1

POLYGON PARTITIONS

1.1. INTRODUCTION

In 1973, Victor Klee posed the problem of determining the minimum number of guards sufficient to cover the interior of an n-wall art gallery room (Honsberger 1976). He posed this question extemporaneously in response to a request from Vasek Chvátal (at a conference at Stanford in August) for an interesting geometric problem, and Chvátal soon established what has become known as "Chvátal's Art Gallery Theorem" (or sometimes, "watchman theorem"): $\lfloor n/3 \rfloor$ guards are occasionally necessary and always sufficient to cover a polygon of n vertices (Chvátal 1975). This simple and beautiful theorem has since been extended by mathematicians in several directions, and has been further developed by computer scientists studying partitioning algorithms. Now, a little more than a decade after Klee posed his question, there are enough related results to fill a book. By no means do all these results flow directly from Klee's problem, but there is a cohesion in the material presented here that is consistent with the spirit of his question.

This chapter examines the original art gallery theorem and its associated algorithm. The algorithm leads to a discussion of triangulation, and a reexamination of the problem brings us to convex partitioning. The common theme throughout the chapter is polygon partitioning. Subsequent chapters branch off into specializations and generalizations of the original art gallery theorem and related algorithmic issues.

1.2. THE ORIGINAL ART GALLERY THEOREM AND ALGORITHM

1.2.1. The Theorem

Problem Definition

A *polygon* P is usually defined as a collection of n vertices $v_1, v_2, \ldots, v_n$ and n edges $v_1v_2, v_2v_3, \ldots, v_{n-1}v_n, v_nv_1$ such that no pair of nonconsecutive edges share a point. We deviate from the usual practice by defining a polygon as the closed finite connected region of the plane bounded by these vertices and edges. The collection of vertices and edges will be referred to as the boundary of P, denoted by ∂P; note that $\partial P \subseteq P$. The term "polygon" is often modified by "simple" to distinguish it from polygons that cross themselves, but in this book all polygons are simple, so we will drop the redundant modifier. The boundary of a polygon is a "Jordan curve": it separates the plane into two disjoint regions, the interior and the exterior of the polygon. A polygon of n vertices will sometimes be called an n-gon.

Let us say that a point $x \in P$ *sees* or *covers* a point $y \in P$ if the line segment xy is a subset of P: $xy \subseteq P$. Note that xy may touch ∂P at one or more points; that is, line-of-sight is not blocked by grazing contact with the boundary. For any polygon P, define $G(P)$ to be the minimum number of points of P that cover all of P: the minimum k such that there is a set of k points in P, $\{x_1, \dots, x_k\}$, so that, for any $y \in P$, some x_i, $1 \le i \le k$, covers y. Finally, define $g(n)$ to be the maximum value of $G(P)$ over all polygons of n vertices.

Klee's original art gallery problem was to determine $g(n)$: the covering points are guards who can survey 360° about their fixed position, and the art gallery room is a polygon. The function $g(n)$ represents the maximum number of guards that are ever needed for an n-gon: $g(n)$ guards always suffice, and $g(n)$ guards are necessary for at least one polygon of n vertices. We will phrase this as: $g(n)$ guards are occasionally necessary and always sufficient, or just necessary and sufficient.

Necessity

A little experimentation with small n quickly establishes a lower bound on $g(n)$. Clearly a triangle needs exactly one guard, so $g(3) = 1$. Even a non-convex quadrilateral can be covered by a single guard, so $g(4) = 1$. It is slightly less obvious that $g(5) = 1$, but there are only three distinct "shapes" of pentagons possible: those with 0, 1, or 2 *reflex* vertices (those with interior angle larger than 180°), and all three can be covered with one guard; see Fig. 1.1. For $n = 6$, there are two shapes (also shown in Fig. 1.1) that need two guards, so $g(6) = 2$. The second shape easily generalizes to a "comb" of k prongs and $n = 3k$ edges that requires k guards (Fig. 1.2) (Chvátal 1975). This establishes that $g(n) \ge \lfloor n/3 \rfloor$.

This situation is typical of the art gallery theorems that we will examine later: it is often easy to establish a lower bound through a generic example that settles the "necessity" of a particular formula. The difficult part is establishing sufficiency, as this needs an argument that holds for *all* polygons. Before showing our first sufficiency proof, we will briefly explore a few approaches that do not work.

False Starts

The formula $g(n) \ge \lfloor n/3 \rfloor$ could be interpreted as: one guard is needed for every three vertices. Phrased in this simple form, it is natural to wonder if

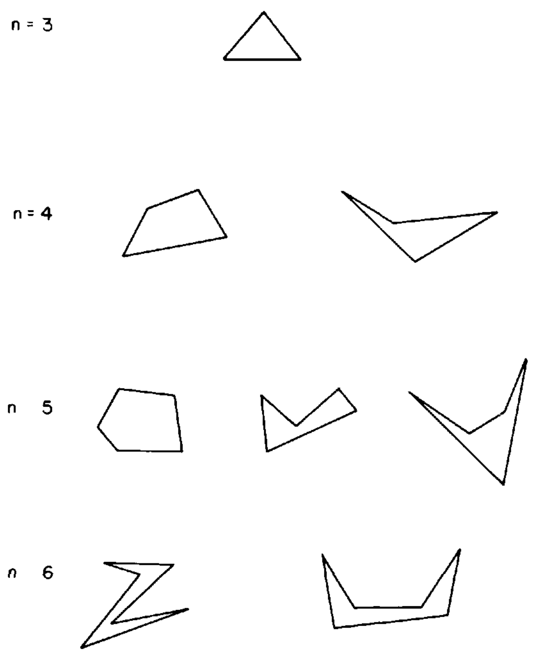

Fig. 1.1. Polygons with 5 or fewer vertices can be covered by a single guard, but some 6-vertex polygons require two guards.

perhaps a guard on every third vertex is sufficient. Figure 1.3 shows that such a simple strategy will not suffice: x_m in the figure will not be covered if guards are placed on all vertices i with $i \equiv m (\mathrm{mod}\ 3)$.

A second natural approach is to reduce visibility of the interior to visibility of the boundary: if guards are placed such that they can see all the paintings on the walls, does that imply that they can see the interior? Not necessarily, as Fig. 1.4 shows: guards at vertices a, b, and c cover the entire boundary but miss the internal triangle Q.

A third natural reduction is to restrict the guards to be stationed only at vertices. Define a *vertex guard* to be a guard located at a vertex; in contrast, guards who have no restriction on their location will be called *point guards.* Define $g_v(n)$ to be the number of vertex guards necessary and sufficient to cover an n-gon. Is $g_v(n) = g(n)$? Certainly there are particular polygons where the restriction to vertices weakens the guards' power: Fig. 1.5 shows one that needs two vertex guards but a single point guard placed at x suffices

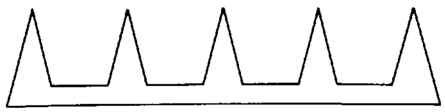

Fig. 1.2. Each prong of the comb requires its own guard. Here $n = 15$ and 5 guards are needed.

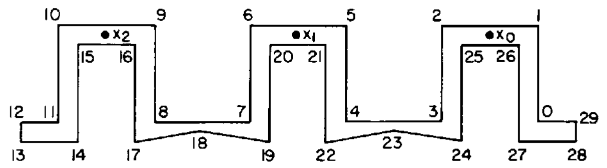

Fig. 1.3. Guards on every third vertex will not cover one of the points x_0, x_1, or x_2.

to cover the entire polygon. But $g(n)$ summarizes information about *all* polygons, so this particular case has no more impact on our question than does the existence of n-gons needing only one guard have on the value of $g(n)$. It turns out that in fact $g_v(n) = g(n)$ and that the reduction is appropriate. Its validity will fall out of the sufficiency proofs presented below, so we will not establish it independently. The reader is forewarned, however, that we will encounter many problems later for which the reduction to vertex guards is a true restriction and changes the problem in a fundamental way.

Fisk's Proof

We will step out of chronological order to sketch Fisk's sufficiency proof, which came three years after Chvátal's original proof (Fisk 1978; Honsberger 1981). Fisk's proof is remarkably simple, occupying just a single journal page. Its explication will introduce several concepts to which we will return later.

The first step in Fisk's proof is to "triangulate" the polygon P by adding internal diagonals between vertices until no more can be added. It is not obvious that a polygon can always be partitioned into triangles without adding new vertices this way; it is even less obvious how to perform the partition with an efficient algorithm. Triangulation is an important topic,

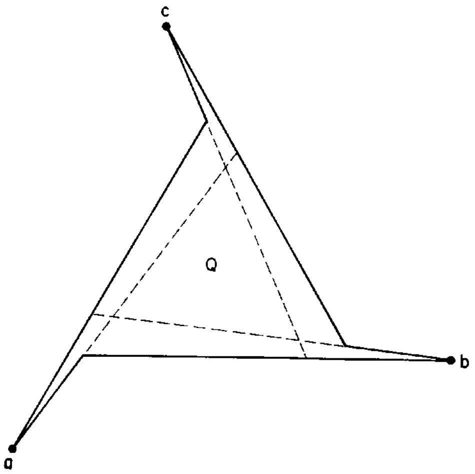

Fig. 1.4. Guards at a, b, and c cover the boundary but not the interior of the polygon.

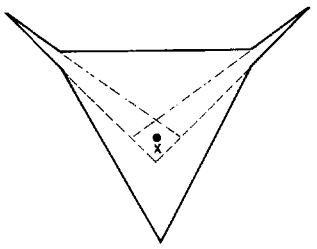

Fig. 1.5. Point guards are more powerful than vertex guards.

and will be covered in depth in Section 1.3. For now we will just assume that a triangulation always exists.

The second step is to "recall" that the graph of a triangulated polygon can be 3-colored. A *k-coloring* of a graph is an assignment of colors to the nodes, one color per node, using no more than *k* colors, such that no two adjacent nodes are assigned the same color. The nodes of the triangulation graph correspond to the vertices of the polygon, and the arcs correspond to the original polygon's edges plus the diagonals added during triangulation. Because a triangulation graph is planar, it is 4-colorable by the celebrated Four Color Theorem (Appel and Haken 1977). We will have to wait for the discussion of triangulation to formally prove that triangulation graphs of polygons are 3-colorable. Let us here just make the claim at least plausible via an example.

Consider the triangulation shown in Fig. 1.6a. Pick an arbitrary triangle, say *acg,* and 3-color it as shown with the colors 1, 2, and 3. The three diagonals *ac, cg,* and *ga* force the nodes *b, e,* and *i* to be colored 3, 1, and 2, respectively. Now diagonals involving the just-colored nodes force other colorings, and so on. The result is the coloring shown in Fig. 1.6b, which is unique given the initial arbitrary coloring of the first triangle: every "move" is forced after that, and since the polygon has no holes, the coloring never wraps around and causes a conflict. This argument will be formalized in Section 1.3.1.

Let us assume that the triangulation graph of a simple polygon can be 3-colored, and finish Fisk's proof.

The third step is to note that one of the three colors must be used no more than 1/3 of the time. Although this is obvious, let me be explicit since

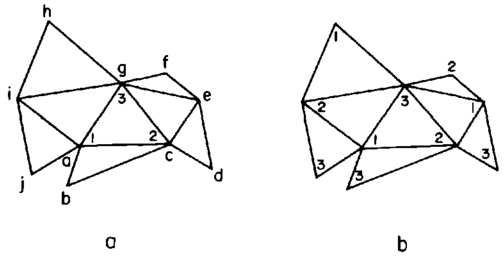

Fig. 1.6. Three-coloring of a triangulation graph starting from *acg.*

variants of this argument are used throughout the book. Let a, b, and c be the number of occurrences of the three colors in a coloring, with $a \leq b \leq c$. The total number of nodes is n, so $a + b + c = n$. If $a > n/3$, then the sum of all three would be larger than n. Therefore, $a \leq \lfloor n/3 \rfloor$ (since a must be an integer).

Let the least frequently used color be red. The fourth and final step is to place guards at every red node. Since a triangle is the complete graph on three nodes, each triangle has all three colors at its vertices. Thus every triangle has a red node and thus a guard in one of its corners. Moreover, since the triangles form a partition of P, every point in the polygon is inside some triangle, and since triangles are convex, every point is covered by a red guard. Thus the guards cover the entire polygon, and there are at most $\lfloor n/3 \rfloor$ of them.

This establishes that $\lfloor n/3 \rfloor$ guards are sufficient to cover the interior of an arbitrary polygon. Together with the necessity proved earlier, we have that $g(n) = \lfloor n/3 \rfloor$.

Chvátal's Proof

The first proof of Chvátal's Art Gallery Theorem was of course given by Chvátal, in 1975 (Chvátal 1975). His proof starts with a triangulation of the polygon, as does Fisk's, but does not use graph coloring. Rather the theorem is proven directly by induction. Although Chvátal's proof is not as concise as Fisk's, it reveals aspects of the problem that are not brought to light by the coloring argument, and we will see in Chapter 3 that Chvátal's argument generalizes in cases where Fisk's does not.

Define a *fan* as a triangulation with one vertex (the fan *center*) shared by all triangles. Chvátal took as his induction hypothesis this statement:

Induction Hypothesis: Every triangulation of an n-gon can be partitioned into $g \leq \lfloor n/3 \rfloor$ fans.

For the basis, note that $n \geq 3$ since we start with an n-gon, and that there is just a single triangulation possible when $n = 3$, 4, and 5, each of which is a fan; see Fig. 1.7. Thus the induction hypothesis holds for $n < 6$.

Given a triangulation with $n \geq 6$, our approach will be to remove part of the triangulation, apply the induction hypothesis, and then put back the deleted piece. We will see in the next section that there is always a diagonal (in fact, there are always at least two) that partitions off a single triangle. But note that this only reduces n by 1, and if we were unlucky enough to start with $n \equiv 1$ or $2 \pmod{3}$, then the induction hypothesis partitions into $g = \lfloor (n-1)/3 \rfloor = \lfloor n/3 \rfloor$ fans, and we will in general end up with $g + 1$ fans

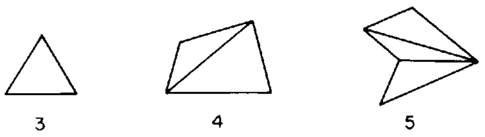

Fig. 1.7. Triangulations of up to five vertices are necessarily fans.

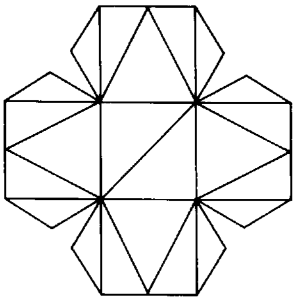

Fig. 1.8. No diagonal of this triangulation cuts off exactly three vertices.

when we put back the removed triangle. The moral is that, in order to make induction work with the formula $\lfloor n/3 \rfloor$, we have to reduce n by at least 3 so the induction hypothesis will yield less than g fans, allowing the grouping of the removed triangles into a fan.

So the question naturally arises: does there always exist a diagonal that partitions off 4 edges of the polygon, and therefore reduces n by 3? The answer is *no*, as established by Fig. 1.8 (this is not the smallest counterexample). Chvátal's brilliant stroke was to realize that there is always a diagonal that cuts off 4 *or* 5 *or* 6 edges:

LEMMA 1.1 [Chvátal 1975]. For any triangulation of an n-gon with $n \geq 6$, there always exists a diagonal d that cuts off exactly 4, 5 or 6 edges.

Proof. Choose d to be a diagonal that separates off a minimum number of polygon edges that is at least 4. Let $k \geq 4$ be the minimum number, and label the vertices of the polygon $0, 1, \ldots, n-1$ such that d is $(0, k)$; see Fig. 1.9. d must support a triangle T whose apex is at some vertex t with $0 \leq t \leq k$. Since $(0, t)$ and (k, t) each cut off fewer than k edges, by the minimality of k we have $t \leq 3$ and $k - t \leq 3$. Adding these two inequalities yields $k \leq 6$. □

Now the plan is to apply the induction hypothesis to the portion on the other side of the special diagonal d. Let G_1 be the triangulation partitioned off by d; it has $k+1$ boundary edges and hence is a $(k+1)$-gon (see Fig.

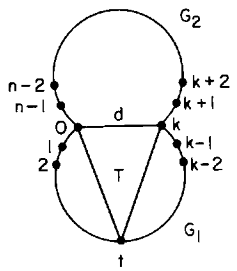

Fig. 1.9. Diagonal d cuts off k vertices in G_1.

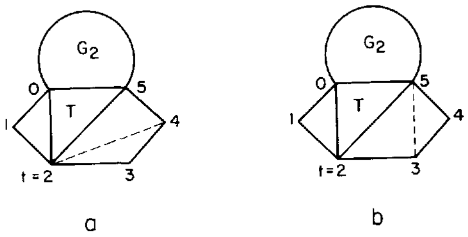

Fig. 1.10. G_1 is a hexagon.

1.9). Let G_2 be the remainder of the original triangulation, sharing d; it has $n-k+1$ vertices. The induction hypothesis says that G_2 may be partitioned into $g' = \lfloor (n-k+1)/3 \rfloor$ fans. Since $k \geq 4$, $g' \leq \lfloor (n-3)/3 \rfloor = \lfloor n/3 \rfloor - 1$. Thus, in order to establish the theorem, we have to show that G_1 need only add one more fan to the partition. We will consider each possible value of k in turn.

Case 1 ($k=4$). G_1 is a 5-gon. We already observed (Fig. 1.7) that every pentagon is a fan. Therefore, G has been partitioned into $\lfloor n/3 \rfloor - 1 + 1 = \lfloor n/3 \rfloor$ fans.

Case 2 ($k=5$). G_1 is an 6-gon. Consider the triangle T of G_1 supported by d, with its apex at t. We cannot have $t=1$ or $t=4$, as then the diagonals $(0,t)$ or $(5,t)$ [respectively] would cut off just 4 edges, violating the assumed minimality of $k=5$. The cases $t=2$ and $t=3$ are clearly symmetrical, so assume without loss of generality that $t=2$; see Fig. 1.10. Now the quadrilateral $(2,3,4,5)$ can be triangulated in two ways:

Case 2a. The diagonal $(2,4)$ is present (Fig. 1.10a). Then G_1 is a fan, and we are finished.

Case 2b. The diagonal $(3,5)$ is present (Fig. 1.10b). Form the graph G_0 as the union of G_2 and T. G_0 has $n-5+1+1=n-3$ edges. Apply the induction hypothesis to it, partitioning it into $g' = \lfloor (n-3)/3 \rfloor = \lfloor n/3 \rfloor - 1$ fans. Now T must be part of a fan F in the partition of G_0, and the center of F must be at one of T's vertices:

*Case 2b.*1. F is centered at 0 or 2. Then merge $(0,1,2)$ into F, and make $(2,3,4,5)$ its own fan. Now all of G is covered with $g'+1 = \lfloor n/3 \rfloor$ fans.

Case 2b.2. F is centered at 5. Merge both $(2,3,5)$ and $(3,4,5)$ into F, and make $(0,1,2)$ a separate fan. The result is $g'+1$ fans.

Case 3 ($k=6$). G_1 is a 7-gon. The tip t of the triangle T supported by d cannot be at 1, 2, 4, or 5, as then a diagonal would exist that cuts off $4 \leq k < 6$ edges, contradicting the minimality of k. Thus $t=3$. Each of the two quadrilaterals $(0,1,2,3)$ and $(3,4,5,6)$ has two possible triangulations, leading to four subcases.

Case 3a. The diagonals $(3,1)$ and $(3,5)$ are present (Fig. 1.11a). Then G_1 is a fan centered at 3, and we are finished.

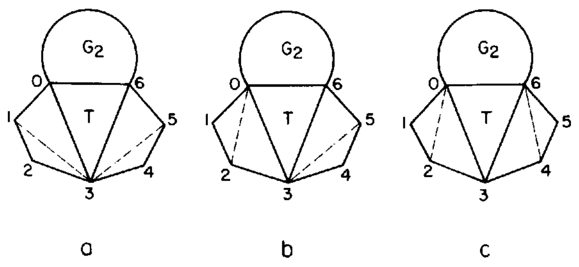

Fig. 1.11. G_1 is a heptagon.

Case 3b. The diagonals $(0,2)$ and $(3,5)$ are present (Fig. 1.11b). Join the quadrilateral $(0,2,3,6)$ to G_2 to form a polygon G_0 with $n-6+1+2=n-3$ vertices, which by the induction hypothesis can be partitioned into $g' = \lfloor n/3 \rfloor - 1$ fans. Let F be the fan of this partition to which the triangle $(0,2,3)$ belongs. The center of F must be at one of its vertices:

Case 3b.1. F is centered at 0 or 2. Merge $(0,1,2)$ into F and make $(3,4,5,6)$ a separate fan.

Case 3b.2. F is centered at 3. Merge $(3,4,5,6)$ into F, and make $(0,1,2)$ a separate fan.

In all cases, G is partitioned into $g'+1 = \lfloor n/3 \rfloor$ fans.

Case 3c. The diagonals $(1,3)$ and $(4,6)$ are present. This is the mirror image of Case 3b.

Case 3d. The diagonals $(0,2)$ and $(4,6)$ are present (Fig. 1.11c). Merge T with G_2 to form a polygon G_0 of $n-6+1+1=n-4$ vertices. Applying the induction hypothesis partitions G_0 into $g' = \lfloor (n-4)/3 \rfloor \le \lfloor n/3 \rfloor - 1$ fans. Let F be the fan of the partition containing T.

Case 3d.1. F is centered at 0. Merge the quadrilateral $(0,1,2,3)$ into F, and make $(3,4,5,6)$ a separate fan.

Case 3d.2. F is centered at 3. Since all of G_2 is behind the $d=(0,6)$ diagonal, it is clear that we can just as well consider F to be centered at 0, falling into Case 3d.1.

Case 3d.3. F is centered at 6. This is the mirror image of Case 3d.1.

In all cases, G is partitioned into $g'+1 = \lfloor n/3 \rfloor$ fans.
This completes the proof. Placing guards at the fan centers establishes the theorem:

THEOREM 1.1 [Chvátal's Art Gallery Theorem 1975]. $\lfloor n/3 \rfloor$ guards are occasionally necessary and always sufficient to see the entire interior of a polygon of n edges.

Note that both Chvátal's and Fisk's proofs incidentally establish by construction that the guards can be chosen to be vertex guards. We now turn to designing an algorithm to perform the stationing of the guards.

1.2.2. The Algorithm of Avis and Toussaint

A naive implementation of the construction used in Chvátal's proof would lead to an algorithm that is quadratic at best: $O(n)$ searches for the special diagonal d would cost $O(n^2)$. However, Avis and Toussaint mimicked Fisk's proof rather directly to obtain an $O(n \log n)$ algorithm (Avis and Toussaint 1981a).

Their algorithm follows the main steps of the proof:

Algorithm 1.1.

(1) Triangulate P, obtaining a graph G.
(2) Three-color the nodes of G.
(3) Place guards at the nodes assigned the least-frequently used color.

Step (1) is a very difficult problem, the topic of the next section. We will see that it can be accomplished in $O(n \log \log n)$ time. Step (2) is easy if you assume that complete triangle adjacency information is contained in the data structure for G output from Step (1). As the triangulation algorithm papers were unconcerned with this issue, Avis and Toussaint assume only that a list of the diagonals of the triangulation is available. Under these minimal assumptions, 3-coloring is not so trivial.

They propose to 3-color by a divide-and-conquer strategy. Their divide step partitions the polygon into two pieces, each of at least $\lfloor n/4 \rfloor$ vertices. Recursively assuming that each piece is 3-colored, the merge step makes a 3-coloring of the whole by relabeling if necessary. Both the divide and the merge steps require only $O(n)$ time, leading to the familiar recurrence equation $T(n) = 2T(n/2) + O(n)$, whose solution is $T(n) = O(n \log n)$.

We now describe the division step.

LEMMA 1.2 [Avis and Toussaint 1981]. Any triangulation of a polygon P of n vertices contains a diagonal d that partitions it into two pieces each containing at least $\lfloor n/4 \rfloor$ vertices.

Proof. Label the vertices of P $1, \ldots, n$. Partition the vertices into four chains C_1, C_2, C_3, C_4, each of length at least $\lfloor n/4 \rfloor$: chain C_i consists of vertices $(i-1)\lfloor n/4 \rfloor + 1$ through $i\lfloor n/4 \rfloor$ for $i = 1, 2, 3$, and C_4 consists of $3\lfloor n/4 \rfloor + 1$ through n.

First note that there must exist an i and j, $i \neq j$, such that a vertex in C_i is connected by a diagonal to a vertex in C_j. Otherwise an interior region would be bound by at least four diagonals, contradicting the assumption that the diagonals form a triangulation.

If there exists such an i and j with $|i-j| = 2$, the lemma is established by the following argument. Let $i = 1$ and $j = 3$ without loss of generality, and let d be a diagonal from C_1 to C_3. Then C_2 is on one side of d and C_4 on the other; thus each piece is of size at least $\lfloor n/4 \rfloor$.

Finally, suppose there do not exist such an i and j with $|i-j| = 2$. Let $i = 1$ and $j = 2$ without loss of generality. Let v_1 be the lowest numbered vertex in C_1 that connects to a vertex in C_2, and let v_2 be the highest numbered

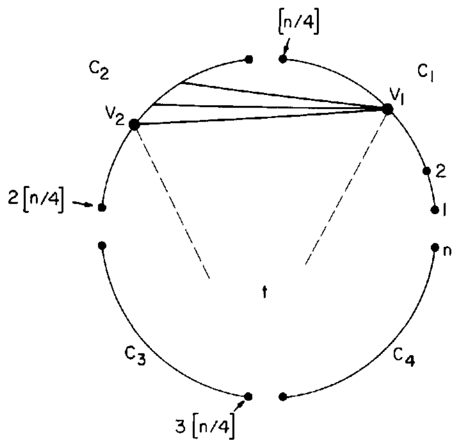

Fig. 1.12. The apex t of the triangle v_1v_2t must lie in either C_3 or C_4.

vertex in C_2 that connects to a vertex in C_1. Clearly v_1v_2 is a diagonal of the triangulation; see Fig. 1.12. Let t be the apex of the triangle supported by v_1v_2 outside of the indices in the range $[v_1, v_2]$. t cannot be in either C_1 or C_2, as that would contradict the extremality of either v_1 or v_2 in their chains. If t is in C_3, then v_1t connects C_1 to C_3; if t is in C_4, then v_2t connects C_2 to C_4. Both cases contradict our assumptions, showing that this last case cannot occur. □

We will encounter a significant extension of this lemma in the next section.

Now that we have established the existence of an appropriate dividing diagonal, it is easy to see how to find one in linear time. Simply check each of the $n-3$ diagonals (see Theorem 1.2 following) and see if its endpoints lie in either C_1 and C_3 or C_2 and C_4.

Finally, we consider the merge step. After recursively applying the algorithm, we have a 3-coloring of G_1 and G_2, the two graphs whose union is G. If the shared diagonal d is colored the same in each part, then no action is necessary. If the diagonal endpoints are assigned different colors in G_1 and G_2, simply relabel the colors in G_2 to accord with G_1's assignment to d. This relabeling will take $O(n)$ time, the size of G_2.

Step (3) of the algorithm clearly takes just linear time, resulting in an $O(n \log n)$ algorithm overall.

1.3. TRIANGULATION

We have encountered triangulations several times, and the concept will be used throughout the book: as the most basic polygon partition possible, its role in the field is analogous to the role of prime factorization in number theory. In this section we will first prove that triangulations exist, and then examine a series of algorithms for constructing a triangulation.

1.3.1. Theorems

When first confronted with the question, "Must all polygons admit a triangulation?," a natural reaction is, "How could they not?" Indeed, they cannot not, but this is still a fact in need of proof; a simple inductive proof follows.

THEOREM 1.2 (Triangulation Theorem). A polygon of n vertices may be partitioned into $n-2$ triangles by the addition of $n-3$ internal diagonals.

Proof. The proof is by induction on n. The theorem is trivially true for $n=3$. Let P be a polygon of $n \geq 4$ vertices. Let v_2 be a convex vertex of P, and consider the three consecutive vertices v_1, v_2, v_3. (We take it as obvious that there must be at least one convex vertex.) We seek an internal diagonal d.

If the segment v_1v_3 is completely interior to P (i.e., does not intersect ∂P), then let $d = v_1v_3$. Otherwise the closed triangle $v_1v_2v_3$ must contain at least one vertex of P. Let x be the vertex of P closest to v_2, where distance is measured perpendicular to v_1v_3 (see Fig. 1.13), and let $d = v_2x$.

In either case, d divides P into two smaller polygons P_1 and P_2. If P_i has n_i vertices, $i = 1, 2$, then $n_1 + n_2 = n + 2$ since both endpoints of d are shared between P_1 and P_2. Clearly $n_i \geq 3$, $i = 1,2$, which implies that $n_i < n$, $i = 1, 2$. Applying the induction hypothesis to each polygon results in a triangulation for P of $(n_1 - 2) + (n_2 - 2) = n - 2$ triangles, and $(n_1 - 3) + (n_2 - 3) + 1 = n - 3$ diagonals, including d. □

COROLLARY. The sum of the interior angles of a polygon is $(n-2)\pi$.

Proof. Each of the $n-2$ triangles consumes π of the total interior angle. □

Next, we make an important observation about the way the triangles in a triangulation fit together.

LEMMA 1.3. The dual graph of a triangulation of a polygon, with a node for each triangle and an arc connecting two nodes whose triangles share a diagonal, is a tree with each node of degree at most 3.

Proof. That each node has degree no greater than 3 is immediate from the

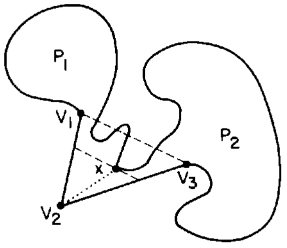

Fig. 1.13. The line segment xv_2 is an internal diagonal.

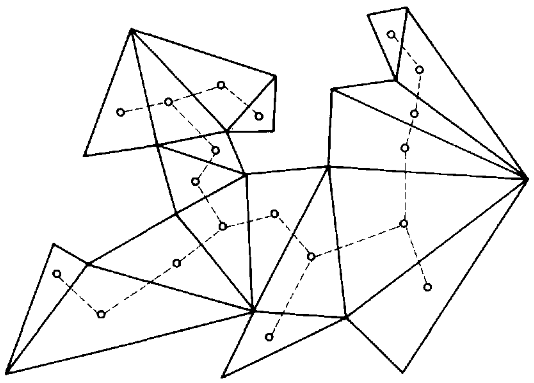

Fig. 1.14. The dual of a polygon triangulation is a tree.

fact that a triangle has 3 sides. Suppose the graph is not a tree. Then it must have a cycle. This cycle encloses some vertices of the polygon, and therefore it encloses points exterior to the polygon. This contradicts the definition of a polygon. □

Nodes of degree 1 are the leaves of the tree, nodes of degree 2 are parts of a path, and nodes of degree 3 are the binary branch points of a tree; see Fig. 1.14. We will see in Chapter 5 that Theorem 1.2 extends to polygons with holes (Lemma 5.1), but Lemma 1.3 does not.

The technical term for the dual used in the above lemma is "weak dual," weak because no node is assigned to the exterior face—that is, the exterior of the polygon. Throughout this book we will use weak duals but call them duals.

Lemma 1.3 yields an easy proof of the "two ears theorem" of Meister (1975). Three consecutive vertices v_1, v_2, v_3 form an *ear* of a polygon P at v_2 if the segment v_1v_3 is completely interior to P. Two ears are *non-overlapping* if the triangle interiors are disjoint.

THEOREM 1.3 [Meister's Two Ears Theorem 1975]. Every polygon of $n \geq 4$ vertices has at least two non-overlapping ears.

Proof. Leaves in the dual of a triangulation correspond to ears, and every tree of two or more nodes must have at least two leaves. □

This theorem in turn leads to a straightforward proof of the 3-colorability of a polygon triangulation graph by induction: cut off an ear triangle from the graph, 3-color the remainder by induction, and put back the removed triangle, coloring its degree 2 tip vertex the color not used on the cut diagonal.

Finally, we should note that in general a polygon has several distinct triangulations; only in special cases is the triangulation unique.

1.3.2. Algorithms

As is often the case, the proof of the existence theorem for triangulations leads directly to an algorithm for constructing one; and, as is again often the case, the algorithm is rather slow. Consider a naive implementation of the proof of Theorem 1.2. Determining whether a given diagonal is interior to the polygon requires $O(n)$ time. The chosen diagonal may partition the polygon into a small and a large piece; in the worst case the smaller piece could be a single triangle. Assuming the worst case at each step, complete triangulation requires

$$\sum_{k=n}^{1} O(k) = O(n^2).$$

Obtaining an optimal algorithm for triangulation is perhaps the outstanding open problem in computational geometry. To 1986, the best algorithms required $O(n \log n)$ time. The number and variety of these algorithms attest to the effort researchers expended on the problem. As this book was being revised, Tarjan and Van Wyk announced a breakthrough: an $O(n \log \log n)$ algorithm. Whether a linear-time algorithm is possible still remains open at this writing. In this section we will present several $O(n \log n)$ triangulation algorithms before sketching the latest algorithm.

The first $O(n \log n)$ algorithm developed proceeds in two stages: it first partitions the polygon into monotone pieces, and then triangulates each monotone piece individually. Thus we must first discuss monotone polygons and partitions, important topics in their own right.

Monotone Polygons

The concept of a *monotone* polygon was introduced in Lee and Preparata (1977) and has since proved to be a very fertile idea; it will be used at several critical junctures throughout this book. Let $p_1, \ldots, p_k$ be a polygonal path or a *chain*. A chain is called *monotone with respect to a line* L if the projections of $p_1, \ldots, p_k$ onto L are ordered the same as in the chain; that is, there is no "doubling back" in the projection as the chain is traversed. Two adjacent vertices p_i and p_{i+1} may project to the same point on L without destroying monotonicity. A chain is called *monotone* if it is monotone with respect to at least one line. We will use the convention that the line of monotonicity is the y-axis. A polygon is *monotone* if it can be partitioned into two chains monotone with respect to the same line. We will call them the *left* and *right* chains; see Fig. 1.15.

Lee and Preparata's monotone partitioning algorithm depends on an "obvious" characterization of monotone polygons, which, like so many such obvious statements, requires a careful proof. Define an *interior cusp* of a polygon as a reflex vertex v whose adjacent vertices either do not both have larger or do not both have smaller y-coordinates than v; picturesquely, interior cusps are stalacitites or stalagmites. The following is proved in (Garey *et al.* 1978).

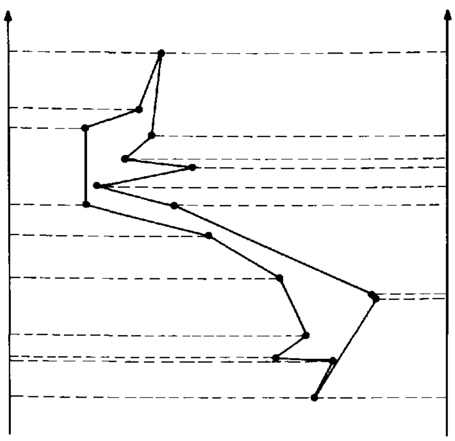

Fig. 1.15. The vertices of a monotone polygon project onto a line in a monotonically increasing sequence.

LEMMA 1.4 [Garey *et al.* 1978]. If a polygon P has no interior cusp, then it is monotone with respect to the y-axis.

Proof. We will prove the contrapositive. Assume therefore that P is not monotone with respect to the y-axis. Then at least one of its two chains, say the right one, is not monotone. Let the vertices of the right chain be $p_1, \ldots, p_k$ from top to bottom, and let p_i be the first vertex of this chain such that the y-coordinate of p_{i+1} is greater than that of p_i; p_i must exist since the chain is not monotone. If p_{i+1} is to the right of the line p_ip_{i-1}, then p_i is an interior cusp and we are finished. So assume that p_{i+1} is to the left of p_ip_{i-1}. Now connect p_i to p_k with line L as shown in Fig. 1.16. Let p_j be a vertex of largest y-coordinate in the chain from p_i to p_k before it crosses L. Then p_j is an interior cusp: it is reflex since it is a local maximum in the y direction with the polygon interior above it, and neither of its adjacent vertices can have larger y-coordinate. □

Lee and Preparata's algorithm removes all interior cusps by the addition of internal diagonals. It uses a general technique called "plane sweep"

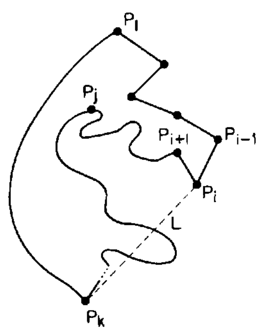

Fig. 1.16. If p_i is not an interior cusp, then p_j is.

(Nievergelt and Preparata 1982), which we will have occasion to use repeatedly in this book. The vertices are sorted by decreasing height and labeled $v_0, \ldots, v_n$, with v_0 highest. We will assume for simplicity that no two points have the same y-coordinate. A horizontal line L is now (conceptually) swept over the polygon from top to bottom. At all times a data structure is maintained reflecting the structure of the polygon in the vicinity of L. Each time a vertex is encountered, the data structure is updated, and perhaps some output is produced.

The data structure S is a list of [*edge*, *vertex*] pairs. Let line L intersect the interior of edges $e_0, \ldots, e_k$ in that order from left to right. Then S is the list

$$[e_0, w_0], [e_1, w_1], \ldots, [e_k, w_k],$$

where w_i is a vertex of minimum y-coordinate between e_i and e_{i+1}—that is, within the trapezoid bounded by L, e_i, e_{i+1}, and the line parallel to L through the lower of the upper endpoints of e_i and e_{i+1} (w_i could be this lower endpoint). An example is shown in Fig. 1.17. The point of this data structure is that, when L encounters an upward-pointing interior cusp such as v_j in Fig. 1.17, it must lie between two edge e_i and e_{i+1}, and v_j can then be connected with a diagonal to w_i.

The algorithm has the following overall structure:

Algorithm 1.2. Monotone Partitioning Algorithm

```
(1) Sort vertices by decreasing height: $v_0, \ldots, v_n$.
(2) {Descending pass}
    **for** $i = 1$ **to** $n$ **do**
        Remove upward-pointing interior cusps.
(3) {Ascending pass}
    **for** $i = n - 1$ **downto** 0 **do**
        Remove downward-pointing interior cusps.
```

We will only describe the descending pass, step (2). An artificial edge e_0 corresponding to the line $x = -\infty$ is used to bound the list S from the left. The algorithm is best described by a mixture of pseudo-code and pictures. The definitions of the symbols will be found in the corresponding figures.

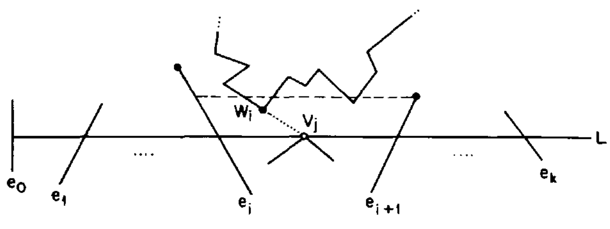

Fig. 1.17. When the sweep line L encounters v_j, the diagonal (v_j, w_i) is output.

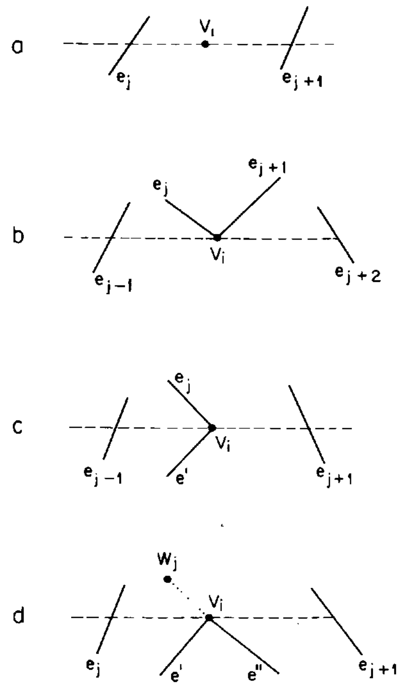

Fig. 1.18. Notation and sweep line events for monotone partitioning algorithm.

```
{Descending pass}
$S \leftarrow [e_0, v_0]$
**for** $i = 1$ **to** $n - 1$ **do**
**begin**
  $j \leftarrow$ smallest index such that $v_i$ is between $e_j$ and $e_{j+1}$, or on $e_{j+1}$ {See Fig.
  1.18a}
  {Let $I$ be the number of edges incident to $v_i$ from above.}
  {$S_{old} = \cdots [e_{j-1}, w_{j-1}], [e_j, w_j], [e_{j+1}, w_{j+1}], [e_{j+2}, w_{j+2}] \cdots$}
  **case** $I$ **of**
  $I = 2$: {Fig. 1.18b}
          $S_{new} \leftarrow \cdots [e_{j-1}, v_i], [e_{j+2}, w_{j+2}] \cdots$
  $I = 1$: {Fig. 1.18c}
          $S_{new} \leftarrow \cdots [e_{j-1}, v_i], [e', v_i], [e_{j+1}, w_{j+1}] \cdots$
  $I = 0$: {Fig. 1.18d}
          $S_{new} \leftarrow \cdots [e_j, v_i], [e', v_i], [e'', v_i], [e_{j+1}, w_{j+1}] \cdots$
          **if** $v_i$ is reflex **then** draw diagonal $(v_i, w_j)$
**end**
```

We will now run through a small example. Consider the polygon in Fig. 1.19. The vertices are labeled by integers in descending order, and the edges

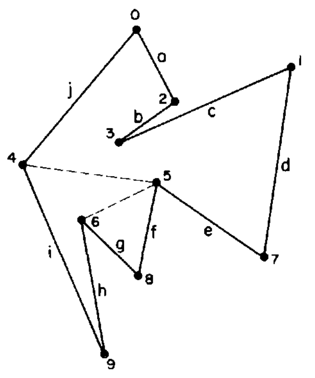

Fig. 1.19. Monotone partitioning example: diagonals (5, 4) and (6, 5) partition the polygon into monotone pieces.

are labeled with letters. Table 1.1 shows the values of the critical variables and the data structure S throughout the execution of the algorithm.

One can easily see that the algorithm is prepared to remove external cusps also, and it is only by checking whether v_i is reflex that we ensure that internal cusps are removed. Their algorithm was originally designed for planar point location, an application for which all cusps need to be removed.

We now turn to an analysis of the time complexity of the algorithm. The initial sorting step takes $O(n \log n)$ time. If the list S is implemented as a *dictionary*, say by a height-balanced tree (Knuth, 1973), then insertions and deletions can be performed in $O(\log n)$ time. As each vertex is processed only when it is passed by the sweeping line L, there are $O(n)$ such insertions and deletions, leading to an $O(n \log n)$ algorithm. The "trapezoidization"

Table 1.1

i	j	I	S	Output
0			$[-\infty, 0][j, 0][a, 0]$	
1	a	0	$[-\infty, 0][j, 0][a, 0][c, 1][d, 1]$	
2	j	1	$[-\infty, 0][j, 2][b, 2][c, 1][d, 1]$	
3	j	2	$[-\infty, 0][j, 3][d, 1]$	
4	$-\infty$	1	$[-\infty, 4][i, 4][d, 1]$	
5	i	0	$[-\infty, 4][i, 5][f, 5][e, 5][d, 1]$	$5 \rightarrow 4$
6	i	0	$[-\infty, 4][i, 6][h, 6][g, 6][f, 5][e, 5][d, 1]$	$6 \rightarrow 5$
7	e	2	$[-\infty, 4][i, 6][h, 6][g, 6][f, 7]$	
8	h	2	$[-\infty, 4][i, 6][h, 8]$	
9	$-\infty$	2	$[-\infty, 9]$	

algorithm in Tarjan and Van Wyk (1986), to be described shortly, improves this time complexity to $O(n \log \log n)$.

Finally we note that Preparata and Supowit have designed an algorithm to decide in linear time whether or not a given polygon is monotone with respect to any direction (Preparata and Supowit 1981).

Triangulation Algorithm of Garey, Johnson, Preparata, and Tarjan

The reason that monotone polygons have proven so useful is not due to their natural shape (they can be rather unnatural), but rather that often algorithms are much simpler if they are designed to work specifically on this restricted class. Triangulation is the best illustration of this: Garey *et al.* demonstrated that monotone polygons may be triangulated in *linear* time. Together with the $O(n \log n)$ algorithm for monotone partitioning just presented, this gives an $O(n \log n)$ algorithm for triangulating a polygon.

One might at first think that the dual of a triangulation of a monotone polygon must just be a simple path rather than the tree guaranteed by Lemma 1.3, but Fig. 1.20 shows that the structure can be quite complicated. Nevertheless, the situation is sufficiently constrained that Garey *et al.* were able to triangulate with a single stack algorithm.

Assume that the polygon P is monotone with respect to the y-axis. The first step of their algorithm is to sort the vertices in descending order by y-coordinate. Normally this would require $O(n \log n)$ time, but as the vertices on both the left and right chains of P are already sorted by y, the total sort can be obtained in linear time by a simple merge of the two sequences. Let $p_0, \ldots, p_n$ be the vertices in sorted order, with p_0 at the top. We will assume that no two vertices have the same y-coordinate to simplify the presentation.

The algorithm successively reduces P by chopping triangles off the top. At all times it maintains a stack of all the vertices examined so far but not yet completely processed. Let $v_0, \ldots, v_t$ be the vertices on the stack, with v_0 on the bottom and v_t on the top of the stack, and let P_i be the polygon remaining as step i commences. Then the following stack properties are

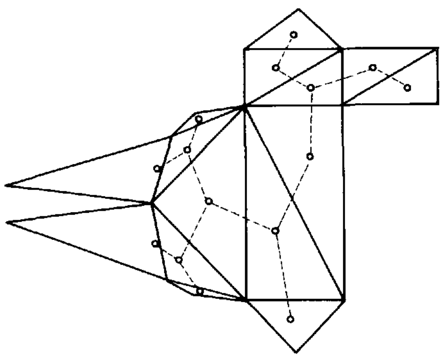

Fig. 1.20. The triangulation dual of a monotone polygon is not necessarily a path.

maintained throughout the processing:

(1) $v_0, \ldots, v_t$ decrease by height, v_t lowest.
(2) $v_0, \ldots, v_t$ form a chain of consecutive vertices on the boundary of P_i.
(3) $v_1, \ldots, v_{t-1}$ are reflex vertices.
(4) The next vertex p_i to be processed is adjacent via a polygon edge of P_i to either v_0 or v_t (or to both).

The algorithm connects diagonals from the next vertex to the vertices on the top of the stack, pops these off the stack, and pushes the just processed vertex onto the stack.

Algorithm 1.3. Triangulation of a Monotone Polygon

```
Sort vertices by decreasing $y$-coordinate, resulting in $p_0, \ldots, p_n$.
Push $p_0$.
Push $p_1$.
for $i = 2$ to $n - 1$ do
  if $p_i$ is adjacent to $v_0$ then {Fig. 1.21a}
    begin
      while $t > 0$ do
      begin
        Draw diagonal $p_i \rightarrow v_t$.
        Pop.
      end
      Pop.
      Push $v_t$.
      Push $p_i$.
    end
  else if $p_i$ is adjacent to $v_t$ then {Fig. 1.21b}
    begin
    while $t > 0$ and $v_t$ is not reflex do
    begin
      Draw diagonal $p_i \rightarrow v_{t-1}$.
      Pop.
    end
    Push $p_i$.
end
```

The stack contents as the algorithm processes the polygon shown in Fig. 1.22 are shown in Table 1.2. The algorithm ends when p_i is adjacent to both v_0 and v_t. Rather than insert special code to handle this case, we permit the redundancy of drawing one diagonal [(17, 16) in the above example] that is superfluous.

We now establish that each diagonal output by the algorithm lies entirely within the polygon. Consider first the diagonal (p_i, v_1) in Fig. 1.21a, which is drawn by the first **while** loop of the algorithm. It forms a triangle $T = (v_0, v_1, p_i)$. None of the vertices $v_2, \ldots, v_t$ can lie inside of T, since

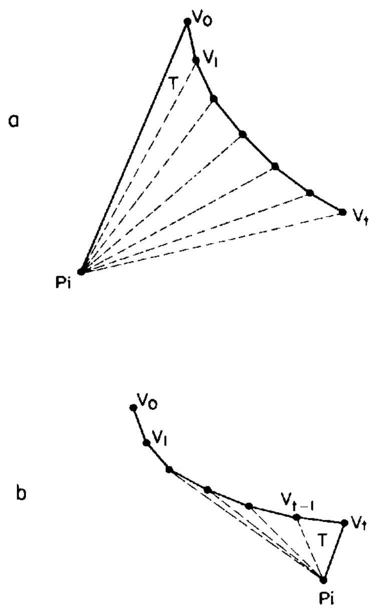

Fig. 1.21. Triangulation algorithm cases: p_i is adjacent to the stack bottom (a) or to the stack top (b).

the reflex angles at those vertices (stack property (3)) force them to lie on the opposite side of v_0v_1 as p_i. Every p_j with $j > i$ has smaller y-coordinate than p_i, so none of these can lie in T. We have thus established that T cannot contain any vertex of P_i. It still could happen that T is crossed by an edge of P_i without any vertices being interior. But this is not possible because v_0p_i and v_0v_1 are boundary edges of P_i. Therefore, (p_i, v_1) is an internal diagonal. Now the remaining diagonals output by the first **while** loop are internal by the same argument (or by induction).

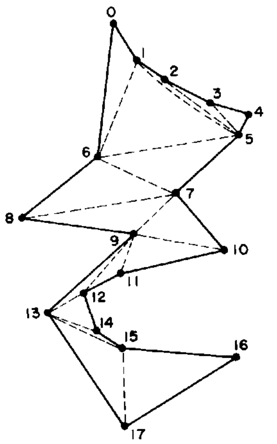

Fig. 1.22. Triangulation algorithm example.

Table 1.2

i	Stack (top→)	Diagonals Drawn
2	0 1	
3	0 1 2	
4	0 1 2 3	
5	0 1 2 3 4	(5, 3) (5, 2) (5, 1)
6	0 1 5	(6, 5) (6, 1)
7	5 6	(7, 6)
8	6 7	(8, 7)
9	7 8	(9, 7)
10	7 9	(10, 9)
11	9 10	(11, 9)
12	9 11	(12, 9)
13	9 12	(13, 12)
14	12 13	(14, 13)
15	13 14	(15, 13)
16	13 15	
17	13 15 16	(17, 16) (17, 15)

Consider second the diagonal (p_i, v_{t-1}) in Fig. 1.21b, drawn by the second **while** loop. Let T be the triangle (v_{t-1}, v_t, p_i). The vertices $v_0, \ldots, v_{t-2}$ are above T and p_j for $j > i$ are below. So no vertex of P_i is inside T. And again, no edge of P_i can cross T since $v_t v_{t-1}$ and $v_t p_i$ are boundary edges of P_i. Thus the diagonal (p_i, v_{t-1}) is internal. The remaining diagonals are internal by the same argument.

Finally, we argue that the four stack properties are maintained by the algorithm. Only p_i and v_t are pushed onto the stack, and when both are pushed they are pushed in the correct vertical order. Thus the vertices are in decreasing order by y-coordinate (1). The vertices form a chain (2) because either (a) the stack is reset to two adjacent vertices (Fig. 1.21a) or (b) by induction (Fig. 1.21b). The internal angles are reflex (3) because p_i is only pushed when v_t is reflex in the second **while**. And finally, p_i is either adjacent to v_0 or v_t (4) because the montonicity of P_i guarantees that p_i has a (unique) neighbor above it, and in the chain $v_0, \ldots, v_t$, only v_0 and v_t do not have all their neighbors accounted for.

Concerning time complexity, each vertex is pushed at most twice on the stack, once as p_i and once as v_t. Examination of the code shows that for each Push there is a corresponding Pop, and thus the algorithm requires $O(n)$ time. Together with the $O(n \log n)$ algorithm for partitioning a polygon into monotone pieces, which adds on $O(n)$ additional edges, this yields the claimed $O(n \log n)$ overall time complexity.

The algorithm has been presented as merely producing diagonals, without the adjacency information contained in the dual graph of the triangulation. It is not difficult, however, to modify the algorithm to produce the complete graph structure for each monotone piece, and then to stitch together the graphs from the pieces, without increasing the time complexity. With this graph structure available, Avis and Toussaint's divide-and-conquer coloring algorithm may be replaced by a straightforward linear recursive graph traversal.

Recent Triangulation Algorithms

In this section we review four recent triangulation algorithms. The algorithms will only be sketched and no proofs will be given; often the authors themselves have only published sketches of their algorithms. Our main point is to illustrate the variety of approaches available.

Plane Sweep Algorithm of Hertel and Mehlhorn. The algorithm presented in the previous two sections uses a plane sweep to partition into monotone pieces, then sweeps over each piece to triangulate it. It is natural to wonder if the triangulation cannot be done during the same sweep that performs the partitioning. Hertel and Mehlhorn showed that indeed a plane sweep algorithm can be constructed (Hertel and Mehlhorn 1983). Moreover, their algorithm is not a trivial merging of the algorithms of Lee and Preparata and of Garey *et al.*; for instance, Hertel and Mehlhorn's algorithm achieves a complete triangulation in a single forward pass, whereas the monotone partitioning algorithm requires a reverse pass as well.

The plane sweep algorithm runs in $O(n \log n)$ time: $O(n \log n)$ to sort the vertices for the sweep, and $O(n)$ instances of data structure updates, each costing $O(\log n)$, so no asymptotic advantage has been gained over the Garey *et al.* algorithm. What makes the Hertel and Mehlhorn approach noteworthy is that they can modify it to achieve $O(n + r \log r)$ time, where r is the number of reflex vertices of the polygon. Since r can be as large as $n-3$, this is no gain in the worst case, but it could be a significant gain in practice. Moreover, it was one of the first hints that *perhaps* better than $O(n \log n)$ might be achievable.

Two changes are made to achieve this new bound. First, the sweep line stops only at the r reflex vertices (and $O(r)$ other vertices that we will not specify here) rather than at all n vertices. Thus only $O(r)$ vertices need to be sorted. Second, the sweep line breaks into pieces, some of which may lag behind others. The data structure representing the state of the polygon "at" this now crooked sweep line is only of size $O(r)$, so that processing each of the $O(r)$ "event" vertices costs $O(\log r)$ each. Of course, $O(n)$ is still needed to output the $n-3$ diagonals. The result is an $O(n + r \log r)$ algorithm.

Chazelle's Polygon Cutting Theorem. We remarked earlier that a naive implementation of the proof of the triangulation theorem results in an inefficient triangulation algorithm. The next algorithm we will discuss is in some sense a sophisticated implementation of the same idea. But rather than depending on the triangulation theorem, it depends on Chazelle's Cutting Theorem. We will present a specialized version of his more general result (Chazelle 1982):

THEOREM 1.4 [Chazelle 1982]. After $O(n \log n)$ preprocessing, it is possible to find, in $O(n)$ time, a diagonal that divides the polygon into two pieces P_1 and P_2 that satisfy $|P_1| \leq |P_2| \leq (2/3)\,|P| + 2$ (where $|Q|$ indicates the number of vertices of Q).

We must have that $|P_1| + |P_2| = |P| + 2$. Solving this equation for $|P_2|$ and substituting into the inequality shows that the theorem implies that $(1/3)\,|P| \leq |P_1|$. Thus the cutting theorem says that a preprocessed polygon can be divided into nearly equal-sized pieces in linear time. This immediately leads to a recursive algorithm for triangulating a polygon: namely, find a cutting diagonal as guaranteed by the theorem, and recurse on the two pieces. If the polygon has fewer than seven vertices, stop the recursion (as the theorem may result in fewer than three vertices in P_1) and triangulate by some brute-force method. Because the search for a cutting diagonal is linear, we have the recurrence relation $T(n) \leq 2T(2n/3) + O(n)$ for the time complexity, whose solution is $O(n \log n)$.

Sinuosity Algorithm of Chazelle and Incerpi. The only supralinear step in the Garey *et al.* algorithm is partitioning into monotone pieces, which costs $O(n \log n)$. Chazelle and Incerpi have shown how the monotone partitioning can be improved to $O(n \log s)$, where s is the "sinuosity" of the polygon (defined below) (Chazelle and Incerpi 1983, 1984). The sinuosity may be $O(n)$, but it is "usually" very small. Their algorithm works by first finding a "trapezoidization" of the polygon, from which it is easy to derive a monotone partition. Indeed, Lee and Preparata's algorithm discussed in the previous section can be viewed as computing a trapezoidization.

The *trapezoidization* $Tr(P)$ of a polygon P is obtained by drawing a horizontal line through every vertex, extended to the point where it first crosses to the exterior of the polygon. Figure 1.23 shows an example. The horizontal lines partition the polygon into trapezoids, or triangles, which can be considered degenerate trapezoids. Each trapezoid T is "supported" on its top and bottom sides by a vertex of P. The vertices v of P that violate monotonicity in the y-direction are those that lie on the interior of a horizontal segment. Connecting each such v to the unique w that is the other support vertex for T partitions P into pieces monotone with respect to y. This is also illustrated in Fig. 1.23.

Chazelle and Incerpi compute the trapezoidization of a polygon by divide-and-conquer. To do this, they first note that a trapezoidization may be defined for any simple, oriented polygonal path, or a chain: it does not have to be a closed polygon. The horizontal partition lines are simply permitted to run to infinity if they meet no obstruction.

Given a polygon P defined by the vertices $p_1, \ldots, p_n$ in counterclockwise order, let P_1 be the chain $p_1, \ldots, p_{\lfloor n/2 \rfloor}$ and P_2 the chain $p_{\lfloor n/2 \rfloor + 1}, \ldots, p_n$.

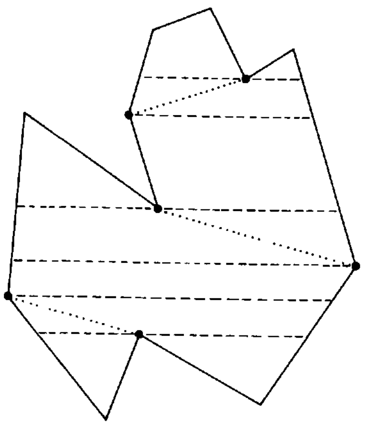

Fig. 1.23. A trapezoidization (dashed lines) leads to a monotone partitioning with the addition of diagonals to "internal" vertices (dotted).

The divide-and-conquer algorithm recursively computes $Tr(P_1)$ and $Tr(P_2)$, and then merges these two into $Tr(P)$. Obviously all the cleverness is embodied in the merge procedure.

Consider the example of Fig. 1.24. Starting from $v_1 = w_1$, the merge processing walks along the chains $v_1, \ldots, v_m$ and $w_1, \ldots, w_m$ simultaneously, stitching together the trapezoids to obtain the trapezoidization for their union. The process has many similarities to merging two sorted lists, but it is of course much more complicated. We will skip the details, and just note one important point: it is possible for the processing to take "short-cuts." For example, one can jump from v_a to v_b on P_1 without examining any of the vertices in between, as P_2 never crosses the $v_a v_b$ line.

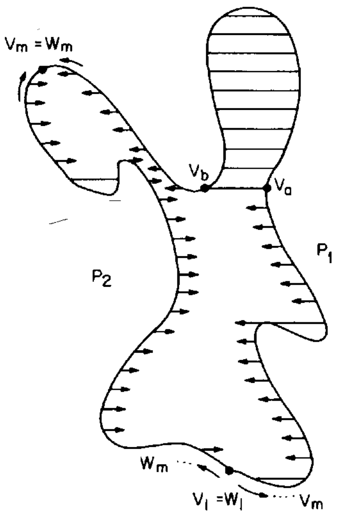

Fig. 1.24. Stitching together trapezoids from separate chains to form a trapezoidization.

We now define the *sinuosity* of a polygonal path $p_1, \ldots, p_k$. Assume for simplicity that no two adjacent vertices have the same y-coordinate. As i moves from 1 to $k-1$, the ray R through $p_i p_{i+1}$ may pass the horizontal (positive x-axis) either counterclockwise (ccw) or clockwise (cw). The path is called *spiraling* if R never passes the horizontal cw twice in a row, and *antispiraling* if it never passes ccw twice in a row. Here by "twice in a row" we mean two successive horizontal crossings, independent of the number of chain vertices between these crossings. Thus a spiraling path winds ccw, with perhaps some cw movements of less than 360°, and an antispiraling path winds cw. It is easy to partition a simple polygon into maximal spiraling and antispiraling chains in linear time. The number of chains is somewhat (±1) dependent on the starting position. The maximum number of chains over all starting positions for a polygon P is defined as the *sinuosity* s of P. For example, the polygon in Fig. 1.25 has $s = 1$: it is a spiral.

Chazelle and Incerpi have established that (a) the horizontal decomposition of any spiraling or antispiraling chain can be computed in linear time using shortcuts, and (b) that this leads to an $O(n \log s)$ algorithm for triangulating a simple polygon of sinuosity s. This result lent further credence to the long-standing conjecture that $O(n \log n)$ is not the lower bound on triangulation.

Triangulation Algorithm of Tarjan and Van Wyk. The conjecture just mentioned was finally settled by Tarjan and Van Wyk, who found an $O(n \log \log n)$ algorithm for triangulation (Tarjan and Van Wyk 1986). As one might suspect from a problem so resistant to solution, their algorithm is rather complex. It would take us very far afield into current data structure theory to explain the algorithm in detail, so we will only sketch it at a high level.

They start with the same observation used by Chazelle and Incerpi (and made independently in (Fournier and Montuno 1984)): triangulation is linear-time reducible to trapezoidization—that is, a triangulation may be

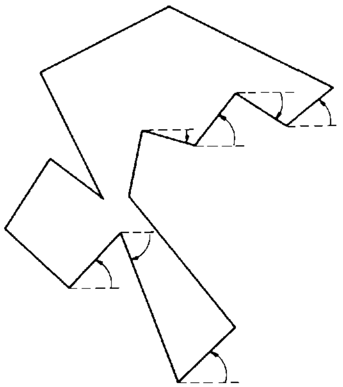

Fig. 1.25. A polygon with sinuosity 1: there are no two successive clockwise transitions across the horizontal.

constructed from a trapezoidization in linear time. Again similar to (Chazelle and Incerpi 1983), Tarjan and Van Wyk construct the trapezoidization by divide-and-conquer. But they divide the polygon, not chains. At any stage of the algorithm, a set S of subpolygons of P are maintained. A polygon P' is removed from S, and a vertex v_{cut} of P' is selected. A horizontal line L is drawn through v_{cut}, and P' is partitioned into pieces that lie above and below L. This is a complicated step, and requires a novel use of "finger search trees" (Brown and Tarjan 1980). The points at which P' crosses L are found in the order in which they occur in a traversal of the boundary of P', which is (in general) not the same as their left-to-right sorted order along L. The intersection points can, however, be sorted in linear time. This is another complicated step, and one of the keys to the algorithm's efficiency. The linear sorting depends on the points forming a "Jordan sequence" (Hoffman *et al.* 1985). After splitting and sorting, all those pieces that are triangles or trapezoids are output; those that are neither are added to S, and the process repeats.

Although it is unclear at this writing if this algorithm is of practical utility, its theoretical impact is felt throughout computational geometry, since so many algorithms depend on triangulation. Even improving on $O(n \log\log n)$ would be a major theoretical advance. The fundamental question of whether a linear-time triangulation algorithm is achievable remains open at this writing.

1.4. CONVEX PARTITIONING

We saw in the preceding sections algorithms whose performance was measured as a function of a variable (r and s) other than n, the number of vertices of the polygon. This suggests asking Klee's original art gallery question, but requesting the answer as a function of something besides n. As a convex n-gon only needs 1 guard, not $\lfloor n/3 \rfloor$, it makes sense to use a variable that is a more accurate measure of the "shape" of the polygon. In this section we investigate the art gallery question as a function of r, the number of reflex vertices of the polygon.

We first note that r can be as large as $n-3$; see Fig. 1.26. This figure

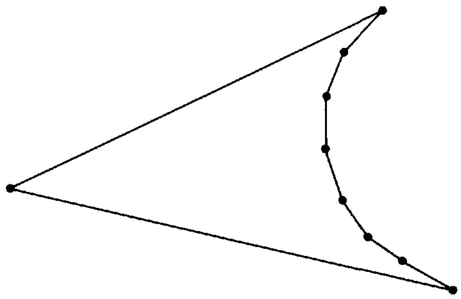
Fig. 1.26. Of a polygon's n vertices, as many as $n-3$ may be reflex.

shows that r no more captures the "shape" of the polygon than n does, since only one guard is needed for this polygon regardless of the size of r. Nevertheless, the pursuit of this issue will draw us into the important topic of convex partitioning.

1.4.1. Theorems

Necessity

Superficially it appears that perhaps no more than roughly $r/2$ guards are ever necessary to see the interior of a polygon of r reflex vertices, but the "shutter" examples in Fig. 1.27 demonstrate that in fact r guards are sometimes necessary.

Sufficiency

Intuition suggests that placing a guard at each reflex vertex suffices to cover any polygon with $r > 1$ reflex vertices. That this is indeed the case can be established by Chazelle's "naive" convex partitioning (Chazelle 1980).

LEMMA 1.5 [Chazelle 1980]. Any polygon can be partitioned into at most $r + 1$ convex pieces.

Proof. The proof is by induction. The lemma is clearly true when $r = 0$. In the general case, draw a ray from a reflex vertex bisecting the internal angle up to its first intersection with the polygon's boundary. This ray divides the polygon into two polygons with r_1 and r_2 reflex vertices, respectively. $r_1 + r_2 \leq r - 1$, since the ray resolved at least one reflex vertex (it may have resolved another at its point of contact with the boundary). Applying the induction hypothesis yields $r_1 + 1 + r_2 + 1 \leq r + 1$ convex pieces. □

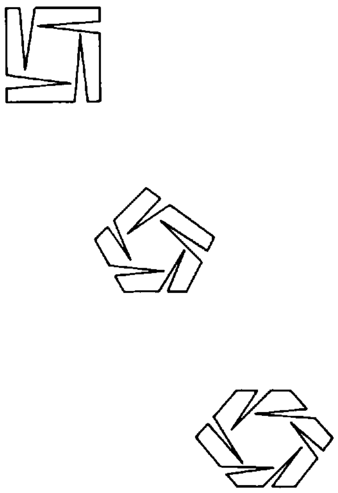

Fig. 1.27. "Shutter" shapes show that r guards can be necessary.

THEOREM 1.5 [O'Rourke 1982]. r guards are occasionally necessary and always sufficient to see the interior of a simple n-gon of $r \geq 1$ reflex vertices.

Proof. Necessity has already been established. For sufficiency, apply Chazelle's naive convex partition lemma. Each convex piece must have at least one reflex vertex on its boundary. Thus guards placed on every reflex vertex see into each convex piece. □

We now turn to a discussion of algorithms for finding convex partitionings.

1.4.2. Algorithms for Convex Partitioning

It is rather easy to compute the naive convex partition in $O(rn) = O(n^2)$ time as follows (Chazelle 1980). For each reflex vertex, intersect every edge of the polygon with the bisection of the reflex angle. Connect the reflex vertex to the closest intersection point. Chazelle shows how this speed can be improved to $O(n + r^2 \log(n/r))$ time, and I believe a plane-sweep algorithm can achieve $O(n \log n)$, but we will not present the details.

Because at most two reflex vertices can be resolved by a single cut, the minimum number of convex pieces into which a polygon may be partitioned is $\lceil r/2 \rceil + 1$. Thus, if an *optimal* partitioning results in OPT pieces, $OPT \geq \lceil r/2 \rceil + 1$. The naive partition achieves no more than $r + 1 \leq 2OPT$ pieces in $O(n^2)$ time. We will discuss two more algorithms, one faster but with a poorer performance ratio, and one slower but optimal.

The first results from an observation of Hertel and Mehlhorn (1983).

THEOREM 1.6 [Hertel and Mehlhorn 1983]. Any triangulation of a polygon can be converted into a convex partitioning of no more than $2r + 1$ pieces by removing diagonals.

Proof. Let d be an internal diagonal of the triangulation incident with a vertex v. Call d *essential* for v if its removal would result in a non-convex interior angle at v. Then a reflex vertex cannot have more than two essential diagonals incident to it: an angle smaller than 360° cannot be partitioned into more than three intervals such that adjacent intervals span more than 180°. If each reflex vertex does have exactly two essential diagonals, and no two reflex vertices share essential diagonals, then $2r$ of the triangulation diagonals cannot be removed, resulting in a partition into $2r + 1$ convex pieces. □

Note that $2r + 1 \leq 4OPT$. Although the performance ratio is lower, the algorithm implied by the theorem runs in $O(n \log \log n)$ time: $O(n \log \log n)$ for triangulation, and linear time for removal of inessential diagonals.

Finally, we briefly mention Chazelle's remarkable optimal algorithm (Chazelle 1980; Chazelle and Dobkin 1985). Construction of an optimal convex partition requires the introduction of "Steiner points": points that

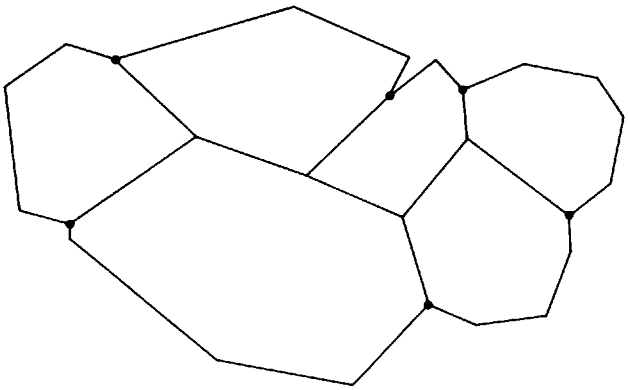

Fig. 1.28. An optimal convex partition may require interactions between the cuts resolving several reflex vertices.

are not vertices of the original polygon.[1] Such points were introduced by the naive partitioning, but in a very controlled manner. The situation for optimal partitions is more complicated, as illustrated in Fig. 1.28. This complexity leads one to believe that perhaps the problem is NP-hard, and indeed, we will see in Chapter 9 that many minimal partition problems are NP-hard. Nevertheless, Chazelle was able to obtain an $O(n^3)$ *optimal* algorithm using dynamic programming, and much else besides. His description fills 97 pages (Chazelle 1980), and we will make no attempt to summarize it here.

Convex partitions will be revisited for three-dimensional polyhedra in Chapter 10.

1. Convex partitions without Steiner points are discussed in Greene (1983).