Chapter 2

Plane Embeddings

Graphs can be represented in various ways, for instance, as an adjacency matrix or using adjacency lists. In this chapter we explore another class of representations that are quite different in nature, namely *geometric* representations. In a geometric representation, vertices and edges are represented by geometric objects, for example points and curves. This approach is appealing because it succinctly visualizes a graph along with its many properties. We have many degrees of freedom in selecting the geometric objects and the details of their geometry. This freedom allows us to tailor the representation to meet specific goals, such as emphasizing certain structural aspects of the graph at hand or reducing the complexity of the obtained representation.

The most common geometric graph representation is a *drawing*, where vertices are mapped to points and edges to curves in $\mathbb{R}^2$. It is desirable to make such a map injective by avoiding edge crossings, both from a mathematically aesthetic viewpoint and for the sake of the practical readability. Those graphs that allow such an *embedding* into the Euclidean plane are known as *planar*. Our goal is to study the interplay between abstract planar graphs and their plane embeddings. Specifically, we want to answer the following questions:

- What is the combinatorial complexity (that is, the number of edges and faces) of planar graphs?
- Under which conditions are plane embeddings unique (up to a certain sense of equivalence)?
- How can we represent plane embeddings in a data structure?
- What is the geometric complexity (that is, the encoding size of the geometric objects used to represent vertices and edges) of plane embeddings?

Most definitions we use directly extend to multigraphs. But for simplicity, we use the term "graph" throughout.

2.1 Drawings, Embeddings and Planarity

A **curve** is a set $C \subset \mathbb{R}^2$ of the form $\{\gamma(t) : 0 \leqslant t \leqslant 1\}$, where $\gamma : [0, 1] \to \mathbb{R}^2$ is a continuous function. The function γ is called a *parameterization* of C. The points $\gamma(0)$ and $\gamma(1)$ are the *endpoints* of the curve. A curve is *closed* if $\gamma(0) = \gamma(1)$. A curve is *simple* if it admits a parameterization γ that is injective on $[0, 1]$; for a closed simple curve we allow as an exception that $\gamma(0) = \gamma(1)$. The following famous theorem describes an important property of the plane. A proof can, for instance, be found in the book of Mohar and Thomassen [24].

Theorem 2.1 (Jordan). *Any simple closed curve C partitions the plane into exactly two regions (connected open sets), each bounded by C.*

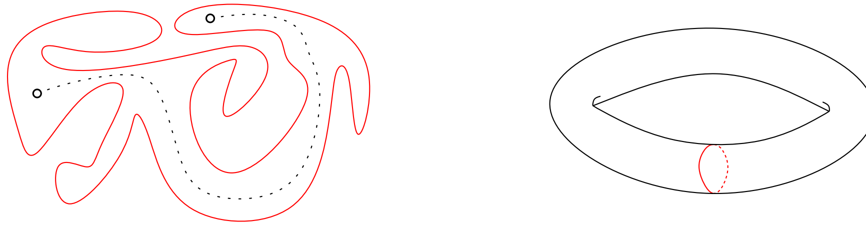


Figure 2.1: *Left: a simple closed curve in the plane and two points in one of its faces. Right: a simple closed curve that does not disconnect the torus.*

Observe that, for instance, on the torus there are simple closed curves that do not disconnect the surface, and thus the theorem does not hold there.

Drawings. As a first criterion for a reasonable geometric representation of a graph, we would like to have a clear separation between different vertices and also between a vertex and nonincident edges. Formally, a *drawing* of a graph $G = (V, E)$ in the plane is a function f that assigns

- a point $f(v) \in \mathbb{R}^2$ to every vertex $v \in V$ and
- a simple curve $f(uv)$ with endpoints $f(u)$ and $f(v)$ to every edge $uv \in E$,

such that

(1) f is injective on V and

(2) $f(uv) \cap f(V) = \{f(u), f(v)\}$, for every edge $uv \in E$.

A common point $f(e) \cap f(e')$ between two curves that represent distinct edges $e, e' \in E$ is called a *crossing* if it is not a common endpoint of e and e'.

Commonly, when discussing a drawing of a graph $G = (V, E)$, we do not differentiate a vertex/an edge from its geometric realization. That is, a vertex $v \in V$ is identified with the point $f(v)$, and an edge $e \in E$ is identified with the curve $f(e)$. For instance, the last

sentence in the previous paragraph may be phrased as "A common point of two edges is called a crossing if it is not their common endpoint."

Often it is convenient to make additional assumptions about edge intersections in a drawing. For example, we may demand *nondegeneracy* in the sense that no three edges can meet at a single crossing, or that any two edges can intersect at only finitely many points.

Planar vs. plane. A graph is *planar* if it admits a drawing in the plane without crossings. Such a drawing is also called a *crossing-free* drawing or a (plane) *embedding* of the graph. A planar graph together with a particular plane embedding is called a *plane* graph. Note the distinction between "planar" and "plane": the former refers to an abstract graph and indicates the possibility of an embedding, whereas the latter refers to a concrete embedding (Figure 2.2).

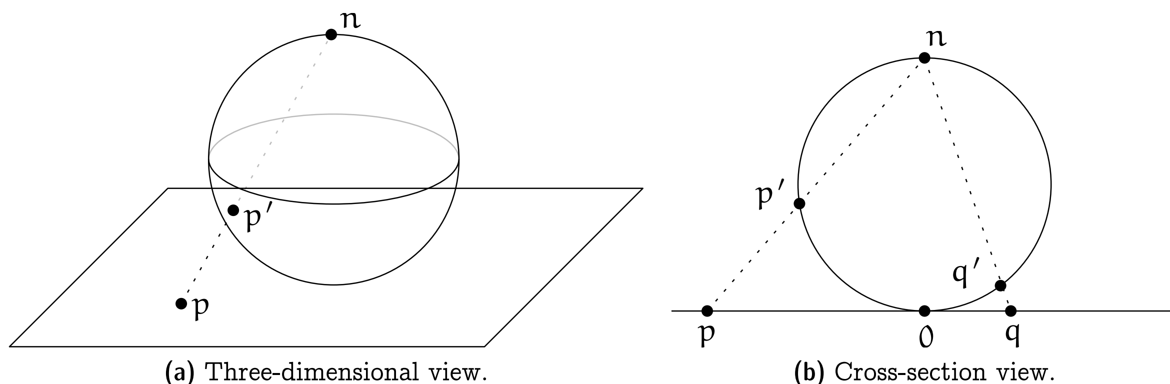
Figure 2.2: *A planar graph (left) and a plane embedding of it (right).*

A *geometric graph* is a graph together with a drawing in which all edges are straight-line segments. Note that such a drawing is fully determined by the vertex positions. A plane graph which is also geometric is called a *plane straight-line graph* (PSLG). On the other hand, a plane graph whose edges are arbitrary simple curves is emphasized as *topological plane graph*.

The *faces* of a plane graph G are the maximally connected regions of $\mathbb{R}^2 \setminus G$, that is, the plane without the points occupied by the embedding (as the image of a vertex or an edge). Each embedding of a finite graph has exactly one *unbounded face*, also called *outer* or *infinite* face. Using stereographic projection, we could show that any face can be swapped out to serve as the unbounded face:

Theorem 2.2. *If a graph G has a plane embedding in which some face is bounded by a cycle $(v_1, \dots, v_k)$, then G also has a plane embedding in which the unbounded face is bounded by the cycle $(v_1, \dots, v_k)$.*

Proof Sketch. Take a plane embedding Γ of G and map it to the sphere using *stereographic projection*: Imagine $\mathbb{R}^2$ being the x/y-plane in $\mathbb{R}^3$ and place a unit sphere S whose south pole touches the origin. We establish a bijection between $\mathbb{R}^2$ and $S \setminus \{n\}$, where $n := (0, 0, 2)$ is the north pole position: A point $p \in \mathbb{R}^2$ is mapped to the intersection p' of the segment $\overline{pn}$ and S, see Figure 2.3. The map is continuous, so it preserves incidence between vertices, edges and faces.

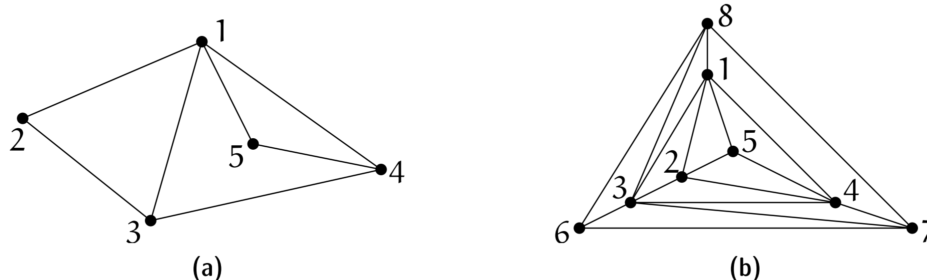


(a) Three-dimensional view. (b) Cross-section view.

Figure 2.3: *Stereographic projection.*

Consider the resulting embedding Γ' of G on S: The infinite face of Γ corresponds to the face of Γ' that contains the north pole n of S. Now rotate the embedding Γ' on S such that the desired face contains n. Mapping back to the plane using stereographic projection results in an embedding in which the desired face is the outer face. □

Exercise 2.3. *Consider the plane graphs depicted in Figure 2.4. For both graphs give a plane embedding in which the cycle* $(1,2,3)$ *bounds the outer face.*



(a) (b)

Figure 2.4: *Make* $(1,2,3)$ *bound the outer face.*

Duality. Every plane graph G has a *dual* G^* whose vertices are the faces of G. For every edge in G, we connect its two incident faces by an edge in the dual G^*. Note that in general, G^* is a multigraph (with loops and multiple edges) and may depend on the embedding. So an abstract planar graph G may have several nonisomorphic duals; see Figure 2.5 for an example. If G is a connected plane graph, then $(G^*)^* = G$. We will see later in Section 2.3 that the dual of a 3-connected planar graph is unique (up to isomorphism).

The Euler Formula and its ramifications. One of the most important tools for planar graphs (and more generally, graphs embedded on a surface) is the Euler–Poincaré Formula.

Figure 2.5: *Two plane drawings G_1 and G_2 of the same abstract planar graph and their duals ${G_1}^*$ and ${G_2}^*$ with ${G_1}^* \not\simeq {G_2}^*$. (To see this, for instance, count the number of vertices of degree greater than three.)*

Theorem 2.4 (Euler's Formula). *For every connected plane graph with n vertices, e edges, and f faces, we have $n - e + f = 2$.*

Proof. Let G be a connected plane graph with n vertices, e edges, and f faces. Note that $e \geqslant n - 1$ as G is connected.

We prove the statement by induction on $e - n$. In the base case $e - n = -1$, the graph G is a (plane) tree and contains exactly one (unbounded) face, and so $n - e + f = 1 + 1 = 2$ as claimed.

In the general case, fix a spanning tree T of G, pick an arbitrary edge e of $G \setminus T$, and consider the graph $G^- = G \setminus e$. By construction it has n vertices and $e - 1$ edges. We claim that it has $f - 1$ faces. To see this observe that $G^- \supset T$ is connected. In particular, the endpoints of e are connected by a path in G^-, which together with e forms a cycle in G. So in G, any two points sufficiently close to but on opposite sides of e are in different faces, whereas they are in the same face of G^-. In other words, the two incident faces of e are distinct in G but merged into one in G^-. All other faces remain untouched. It follows that G^- has $f - 1$ faces, as claimed. Then by the inductive assumption on G^-, we have $n - e + f = n - (e - 1) + (f - 1) = 2$, which concludes the induction. □

In particular, this shows that every plane embedding of a planar graph has the same number of faces. In other words, the number of faces is an invariant of an abstract planar graph. It also follows (as the corollary below) that planar graphs are *sparse*, that is, they have a linear number of edges and faces only. So the asymptotic complexity of a planar graph is already determined by its number of vertices.

Corollary 2.5. *A simple planar graph on $n \geqslant 3$ vertices has at most $3n - 6$ edges and at most $2n - 4$ faces.*

Proof. Without loss of generality we may assume that G is connected. (If not, add edges between components of G until the graph is connected. The number of edges increases and the number of faces remains unchanged.) The statement is easily checked for $n = 3$, where G is either a triangle or a path and therefore has no more than $3 \leqslant 3 \cdot 3 - 6$ edges and no more than $2 \leqslant 2 \cdot 3 - 4$ faces. Next consider a simple connected planar graph G

on $n \geqslant 4$ vertices, and fix any plane embedding of it. Denote by E its set of edges and by F its set of faces. Let

$$X = \{(e, f) \in E \times F : e \text{ bounds } f\}$$

denote the set of incident edge-face pairs. We count X in two different ways.

First note that each edge bounds at most two faces and so $|X| \leqslant 2 \cdot |E|$.

Second note that every face is bounded by at least three edges: If G contains a cycle, then the boundary of every face shall contain a cycle and hence at least three edges. If G is acyclic, then it must be a tree since we assumed it to be connected. Its only face (the outer face) is bounded by all edges; and there are at least three since G contains at least four vertices. In both cases we have $|X| \geqslant 3 \cdot |F|$.

Therefore $3|F| \leqslant 2|E|$. Using Euler's Formula we conclude that

$$\begin{aligned} 4 &= 2(n - |E| + |F|) \leqslant 2n - 3|F| + 2|F| = 2n - |F| \text{ and} \\ 6 &= 3(n - |E| + |F|) \leqslant 3n - 3|E| + 2|E| = 3n - |E| , \end{aligned}$$

which yield the claimed bounds. □

Corollary 2.5 implies that the degree of a "typical" vertex in a planar graph is a small constant.

Corollary 2.6. *The average vertex degree in a simple planar graph is less than six.*

Exercise 2.7. *Prove Corollary 2.6.*

There exist several variations of this statement, a few more of which we will encounter during this course.

Exercise 2.8. *Show that neither K_5 (the complete graph on five vertices) nor $K_{3,3}$ (the complete bipartite graph where both classes have three vertices) is planar.*

Exercise 2.9. *Let P be a set of $n \geqslant 3$ points in the plane such that the distance between every pair of points is at least one. Show that there are at most $3n - 6$ pairs of points in P at distance exactly one.*

Characterizing planarity. The classical theorems of Kuratowski and Wagner provide a characterization of planar graphs in terms of forbidden substructures. A *subdivision* of a graph $G = (V, E)$ is obtained from G by replacing each edge with a path.

Theorem 2.10 (Kuratowski [22, 31]). *A graph is planar if and only if it does not contain a subdivision of $K_{3,3}$ or K_5.*

A *minor* of a graph $G = (V, E)$ is obtained from G using zero or more edge contractions, edge deletions, and/or vertex deletions.

Theorem 2.11 (Wagner [34]). *A graph is planar if and only if it does not contain $K_{3,3}$ or K_5 as a minor.*

In some sense, Wagner's Theorem is a special instance[1] of a much more general theorem.

Theorem 2.12 (Graph Minor Theorem, Robertson/Seymour [28]). *Every minor-closed family of graphs can be described in terms of a finite set of forbidden minors.*

Being *minor-closed* means that any minor of any graph from the family also belongs to the family. For instance, the family of planar graphs is minor-closed because planarity is preserved under removal of edges and vertices and under edge contractions.

Exercise 2.13. *A graph is 1-planar if it admits a drawing in the plane in which every edge has at most one crossing. Prove or disprove: The family of 1-planar graphs is minor-closed.*

The Graph Minor Theorem is a celebrated result established by Robertson and Seymour in a series of twenty papers, see also the survey by Lovász [23]. They also describe an $O(n^3)$ algorithm (with horrendous constants, though) to decide whether a graph on n vertices contains a fixed (constant-size) minor. As a consequence, every minor-closed property can be tested in polynomial time. Later, Kawarabayashi et al. [20] showed that this problem can be solved in $O(n^2)$ time.

Unfortunately, the Graph Minor Theorem is nonconstructive in the sense that in general we do not know how to obtain the set of forbidden minors for a given family. For instance, for the family of toroidal graphs (graphs that can be embedded without crossings on the torus) more than 16′000 forbidden minors are known, and the theorem tells us that the number is finite, but we still do not know the concrete number. So while we know that there exists a quadratic time algorithm to test membership for minor-closed families, we have no idea what such an algorithm looks like in general.

Graph families other than planar graphs for which the forbidden minors are known include forests (free of K_3 minors) and outerplanar graphs (free of $K_{2,3}$ and K_4 minors). A graph is *outerplanar* if it admits a plane embedding in which all vertices appear on the outer face (Figure 2.6).

Figure 2.6: *An outerplanar graph (left) and a plane embedding of it in which all vertices are incident to the outer face (right).*

Exercise 2.14. *(a) Give an example of a 6-connected planar graph or argue that no such graph exists.*

[1] It is more than just a special instance because it also specifies the forbidden minors explicitly.

(b) Give an example of a 5-connected planar graph or argue that no such graph exists.

(c) Give an example of a 3-connected outerplanar graph or argue that no such graph exists.

Planarity testing. To test a given graph for planarity we do not have to contend ourselves with a quadratic-time algorithm. In fact, there exist a number of different linear time algorithms that decide if a given abstract graph is planar; all of them—from a very high-level point of view—can be regarded as an annotated depth-first-search. The first such algorithm was described by Hopcroft and Tarjan [19], while the current state-of-the-art is probably among the "path searching" method by Boyer and Myrwold [6] and the "LR-partition" method by de Fraysseix et al. [14]. Although the overall idea in all these approaches is easy to convey, many technical details make an in-depth discussion rather painful to go through.

2.2 Graph Representations

There are two standard representations for an abstract graph $G = (V, E)$ on $n = |V|$ vertices. For the *adjacency matrix* representation we consider the vertices to be ordered as $V = \{v_1, \ldots, v_n\}$. The adjacency matrix of an undirected graph is a symmetric $n \times n$-matrix $A = (a_{ij})_{1 \leqslant i,j \leqslant n}$ where $a_{ij} = a_{ji} = 1$, if $\{v_i, v_j\} \in E$, and $a_{ij} = a_{ji} = 0$ otherwise. Storing such a matrix explicitly requires $\Omega(n^2)$ space, but it allows testing in constant time whether or not two given vertices are adjacent.

In an *adjacency list* representation, we store for each vertex a list of its neighbors in G. This requires only $O(n+|E|)$ storage, which is better than for the adjacency matrix in case that $|E| = o(n^2)$. On the other hand, the adjacency test for two given vertices is not a constant-time operation, because it requires a search in one of the lists. Depending on the implementation of the lists, the search time ranges from $O(d)$ (for an unsorted list) to $O(\log d)$ (for a sorted dynamic data structure such as a balanced search tree), where d is the minimum degree of the two vertices.

Both representations have their merits. The choice typically depends on what one wants to do with the graph. When dealing with embedded graphs, however, additional information about the embedding is needed beyond the pure incidence structure of the graph. The next section discusses a standard data structure to represent embedded graphs.

2.2.1 The Doubly-Connected Edge List

The *doubly-connected edge list* (DCEL) is a data structure to represent a plane graph in such a way that it is easy to traverse and to manipulate. To avoid complications, let us discuss only connected graphs that contain at least two vertices. It is not hard to extend the data structure to be able to represent all plane graphs. We also assume

that we deal with a straight-line embedding and thus the geometry of edges is defined by the positions of their endpoints already. For more general embeddings, the geometric description of edges has to be stored in addition.

The main building block of a DCEL is a list of *halfedges*. Every actual edge is split into two halfedges going in opposite direction, and these are called *twins*, see Figure 2.7. Along the boundary of each face, halfedges are oriented counterclockwise, that is, the face always stays to the left.

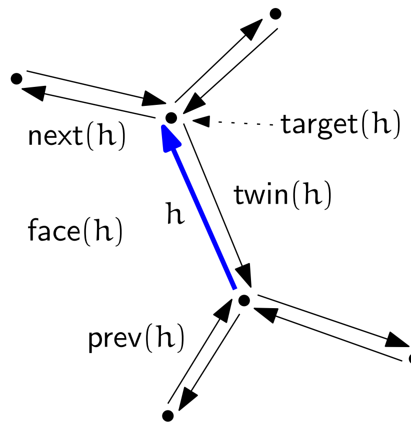


Figure 2.7: *A halfedge in a DCEL.*

A DCEL also stores a list of vertices and a list of faces. These three lists are unordered but interconnected by various pointers. A vertex v stores a pointer halfedge(v) to an arbitrary halfedge originating from v. Every vertex also records its coordinates point(v), that is, the point it is mapped to in the embedding. A face f stores a pointer halfedge(f) to an arbitrary halfedge within the face. A halfedge h stores *five* pointers:

- a pointer target(h) to its target vertex,
- a pointer face(h) to its incident face,
- a pointer twin(h) to its twin halfedge,
- a pointer next(h) to the halfedge following h along the boundary of face(h), and
- a pointer prev(h) to the halfedge preceding h along the boundary of face(h).

A constant amount of information is stored for every vertex, (half-)edge, and face of the graph. Therefore the whole DCEL needs storage proportional to $|V| + |E| + |F|$, which is $O(n)$ for a plane graph with n vertices by Corollary 2.5.

This information is sufficient for most tasks. For example, traversing all edges around a face f can be done as follows:

$s \leftarrow$ halfedge(f)
$h \leftarrow s$
do

something with h
$h \leftarrow$ next(h)
while $h \neq s$

Exercise 2.15. *Give pseudocode to traverse all edges incident to a given vertex* v *of a DCEL.*

Exercise 2.16. *Why is the previous halfedge* prev($\cdot$) *stored explicitly whereas the source vertex of a halfedge is not?*

2.2.2 Manipulating a DCEL

In many applications, plane graphs do not just appear as static objects but rather evolve over the course of an algorithm. Therefore the data structure must allow for efficient updates. These include, but are not limited to, appending new vertices, edges and faces to the corresponding list within the DCEL and—symmetrically—the ability to delete an existing entity.

First, it should be easy to add a new vertex v to the graph within a given face f and (as we maintain a connected graph) connect v to an existing vertex u. For such a connection to be valid, we require that the open line segment $\overline{uv}$ lies completely in f. Given that we need access to both f and u, it would be convenient to pass a halfedge h as an argument, which is supposed to satisfy face(h) $= f$ and target(h) $= u$. Then our operation becomes

add-vertex-at(v, h)
Precondition: the open line segment $\overline{\text{point}(v)\text{point}(u)}$, where $u :=$ target(h), lies completely in $f :=$ face(h).
Postcondition: the new vertex v has been inserted into f, connected by an edge to u.

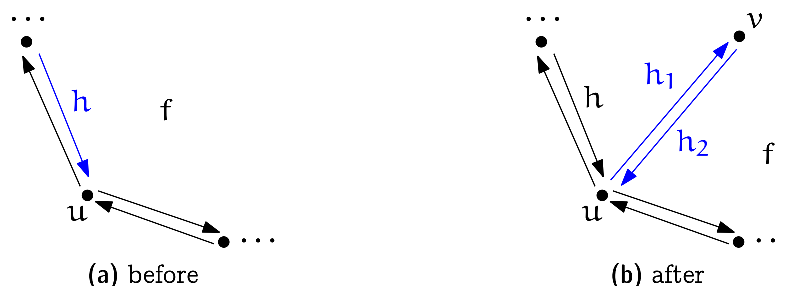


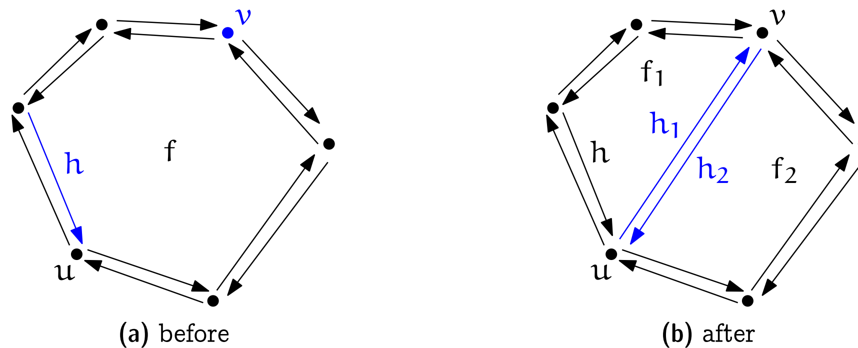
Figure 2.8: *Add a new vertex connected to an existing vertex* u.

See also Figure 2.8. It can be realized by manipulating a constant number of pointers as follows.

```
add-vertex-at(v, h) {
    h1 ← a new halfedge
    h2 ← a new halfedge
    halfedge(v) ← h2
    twin(h1) ← h2
    twin(h2) ← h1
    target(h1) ← v
    target(h2) ← u
    face(h1) ← f
    face(h2) ← f
    next(h1) ← h2
    next(h2) ← next(h)
    prev(h1) ← h
    prev(h2) ← h1
    next(h) ← h1
    prev(next(h2)) ← h2
}
```

Similarly, it should be possible to add an edge between two existing vertices u and v, provided the open line segment $\overline{uv}$ lies completely within a face f of the graph, see Figure 2.9. Since such an edge insertion splits f into two faces, the operation is called *split-face*. Again we pass as an argument the halfedge h satisfying $\mathrm{face}(h) = f$ and $\mathrm{target}(h) = u$.

split-face(h, v)
Precondition: v is incident to $f := \mathrm{face}(h)$ but not adjacent to $u := \mathrm{target}(h)$.
The open line segment $\overline{\mathrm{point}(v)\mathrm{point}(u)}$ lies completely in f.
Postcondition: f has been split by a new edge uv.



(a) before (b) after

Figure 2.9: *Split a face by an edge* uv.

The implementation is slightly more complicated compared to add-vertex-at above, because the face f is destroyed and so we have to update the face information of all incident

halfedges. In particular, this is not a constant time operation and has complexity proportional to the size of f.

```
split-face(h, v) {
    f1 ← a new face
    f2 ← a new face
    h1 ← a new halfedge
    h2 ← a new halfedge
    halfedge(f1) ← h1
    halfedge(f2) ← h2
    twin(h1) ← h2
    twin(h2) ← h1
    target(h1) ← v
    target(h2) ← u
    next(h2) ← next(h)
    prev(next(h2)) ← h2
    prev(h1) ← h
    next(h) ← h1
    i ← h2
    loop
        face(i) ← f2
        if target(i) = v break the loop
        i ← next(i)
    endloop
    next(h1) ← next(i)
    prev(next(h1)) ← h1
    next(i) ← h2
    prev(h2) ← i
    i ← h1
    do
        face(i) ← f1
        i ← next(i)
    until target(i) = u
    delete the face f
}
```

In a similar fashion one can realize the inverse operation join-face(h) that removes the edge represented by h, thereby joining the faces face(h) and face(twin(h)).

It is easy to see that every connected plane graph on at least two vertices can be constructed using the operations add-vertex-at and split-face, starting from an embedding of K_2 (two vertices connected by an edge).

Exercise 2.17. *Give pseudocode for the operation join-face*(h)*. Specify preconditions if needed.*

Exercise 2.18. *Give pseudocode for the operation split-edge*(h)*, that splits the edge represented by* h *into two by a new vertex* w*, see Figure 2.10.*

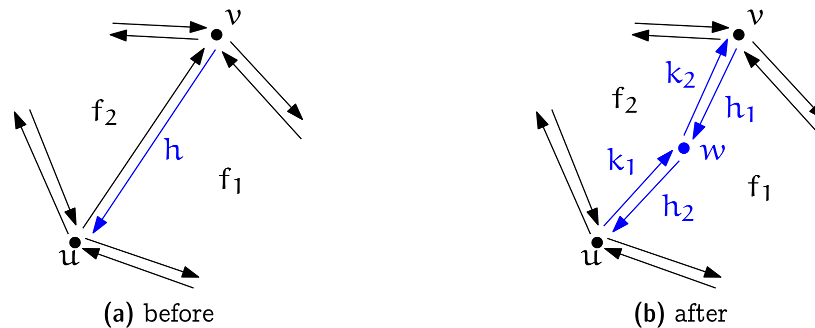


Figure 2.10: *Split an edge by a new vertex.*

2.2.3 Graphs with Unbounded Edges

In some cases it is convenient to consider plane graphs in which some edges are not mapped to a line segment but to an unbounded curve, such as a ray. This setting is not really much different from the one we studied before, except that one special vertex is placed "at infinity". One way to think of it is in terms of *stereographic projection* (see the proof of Theorem 2.2). The further away a point in $\mathbb{R}^2$ is from the origin, the closer its image on the sphere S gets to the north pole n of S. But there is no way to reach n except in the limit. Therefore, we can imagine drawing the graph on S instead of in $\mathbb{R}^2$ and putting the "infinite vertex" at n.

All this is just for the sake of a proper geometric interpretation. As far as a DCEL of such a graph is concerned, there is no need to consider spheres or anything beyond what we have discussed. The only difference to the case with all finite edges is that there is this special infinite vertex, which does not have any point/coordinates associated to it. Other than that, the infinite vertex is treated in exactly the same way as the finite vertices: it has in- and out-going halfedges along which the unbounded faces can be traversed (Figure 2.11).

Remarks. It is actually not so easy to point exactly to where the DCEL data structure originates from. Often Muller and Preparata [25] are credited, but while they use the term DCEL, the data structure they describe is different from what we discussed above and from what people usually consider a DCEL nowadays. Overall, there are a large number of variants of this data structure, which appear under the names *winged edge* data structure [3], *halfedge* data structure [35], or *quad-edge* data structure [16]. Kettner [21] provides a comparison of all these with some additional references.

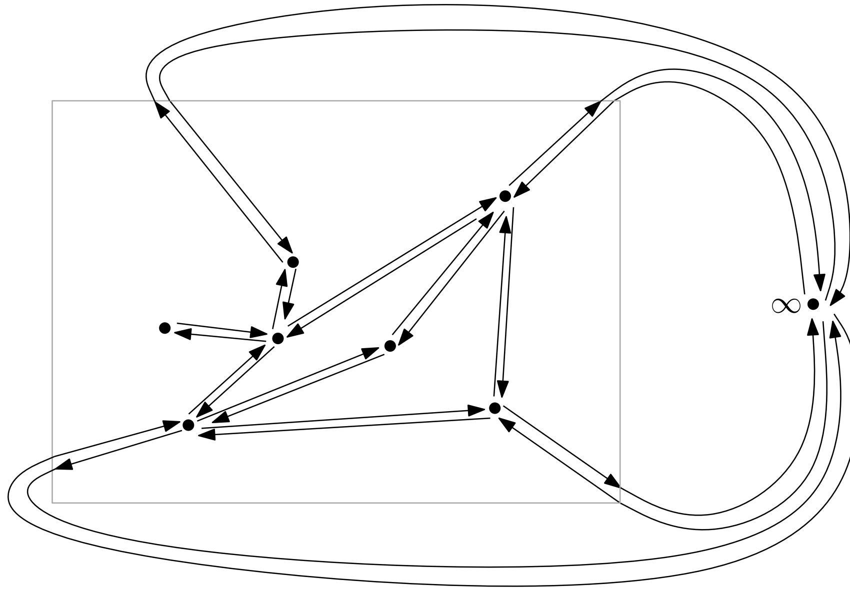


Figure 2.11: *A DCEL with unbounded edges. Usually, we will not show the infinite vertex and draw all edges as straight-line segments. This yields a geometric drawing, like the one within the gray box.*

2.2.4 Combinatorial Embeddings

The basic DCEL omits geometric aspects (that is, positions and shapes of a vertex/edge/face) and only stores incidences and adjacencies between vertices, edges, and faces of an embedding. We call such information the *combinatorial embedding* of the actual plane graph. Conventionally, we write it as a set of face boundaries, where each boundary is encoded as a circular sequence of vertices in counterclockwise order. For instance, the combinatorial embeddings of the plane graphs in Figure 2.12a are

$$\begin{aligned}(a) &: \{(1,2,3),(1,3,6,4,5,4),(1,4,6,3,2)\},\\ (b) &: \{(1,2,3,6,4,5,4),(1,3,2),(1,4,6,3)\}, \text{and}\\ (c) &: \{(1,4,5,4,6,3),(1,3,2),(1,2,3,6,4)\}.\end{aligned}$$

Note that a vertex can appear several times along the boundary of a face (if it is a cut-vertex).

This view allows us to compare embeddings easily. Two embeddings (plane graphs) are *combinatorially equivalent* if their combinatorial embeddings are equal up to a global change of orientation (reversing the order of all sequences simultaneously). For example, (b) is not equivalent to (a) nor (c), because it is the only one with a face bounded by seven vertices. However, (a) and (c) turn out to be equivalent: after reverting orientations f_1 takes the role of h_2, f_2 takes the role of h_1, and f_3 takes the role of h_3.

Exercise 2.19. *Let* G *be a planar graph with vertex set* $\{1,\ldots,9\}$*. Try to find an embedding corresponding to the following list of circular sequences of faces:*

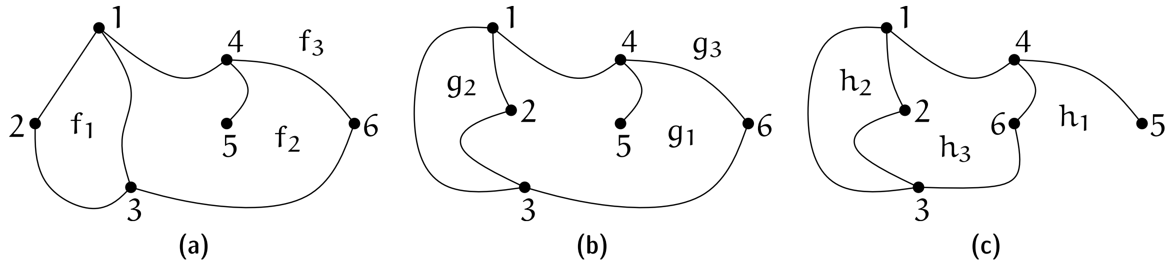


Figure 2.12: *Equivalent embeddings?*

(a) $\{(1,4,5,6,3),(1,3,6,2),(1,2,6,7,8,9,7,6,5),(7,9,8),(1,5,4)\}$

(b) $\{(1,4,5,6,3),(1,3,6,2),(1,2,6,7,8,9,7,6,5),(7,9,8),(1,4,5)\}$

Combinatorial embeddings are not only used to categorize plane graphs. They also play a role in algorithm design. Quite often, algorithms dealing with planar graphs do not need a full-fledged embedding to proceed. It is sufficient to operate on a combinatorial embedding, which is more efficient to handle.

Many people prefer a dual representation which, instead of listing face boundaries, enumerates the neighbors of v in cyclic order for each vertex v. It can avoid the issue of a vertex appearing multiple times in the sequence. However, the following lemma shows that such an issue does not arise when dealing with biconnected graphs.

Lemma 2.20. *In a biconnected plane graph every face is bounded by a cycle.*

We leave the proof as an exercise. Intuitively the statement is clear, but we believe it is instructive to think about a formal argument. An easy consequence is stated below, whose proof is also an exercise.

Corollary 2.21. *For any vertex v in a 3-connected plane graph, there is a cycle that contains all neighbours of v.*

Exercise 2.22. *Prove Lemma 2.20 and Corollary 2.21.*

Given Lemma 2.20, one might wonder the converse question: Which cycles in a planar graph G bound a face (in some plane embedding of G)? Such cycles are said to be *facial*; see Figure 2.13.

Exercise 2.23. *Describe a linear time algorithm that, given an abstract planar graph G and a cycle C in G, tests whether C is a facial cycle. (You may assume that planarity can be tested in linear time.)*

2.3 Unique Embeddings

As we have seen, an abstract planar graph may admit many different embeddings, even in the combinatorial sense. Under what condition does it admit a unique combinatorial embedding?