CHAPTER 2

Planar Graphs

Es nimmt mich Wunder, dass diese allgemeinen proprietates in der Stereometrie noch von Niemand, so viel mir bekannt, sind angemerkt worden; doch viel mehr aber, dass die fürnehmsten davon als theor. 6 und theor. 11 so schwer zu beweisen sind, denn ich kann dieselben noch nicht so beweisen, dass ich damit zufrieden bin.

[It seems miraculous to me that these general results in solid geometry have not been noticed by anyone yet, as far as I know; but even more, that results as fundamental as Theorem 6 and Theorem 11 are so difficult to prove, because I cannot prove them in a way that makes me happy.]

— Leonhard Euler, letter to Christian Goldbach, November 14, 1750.

(Theorem 6 is the angle-defect formula; Theorem 11 is Euler’s formula.)

Mirabilis... est haec inter angulos solidos et triangula lateralia conspiratio; mihi certe eius notitia, vel sola, remunerari videbatur laborem in haec impensum.

[This harmony between solid angles and triangular sides is astonishing; certainly for me, just observing it seems to repay the effort spent on them.]

— Albrecht Ludwig Friedrich Meister, “Commentatio de solidis geometricis...” (1785)

2.1 Graphs

Basic Definitions

A graph is an abstract combinatorial structure that models pairwise relationships. Graphs are traditionally defined as pairs \((V, E)\), where \(V\) is an arbitrary finite set of so-called vertices, and \(E\) is a set of unordered pairs of vertices, called edges. While admirably terse, this definition is both unnecessarily restrictive and inconsistent with the usual
2. Planar Graphs

For any dart $d$, we call the dart $\text{rev}(d)$ the reversal of $d$, and we call the vertex $\text{head}(d)$ the head of $d$. The tail of a dart is the head of its reversal: $\text{tail}(d) := \text{head}(\text{rev}(d))$. The head and tail of a dart are its endpoints. Intuitively, a dart is a directed path from its tail to its head; in keeping with this intuition, we say that a dart $d$ leaves its tail and enters its head. We often write $u \rightarrow v$ to denote an dart with tail $u$ and head $v$, even (at the risk of confusing the reader) when there is more than one such dart.

For any dart $d \in D$, the unordered pair $\{d, \text{rev}(d)\}$ is called an edge of the graph. We often write $E$ to denote the set of edges of a graph, and we write $e^+$ and $e^-$ to denote the constituent darts of an edge $e$. The endpoints of an edge $e = \{e^+, e^-\}$ are the endpoints (equivalently, just the tails) of its constituent darts. Intuitively, each dart is an orientation of some edge from one of its endpoints to the other. A vertex $v$ and an edge $e$ are incident if $v$ is an endpoint of $e$; two vertices are neighbors if they are endpoints of the same edge. We often write $uv$ to denote an edge with endpoints $u$ and $v$, even (at the risk of confusing the reader) when there is more than one such edge.

A loop is an edge $e$ with only one endpoint, that is, $\text{head}(e^+) = \text{tail}(e^+)$. Two edges are parallel if they have the same endpoints. A graph is simple if it has no loops or parallel edges and non-simple otherwise. (Non-simple graphs are sometimes called generalized graphs or multigraphs.) A simple graph is more tersely defined as a pair of sets $(V, E)$, where each element of $E$ is a set of two elements of $V$; our more complex definition is necessary because we do not want to assume a priori that all graphs are simple.

The degree of a vertex $v$, denoted $\text{deg}_G(v)$ (or just $\text{deg}(v)$ if the graph $G$ is clear from context), is the number of darts whose head is $v$, or equivalently, the number of incident edges plus the number of incident loops. A vertex is isolated if it is not incident to any edge.

It is often convenient to regard graphs as continuous topological spaces, in which vertices are points and edges are interior-disjoint paths between their endpoints, rather than strictly combinatorial objects. More formally, let $V^\top$ be a set of distinct points $v^\top$, one for each vertex $v \in V$, and let $E^\top$ be a set of disjoint closed real intervals $e^\top$, one for each edge $e \in E$. The topological graph $G^\top$ is the quotient space $(V^\top \sqcup E^\top) / \sim$, where for each edge $e$, we have $a \sim \text{head}(e^+)\top$ and $b \sim \text{tail}(e^-)\top$, where $e^\top = [a, b]$. To avoid excessive formality, however, we rarely distinguish between an abstract graph $G$ and the corresponding topological graph $G^\top$.
Data Structures

In implementations of graph algorithms, graphs are normally represented using a data structure called an incidence list; for simple graphs, the same data structure is more commonly known as an adjacency list. A standard incidence list is an array of linked lists, indexed by the vertices, where each record in each linked list corresponds to one of the darts entering the corresponding vertex. The record for each dart $d$ contains the index of $\text{tail}(d)$, a pointer to the record for $\text{rev}(d)$, and a constant amount of other algorithm-dependent auxiliary data. Auxiliary data associated with the vertices is stored in the main array; auxiliary data associated with the edges, if any, is stored in the dart records. Storing a graph with $n$ vertices and $m$ edges in an incidence list requires $O(n + m)$ space altogether.

An incidence list representation of a graph, with the dart records for two edges emphasized. For clarity, most reversal pointers are omitted.

If a graph is stored in an incidence list, we can insert a new edge in $O(1)$ time, delete an edge in $O(1)$ time (given a pointer to one of its darts), and visit all the edges incident to any vertex $v$ in $O(1)$ time per edge, or $O(\deg(v))$ time altogether. There are several standard operations that incidence lists do not support on $O(1)$ time, the most glaring of which is testing whether two vertices are neighbors. Surprisingly, however, most efficient graph algorithms do not require this operation. For those few that do, we can store the darts entering each vertex in a more efficient data structure, such as a balanced binary search tree or a hash table, instead of a linked list.

Deletion, Contraction, Subgraphs, and Minors

Let $G$ be a graph with $n$ vertices and $m$ edges. Deleting an edge $e$ from $G$ yields a smaller graph $G \setminus e$ with $n$ vertices and $m - 1$ edges. More generally, deleting any subset of edges $F \subseteq E$ yields a smaller graph $G \setminus F$. We also write $G \setminus v$ to denote the graph obtained from $G$ by deleting a vertex $v$ and all its incident edges.

If $e$ is not a loop, then contracting $e$ merges the endpoints of $e$ into a single vertex and destroys the edge, yielding a smaller graph $G/e$ with $n - 1$ vertices and $m - 1$ edges. Contracting a loop is simply forbidden. More generally, contracting any subset $F \subseteq E$ of non-loop edges yields a smaller graph $G/F$. 
subgraph
subgraph
minor of a graph
proper minor
vertex-disjoint
edge-disjoint
walk
tail of an walk
head of an walk
closed walk
open walk
length of a walk
simple walk
path in a graph
cycle in a graph
even subgraph
connected graph
component of a graph
acyclic graph
forest
tree
cut in a graph
crossing a cut
boundary of a cut
edge cut
bond
bridge
spanning tree
exercise for the reader

A **subgraph** of a graph $G$ is another graph obtained from $G$ by deleting edges and vertices; a **proper subgraph** of $G$ is any subgraph other than $G$ itself. Similarly, a **minor** of $G$ is any graph obtained from a subgraph of $G$ by contracting edges; a **proper minor** of $G$ is any minor other than $G$ itself. Two or more subgraphs of $G$ are **vertex-disjoint** if they have no vertices in common, and **edge-disjoint** if they have no edges in common.

**Walks, Paths, Cycles, Cuts, Bonds, and Trees**

A **walk** in a graph $G$ is an alternating sequence $\omega = (v_0, d_1, v_1, d_2, \ldots, d_k, v_k)$ of vertices and darts of $G$, where for every index $i$, we have $v_{i-1} = tail(d_i)$ and $v_i = head(d_i)$. The initial vertex $v_0$ and the final vertex $v_k$ are respectively the **tail** and **head** of the walk; informally, we say that $\omega$ is a walk from $v_0$ to $v_k$. A walk is **closed** if it has at least one dart and its first and last vertices coincide; otherwise, the walk is **open**. The **length** of a walk is the number of darts. If the graph $G$ has no loops, we can safely regard any walk as an alternating sequence of vertices and edges; on the other hand, when a walk traverses a loop, the choice of dart specifies the direction of traversal. At the risk of ambiguity, we sometimes use the more mnemonic notation $v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_k$ to describe a walk.

A walk is **simple** if all its vertices are distinct, except possibly the first and last. A **path** is a simple open walk; and a **cycle** is a simple closed walk. For example, a loop is a cycle of length 1, and a single vertex is a path (but not a cycle!) of length 0. We can safely regard paths and cycles as subgraphs rather than walks.

An **even subgraph** of a graph $G$ is a subgraph in which every vertex has positive, even degree. Every even subgraph is the union of one or more edge-disjoint cycles [25].

A graph is **connected** if it contains a path from any vertex to any other. A **component** of a graph is a maximally connected subgraph. A graph is **acyclic** if the graph does not contain a cycle; acyclic graphs are also called **forests**. Finally, a **tree** is any graph that is both connected and acyclic; thus, any forest is the disjoint union of trees.

A **cut** $G$ is a partition of the vertices $V$ of $G$ into two non-empty subsets $S$ and $V \setminus S$. An edge **crosses** the cut $(S, V \setminus S)$ if it has one endpoint in $S$ and the other in $V \setminus S$. The set of all edges that cross the cut is called the **boundary** of the cut and denoted $\partial S$. An **edge cut** is the boundary of some cut. An edge cut $C$ is called a **bond** if no proper subset of $C$ is also an edge cut; thus, a graph is connected if and only if it has a non-empty bond. Every non-empty edge cut is the union of one or more edge-disjoint bonds. For any connected graph $G$, an edge cut $C$ is a bond if and only if the subgraph $G \setminus C$ has exactly two components. A **bridge** is an edge cut consisting of a single edge.

**Spanning Trees**

A **spanning tree** of $G$ is a connected, acyclic subgraph of $G$ that includes every vertex of $G$. We leave the following lemma as an exercise for the reader.

**Lemma 2.1.** Let $G$ be a connected graph, and let $e$ be an edge of $G$. 

2.1. Graphs

(a) If $e$ is a loop, then every spanning tree of $G$ excludes $e$.
(b) If $e$ is not a loop, then for any spanning tree $U$ of $G / e$, the subgraph $U \cup e$ is a spanning tree of $G$.
(c) If $e$ is a bridge, then every spanning tree of $G$ includes $e$.
(d) If $e$ is not a bridge, then every spanning tree of $G \setminus e$ is also a spanning tree of $G$.

This lemma immediately suggests the following general strategy to compute a spanning tree of any connected graph: For each edge $e$, either contract $e$ or delete $e$. Loops must be deleted and bridges must be contracted; otherwise, the decision to contract or delete is arbitrary. Lemma 2.1 inductively implies that the set of contracted edges is a spanning tree of $G$, regardless of the order that edges are visited, or which non-loop non-bridge edges are deleted or contracted.

In practice, most algorithms that compute spanning trees do not actually contract or delete edges; rather, they simply label the edges as belonging to the spanning tree or not. In this context, Lemma 2.1 can be rewritten as follows:

**Lemma 2.2.** Let $G$ be a connected graph.

(a) Every spanning tree of $G$ excludes at least one edge from every cycle in $G$.
(b) For every edge $e$ of every cycle of $G$, there is a spanning tree of $G$ that excludes $e$.
(c) Every spanning tree of $G$ includes at least one edge from every bond in $G$.
(d) For every edge $e$ of every bond of $G$, there is a spanning tree of $G$ that includes $e$.

**Corollary 2.3.** If the edges of a connected graph $G$ are arbitrarily colored red or blue, so that each cycle in $G$ has at least one red edge and each bond in $G$ has at least one blue edge, then the subgraph of blue edges is a spanning tree of $G$.

Given a connected graph with $n$ vertices and $m$ edges, we can compute a spanning tree in $O(n + m)$ time using either depth-first search or breadth-first search; both algorithms can be seen as variants of the red-blue coloring algorithm, where the order in which edges are colored is determined on the fly. Similarly, given a disconnected graph, we can compute a spanning tree for each component in $O(n + m)$ time; this is the most efficient method to determine the number of connected components in a graph.
2.2 Planar Graphs

Embeddings

A planar embedding of a graph \( G \) is a continuous injective function from the topological graph \( G^\top \) to the plane. More explicitly, a planar embedding maps the vertices of \( G \) to distinct points in the plane and maps the edges of \( G \) to simple paths in the plane between the images of their endpoints, such that the paths do not intersect except at common endpoints. A planar graph is an abstract graph that has at least one planar embedding. Somewhat confusingly, a planar embedding of a planar graph is also called a plane graph.

For many proofs, it is actually more natural to consider graph embeddings on the sphere \( S^2 = \{(x, y, z) \mid x^2 + y^2 + z^2 = 1\} \) instead of the plane. Consider the standard stereographic projection map \( st: S^2 \setminus (0, 0, 1) \rightarrow \mathbb{R}^2 \), where \( st(x, y, z) := \left( \frac{x}{1-z}, \frac{y}{1-z} \right) \).

The projection \( st(p) \) of any point \( p \in S^2 \setminus (0, 0, 1) \) is the intersection of the line through \( p \) and the “north pole” \((0, 0, 1)\) with the \( x\,y \)-plane. Given any spherical embedding, if we rotate the sphere so that the embedding avoids \((0, 0, 1)\), stereographic projection gives us a planar embedding; conversely, given any planar embedding, inverse stereographic projection immediately gives us a spherical embedding. Thus, a graph is planar if and only if it has an embedding on the sphere.

The components of the complement of the image of a planar or spherical embedding are called the faces of the embedding. The Jordan curve theorem implies that every face of a spherical embedding of a connected graph is homeomorphic to an open disk. A planar embedding of a connected graph has a single unbounded outer face, which
is homeomorphic to the complement of a closed disk. For disconnected graphs, some faces are homeomorphic to an open disk with a finite number of open disks removed.

The faces on either side of an edge of a planar embedding are called the shores of that edge. For any dart $d$, the face just to the left of the image of $d$ in the embedding is called the left shore of $d$, denoted $\text{left}(d)$; symmetrically, the face just to the right is the right shore of $d$, denoted $\text{right}(d)$. The same face may be both the left shore and right shore of a dart. We say that an edge $e$ and a face $f$ are incident if $f$ is one of the shores of $e$; similarly, an vertex $v$ and a face $f$ are incident if $v$ and $f$ have a common incident edge. The degree of a face $f$, denoted $\deg_G(f)$ (or just $\deg(f)$ if $G$ is clear from context), is the number of darts whose right shore is $f$.

Let $F$ be the set of faces of a planar embedding of a connected graph with vertices $V$ and edges $E$. We refer to the triple $(V, E, F)$ as a planar map. Similarly, a spherical map consists of the vertices, edges, and faces of an embedding of a connected graph onto the sphere. Trapezoidal decompositions and triangulations of polygons are both examples of planar maps. A planar or spherical map is called a triangulation if every face (including, for planar maps, the outer face) has degree 3. The underlying graph of a triangulation is not necessarily simple.

At the risk of further confusing the reader, but following standard practice, we often use the same symbol $G$ to simultaneously denote an abstract planar graph $G$, the corresponding topological graph $G^\top$, the image of a planar or spherical embedding of $G$ (which, by definition, is homeomorphic to $G^\top$), and the resulting planar map $(V, E, F)$. In particular, a vertex of an embedding of $G$ is the point associated with a vertex of $G$, and an edge of the embedding is the path associated with an edge of $G$.

Rotation Systems

As usual in topology, we are not really interested in particular embeddings, but rather equivalence classes of embeddings, where two embeddings are equivalent if one can be continuously deformed into the other. Fortunately, every equivalence class of embeddings has a concrete combinatorial representation, called a rotation system.
Recall that a permutation of a finite set $X$ is a bijection $\pi : X \leftrightarrow X$. For any permutation $\pi$ and any element $x \in X$, let $\pi^0(x) := x$ and $\pi^k(x) := \pi(\pi^{k-1}(x))$ for any integer $k > 0$. The orbit of an element $x$ is the set \{ $\pi^k(x) \mid k \in \mathbb{N}$ \} = \{ $x, \pi(x), \pi^2(x), \ldots$ \}. The restriction of $\pi$ to any of its orbits is a cyclic permutation; the infinite sequence $x, \pi(x), \pi^2(x), \ldots$ repeatedly cycles through the elements of the orbit of $x$. Thus, the orbits of any two elements of $X$ are either identical or disjoint.

The rotation system for a graph embedding is a permutation of its darts, called the successor permutation. The successor $\text{succ}(d)$ of any dart $d$ is the next dart entering $\text{head}(d)$ in counterclockwise order after $d$.²

The faces of any connected graph embedding are also encoded in its rotation system. Recall that $\text{rev}$ is the reversal permutation of the darts of a graph. For any dart $d$, the dual successor $\text{next}(d) := \text{rev}(\text{succ}(d))$ is the next dart after $d$ in clockwise order around the boundary of $\text{right}(d)$.

For disconnected graphs, it is impossible to determine from a rotation system which boundary components belong to the same face of an embedding, or even whether the components of the graph are embedded on the same sphere or on different spheres; this information must be recorded separately. (Many authors prefer to define the faces of any embedding to be the orbits of the permutation next, even when the graph is disconnected; see, for example, Klein [52]. This is equivalent to declaring that each component of a disconnected graph is embedded on its own surface.) On the other hand, since almost all graph algorithms consider each component of the input graph independently, this extra information is rarely actually used. Thus, from now on we implicitly assume that all embedded graphs are connected, unless explicitly stated otherwise. With this assumption in place, the orbits of $\text{next}$ correspond precisely to the faces of the embedding.

We call a rotation system planar if it describes a planar (or spherical) embedding. Every rotation system corresponds to a embedding of a graph on some orientable surface, but not necessarily the sphere or the plane. Later in this chapter, we describe how to construct a planar embedding that is consistent with a given planar rotation system.

Rotation systems trace their origins to Hamilton’s Icosian Calculus [40, 41, 42], which can be seen as a rotation system for the regular dodecahedron, and to early research by Kirkman [48, 49, 50] and Cayley [14] on enumerating convex polyhedra. A more complete account of the history of rotation systems appears in Chapter ??.
2.3 Straight-Line Embeddings

So far we have not assumed that planar embeddings are in any way well-behaved; edges may be embedded as arbitrarily pathological paths. It is not hard to prove using a compactness argument that any planar graph has a piecewise-linear planar embedding, meaning a planar embedding in which every edge is embedded as a simple polygonal path. (We will see a similar compactness argument in the next chapter.)

However, with the Jordan curve theorem in hand, we can actually prove a stronger result, namely that every simple planar graph has a straight-line embedding, meaning an embedding in which every edge is embedded as a single line segment. This result was first proved (indirectly) by Steinitz [80], and later independently rediscovered by Wagner [87], Fáry [29], Stein [81], and Stojaković [82]. We say that two planar graph embeddings are equivalent if they have the same rotation system.

Theorem 2.4. Every planar embedding of a simple planar graph $G$ is equivalent to a straight-line embedding of $G$.

Proof: Fix a simple plane graph $G$ with at least four vertices (since otherwise the theorem is trivial). If any face of $G$ has degree greater than 3, it must contain a path between two non-adjacent vertices; adding this path as a new embedded edge yields a planar embedding of a larger simple plane graph. Thus, without loss of generality, we can assume that $G$ is a simple triangulation.

Now we actually prove the following even still more stronger result. Let $H$ be a simple plane graph in which every bounded face has degree 3 and whose outer face is bounded by a simple cycle of length $h \geq 3$. Suppose the outer face of $H$ has vertices $v_1, v_2, \ldots, v_h$ in counterclockwise order. Let $P$ be an arbitrary convex polygon with $h$ vertices $p_1, p_2, \ldots, p_h$ in counterclockwise order. We claim that there is a straight-line embedding of $H$, equivalent to the given embedding, such that $P$ is the boundary of the outer face, and each vertex $v_i$ is mapped to the corresponding point $p_i$.

If $H$ is a single triangle, the claim is trivial, so assume otherwise. There are two other cases to consider.

Case 1. Suppose the only neighbors of $v_h$ on the outer face are $v_1$ and $v_{h−1}$. In this case, we recursively compute an embedding of the subgraph $H' = H \setminus v_h$ and then embed the edges incident to $v_h$ as line segments.

Specifically, let $w_1, w_2, \ldots, w_d$ be the neighbors of $v_h$, indexed in clockwise order around $v_h$ so that $w_1 = v_{h−1}$ and $w_d = v_1$. The vertices $w_2, \ldots, w_{d−1}$ all lie in the complement of the outer face, and $H$ contains the edge $w_iw_{i+1}$ for every index $i$. It follows that the outer face of the subgraph $H' = H \setminus v_h$ is bounded by a simple cycle.
2. Planar Graphs

Case 1: Only two neighbors on the outer face; remove the vertex and recurse.

with vertices $v_1, v_2, \ldots, v_{h-1} = w_1, w_2, \ldots, w_d = v_1$. Every bounded face of $H'$ is also a bounded face of $H$ and thus has degree 3.

Now let $\alpha$ be a convex arc from $p_1$ to $p_{h-1}$ inside the triangle $p_1p_hp_{h-1}$. (For example, $\pi$ could be a circular arc tangent to $p_1p_h$ at $p_1$ and tangent to $p_hp_{h-1}$ at $p_{h-1}$.) Place $d$ evenly spaced points $q_1, q_2, q_3, \ldots, q_d$ along $\alpha$, with $q_1 = p_{h-1}$ and $q_d = p_1$. Finally, let $P'$ be the convex polygon obtained by replacing the edges $p_{h-1}p_h$ and $p_hp_1$ with the polygonal chain $q_1q_2\cdots q_d$.

The inductive hypothesis implies that there is a straight-line embedding of $H'$ whose outer face is $P'$, that maps each vertex $v_i$ (with $i \neq h$) to the corresponding point $p_i$ and each vertex $w_i$ to the corresponding point $q_i$. Mapping the vertex $v_h$ to the point $p_h$ and mapping each edge $v_hw_i$ to the line segment $p_hq_i$ gives us the required embedding of $H$.

Case 2. Now suppose $v_h$ is adjacent to some vertex $v_j$ on the outer face besides $v_1$ and $v_{h-1}$. In this case, we split $H$ into two subgraphs along the edge $v_hv_j$, split the polygon $P$ into two smaller polygons along the diagonal $p_hp_j$, and recursively embed each subgraph of $H$ into the corresponding fragment of $P$.

This case is actually redundant, but it resembles the polygon-triangulation proof, so it’s worth keeping for intuition.

Specifically, let $H^\dagger$ be the subgraph of $H$ obtained by deleting every vertex outside the simple cycle $v_hv_1v_2\cdots v_jv_h$ in the given embedding. (The outside of this cycle is well-defined by the Jordan Curve Theorem.) Similarly, let $H^\ddagger$ be the subgraph of $H$ obtained by deleting every vertex outside the cycle $v_hv_jv_{j-1}\cdots v_{h-1}v_h$. Both $H^\dagger$ and $H^\ddagger$ satisfy the conditions of our claim.

The line segment $p_hp_j$ partitions the polygon $P$ into two smaller convex polygons $P^\dagger$ and $P^\ddagger$. The induction hypothesis implies that $H^\dagger$ and $H^\ddagger$ have straight-line embeddings,
equivalent to their given embeddings, with respective outer faces bounded by $P^\flat$ and $P^\sharp$, mapping each vertex $v_i$ to the corresponding point $p_i$. In particular, both embeddings map the edge $v_h v_j$ to the line segment $p_h p_j$. Combining the straight-line embeddings of $H^\flat$ and $H^\sharp$ gives us the required straight-line embedding of $H$. □

The following corollaries are now immediate; we leave the omitted details as an exercise for the reader.

**Corollary 2.5.** Every planar embedding of a planar graph is equivalent to a piecewise-linear embedding.

**Corollary 2.6.** (a) Every component of a planar graph is planar.
(b) Every subgraph of a planar graph is planar.
(c) Every minor of a planar graph is planar.

**Corollary 2.7.** Given a planar rotation system for a planar graph $G$ with $n$ vertices and $m$ edges, we can compute an equivalent piecewise-linear embedding (or an equivalent straight-line embedding if $G$ is simple) in $O(n + m)$ time.

Steinitz [80] actually proved a much more subtle and difficult generalization of the straight-line embedding theorem. A graph is **3-connected** if it remains connected after deleting any two vertices.

**Theorem 2.8 (Steinitz [80]).** A simple planar graph is the graph of a 3-dimensional convex polytope if and only if it is 3-connected.

In fact, Steinitz’s theorem implies that every 3-connected simple planar graph has a **convex** embedding, meaning an embedding in which every bounded face is convex and the complement of the outer face is convex. This corollary was further strengthened, and given a more direct proof, by Tutte [84,85].

**Theorem 2.9 (Tutte [84,85]).** Any planar embedding of a simple 3-connected planar graph is equivalent to a convex embedding, in which every vertex not on the outer face lies at the center of mass of its neighbors. Moreover, the outer face can be chosen to be the complement of any convex polygon with the correct number of vertices.

Our proof of Theorem 2.4 roughly follows an argument of de Fraysseix et al. [33,34], who actually prove that any $n$-vertex planar graph has a straight-line embedding whose vertices lie on an $O(n) \times O(n)$ integer grid.
2. Planar Graphs

2.4 Duality

Definition

Fundamentally, a rotation system is just a pair \((\text{succ}, \text{rev})\) of permutations of a finite set of abstract objects, such that every orbit of \(\text{rev}\) has exactly two elements. The objects being permuted can be interpreted as the darts of an embedded graph \(G\) whose vertices are the orbits of \(\text{succ}\), whose edges are the orbits of \(\text{rev}\), and whose faces are the orbits of \(\text{succ} \circ \text{rev}\).

The dual of a rotation system \((\text{succ}, \text{rev})\) is the rotation system \((\text{rev} \circ \text{succ}, \text{rev})\). The embedded graph \(G^*\) determined by this dual rotation system is called the dual graph of \(G\). That is, the vertices of \(G^*\) are orbits of \(\text{rev} \circ \text{succ}\); the edges of \(G^*\) are the orbits of \(\text{rev}\); and the faces of \(G^*\) are the orbits of \(\text{succ}\). Thus, each vertex \(v\), edge \(e\), dart \(d\), or face \(f\) of the original graph \(G\) corresponds to—or more evocatively, “is dual to”—a distinct face \(v^*\), edge \(e^*\), dart \(d^*\), or vertex \(f^*\) of the dual graph \(G^*\), respectively. The endpoints of any primal edge \(e\) are dual to the shores of the corresponding dual edge \(e^*\), and vice versa. Specifically, for any dart \(d\), we have \(\text{tail}(d^*) = \text{left}(d)\) and \(\text{head}(d^*) = \text{right}(d)\), and symmetrically, \(\text{left}(d^*) = \text{tail}(d)\) and \(\text{right}(d^*) = \text{head}(d)\).

![A planar embedded graph and its dual. One dart and its dual are emphasized.](image)

Attentive readers may have noticed that the rotation system of a graph encodes the \textit{counterclockwise} order of darts leaving each vertex, while the dual rotation system encodes the \textit{clockwise} order of darts around each face. Formally, to avoid this inconsistency, the graphs \(G\) and \(G^*\) use opposite orientations of the plane or sphere to distinguish left from right, and clockwise from counterclockwise. Intuitively, \(G\) and \(G^*\) are drawn on opposite sides of the plane or sphere.

We can also define the dual graph \(G^*\) directly in terms of a topological embedding of \(G\), as follows. Choose an arbitrary point \(f^*\) in the interior of each face \(f\) of \(G\). Let \(F^*\) denote the collection of all such points. For any edge \(e\) of \(G\), choose a path \(e^*\) between the chosen points in the shores of \(e\), such that \(e^*\) intersects \(e\) once transversely and does not intersect any other edge of \(G\). Let \(E^*\) denote the collection of all such paths. Then the dual graph \(G^*\) is the topological graph with vertices \(F^*\) and edges \(E^*\). By
construction, $G^*$ is embedded in the sphere; one can verify mechanically that each face of $G^*$ contains exactly one vertex of $G$.

We must emphasize that duality is a correspondence between embedded graphs, not between abstract graphs. An abstract planar graph can have many non-isomorphic planar embeddings, each of which defines a different abstract dual graph. Moreover, the dual of a simple embedded graph is not necessary simple; any vertex of degree 2 in $G$ gives rise to parallel edges in $G^*$, and any bridge in $G$ is dual to a loop in $G^*$. This is why we don’t want graphs to be simple by definition!

Duality is an involution; the dual of $G^*$ is the original graph $G$. This observation is a trivial consequence of the combinatorial definition (Proof: $\text{rev} \circ \text{rev} \circ \text{succ} = \text{succ}$ □), but with some care, it can also be proved directly from the topological formulation.

When the graph $G$ is clear from context, we abuse notation by writing $H^*$ to denote the subgraph of $G^*$ containing the edges dual to the edges of a subgraph $H$ of $G$.

### Correspondences

The correspondence between primal and dual edges easily extends to larger structures within any connected planar graph $G$. For example, as we already observed, an edge $e$ is a bridge in $G$ if and only if the dual edge $e^*$ is a loop in $G^*$. The following tables summarize some of these correspondences; we develop these further in the next several lemmas.

<table>
<thead>
<tr>
<th>primal $G$</th>
<th>dual $G^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>vertex $v$</td>
<td>face $v^*$</td>
</tr>
<tr>
<td>dart $d$</td>
<td>dart $d^*$</td>
</tr>
<tr>
<td>edge $e$</td>
<td>edge $e^*$</td>
</tr>
<tr>
<td>face $f$</td>
<td>vertex $f^*$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>primal $G$</th>
<th>dual $G^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>empty loop</td>
<td>spur</td>
</tr>
<tr>
<td>loop</td>
<td>bridge</td>
</tr>
<tr>
<td>cycle</td>
<td>bond</td>
</tr>
<tr>
<td>even subgraph</td>
<td>edge cut</td>
</tr>
<tr>
<td>spanning tree</td>
<td>complement of spanning tree</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>primal $G$</th>
<th>dual $G^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G \setminus e$</td>
<td>$G^* \setminus e^*$</td>
</tr>
<tr>
<td>$G / e$</td>
<td>$G^* / e^*$</td>
</tr>
<tr>
<td>$\text{minor } G \setminus X / Y$</td>
<td>$\text{minor } G^* \setminus Y^* / X^*$</td>
</tr>
</tbody>
</table>

Correspondences between features of primal and dual planar maps
Lemma 2.10 (Contraction ↔ deletion). Fix a connected plane graph $G$. For any edge $e$ of $G$ that is not a loop, we have $(G / e)^* = G^* \setminus e^*$. Symmetrically, for any edge $e$ of $G$ that is not a bridge, we have $(G \setminus e)^* = G^* / e^*$.

Proof: Let $\text{succ}$ denote the rotation system of $G$, and let $\text{next} = \text{succ} \circ \text{rev}$ denote its dual rotation system. Pick an arbitrary edge $e$ of $G$. There are two cases to consider.

First, suppose $e$ is not a loop. Then $G / e$ is a connected plane graph that contains every dart in $G$ except $e^+$ and $e^-$. Let $\text{succ} / e$ and $\text{next} / e$ denote the induced primal and dual rotation systems of $G / e$. Then for any dart $d$ of $G / e$, we have

$$(\text{succ} / e)(d) = \begin{cases} \text{succ}(e^-) & \text{if } \text{succ}(d) = e^+, \\ \text{succ}(e^+) & \text{if } \text{succ}(d) = e^-, \\ \text{succ}(e) & \text{otherwise} \end{cases}$$

and

$$(\text{next} / e)(d) = \begin{cases} \text{next}(e^-) & \text{if } \text{next}(d) = e^+, \\ \text{next}(e^+) & \text{if } \text{next}(d) = e^-, \\ \text{next}(e) & \text{otherwise} \end{cases}$$

On the other hand, suppose $e$ is not a bridge. Then $G \setminus e$ is a connected plane graph that contains every dart in $G$ except $e^+$ and $e^-$. Let $\text{succ} \setminus e$ and $\text{next} \setminus e$ denote the induced primal and dual rotation systems of $G \setminus e$. Then for any dart $d$ of $G \setminus e$, we have

$$(\text{succ} / e)(d) = \begin{cases} \text{succ}(e^+) & \text{if } \text{succ}(d) = e^+, \\ \text{succ}(e^-) & \text{if } \text{succ}(d) = e^-, \\ \text{succ}(e) & \text{otherwise} \end{cases}$$

and

$$(\text{next} \setminus e)(d) = \begin{cases} \text{next}(e^-) & \text{if } \text{next}(d) = e^+, \\ \text{next}(e^+) & \text{if } \text{next}(d) = e^-, \\ \text{next}(e) & \text{otherwise} \end{cases}$$

These two cases are obviously symmetric. By definition, the dual rotation system next is the rotation system of $G^*$. \qed

Lemma 2.11 (Even subgraph ↔ edge cut). Fix a connected plane graph $G$. A subgraph $H$ is an even subgraph of $G$ if and only if $H^*$ is an edge cut in $G^*$.

Proof: Let $H$ be an even subgraph of $G$. Let $C_1, C_2, \ldots, C_k$ be edge-disjoint cycles in $G$ whose union is $H$. Color each vertex of $G^*$ black if it lies in the interior of an odd number of cycles $C_i$, and white otherwise. Any path in $G^*$ from a white vertex to a black vertex
must cross some edge in $H$, and therefore must contain some dual edge in $H^*$. We conclude that $H^*$ is a cut in $G^*$.

On the other hand, let $H^*$ be an edge cut in $G^*$. Let $S^*$ be a subset of vertices of $G^*$ such that $H^* = \partial S^*$. Color a face of $G$ black if its dual vertex lies in $S^*$ and white otherwise. The primal subgraph $H$ contains precisely the edges of $G$ with one white shore and one black shore. Every vertex in $G$ is clearly incident to an even number of such edges. We conclude that $H$ is an even subgraph of $G^*$.

Corollary 2.12 (Cycle $\iff$ bond). Fix a connected plane graph $G$. A subgraph $H$ is a cycle in $G$ if and only if $H^*$ is a bond in $G^*$.

Proof: A cycle is a minimal even subgraph, and a bond is a minimal edge cut. □

Corollary 2.12 was first proved by Whitney [89,90]. Whitney actually proved the converse of this result as well; yielding the following result. An algebraic dual of an abstract graph $G$ is another abstract graph $G^*$ with the same set of edges, such that a subset of edges defines a cycle in $G$ if and only if the same subset defines a bond in $G^*$.

Theorem 2.13 (Whitney [89,90]). A connected abstract graph is planar if and only if it has an algebraic dual.

Corollary 2.14 (Spanning tree $\iff$ spanning cotree). Fix a connected plane graph $G$. A subgraph $T$ is a spanning tree of $G$ if and only if $G^* \setminus T^*$ is a spanning tree of $G^*$.

Proof: Let $T$ be an arbitrary spanning tree of $G$, and let $C^* = G^* \setminus T^*$ be the complementary dual subgraph of $T$. Lemma 2.2 implies that every cycle of $G$ excludes at least one edge in $T$, and every bond of $G$ contains at least one edge in $T$. Thus, Corollary 2.12 implies that every bond of $G^*$ contains at least one edge in $C^*$, and every cycle of $G^*$ excludes at least one edge in $C^*$. We conclude from Lemma 2.2 that $C^*$ is a spanning tree of $G^*$.
Self-Dual Data Structures

Recall that the incidence list data structure for any graph $G$ stores a permutation of the darts entering each vertex of $G$, in the form of a linked list; thus, any incidence list actually represents a (not necessarily planar) rotation system for $G$. If we add a predecessor pointer to $\text{succ}^{-1}(d)$ to the record of every dart $d$, to complement the existing pointers to $\text{succ}(d)$ and $\text{rev}(d)$, the resulting data structure allows us to quickly navigate either the embedding of $G$ or the induced dual embedding $G^*$. To emphasize the importance of the order of the linked lists, we refer to this data structure as a **sorted incidence list**.

Although the dual embedding $G^*$ is implicitly encoded in the sorted incidence list of $G$, it is often more convenient to store the dual embedding explicitly. A **self-dual incidence list** of $G$ is essentially an overlay of the sorted incidence lists of $G$ and $G^*$. This data structure consists of two arrays, one indexed by vertices of $G$ and the other by faces of $G$, and a set of dart records. The record for each dart $d$ stores the index of the vertex $\text{head}(d)$, the index of the face $\text{left}(d)$, and pointers to five darts $\text{rev}(d)$, $\text{succ}(d)$, $\text{succ}^{-1}(d)$, $\text{next}(d)$, and $\text{next}^{-1}(d)$. For each vertex $v$, the corresponding entry in the vertex array points to an arbitrary dart $d$ with $\text{head}(d) = v$; symmetrically, for each face $f$, the corresponding entry in the face array points to an arbitrary dart $d$ with $\text{right}(d) = f$. Any self-dual incidence list of $G$ is also a self-dual incidence list list of $G^*$, in which the interpretations of the $\text{succ}$ and $\text{next}$ pointers, the $\text{head}$ and $\text{left}$ pointers, and the vertex and face arrays are swapped; no actual modification of the data structure is required.

Sorted and self-dual incidence lists are two examples of a wide family of so-called **half-edge** data structures. Several variants appear in the literature, some omitting the...
vertex and face arrays, some storing the tails of darts instead of heads, some combining both dart records for each edge into a single edge record, some sorting only by the dual rotation system next instead of succ, some storing darts in arrays instead of linked lists, some storing vertices and faces in linked lists instead of arrays, and so on. These low-level design choices have no impact on the design and theoretical analysis of the algorithms that use the resulting data structures, and so we will largely ignore them. From a practical standpoint, however, these variants reflect significant tradeoffs between computation speed, compactness, ease of implementation, and ease of use.

One of the earliest examples of a half-edge data structure is the winged-edge data structure proposed by Baumgart in 1975 [3, 35], which keeps a single record for each edge, with pointers to both endpoints, both shores, and the four neighboring edges. In 1977, Preparata and Hong described an algorithm for computing three-dimensional convex hulls, represented by sorted incidence lists [72]; only a year later, Muller and Preparata describe this data structure as “one of the most commonly used representations for a planar graph” [67]. Other examples include an unnamed data structure of Danaraj and Klee [19], Muller and Preparata’s doubly-connected edge list [67], Eastman’s half-edge and split-edge structures [23, 62], Weiler’s vertex-edge and face-edge structures [88], Chen’s doubly-connected face list [16], and the doubly-connected edge list described by de Berg et al. [4] (which is slightly different from Muller and Preparata’s structure of the same name).

Another family of data structures represent each edge by four records rather than two. Each record is associated with a blade [9]: an edge with a direction, specifying which endpoint is the tail, and an independent orientation, specifying which shore is on the left. Examples of blade-based data structures include the graph-encoded maps or gem representation proposed by Lins [59], Eastman’s corner structure [23], Guibas and Stolfi’s popular quad-edge data structure [39], Lienhardt’s generalised maps [57, 58], and Brisson’s cell-tuple structure [7]. Blade-based data structures offer some additional flexibility that half-edge data structures do not, which we will revisit in Chapter ??.

2.5 Euler’s Formula

Arguably the earliest fundamental result in combinatorial topology is a simple formula first published by Leonhard Euler. We provide three short proofs, one directly inductive, one relying on tree-cotree decompositions, and one relying on straight-line embeddings.

Euler’s Formula for Planar Graphs. For any connected plane graph $G$ with $n$ vertices, $m$ edges, and $f$ faces, we have $n - m + f = 2$.

Proof (induction): If $G$ has no edges, it has one vertex and one face. Otherwise, let $e$ be any edge in $G$; there are two overlapping cases to consider.
2. Planar Graphs

exercises for the reader
maximally planar

- If \( e \) is not a loop, then contracting \( e \) yields a connected plane graph \( G / e \) with \( n - 1 \) vertices, \( m - 1 \) edges, and \( f \) faces. The induction hypothesis implies that 
  \[(n - 1) - (m - 1) + f = 2.\]

- If \( e \) is not a bridge, then deleting \( e \) yields a connected plane graph \( G \setminus e \) with 
  \( n \) vertices, \( m - 1 \) edges, and \( f - 1 \) faces. The induction hypothesis implies that 
  \[n - (m - 1) + (f - 1) = 2.\]

In all cases, we conclude that \( n - m + f = 2. \)

\[\Box\]

Proof (von Staudt [78]): Let \( T \) be a spanning tree of \( G \). Because \( T \) has \( n \) vertices, it 
also has \( n - 1 \) edges. The complementary dual subgraph \( C^* = (G \setminus T)^* \) is a spanning 
tree of \( G^* \). Because \( C^* \) has \( f \) vertices, it also has \( f - 1 \) edges. Every edge in \( G \) is either 
an edge of \( T \) or the dual of an edge in \( C^* \), but not both. Thus, \( m = (n - 1) + (f - 1). \)

Proof (l’Huiller [55, 56]): It suffices to prove the theorem for simple triangulations. 
If \( G \) is not a simple graph, we can make it simple by splitting each edge into a path of 
three edges, by introducing two new vertices; the resulting simple graph has \( n + 2m \) 
vertices, \( 3m \) edges, and \( f \) faces. Similarly, if any face of \( G \) has degree greater than 3, 
it must contain a path between two non-adjacent vertices; adding this path to the 
embedding yields a new plane graph with \( n \) vertices, \( m + 1 \) edges, and \( f + 1 \) faces. 
Finally, Theorem 2.4 implies that we only need to consider straight-line embeddings.

So let \( G \) be a simple straight-line plane triangulation with \( n \) vertices, \( m \) edges, and 
\( f \) faces (including the outer face). We compute the sum of the interior angles around 
the vertices of \( G \) in two different ways. First, each face is a triangle, so the sum of all 
angles is \( \pi f \). Second, the angles around each interior vertex sum to \( 2\pi \), and the angles 
around the three boundary vertices also sum to \( 2\pi \) (specifically, \( \pi \) from the bounded 
faces plus \( \pi \) from the inverted outer face). Thus, \( \pi f = 2\pi(n - 3) + 2\pi \), which implies 
that \( f = 2n - 4 \). Finally, because every face is a triangle, we have \( 2m = 3f = 3n - 6 \). 
We conclude that \( n - m + f = n - (3n - 6) + (2n - 4) = 2. \)

\[\Box\]

- Proof by Schnyder wood!

Euler’s formula has several straightforward but useful consequences, whose proofs 
we leave as exercises for the reader. A simple planar graph \( G \) is **maximally planar** if 
inserting a new edge between any two distinct non-adjacent vertices makes the graph 
non-planar.

Corollary 2.15. Every simple planar graph \( G \) with \( n \geq 3 \) vertices has at most \( 3n - 6 \) edges 
and at most \( 2n - 4 \) faces, with equality if some embedding of \( G \) is a triangulation.

Corollary 2.16. A simple planar graph \( G \) is maximally planar if and only if every planar 
embedding of \( G \) is a triangulation.

Corollary 2.17. Every simple planar graph with at least six vertices has a vertex with 
degree less than 6.
2.5. Euler’s Formula

Some Muddled History

Like the Jordan curve theorem, the early history of Euler’s formula is complex and confusing. Literally dozens of proofs were published in the 19th century alone; the precise conditions of the formula are themselves a subject of intense debate; most of the early proofs were either incorrect or incomplete; even when correct, many early authors overstated the generality of their results; and finally, a complete formal proof requires the Jordan curve theorem! Here I briefly sketch only a few important early landmarks, using modern terminology and notation. Brückner [8] provides a much more thorough (but still incomplete) survey of the literature up to about 1900; see also the more recent survey by Eppstein [24].

Early statements of Euler’s formula did not consider arbitrary planar graphs, but only convex polyhedra. The first incarnation of the formula appears an unpublished manuscript of Descartes [21,22], written more than 200 years before Euler’s rediscovery. Descartes observed without proof that the sum of all the plane angles in any convex polyhedron is \( 2\pi(n-2) \); starting from this observation, Descartes proved that \( f = 2n-4 \) if every face is a triangle and that the number of plane angles is always \( 2f + 2n - 4 \).\(^5\)

Euler first described both his formula \( n + f = m + 2 \) and the angle-sum formula \( \Sigma = 2\pi(n-2) \) in a letter to Goldbach in 1750 [26]. Twelve days later, he presented both formulas to the St. Petersburg Academy of Sciences [28]; he did not prove his formula, but he did derive the angle-sum formula from it. Two years later, Euler proposed an inductive proof [27]. Specifically, Euler proposed a method for removing an arbitrary vertex and its incident faces, and then patching the resulting hole with new triangles, to obtain a new polyhedron with \( n - 1 \) vertices, \( m - 3 \) edges, and \( f - 2 \) faces; the polyhedral formula immediately follows by induction. However, Euler’s vertex-deletion algorithm sometimes yields a non-convex "polyhedron" with disconnected interior, which eventually makes further progress impossible; consequently, Euler’s proof is incorrect. In hindsight, Euler’s argument is easy to fix; to retriangulate the hole, it suffices to compute the convex hull of the undeleted vertices, and then arbitrarily triangulate any non-triangular faces.

Karsten included Euler’s formula in his influential series of textbooks on mathematics and physics [47], after learning of the formula directly from Euler [76]. Karsten offered the following inductive proof. Decompose the polyhedron into pyramids, by joining each facet to a common interior point. By a direct counting argument, each individual pyramid satisfies Euler’s formula. Now delete the pyramids one at a time until only one pyramid remains, and consider the number of vertices and facets of the remaining solid. Deleting a pyramid whose base has \( b \) edges and that shares \( c \) contiguous triangular faces with the remaining undeleted pyramids yields a polyhedron with \( n - b + c + 1 \) vertices, \( m - 2b + 3c \) edges, and \( f + 2c - b - 1 \) faces. Unfortunately, Karsten’s argument is incomplete, because it requires that each deleted pyramid share a connected set of faces with the remaining solid; carelessly removing pyramids can easily violate this invariant. In more modern terminology, Meister assumes that any plane graph has a
shelling. In hindsight, Karsten’s proof is easy to fix; a suitable deletion order can be derived from any spanning cotree. Nearly identical proofs, with the same flaw, were later independently proposed by Meister [66] (for polyhedra with triangular facets) and L’Huillier [55, 56] (in addition to the angle-based proof given above).

The first complete proof of Euler’s polyhedral formula was given by Legendre [54]. Legendre projects the vertices and edges of the polyhedron onto the unit sphere from an arbitrary interior point, and then applies the already well-known fact that a spherical triangle with interior angles $\alpha$, $\beta$, and $\gamma$ has area $\alpha + \beta + \gamma - \pi$. Suppose the original polyhedron has $n$ vertices and $f$ facets, all triangles. The angles at each vertex of the resulting spherical triangulation sum to exactly $2\pi$; thus, the total area of all $f$ spherical triangles is $2\pi n - \pi f$. We immediately conclude that $f = 2n - 4$, because the surface area of the unit sphere is $4\pi$. The proof for more general polyhedra follows by triangulating the faces. Essentially the same proof was later given by Hirsch [43].

Cauchy [13] is usually cited as the author of the first purely combinatorial proof of Euler’s formula. However, all three of Cauchy’s proofs closely follow the inductive shelling strategy published by Karten almost 50 years earlier, and suffer from precisely the same flaw. In short, Cauchy’s proof is neither Cauchy’s nor a proof—it is a fungus! In two of his proofs (one with triangular faces, the other for arbitrary faces), Cauchy removes a single face of the polyhedron, projects the remaining faces into the plane, and then recursively removes faces from the resulting planar map. Cauchy’s third proof recursively removes tetrahedra from a decomposition of the polyhedron, obtained by joining a vertex to all non-incident faces. Cauchy’s planar arguments were slightly simplified, but not repaired, by Grunert [38]. The first correct combinatorial proof appears to be von Staudt’s tree-cotree proof [78]. Again, in hindsight, von Staudt’s proof can be interpreted as a formalization of the Karsten-Meister-L’Huillier-Cauchy-Grunert inductive shelling argument; indeed, Brückner [8] incorrectly attributes the tree-cotree proof to Cauchy.

The first proofs that explicitly consider arbitrary plane graphs are due to Cayley [15] and Listing [60]. In fact, both Cayley and Listing allowed their graphs to include isolated closed “edges” with no vertices, and Listing considered much more general “acyclic spatial complexes” constructed by gluing disks to cycles in graphs. Cayley’s argument is a prototype for our first inductive proof; he observed that the quantity $n - m + f$ does not change when one inserts a new vertex in the interior of an edge or inserts a new edge in the interior of a face. Listing repeats (and further generalizes) Cauchy’s proof, using a global counting argument instead of induction, but again assuming without proof the existence of a shelling. Both of these proofs implicitly assume the Jordan curve theorem, and therefore are technically incomplete (shelling issues aside).

Jordan [45] neatly sidestepped the Jordan curve theorem by defining a surface map (or “polyhedron”) to be “Eulerian” if every simple cycle separates its surface into two components. Jordan’s proof that all “Eulerian” polyhedra satisfy Euler’s formula can be applied without modification to arbitrary planar maps. Jordan also avoided the shelling issue by using a divide-and-conquer strategy, similar to the second case of our proof of
Theorem 2.4, instead of removing faces one by one.

### 2.6 Minimum Spanning Trees

In many applications of graphs, there is a numerical weight associated with each edge of the graph. For example, the graph might model a road network, where each vertex represents a city, each edge represents a road, and the weight of an edge is the length of the corresponding road. Or the graph might represent an digital image, where each vertex is a pixel, edges join adjacent pixels, and the weight of each edge reflects the similarity between the corresponding pixels.

A common task in many graph algorithms is finding a minimum spanning tree of an edge-weighted graph $G$; this is a spanning tree of $G$ whose total weight is no bigger than total weight of any other spanning tree of $G$. Minimum spanning trees were originally studied as a model of efficient communication networks [5, 6], but have since proved useful in many other contexts, including more complex network design and optimization problems, clustering, image processing, preconditioning linear systems, and computing approximate solutions of several NP-hard problems. Graham and Hell [36] give a detailed early history of the minimum spanning tree problem, tracing several classical algorithms to their (multiple) original sources. The more recent survey Mareš [64] gives a thorough overview of the state of the art. Finding minimum spanning trees is relatively straightforward in arbitrary graphs, but Euler’s formula implies even simpler and faster algorithms for planar graphs.

Following standard practice, we report running times of graph algorithms as functions of two variables $n$ and $m$, which respectively denote the number of vertices and the number of edges of the input graph. If the input graph is simple and planar, Euler’s formula implies $m = O(n)$; thus, we report running times for simple planar graphs only as a function of $n$. On the other hand, if the input graph is connected, then $m \geq n - 1$.

To simplify exposition, we implicitly assume that all edge weights are distinct; this assumption implies that the minimum spanning tree is unique. Our assumption can be enforced if necessary by the following simple tie-breaking rule. Arbitrarily index the edges from 1 to $m$; if two edges have the same weight, proceed as though the edge with smaller index has smaller weight.

**Tarjan’s Red-Blue Meta-Algorithm**

Tarjan [83] observed that several classical minimum spanning tree algorithms can be described by a single general strategy. Color each edge of $G$ blue if it is the lightest edge in some bond, or red if it is the heaviest edge in some cycle.

**Lemma 2.18 (Tarjan’s “blue rule”).** Let $e$ be any blue edge in any edge-weighted graph $G$, and let $T$ be the minimum spanning tree of $G / e$. Then $T \cup e$ is the minimum spanning tree of $G$. 

minimum spanning tree

$n$

$m$
**Proof:** Let $G$ be any edge-weighted graph, let $B$ be an arbitrary bond in $G$, and let $e$ be the lightest edge in $B$. Let $T$ be any spanning tree of $G$ that *excludes* the blue edge $e$. The spanning tree $T$ contains a unique path between the endpoints of $e$; at least one edge $e'$ in this path must lie in the bond $B$. The subgraph $T' = (T \cup e) \setminus e'$ is a spanning tree of $G$ with smaller total weight than $T$, so $T$ cannot be the minimum spanning tree of $G$.

We conclude that $e$ is an edge in the minimum spanning tree; the lemma now follows from Lemma 2.2. \qed

We leave the symmetric proof of Tarjan’s red rule as an *exercise for the reader*.

**Lemma 2.19 (Tarjan’s “red rule”).** Let $e$ be any red edge in any edge-weighted graph $G$. The minimum spanning tree of $G / e$ is also the minimum spanning tree of $G$.

**Corollary 2.20.** In any edge-weighted graph $G$, every edge is either red or blue, but not both, and the subgraph of blue edges is the minimum spanning tree of $G$.

With these rules in place, Tarjan’s general strategy is simple: If the input graph $G$ has no edges, there is nothing to do; otherwise, either contract an arbitrary blue edge or delete an arbitrary red edge, and then recurse. The correctness of this strategy follows inductively from Tarjan’s blue and red rules, regardless of which rule is applied at each recursive call, or to which bond or cycle the rule is applied. For the sake of efficiency, we may prefer to contract several blue edges at once, or delete several red edges at once, rather than one at a time.

**Flattening**

Call an edge of $G$ redundant if it is a loop, or if it is not the lightest edge between its endpoints. Every redundant edge is the heaviest edge in a cycle of length 1 or 2; thus, redundant edges are in fact red. Thus, we can safely delete all redundant edges from a graph without changing its minimum spanning tree; we call this process *flattening* the graph.

**Lemma 2.21.** Any edge-weighted graph can be flattened in $O(n + m)$ time.

**Proof:** As usual, we assume that the graph is stored in an incidence list, with the weight of each edge stored in both of the dart records for that edge. To simplify the description of the algorithm, we treat each individual linked list as a *stack*, accessible only through the following operations:

- **EMPTY?(S):** Return True if the stack is empty and False otherwise.
- **PUSH(S, d):** Push a new dart $d$ onto stack $S$
- **POP(S):** Remove and return the newest dart in stack $S$
Each of these operations can be performed in $O(1)$ time if the stack is represented as a standard linked list. We also assume that each dart stores both its head and its tail.

Our Flatten algorithm performs two passes over the edges. The first pass transposes the adjacency list data structure, transforming the lists of darts entering each vertex into lists of darts leaving each vertex; the first pass also removes loops from the graph. An important side-effect of our transposition algorithm is that parallel darts are clustered together consecutively. The second pass transposes the graph again and removes parallel edges. Pseudocode for our Flatten algorithm appears below. The algorithm spends $O(1)$ time for each vertex and edge, so its overall running time is $O(n + m)$.

Flatten $G$:

1. ((Transpose and remove loops))
   for $i \leftarrow 1$ to $n$
     while $\neg$EMPTY($G[i]$)
       $d \leftarrow$ POP($G[i]$)
       if head($d$) $\neq$ tail($d$)
         PUSH($H[tail(d)]$, $d$)
       else
discard $d$

2. ((Transpose and remove parallel edges))
   for $i \leftarrow 1$ to $n$
     $d \leftarrow$ POP($H[i]$)
     while $\neg$EMPTY($H[i]$)
       $d' \leftarrow$ POP($H[i]$)
       if head($d$) $\neq$ head($d'$)
         PUSH($G[head(d)]$, $d$)
       else
         weight($d$) $\leftarrow$ min{weight($d$), weight($d'$)}
         discard $d'$
         PUSH($G[head(d)]$, $d$)
   return $G$

Flattening a graph in $O(m + n)$ time.

Borůvka’s Algorithm

The earliest algorithm to compute minimum spanning trees of arbitrary graphs was described by the Czech mathematician Otakar Borůvka in his 1926 PhD thesis [5, 6, 68], and later independently rediscovered several times [18, 31, 77]. Borůvka’s algorithm has the following simple description: Simultaneously contract the lightest edge incident to every vertex, flatten the contracted graph, and recurse on the resulting minor. The recursion halts when the graph has no edges. The set of edges incident to a vertex is a bond, so the lightest such edge is blue. Thus, Borůvka’s algorithm is an instance
of Tarjan’s general red-blue strategy, and therefore correctly computes the minimum spanning tree.

```
BORŮVKA(G):
if G has no edges
    return ∅
L ← ∅
for each vertex v of G
    add the lightest edge incident to v to L
return L ∪ BORŮVKA(FLATTEN(G / L))
```

Borůvka’s minimum spanning tree algorithm.

We can find the lightest edge incident to each vertex in $O(m)$ time by a brute-force traversal of the incidence list of $G$. We compute the contracted graph $G / L$ in $O(m)$ time as follows. First, we compute a label $\ell(v)$ for every vertex $v$, such that any two vertices have the same label if and only if they lie in the same component of the subgraph $L$. These labels can be computed in $O(m)$ time by a depth- or breadth-first search of the subgraph $L$; the set of labels is also the vertex set of $G / L$. Then we copy each dart of $G$ into a new incidence list for $G / L$, using the endpoint labels as vertices; that is, each dart $u \rightarrow v$ in $G$ becomes a dart $\ell(u) \rightarrow \ell(v)$ in $G / L$ with the same weight. In particular, each edge in $L$ becomes a loop, which is deleted when the graph is flattened. Finally, flattening $G / L$ requires $O(m)$ time. Thus, ignoring the cost of the recursive call, Borůvka’s algorithm runs in $O(m)$ time. Each round of contraction reduces the number of vertices by at least a factor of 2, so the algorithm ends after $O(\log n)$ recursive calls. Thus, for arbitrary graphs, the entire algorithm runs in $O(m \log n)$ time.

However, Cheriton and Tarjan [17] observed that Borůvka’s algorithm actually runs in $O(m)$ time when the input graph is planar. After the first contraction round, Corollary 2.6(c) implies that the contracted graph $G / L$ is planar, so Euler’s formula implies that $\text{FLATTEN}(G / L)$ has $O(n)$ edges. Thus, the second contraction round requires only $O(n)$ time. Moreover, because each contraction round removes at least half the vertices, the running times of successive rounds decrease geometrically. Thus, the total time for all rounds after the first is only $O(n)$.

**Theorem 2.22.** Borůvka’s algorithm computes the minimum spanning tree of any connected edge-weighted planar graph in $O(m)$ time.

**Mareš’s Algorithm**

The following “local” variant of Borůvka’s algorithm, proposed by Mareš [63], avoids the difficulty of computing the contraction $G / L$ by contracting edges one at a time and only contracting edges incident to low-degree vertices. Unlike Borůvka’s original algorithm, Mareš’s algorithm does not work for arbitrary graphs.

Any edge $vw$ can be contracted in time $O(\deg(v))$ by moving all the darts from $v$’s linked list into $w$’s, and then marking $v$ as deleted in the top-level array. Thus, each
2.6. Minimum Spanning Trees

MAREŠ(G):
if G has no edges
    return ∅
L ← ∅
while G has at least two vertices, one of which has degree at most 6
    v ← any vertex of G with degree at most 6
    e ← lightest non-loop edge incident to v
    G ← G / e
    Add e to L
return L ∪ MAREŠ(FLATTEN(G))

Mareš's minimum spanning tree algorithm for planar graphs.

contraction in Mareš requires only \(O(1)\) time, and because we cannot contract more
than \(n - 1\) edges, the entire while loop runs in \(O(n)\) time. The following lemma implies
that if the input graph is simple, then the while loop reduces the number of vertices by
a constant factor.

Lemma 2.23. Any simple planar graph with \(n\) vertices has at least \(n/4\) vertices of degree
at most 6.

Proof: Without loss of generality, let \(G\) be a triangulation. For each vertex \(v\), the number
\(δ(v) := \deg(v) - 3\) is non-negative. Euler's formula implies that \(\sum_v δ(v) ≤ 3n - 12\).
Thus, there are at most \(3n/4 - 3\) vertices \(v\) such that \(δ(v) ≥ 4\).

Suppose the input graph \(G\) is simple. At the start of the algorithm, \(G\) has at least
\(n/4\) vertices with degree at most 6, and each contraction reduces the number of such
vertices by at most 2. Thus, the while loop iterates at least \(n/8\) times, leaving a graph
with at most \(7n/8\) vertices. We conclude that the running times for successive rounds
decrease geometrically, and thus the total running time of Mareš's algorithm is \(O(n)\).
Again, if \(G\) is simple but non-planar, the first round requires \(O(m)\) time, and the rest
of the algorithm takes only \(O(n)\) time.

Algorithms for Planar Maps

Both Borůvka's algorithm and Mareš's incremental variant compute the minimum
spanning tree of any simple abstract planar graph in \(O(n)\) time; neither algorithm
requires an embedding. However, if we are lucky enough to be given a planar embedding,
there are even simpler linear-time algorithms.

The first algorithm is a further simplification of Mareš's incremental algorithm:
repeatedly find a vertex with degree at most 5, contract the lightest edge leaving that
vertex, and flatten the resulting graph. The key insight is that if the graph \(G\) is simple,
we can flatten the contracted graph \(G / e\) in only \(O(1)\) time. \(G / e\) has at most two
pairs of parallel edges. Specifically, if \(\text{right}(e^+)\) is a triangle, then the edges carrying
\(\text{next}(e^+)\) and \(\text{next}^{-1}(e^+)\) are parallel in \(G / e\); symmetrically, if \(\text{right}(e^-)\) is a triangle,
then the edges carrying $\text{next}(e^-)$ and $\text{next}^{-1}(e^-)$ are parallel in $G / e$. If the graph $G$ is simple, these are the only possible parallel edges in $G / e$. Each parallel pair appears consecutively in the rotation system of $G / e$ and thus are accessible in $O(1)$ time from the records associated with the contracted edge $e$.

A minimum spanning tree algorithm for embedded planar graphs.

Arguably an even easier approach is to choose a \textit{random} vertex at each iteration. Euler's formula implies that the expected degree of a random vertex in a simple planar graph is less than 6, so the total expected running time of the resulting algorithm is still $O(n)$.

The second algorithm, proposed by Matsui [65], eliminates flattening altogether by exploiting duality. Corollary 2.14 implies that if $T$ is the minimum spanning tree of a plane graph $G$, then the complementary dual subgraph $G^* \setminus T^*$ is the \textit{maximum} spanning tree of $G^*$. Euler's formula implies that every planar map, simple or not, contains either a vertex degree at most 3 or a face of degree at most 3. Thus, we immediately obtain a simple "self-dual" algorithm: at each iteration, either contract the lightest edge incident to a low-degree vertex (unless that edge is a loop), or delete the heaviest edge incident to a low-degree face (unless that edge is a bridge).

### 2.7 Exercises

1. Collected “exercises for the reader”:
   
   a) Prove Lemma 2.1.
   b) Prove Corollary 2.5.
   c) Prove Corollary 2.6.
   d) Prove Corollary 2.7.
   e) Prove Corollary 2.15.
   f) Prove Corollary 2.16.
   g) Prove Corollary 2.17.
   h) Prove Lemma 2.19.

2. Prove that every planar map has either a vertex with degree at most 3 or a face with degree at most 3.
2.7. Exercises

MATSUIEMBEDDED(G):
\[ T \leftarrow \emptyset \]
while \( G \) has more than one vertex or more than one face
\[ e \leftarrow \text{lightest edge incident to } v \]
\[ G \leftarrow G / e \]
add \( e \) to \( T \)

A self-dual minimum spanning tree algorithm for embedded planar graphs.

3. A graph \( G \) is **bipartite** if its vertices can be partitioned into disjoint subsets \( L \) and \( R \), such that every edge has one endpoint in \( L \) and one endpoint in \( R \). Prove that every simple bipartite planar graph has at most \( 2n - 4 \) edges.

4. A **coloring** of a graph \( G \) is an assignment of colors (elements of some finite set) to the vertices of \( G \) such that the endpoints of each edge have distinct colors. A **\( k \)-coloring** is a coloring that uses at most \( k \) distinct colors. The infamous **Four Color Theorem** states that every planar graph has a 4-coloring.
   a) Prove that every loopless planar graph has a 6-coloring.
   b) Prove that every loopless planar graph has a 5-coloring.

5. A three-dimensional convex polyhedron is **regular** if all faces have the same degree and all vertices have the same degree. To rule out degenerate cases like line segments and two-sided regular polygons, we insist that all vertex and face degrees are at least 3. Prove that the regular tetrahedron, the cube, the regular octahedron, the regular dodecahedron, and the regular icosahedron are the only regular convex polyhedra.

6. Kirkpatrick’s planar point-location data structure [51]:
   a) Prove that every simple planar graph with \( n \) vertices has an independent subset of at least \( n/12 \) vertices, each with degree less than 12.
   b) Describe an algorithm to compute such an independent set in \( O(n) \) time.
   c) Let \( G \) be a simple straight-line plane triangulation with \( n \) vertices. Describe a method to preprocess \( G \) in \( O(n) \) time, into a data structure of size \( O(n) \), so
that given an arbitrary point \( q \) in the plane, we can determine in \( O(\log n) \) time which triangle of \( G \) (if any) contains \( q \).
2.7. Exercises

Notes

1. (page 3) Readers bothered by the fact that an “incidence list” is not actually a list should remember Hirsch’s Red Herring Principle [44]. In computer science, as in mathematics, a red herring is neither necessarily red nor necessarily a fish; conversely, a herring that happens to be red is not necessarily a red herring.

   The ring worm is not ringed, nor is it worm. It is a fungus.
   The puff adder is not a puff, nor can it add. It is a snake.
   The funny bone is not funny, nor is it a bone. It is a nerve.
   The fishstick is not a fish, nor is it a stick. It is a fungus.


2. (page 8) Because the edges of a planar embedding can be arbitrary continuous paths, it is not immediately obvious that every planar embedding has a well-defined rotation system. The existence of such a rotation system follows from the observation that every embedding is isotopic to a piecewise-linear embedding; see note 3.

3. (page 9) Fix an arbitrary planar embedding of an abstract planar graph $G$. We will locally modify the embedding, first in the neighborhoods of the vertices, and then in the neighborhoods of the edges, so that it becomes piecewise-linear.

   First, let $\epsilon$ be the minimum distance between any two vertices of $G$. For each vertex $v$, let $C_v$ be a circle of radius $\epsilon/3$ centered at $v$. For each dart $d$, let $\pi_d$ denote the corresponding path from tail$(d)$ to head$(d)$ in the embedding. For each dart $d$, let $\sigma_d$ be a minimal subpath of $\pi_d$ that starts on $C_{\text{tail}(d)}$ and ends on $C_{\text{head}(d)}$; we can choose these paths so that $\sigma_{\text{rev}(d)}$ is the reversal of $\sigma_d$. Finally, let $\tau_d$ be the path consisting of the line segment from tail$(d)$ to $\sigma_d(0)$, the subpath $\sigma_d$, and the line segment from $\sigma_d(1)$ to head$(d)$. By construction, the new paths $\tau_d$ are interior-disjoint, and thus define a new planar embedding of $G$.

   Now let $\delta$ be the minimum distance between any two paths $\sigma_d$ and $\sigma_d'$. Because each path $\sigma_d$ is compact, $\sigma_d$ can be covered by a finite number of balls of radius $\delta/3$, each centered at a point on $\sigma_d$. It follows that there is a finite sequence of points $x_0,x_1,\ldots,x_k$ on $\sigma_d$, such that $x_0$ and $x_k$ are the endpoints of $\sigma_d$, and any two neighboring points $x_i$ and $x_{i+1}$ have distance at most $\delta/3$. The polygonal chain with vertices $\text{tail}(d), x_0, x_1, \ldots, x_k, \text{head}(d)$ may not be simple, but after removing a finite number of subloops, we obtain a simple polygonal chain $\sigma_d$ from $\text{tail}(d)$ to $\text{head}(d)$. By construction, the new piecewise-linear paths $\sigma_d$ are interior-disjoint, and thus define a piecewise-linear embedding of $G$.

   With more care, one can show that any planar embedding of $G$ is isotopic to a piecewise-linear embedding of $G$, meaning that there is a continuous deformation of the entire plane that transforms one embedding into the other.

ORLY? Citation needed!
4. (page 19) I will not add to the massive sea of ink that has already been spilled disputing the proper definition of “polyhedron”—and therefore the correct statement of Euler’s formula for all polyhedra—except to point to the detailed discussions by Lakatos [53] and Grünbaum [37].

5. (page 19) Descartes’ unpublished manuscript Progymnasmata de solidorum elementis [Exercises in the Elements of Solids] was most likely written around 1630 [22]. After Descartes’ death in Sweden in 1650, his possessions were shipped to his friend Claude Clerselier in Paris; upon arrival, a box of manuscripts, including the Progymnasmata, fell into the Seine and was not recovered for three days. After carefully drying them, Clerselier made Descartes’ manuscripts available to other scholars. Gottfried Leibniz transcribed several of these manuscripts, including the Progymnasmata, during a trip to Paris in 1676, most likely in an effort to collect evidence against recent charges by English mathematicians that his results were merely elaborations of Descartes’ ideas. (Isaac Newton charged Leibniz of plagiarizing his calculus later that same year.) Descartes’ original manuscript was then lost forever. Leibniz’s hand-written copy vanished into his archives for almost two centuries; its existence was unknown until 1859, when it was discovered in an uncatalogued pile of Leibniz’s papers by Louis Alexandre Foucher de Careil. Foucher de Careil published Leibniz’s transcription [32], but his re-transcription introduced several significant errors, rendering it essentially useless. An accurate transcription of the Progymnasmata finally appeared in 1908, thanks to the combined efforts of several Cartesian and Leibnizian scholars [2]. The remarkable story is told in more detail by Federico [30], Richeson [74], and (with some creative embellishment) Aczel [1].

It is a matter of considerable dispute whether Descartes actually stated Euler’s formula, and therefore deserves to share credit with Euler, or only came close, and therefore does not. The result that Descartes actually proves is the following [21, p. 269]:

\[
\text{Ponam semper pro numero angulorum solidorum } \alpha \text{ & pro numero facirum } \varphi \ldots.
\]
\[
\text{Numerus verorum angulorum planorum est } 2\varphi - 2\alpha - 4.
\]
\[
\text{[I always write } \alpha \text{ for the number of solid angles and } \varphi \text{ for the number of faces. . . .}
\]
\[
\text{The total number of plane angles is } 2\varphi - 2\alpha - 4.\]

In my opinion, the difference between Descartes’ and Euler’s formulas is one of notational emphasis, not content. As both Descartes [21, p. 268] and Euler [28, Proposition I] observe, the number of plane angles is exactly twice the number of edges. Had Descartes published his result, even Euler (who exhibited surprise that the formula was not already known) would have called it “Descartes’ formula”.

Lakatos [53], Malkevich [61], and several others argue that Descartes should not share credit with Euler, because he did not identify edges as useful components of polyhedra in their own right. But this assertion is actually false. In the second half of the Progymnasmata, Descartes derives closed-form polynomial expressions for figurate numbers based on the Platonic and Archimedean solids; Descartes describes these solids
by listing the number and shapes of the faces, the number of solid angles, and the
number of edges in each [21, pp. 271–275].

Moreover, the phrase “Euler’s formula” is commonly used to describe much more
general results that Euler never even suggested, including L’Huillier’s formula for poly-
hedra of higher genus [55, 56], Shläfi’s formula for higher-dimensional polytopes [75],
and Poincaré’s formula relating face counts and Betti numbers of polyhedral com-
plexes [69, 70]. In light of this sloppy generosity toward Euler, refusing to share credit
with Descartes because of a minor notational difference seems remarkably inconsistent.

6. (page 20) Meister cites both Euler [27, 28] and Karsten [47] in a footnote, in which
he claims ignorance of their prior work before Kästner brought it to Meister’s attention
shortly before publication. Both Karsten and Meister reproduce Euler’s argument that
any polyhedron with \( n \) vertices has between \([n/2] + 2\) and \(2n - 4\) facets. Meister
also proved the by construction that for any integers \( n \) and \( f \) such that \( n \geq 4 \) and
\([n/2] + 2 \leq f \leq 2n - 4\), a convex polyhedron with \( n \) vertices and \( f \) facets actually exists;
this result is usually attributed to Steinitz [79], who independently proved it 120 years
later.

Both Karsten and Meister appear to be almost completely forgotten. One of the
few places where Meister’s results are cited is Brückner’s 1900 treatise on polygons
and polyhedra [8], which includes an extremely detailed historical survey. Brückner
correctly observed that Meister’s work “…seems to have received little attention, be-
cause its results are later presented as new by others.” (“Die Abhandlung scheint aber
wenig beachtet worden zu sein, denn die in ihr niedergelegten Resultate werden ver-
schiedentlich später von anderen als neu vorgetragen.”)

In strict accordance with Stigler’s Law of Eponymy, recursively applied, Brückner’s
otherwise exhaustive survey does not mention Karsten at all. In fact, the only citation of
Karsten’s work on Euler’s formula I have seen is the footnote in Meister’s paper. Karsten
is slightly better known for his geometric interpretation of logarithms of negative and
complex numbers [46], although even this interpretation is usually attributed to De
Morgan [20]. Cajori [11, 12] describes the obscurity of Karsten’s discovery as follows:
“It looks very much as if the transactions of academies had been in some cases the safest
places for the concealment of scientific articles from the scientific public.”

7. (page 20) The lacuna in Cauchy’s proof argument is often ignored even by modern
mathematicians, even in the context of discussing formal proofs of Euler’s formula. For
example, Lakatos mentions the issue in passing [53, pp. 10–13] but clearly considers it
less worthy of discussion than Cauchy’s sloppy definition of “polyhedron”. Lakatos never
justifies Cauchy’s claim that a shelling order exists for any convex polyhedron; instead,
he cites a more recent proof by Reichart [73], written in answer to an “exercise” posed
by van der Waerden [86]. (In fact, Reichart proves a much stronger result: The triangles
in any infinite simply-connected surface triangulation can be indexed \( \triangle_1, \triangle_2, \ldots \) so that
for all \( n \geq 1 \), the union \( \bigcup_{i=1}^{n-1} \triangle_i \) is homeomorphic to a disk.)
More recently, in a discussion of persistent errors in mathematical proofs, Bundy et al. [10] correctly observe that Cauchy’s argument does not apply to all polyhedra (whatever that means), but like Lakatos, they do not notice that his argument is incomplete even for convex polyhedra. Categorical imprecision is apparently more philosophically interesting than simple omission.
2.7. Exercises

*I think the most annoying thing about secondary sources is not that they contain errors or off-the-wall interpretations, but that they NEVER include the specific information one is looking for.*  

Bibliography

2. Planar Graphs


2.7. Exercises


2. Planar Graphs


2. Planar Graphs


2.7. Exercises


