (V, E)$, where the segment endpoints are the *vertices* V and the segments define *edges* in E, and collectively, these vertices and edges partition the plane to form cells that define a set of *faces* F. If the graph G is connected, as will often be the case in geometric applications, then the sizes of these respective sets are related according to an equation known as *Euler's formula*.

Theorem 1 (Euler's Formula) *If G is a connected embedded planar graph, then*

$$|V| - |E| + |F| = 2.$$

Proof: We prove this formula by induction on the number of faces in G. If $|F| = 1$, then G is a tree; hence $|E| = |V| - 1$, which establishes the formula. Consider, then, the case $|F| \geq 2$, assuming the formula is true for graphs with fewer than $|F|$ faces. Let e be an edge in G whose removal does not disconnect G (such an edge must exist). Remove e from G, and note that this results in two faces of G "fusing" together to form a new face. That is, the resulting graph G' has $|F| - 1$ faces; hence, we can apply the induction hypothesis, observing that G' also has one fewer edge, so that for G' we have

$$|V| - (|E| - 1) + (|F| - 1) = 2.$$

This, of course, implies that for G we have

$$|V| - |E| + |F| = 2,$$

which establishes the theorem. ∎

An important implication of this formula, which is often used in the analysis of algorithms, is the following.

Corollary 2 *Let G is a connected embedded planar graph with no parallel edges or self-loops. Then $|E| \leq 3|V| - 6$ and $|F| \leq 2|V| - 4$.*

Proof: Let $d(f)$ denote the *degree* of a face f in F, that is, the number of times one transitions from one edge to another that occur in a walk around the boundary of f. So,

$$\sum_{f \in F} d(f) = 2|E|,$$

since each edge will be counted twice by faces in G—once on each side. Also, since G has no parallel edges or self-loops, each face of G must have at least three edges; hence,

$$\sum_{f \in F} d(f) \geq 3|F|,$$

which implies that $2|E| \geq 3|F|$. Plugging this into Euler's formula, we get that

$$|V| - |E| + 2|E|/3 \geq 2,$$

which implies that $|E| \leq 3|V| - 6$. Plugging the relation $2|E| \geq 3|F|$ into Euler's formula a different way, we get that

$$|V| - 3|F|/2 + |F| \geq 2,$$

which implies that $|F| \leq 2|V| - 4$. ∎

So, in other words, if G is an embedded planar graph with n vertices, then the number of edges in G is $O(n)$ and the number of faces in G is $O(n)$.

1.2 Representations for Planar Subdivisions

There are a number of different, but strongly related, methods for representing embedded planar graphs. They all provide for efficiently traversing around vertices and faces, as well as crossing edges between adjacent faces or adjacent vertices. We review the most well-known such structures.

1.2.1 The Winged-Edge Data Structure

The first structure we review is the *winged edge* data structure proposed by Baumgart. It stores a record for each vertex, face, and edge, with the edge record being the most important. The record for a vertex $v \in V$ stores v's coordinate position and a pointer to an arbitrary edge incident upon v. Likewise, the record for a face $f \in F$ stores the name of f and a pointer to an arbitrary edge on the boundary of f. The record for an edge $e \in E$ stores many fields, however, including

- Pointers to the two vertex endpoints v_1 and v_2 of e,
- Pointers to the two faces f_1 and f_2 on either side of e,
- Pointers to the predecessor and successor edges of e on the boundaries of f_1 and f_2, respectively. These are the so-called "wing edges."

Thus, the total number of pointers needed, therefore, is $|V| + |F| + 8|E|$. Most importantly, just by following the appropriate pointers, one can traverse the boundary of a face f in either clockwise or counterclockwise order in time proportional to the number of edges on the boundary, and, likewise, one can list all the edges incident upon a vertex v in clockwise or counterclockwise order in time proportional to their number. The details for implementing such searches, of course, requires careful attention to the orientation of the wing edge records.

1.2.2 Doubly-Connected Edge List

The next data structure we review is the *doubly-connected edge list* (DCEL) structure of Muller and Preparata, which will provide similar, but more-limited, functionality compared to the winged-edge structure, but it is more space efficient. It maintains the records for vertices and faces as in the winged-edge structure, each pointing to an arbitrary incident edge. Each edge e stores pointers to the records of its two incident vertices v_1 and v_2 and its two incident faces f_1 and f_2 (as in the winged edge structure). In addition, each edge stores pointers to the next edge incident to v_1 (resp., v_2) in a counterclockwise ordering. Thus, the total number of pointers need is $|V| + |F| + 6|E|$, which improves that of the winged edge structure, yet it still allows the edges incident upon the same vertex or face to be output in counterclockwise order in time proportional to their number. Intuitively, the DCEL deletes two pointer fields from the record for each edge in the winged-edge structure. The details for traversing faces and vertices is therefore a little more complicated, however, and it is in general not so easy to traverse the edges around a vertex or face efficiently in clockwise order (unless, of course, one also stores for each edge e an edge e' oriented in the opposite direction).

1.2.3 The Quad-Edge Structure

The final subdivision data structure that we review is the quad-edge structure of Guibas and Stolfi. It is just as space efficient as the winged-edge structure (actually it is isomorphic to the winged-edge structure), but it is organized in a way that allows the details for face and vertex traversal to be conceptually simpler than those for the other structures. It also has the added benefit of automatically encoding both the planar subdivision and its graph-theoretic planar dual. In addition, it can handle the "degnerate" cases of self loops and dangling edges better than the other two structures. The records for vertices and faces are as in the other structures. The record for an edge e also stores pointers to e's incident vertices v_1 and v_2 and faces f_1 and f_2, as in the other structures. In addition, each edge record stores a pointer to the next edge in a counterclockwise listing of edges around v_1, f_1, v_2, and f_2. Thus, the total number of pointers is $|V|+|F|+8|E|$, as with the winged-edge structure. Moreover, as mentioned above, the symmetry of representation between vertices and faces imply that the graph-theoretic planar dual of the subdivision is automatically represented in the quad-edge structure. Also, the details for traversing a vertex or face in counterclockwise order in time proportional to the number of incident edges are arguably simpler than those for the other structures, as one simply marches through the linked list for that vertex or face. In addition, it is fairly straightforward to traverse a vertex or face in clockwise order, as well, in time proportional to the number of incident edges by a similar list traversal. Indeed, the pointers out from an edge record in a quad-edge representation point to exactly the same edge records as they would in the winged-edge structure. Yet by explicitly organizing these pointers into four circularly linked lists we gain a conceptual simplicity that should aid significantly in implementations.