Please scan and finish this article for Article Express located at Fine Library (SM). Thank you.

ILLiad Transaction Number: 235563
This request is for: Robert Tarjan (ret)
Delivery Method: Mail to Address
Electronic Delivery: Yes

Call Number: QA448.D38 D57 1991
Branch Location: Fine Library (SM)

Journal Title: Discrete and computational geometry : papers from the DIMACS special year
Article Author: Bernard Chazelle Article Title: Triangulating a Simple Polygon in Linear Time

Journal Vol: 6 Journal Issue: 
Journal Month: Journal Year: 1991
Article Pages: pp. 485--524

This request has been sent by #S - Borrowing.
ILLiad Transaction Number: 235563

Thank you,
Shauna Shaw
Engineering Library
Triangulating a Simple Polygon in Linear Time*

Bernard Chazelle
Department of Computer Science, Princeton University,
Princeton, NJ 08544, USA

Abstract. We give a deterministic algorithm for triangulating a simple polygon in linear time. The basic strategy is to build a coarse approximation of a triangulation in a bottom-up phase and then use the information computed along the way to refine the triangulation in a top-down phase. The main tools used are the polygon-cutting theorem, which provides us with a balancing scheme, and the planar separator theorem, whose role is essential in the discovery of new diagonals. Only temporary data structures are required by the algorithm. In particular, no dynamic search trees, finger trees, or point-location structures are needed. We mention a few applications of our algorithm.

1. Introduction

Triangulating a simple polygon has been one of the most outstanding open problems in two-dimensional computational geometry. It is a basic primitive in computer graphics and, generally, seems the natural preprocessing step for most nontrivial operations on simple polygons [5], [14]. Recall that to triangulate a polygon is to subdivide it into triangles without adding any new vertices. Despite its apparent simplicity, however, the triangulation problem has remained elusive. In 1978 Garey et al. [12] gave an O(n log n)-time algorithm for triangulating a simple n-gon. While it was widely believed that triangulating should be easier than sorting, no proof was to be found until 1986, when Tarjan and Van Wyk [27] discovered an O(n log log n)-time algorithm. Following this breakthrough, Clarkson et al. [7] discovered a Las Vegas algorithm, recently simplified by Seidel [25], with O(n log^2 n) expected time. In 1989 Kirkpatrick et al. [30] gave a new,

* The author wishes to acknowledge the National Science Foundation for supporting this research in part under Grant CCR-8700817.
conceptually simpler $O(n \log \log n)$-time algorithm, and they also derived an $O(n \log^2 n)$ bound for the case where vertices have polynomially bounded integer coordinates. Other results on the triangulation problem include linear or quasilinear algorithms for restricted classes of polygons [6], [11], [17], [28], [30].

Our main result is a linear-time deterministic algorithm for triangulating a simple polygon. The algorithm is elementary in that it does not require the use of any complicated data structure; in particular, it does not need dynamic search trees, finger trees, or any fancy point-location structures.

What makes fast polygon triangulation a difficult problem are the basic inadequacies of either a pure top-down or a pure bottom-up approach. To proceed top-down is to look at the whole polygon and infer global information right away. We can rely on the polygon-cutting theorem [4] which says that the polygon can be cut along a diagonal into two roughly equal-size pieces. The immediate dilemma is that to find such a diagonal appears just as difficult as triangulating the whole polygon to begin with. Besides, we would actually need to find such a diagonal in sublinear amortized time (say, bounded by $O(n/\log^2 n)$) to keep our hopes for an optimal triangulation algorithm alive. A bottom-up approach, on the other hand, involves computing, say, triangulations of subpieces of the polygon's boundary; this suffers from the obvious flaw that too much information gets to be computed. Indeed, diagonals for small pieces of the boundary are not guaranteed to be diagonals of the whole polygon and might therefore be wasted. Our solution is to mix bottom-up and top-down approaches together. The basic strategy is to build a coarse approximation of a triangulation in a bottom-up phase and then use the information computed along the way to refine the triangulation in a top-down phase. The main tools we use are

(i) the polygon-cutting theorem, which provides us with a balanced scheme, and

(ii) the planar separator theorem [21], whose role is essential in the discovery of new diagonals.

Here is a more detailed overview of the algorithm. As was observed in [6] and [11] a triangulation of a polygon can be derived in linear time from its horizontal visibility map, sometimes referred to in the literature as trapezoidal decomposition: this is the partition of the polygon obtained by drawing horizontal chords from the vertices. We can extend this notion easily and speak, more generally, of the visibility map of any simple polygonal curve (Fig. 2.1). Chazelle and Incerpi [6] showed how to build the visibility map of an $n$-vertex curve in $O(n \log n)$ time, using divide-and-conquer. Their algorithm mimics mergersort: assuming that $n$ is a power of 2, at the $k$th stage ($k = 1, 2, \ldots, \log n$), the boundary of the polygon is decomposed into chains of size $2^k$, whose visibility maps are computed by piecing together the maps obtained at the previous stage. Each stage can be accomplished in a linear number of operations, so computing the visibility map of the polygon takes $O(n \log n)$ time.

The new algorithm follows the same pattern: it goes through an up-phase of $\log n$ stages, each involving the merging of maps obtained at the previous stage. The novelty we bring into this process is to use only coarse samples of the visibility maps during the merge. In this way we can carry out an $O(n \log n)$-time and beat the $n \log n$ barrier. The samples are uniform maps; uniform in the sense that they approximate the visibility map equally well. Of course, in the end, we also need an efficient submap derived for the whole polygon into its full-fledged triangulation. This is done, it takes only linear time to compute a triangulation of the whole polygon, we go down through stages in reverse (a down phase), redefines the submap incrementally, until we get back to the initial point where the full visibility map emerges at last. Figure 1.1 illus-

The difference is rooted in a notion which we call a 2-division of the single most important concept in our algorithm, fe-

Fig. 1.1

Up phase

Down phase
On log log n-time algorithm, and they also derived an algorithm where vertices have polynomially bounded integer values. The triangulation algorithm is elementary in that it does not require the use of a linear-time deterministic algorithm for triangulating a polygon. The algorithm is based on a bottom-up approach. To proceed, the whole polygon is divided into pieces, each of which is a triangle. The immediate subproblems to such a division appear as difficult as triangulating a polygon with n vertices.

On log log n-time algorithm, and they also derived an algorithm where vertices have polynomially bounded integer values. The triangulation algorithm is elementary in that it does not require the use of a linear-time deterministic algorithm for triangulating a polygon. The algorithm is based on a bottom-up approach. To proceed, the whole polygon is divided into pieces, each of which is a triangle. The immediate subproblems to such a division appear as difficult as triangulating a polygon with n vertices.

...
mergesort and triangulation part ways. Recall that the polygon-cutting theorem is a geometric analog of the centroid theorem for free trees: a visibility map has a tree structure and, so, can be written as a collection of "blobs" of roughly equal size, themselves interconnected in a tree pattern. These blobs are the constituents of a submap. Merging two submaps can thus be equated with "merging" two trees together. The mergesort equivalent of a submap would be a sublist (of one of the lists to be merged) obtained by picking keys at regular intervals. Notice that merging two such sublists might in the worst case produce a new sublist whose corresponding intervals are up to twice the size of the original intervals. This coarsening effect prevents us from speeding up mergesort, because repairing the damage might involve computing medians or things of that nature for which no shortcuts can be found. To be sure, as we shall see, equally bad things can happen with submaps; repairing the damage, however, can be done by simply adding new chords to submaps, which can be made to take only sublinear time. To make this possible we must keep the coarseness of submaps under control by requiring that the tree structure of a submap be of bounded degree: in our terminology "conformal" actually means degree at most 4. Restoring conformality after merging two submaps is the linchpin of the algorithm and, as we should expect, its most delicate and subtle part as well: it can be viewed as a geometric "two-dimensional" analog of rotations in balanced dynamic search trees.

Although our algorithm is an outgrowth of the mergesort-like method of Chazelle and Incerpi, it goes far beyond it. For example, the algorithm never actually merges visibility maps but only submaps (which is done quite differently). We must also borrow ideas from a number of other sources: as we mentioned before, one of them is the polygon-cutting theorem, and, more specifically, the hierarchical polygon decomposition of Chazelle and Guibas [5]. There exist optimal algorithms for computing such decompositions [3, 14] but, as it turns out, standard suboptimal methods work just as well for our purposes. Merging submaps requires a primitive to detect new chords. In the Chazelle-Incerpi method the detection is limited to constant-size domains, so it can be done naively. But the domains can be arbitrarily large so we need a sublinear ray-shooting method. This cannot be done by using fast planar point location, which is the approach followed in the algorithm of Kirkpatrick et al. [20]. The reason is that we need to support merging of two point-location structures, and the known methods, even the dynamic ones, are inadequate in that regard. We turn the problem around by using a weak form of divide-and-conquer based on Lipton and Tarjan's planar separator theorem [21].

2. Visibility Maps and Submaps

Following the tradition of visibility algorithms, we begin by restricting their applicability to polygons where no two distinct vertices have the same y-coordinate. Of course, the standard excuse still works: we can easily get around this assumption by applying the symbolic perturbation techniques of [10] and [31]. Turning now to visibility maps, recall that, in the Chazelle-Incerpi method, chords are extended only toward the interior of the polygon, to determine how far they should extend. Kirkpatrick et al. scheme of extending chords on both sides. This amounts to treating the curve as a very thin polygon embedded in a cylindrical plane around infinity. For the reasons we give below, we find it more natural to treat our objects in a topological manifold, called the s-sphere, equivalent to a 2-sphere. Proofs of correctness in the polygon case are trickier and riddled with painful case-analyses. On the other hand, proofs of correctness in this case are often simple, proofs become much simpler, prove the ambient space has the "right" topology.

What is the right topology in this context? One problem plane is that although it has a Jordan curve theorem, the removing a simple closed curve may now have one of three possibilities:

(1) an open disk,
(2) a cylinder $S^1 \times (0, +\infty)$, or
(3) a perforated cylinder.

We simplify all that by defining our ambient space to be $\mathbb{R}^3 \setminus \bar{\gamma}$, where $\gamma$ is the product space $[−\infty, +\infty]^2$ with the following identification:

(i) $(−x, y) = (x, y)$,
(ii) $(x, −\infty) = (x, \infty)$,
(iii) $(x, +\infty) = (x, −\infty)$,

for all $x, y \in [−\infty, +\infty]$. The spherical plane is homeomorphic to the n-sphere, we now have the nicest Jordan curve theorem of all: the closed curve creates two open disks.

The only remaining difficulty is the orientability of Jordan curves. The orientability of the spherical plane, the boundary of any rectangle in a disk, being of codimension 1, has an induced orientation. The reverse orientation is called counterclockwise. Conversely, a clockwise tour of the curve refers to the orientation induced by a given disk. Unless one of these two possibilities is chosen unambiguously by fixing a reference on the plane and looking at the unique disk that does not contain the reference, it means restricting ourselves to curves that avoid that point, from the reference point of view. From now on, we assume that the reference point is chosen so that the clockwise traversal of a Jordan curve in an enclosed region, makes full sense.

2.1. The Visibility Map

Given a simple (nonclosed) polygonal curve $C$ with vertices $V(C)$, the visibility map of $C$, denoted $V(C)$, is the planar subdivision...
are extended only toward the interior of the polygon, with a special rule to determine how far they should extend. Kirkpatrick et al. [20] use the simpler scheme of extending chords on both sides. This amounts to thinking of a polygonal curve as a very thin polygon embedded in a cylindrical plane that lets chords wrap around infinity. For the reasons we give below, we find it more convenient to embed our objects in a topological manifold, called the spherical plane, which is equivalent to a 2-sphere. Proofs of correctness in the polygonal merging business tend to be tricky and riddled with painful case-analyses. One reason for this is that these proofs often attempt to establish topological facts by geometric means, thus adding unnecessary complications. By sticking to topological considerations as much as possible, proofs become much simpler, provided, of course, that the ambient space has the “right” topology.

What is the right topology in this context? One problem with the cylindrical plane is that although it has a Jordan curve theorem, the two regions created by removing a simple closed curve may now have one of three types:

1. an open disk,
2. a cylinder $S^1 \times (0, +\infty)$, or
3. a perforated cylinder.

We simplify all that by defining our ambient space to be the spherical plane: this is the product space $[-\infty, +\infty]^2$ with the following identification rules:

\begin{align*}
& (i) \quad (-\infty, y) = (+\infty, y), \\
& (ii) \quad (x, -\infty) = (-x, -\infty), \\
& (iii) \quad (x, +\infty) = (-x, +\infty),
\end{align*}

for all $x, y \in [-\infty, +\infty]$. The spherical plane is homeomorphic to a 2-sphere, so we now have the most Jordan curve theorem of all: the removal of any simple closed curve creates two open disks.

The only remaining difficulty is the orientability of Jordan curves. If we fix an orientation of the spherical plane, the boundary of any region homeomorphic to a disk, being of codimension 1, has an induced orientation, called clockwise; the reverse orientation is called counterclockwise. Conversely, given a Jordan curve, a clockwise tour of the curve refers to the orientation induced by one of the two homeomorphic disks that it bounds. Unless one of these disks is already understood, we choose one unambiguously by fixing a reference point somewhere in the plane and looking at the unique disk that does not contain it. Of course, this means restricting ourselves to curves that avoid that point, but this is only a minor inconvenience. From now on, we assume that the reference point is at $(0, +\infty)$.

In this way, to speak of the clockwise traversal of a Jordan curve, without reference to an enclosed region, makes full sense.

2.1. The Visibility Map

Given a simple (nonclosed) polygonal curve $C$ with vertices $v_0, \ldots, v_n$, we define the visibility map of $C$, denoted $V(C)$, as the planar subdivision formed by extending
two horizontal segments from each vertex \( v_i \), one in each direction. Each segment, which we call a chord, is extended until it meets another point of \( C \). If it were to go to infinity in the Cartesian plane, then it would actually wrap around in the spherical plane until it hits \( C \) again. Adding chords to \( C \) subdivides the spherical plane into regions: triangles or trapezoids with two horizontal segments and two nonhorizontal segments, all of which are possibly divided into several collinear edges (Fig. 2.1).

In order to distinguish between the two sides of an edge, we give each edge of \( C \) an infinitesimal width so as to make the curve \( C \) into a very thin simple polygon. (This is just a conceptual device, so, in particular, no actual perturbation of the polygon needs to be performed on the computer.) The boundary of that polygon is called the double boundary of \( C \) and is denoted \( \partial C \). By abuse of terminology we refer to the two sides of the double boundary as one would speak of the left and right sides of a snake; of course, this is meaningful geometrically but not topologically. Each vertex of \( C \) that is not a local extremum in the \( y \)-direction gives rise to two companion vertices in \( 2C \), one on each side of the curve (Fig. 2.2.1). In this way, each vertex is incident upon exactly one chord (possibly the same chord for the two companions). But what about local extrema? For each such vertex of \( C \), if it is not one of the two endpoints, we create a total of four vertices in \( 2C \) (Fig. 2.2.2): two companion pairs of duplicate vertices; one pair on each side of \( \partial C \). The duplicate vertices in a given pair are next to each other along \( \partial C \). By convention, we say that one of these pairs, the one on the "inside" of the turn, gives rise to a chord of null length. Finally, as shown in Fig. 2.2.3, for each endpoint of \( C \) we create two companion vertices; these can also be called duplicates since they are next to each other as well as on both sides of \( \partial C \). Figure 2.3 illustrates these definitions. Note that for simplicity we have not numbered vertices connected to null-length chords, but we have numbered all chord endpoints for later use. We have now ensured that each vertex of \( \partial C \) is incident upon exactly one chord of the visibility map. Therefore, any vertex—and actually also any point—of \( \partial C \) has a unique horizontal "chord direction" (left or right) assigned to it. Roughly speaking, this direction points to the left of an observer with respect to \( \partial C \).

If a chord of \( V(C) \) has \( p \) and \( q \) as endpoints, we say that \( p \) is with respect to \( C \), or simply, see each other, when \( C \) is unoriented, we say that \( p \) and \( q \) are mutually visible. More generally, (not necessarily vertices) of \( \partial C \) with the same \( y \)-coordinates see each other with respect to \( C \) if one of the two (relative) \( p \) and \( q \) lies completely outside of \( C \) (regarded as a convex extension the segment \( pq \) is also called a chord. For example, labeled 6 and 7 see each other, whereas points 6 and 16 and points 1 and 16 do not see each other. We close our discussion with a simple but useful fact.
from each vertex $v_i$, one in each direction. Each segment, extended until it meets another point of $C$. If it were to return to the vertex where it started, it would actually wrap around the curve $C$ again. Adding chords to $C$ subdivides the spherical polygon into triangles or trapezoids with two horizontal segments and two vertical segments, all of which are possibly divided into several collinear parts. Each part between the two sides of an edge, we give each edge of $C$ the property of being incident upon exactly one chord (possibly the edge itself). But what about local extrema? For each vertex, not one of the endpoints, we create a total of four vertices, two companion pairs of duplicate vertices; one pair on each side of the edge of null length. Finally, as shown in Fig. 2.2.3, for each vertex, the points of incidence of duplicate vertices; these can also be called duplicates of the same type as well as on both sides of $C$. Figure 2.3 illustrates the idea of the visibility relationship between a vertex and its neighbors. For simplicity we have not numbered the vertices connected to a given vertex. We have numbered all chord endpoints for later use.

If a chord of $V(C)$ has $p$ and $q$ as endpoints, we say that $p$ and $q$ see each other with respect to $C$, or simply, see each other, when $C$ is understood. Equivalently, we say that $p$ and $q$ are mutually visible. More generally, if $p$ and $q$ are two points (not necessarily vertices) of $C$ with the same $y$-coordinates, we say that $p$ and $q$ see each other with respect to $C$ if one of the two (relatively open) segments joining $p$ and $q$ lies completely outside of $C$ (regarded as a thin simple polygon). By extension the segment $pq$ is also called a chord. For example, in Fig. 2.3 points 6 and 7 see each other, whereas points 6 and 14 as well as points 3 and 4 and points 1 and 1' do not see each other. We close our discussion of visibility with a simple but useful fact.
Lemma 2.1. If we remove a pair of mutually visible points from the double boundary of a simple polygonal curve, then no chord can connect the two resulting pieces.

Proof. Let $C_1$ and $C_2$ be the two pieces of the double boundary resulting from the removal of a pair of visible points. Together with $C_1$ and $C_2$, the chord $c$ connecting the two points subdivides the spherical plane into three polygonal regions (i.e., regions bounded by simple polygonal curves), one of which is the infinitely small thin polygon $C$ itself. Any chord connecting $C$ and $C_i$ lies outside the “polygon” $C$, so it must cross $c$. But this is impossible because chords are horizontal.

The circular sequence of chord endpoints in $V(C)$ encountered during a clockwise traversal of $SC$ is called the canonical vertex enumeration of $V(C)$; note that it contains other points besides the vertices of $SC$. Figure 2.3 provides the sequence $1, 1', \ldots, 17, 18$, with the primes indicating duplicated vertices. Recall that we have refrained from numbering the endpoints of null-length chords. Speaking of which, note that null-length chords create empty regions. The traversal of the double boundary leads to a canonical enumeration of the regions (with repetitions). In the case of Fig. 2.3 we have the list (including only the nonempty regions for simplicity)


It is easy to see that the dual graph of the subdivision is a tree, naturally called the visibility tree of $C$. This graph is defined by associating a distinct node with each region (empty or nonempty) of the visibility map and connecting any pair of nodes whose corresponding regions share a common chord. Note that this also includes null-length chords. Figure 2.1 shows the tree without the nodes associated with empty regions. Since any two consecutive regions in the canonical region enumeration have a common chord, the visibility tree is indeed connected. Why can it not have cycles? It suffices to show that removing any chord $ab$ would disconnect the graph.

First, assume that $a$ and $b$ are not duplicates of each other (although they might be companions). Then, from Lemma 2.1, removing $a$ and $b$ splits the double boundary into two pieces which are closed under visibility. It follows that the boundary of any region contains segments from exactly one of the two pieces, and therefore can be classified by the piece to which these segments belong. Any chord separating a region associated with one piece from a region associated with the other piece must join two points at the juncture between the two pieces, and only $ab$ satisfies this requirement. This concludes the first case. Assume now that $a$ and $b$ are duplicates of each other. If the chord $ab$ has zero length, then removing it isolates an empty region from the rest, and our claim holds. So, suppose that $ab$ has nonzero length. Then $a$ is either the highest or lowest point of $SC$, therefore removing $ab$ would disconnect the upper or lower part of the spherical plane from the other regions. This completes the proof that the visibility tree is indeed a tree. (There is also a simple topological proof, which is given in Lemma 2.2 below.)

2.2. Visibility Submaps

An operation which we will find useful is the removal of chords we go any further, however, we wish to prepare for the been augmented with some additional chords (something later). Obviously, these new chords cannot connect vertices been used up) but rather arbitrary points on the curve. After $V(C)$ as well as the visibility tree, everything we said previous enumerations can be trivially extended to this new situation; we treat $V(C)$ either in its original state or in some augmented which applies whenever the distinction needs to be made.

The operation we want to discuss now involves removing $V(C)$ for a chord $a$ may have additional chords, we need meaning of a removal. A chord has two endpoints; none are vertices of $SC$. So, the removal of a chord entails removing itself but also those endpoints that are not vertices of $SC$ those points. This cleanup operation is meant to prevent stranded in the middle of any edge of $SC$; in other words, vertex of $SC$ must be the endpoint of a nonremoved chord.

Removing one (or several chords (of zero or nonzero) produces a polygonal subdivision of the spherical plane, (Fig. 2.4). The boundary of a nonempty region of the subsequence of horizontal segments, called exit chords, alter and not of $C$. For example, let us assume that all null removed in the submap of Fig. 2.4. Then the boundary consists of a two-edge arc (beginning at the big dot), for a one-edge arc, an exit chord, a three-edge arc (not a four a one-edge arc, and one final exit chord. Note that so length, as it is the case in the region labeled $V$. As illustrated, traversal of $SC$ induces a traversal of the boundary of each which is counterclockwise with respect to the orientation of the plane, we have the following fact.

---

Fig. 2.4
have a pair of mutually visible points from the double boundary curve, then no chord can connect the two resulting pieces.

be the two pieces of the double boundary resulting from visible points. Together with $C_1$ and $C_2$, the chord $c$ splits the spherical plane into three polygonal bounded by simple polygonal curves), one of which is the polygon $C$ itself. Any chord connecting $C_1$ and $C_2$ lies outside $C$ but this is impossible because chords are extensions of chord endpoints in $V(C)$ encountered during a clockwise traversal of the canonical vertex enumeration of $V(C)$. Note that $V(C)$ contains vertices of $C$. Figure 2.3 shows the sequence of vertices indicated by dashed lines. Recall that we have the endpoints of null-length chords. Speaking of which, chords create empty regions. The traversal of the sequence of vertices indicates the regions (with repetitions). In the list, only the nonempty regions for $V(C)$ are shown.

The dual graph of the subdivision is a tree, naturally called the canonical vertex enumeration of $V(C)$. This graph is defined by associating a distinct node with each vertex of $V(C)$. Each node is labeled with the coordinates and radius of the vertex. The graph consists of edges connecting adjacent vertices. Each edge represents a chord connecting two vertices. The tree structure of the graph permits efficient traversal and querying of the visibility information.

The visibility map connects any pair of non-adjacent regions with a common chord. Note that this also applies to the dual graph, Figure 2.1. The tree structure of the graph permits efficient traversal and querying of the visibility information.

The dual graph of the subdivision is a tree, naturally called the canonical vertex enumeration of $V(C)$. This graph is defined by associating a distinct node with each vertex of $V(C)$. Each node is labeled with the coordinates and radius of the vertex. The graph consists of edges connecting adjacent vertices. Each edge represents a chord connecting two vertices. The tree structure of the graph permits efficient traversal and querying of the visibility information.

The visibility map connects any pair of non-adjacent regions with a common chord. Note that this also applies to the dual graph, Figure 2.1. The tree structure of the graph permits efficient traversal and querying of the visibility information.

The dual graph of the subdivision is a tree, naturally called the canonical vertex enumeration of $V(C)$. This graph is defined by associating a distinct node with each vertex of $V(C)$. Each node is labeled with the coordinates and radius of the vertex. The graph consists of edges connecting adjacent vertices. Each edge represents a chord connecting two vertices. The tree structure of the graph permits efficient traversal and querying of the visibility information.

2.2. Visibility Submaps

An operation which we will find useful is the removal of chords from $V(C)$. Before we go any further, however, we wish to prepare for the possibility that $V(C)$ has been augmented with some additional chords (something that will often happen later). Obviously, these new chords cannot connect vertices of $C$ (since all have been used up) but rather arbitrary points on the curve. Although this clearly affects $V(C)$ as well as the visibility tree, everything we said previously regarding canonical enumerations can be trivially extended to this new situation. So, from now on, let us treat $V(C)$ either in its original state or in some augmented form. We specify which applies whenever the discussion needs to be made.

The operation we want to discuss now involves removing a given chord from $V(C)$. Since $V(C)$ may have additional chords, we need to be careful about the meaning of a removal. A chord has two endpoints; none, one, or two of which are vertices of $C$. So, the removal of a chord entails removing not only the chord itself but also those endpoints that are not vertices of $C$, and gluing back $C$ at those points. This cleanup operation is meant to prevent the presence of vertices stranded in the middle of an edge of $C$: in other words, any vertex that is not a vertex of $C$ must be the endpoint of a nonremoved chord.

Removing one or several chords (of zero or non-zero length) from a map produces a polygonal subdivision of the spherical plane, called a submap of $V(C)$ (Fig. 2.4). The boundary of a nonempty region of the submap is an oriented circular sequence of horizontal segments, called exit chords, alternating with pieces of $C$ and not of $C$. For example, let us assume that all null-length chords have been removed in the submap of Fig. 2.4. Then the boundary of the region labeled I consists of two two-edge arcs (beginning at the big dot), followed by an exit chord, a one-edge arc, an exit chord, a three-edge arc (not a four-edge arc!), an exit chord, a one-edge arc, and one final exit chord. Note that some arcs may be of zero length, as is the case in the region labeled V. As illustrated in Fig. 2.4, a clockwise traversal of $C$ induces a traversal of the boundary of each region of the submap which is counterclockwise with respect to the orientation of the region. More formally, we have the following fact.

Fig. 2.4
Lemma 2.2. Let \( A_1, \ldots, A_k \) be the counterclockwise enumeration of the (oriented) arcs of a nonempty submap region (as induced from the region's orientation). Then each \( A_i \) is oriented clockwise with respect to \( \partial C \). Moreover, the sequence \( A_1, \ldots, A_k \) also occurs clockwise around \( \partial C \).

Proof. The curve \( \partial C \) is homeomorphic to a circle embedded in the spherical plane. Adding a chord is topologically equivalent to connecting two points on the circle by a simple curve lying entirely on one side of the circle. The requirement that all these curves should be mutually disjoint induces a parenthesis system which immediately reveals the tree structure of the dual graph. This is similar to the parenthesis systems in Jordan sorting [18]. As an example, Fig. 2.5 depicts the topological equivalent of the submap of Fig. 2.4. From this perspective, the lemma should be completely obvious.

As was the case with \( V(C) \), a clockwise traversal of \( \partial C \) induces canonical vertex/region enumerations of the submap. Figure 2.4 gives the region enumeration: I, II, III, IV, V, VI, I. (Recall that that particular submap is supposed to have had all its null-length chords removed and therefore has no empty regions.) Bold dots mark the points during the clockwise traversal of the double boundary where the canonically enumerated regions are first discovered. An important requirement is that a vertex enumeration of a submap should list only the endpoints of actual exit chords and thus might skip over many vertices of \( \partial C \). In this way, canonical enumerations of any type take time proportional to the number of regions and not to the number of vertices (which might be much higher). We define the weight of a region as 0 if the region is empty, or else as the maximum number of nonnull length edges in any of its arcs. For example, regions I and II in Fig. 2.4 have weights 4 and 3, respectively.

Although weights count only edges of nonzero length, chords of zero length (if any) are taken into account inasmuch as they separate arcs. In other words, an arc never contains any chord, whether that chord be of nonzero length or not. Of course, once removed, a chord of zero length ceases to separate any arcs. Also, note the important role played by the double boundary in the definition of a region's weight. Indeed, a region may have a very small weight because its arcs

are all small; but this does not prevent any one of these number of vertices on the other side of the double bound of \( \partial C \) have companions those extra vertices have chords.

Combinatorially, a region corresponds to a subtree of

The dual graph of a submap is obtained by contracting a tree that correspond to the removed chords (Fig. 2.6), visibility tree by graph-minor operations, the dual graph (as was clear in the proof of Lemma 2.2), which we submap. Note that, conversely, contracting any edge of tree to removing the corresponding chord from the visibility node naturally refers to the weight of its corresponding

2.3. Conformality and Granularity

Since no two distinct vertices of \( C \) have the same \( x \)-coordinate, a node in a visibility tree cannot exceed 4. We should not talk of submap trees, however, so we distinguish the trees as those with node-degree at most 4. By analogy with the [4], we can decompose a conformal submap in an hierarchy, is to pick the centroid of the submap's tree and observe one incident edge whose removal leaves two subtrees, each at most three-quarters the original number. Associating the root of a binary tree and recursing in this fashion we get a tree decomposition of the submap logarithmic in the number of regions (which is the number of the submap). Figure 2.7 illustrates the correspondence between
... $A_1$ be the counterclockwise enumeration of the (oriented) map region (as induced from the region's orientation). There
wise with respect to $O_C$. Moreover, the sequence $A_1, \ldots, A_4$
bound $O_C$.

is homeomorphic to a circle embedded in the spherical
structure so that each point on the boundary is topologically equivalent to connecting two points on the
boundary entirely on one side of the circle. The requirement
on the interior points of the cycle is that they must lie in Jordan sorting [18]. As an example, Fig. 2.5 depicts
the $V(C)$, a clockwise traversal of $O_C$ induces canonical
presentations of the submap. Figure 2.4 gives the region enumeration
V, II, I. (Recall that that particular submap is supposed
length chords removed and therefore has no empty regions.)
In general the clockwise traversal of the double boundary
enumerated regions are first discovered. An important
vertex enumeration of a submap should list only the
chords and thus might skip over many vertices of $O_C$. In
representations of any type take time proportional to the number
of vertices (which might be much higher). We
give as 8 if the region is empty, or else as the maximum
in any of its arcs. For example, regions I and II
4 and 3, respectively.

not only edges of nonzero length chords of zero length (if
unt inasmuch as they separate arcs. In other words, an
edge, whether that chord be of nonzero length or not. Of
chords of zero length ceases to separate any arcs. Also,
played by the double boundary in the definition of a
region may have a very small weight because its arcs

are all small, but this does not prevent any one of these arcs from having a huge
number of vertices on the other side of the double boundary. Of course, because
vertices of $O_C$ have companions those exit vertices have to be endpoints of exit
chords.

Combinatorially, a region corresponds to a subtree of the visibility tree of $C$.
The dual graph of a submap is obtained by contracting the edges of the visibility
tree that correspond to the removed chords (Fig. 2.6). Being derived from the
visibility tree by graph-theoretic operations, the dual graph of a submap is itself a
tree (as was clear in the proof of Lemma 2.2), which we simply call the tree of the
submap. Note that, conversely, contracting any edge of the visibility tree amounts
to removing the corresponding chord from the visibility map. The weight of a
node naturally refers to the weight of its corresponding region.

2.3. Conformality and Granularity

Since no two distinct vertices of $C$ have the same $y$-coordinate, the degree of any
node in a visibility tree cannot exceed 4. We should not expect this to be always
true of submap trees, however, so we distinguish the trees of conformal submaps
as those with node-degree at most 4. By analogy with the polygon-cutting theorem
[4] we can decompose a conformal submap in an hierarchical manner. The idea
is to pick the centroid of the submap's tree and observe that there exists at least
one incident edge whose removal leaves two subtrees, each with a number of edges
at most three-quarters the original number. Associating the removed edge with
the root of a binary tree and recursively proceeding in this fashion with respect to the root's
children provides a tree decomposition of the submap. The tree has depth
logarithmic in the number of regions (which is the number of chords plus one).
The internal nodes (resp. leaves) are in bijection with the exit chords (resp. regions)
of the submap. Figure 2.7 illustrates the correspondence (leaves have been omitted).
By using straightforward tree-labeling techniques we can find the centroid node, and from there, the first edge to be removed, in linear time. Proceeding recursively gives us a simple $O(m \log m + 1)$-time algorithm for computing the tree decomposition of a submap of $m$ regions. We use this result below because it is simple and practical. Optimal methods exist but they are all fairly complicated and unnecessary for our purposes [3], [14].

One final piece of classification addresses our desire to make submaps coarse but uniform approximations of visibility maps. We say that a submap is $\gamma$-granular if

(i) every node of its tree has weight at most $\gamma$ and

(ii) contracting any edge incident upon at least one node of degree less than 3 produces a new node whose weight exceeds $\gamma$.

Note that this weight might be less than the added weight of the two nodes of the contracted edge. This is either because the arcs incident upon the chord removed did not determine the weights, or more interestingly, because one or both endpoints of the chord might not be vertices of $\mathcal{C}$ and might thus disappear. If only condition (i) holds, then the submap is $\gamma$-semigranular. Adding condition (ii) makes the semigranularity maximal in some sense. Finally, by default, if (i) holds but the submap has no exit chord, it is still said to be $\gamma$-granular. The following result asserts that, as we would expect, a $\gamma$-granular conformal submap is more economical to encode than its full visibility map by a factor of $\gamma$.

Lemma 2.3. If $C$ is a polygonal curve with $n$ vertices, any $\gamma$-granular conformal submap of the (possibly augmented) visibility map of $C$ has $O(n/\gamma + 1)$ regions and each region is bounded by $O(\gamma)$ edges.

Proof. We can assume that the tree of the submap has at most $\gamma$ siblings. If the lemma is trivial because of the $\gamma$-granularity. Among them, let $E$ be the set of those incident upon at least one node that is trivial to show by induction on the size of the tree that has a fixed fraction of all the edges. Now, contracting any edge removing a chord associated with $E$ produces a merged tree $\gamma$-meaning that it has an arc with more than $\gamma$ edges. A vertex of $C$ can give rise to at most 4 vertices of $\mathcal{C}$. If not, extra vertices behind itself, those of $\mathcal{C}$, such at least $O(\gamma)$ distinct vertices of $C$. If contracting any edge of $E$ a disjoint merged region, then it would follow from the path $E$, and hence the whole tree, has $O(n/\gamma + 1)$ edges. Unfortunatly might produce overlapping merged regions (i.e., if they From the conformity of the submap, however, we know $C$ can be used at most $\gamma$ constant number of times in therefore $E$ has indeed $O(n/\gamma + 1)$ edges and the first established. The second part derives from the conformity ensures that there is a bounded number of arcs per region. total number of bounding edges is at most proportional to.

2.4. Representation Issues

How do we represent visibility maps and submaps as we describe our mode of representation, then we point out some and explain why they are needed. Let $P$ be the input polygon whose visibility map is sought and let $C$ be the submap (or submap) we wish to represent. We assume that $P$ is not restrictive since a little hole can always be punctured. We assume that the edges of $P$ are stored in a table, in the order in which they occur along the boundary of $P$. A (and also do.) Note that the notion of double boundary need not be i.e., no edges are duplicated in the table. The input table is to be modified or even copied. A visibility submap of $P$ of its own data structure: arcs are encoded by pointing directly. More precisely, each arc is represented by a separate arc data arcs can be represented explicitly so let us assure that the arcs, where $e_i$ and $e_j$ are the edges adjacent to the two chords $t_i = 1$, then the arc-structure consists of a single point representative edge $e$ of $P$ that contains $e_i$. Since $e_i$ is an edge of the data need to indicate by a flag which side of $e_i$ is to be underrecord the endpoints of the arc because chords take care of the same information as above but now with respect to
Triangulating a Simple Polygon in Linear Time

B. Chazelle

Proof. We can assume that the tree of the submap has at least one edge, otherwise the lemma is trivial because of the $\gamma$-granularity. Among the edges of that tree, let $E$ be the set of those incident upon at least one node of degree less than 3. It is trivial to show by induction on the size of the tree that $E$ accounts for at least a fixed fraction of all the edges. Now, contracting any edge in $E$, or equivalently, removing a chord associated with $E$ produces a merged region of weight greater than $\gamma$ meaning that it has an arc with more than $\gamma$ edges of nonzero length. Since a vertex of $C$ can give rise to at most four vertices of $\mathcal{C}$, and removed chords do not leave extra vertices behind except those of $\mathcal{C}$, such an arc must involve at least $O(\gamma)$ distinct vertices of $C$. If contracting any edge of $E$ were always to produce a disjoint merged region, then it would follow from the pigeon-hole principle that $E$, and hence the whole tree, has $O(n\gamma^3)$ edges. Unfortunately, two edges of $E$ might produce overlapping merged regions (i.e., if they share a common node). From the conformality of the submap, however, we know that a given vertex of $C$ can be used at most a constant number of times in this counting argument, therefore $E$ has indeed $O(n\gamma^3)$ edges and the first part of the lemma is established. The second part derives from the conformality of the submap, which ensures that there is a bounded number of arcs per region and, hence, that the total number of bounding edges is at most proportional to the weight of the region.

2.4. Representation issues

How do we represent visibility maps and submaps as data structures? We first describe our node of representation, then we point out some of its idiosyncrasies and explain why they are needed. Let $P$ be the input polygonal curve (the one whose visibility map is sought) and let $C$ be the subchain of $P$ whose visibility map (or submap) we wish to represent. We assume that $P$ is nonclosed; this is not restrictive since a little hole can always be punctured if it is closed to begin with. We assume that the edges of $P$ are stored in a table (the input table) in the order in which they occur along the boundary of $P$. (A doubly linked list would also do.) Note that the notion of double boundary need not be encoded explicitly, i.e., no edges are duplicated in the table. The input table is read-only; it is never to be modified or even copied. A visibility submap of $V(C)$ is represented by its own data structure: arcs are encoded by pointing directly into the input table. More precisely, each arc is represented by a separate arc-structure. Null-length arcs can be represented explicitly so let us assume that the arc has nonzero length. Let $e_1, \ldots, e_t$ be the edges of an arc in clockwise order along the double boundary, where $e_1$ and $e_t$ are the edges adjacent to the two chords connected by the arc. If $t = 1$, then the arc-structure consists of a single pointer into the input table to the edge $e$ of $P$ that contains $e$. Since $e_1$ is an edge of the double boundary, we also need to indicate by a flag which side of $e_1$ is to be understood. We do not need to record the endpoints of the arc because chords take care of that. If $t > 1$, we store the same information as above but now with respect to both $e_1$ and $e_t$ in that
order. We say that a submap (or map) is given in normal form if the following information is provided:

(i) The tree of the submap (or map) is represented in standard edge/node adjacency fashion.

(ii) Each edge of the tree contains a record describing the corresponding chord as well as pointers to the arc-structures of the two, three, or four arcs adjacent to it. Conversely, each arc-structure has a pointer to the node of the tree whose corresponding region is incident upon the arc in question.

(iii) The arc-structures are stored in a table (the arc-sequence table) in the order corresponding to a canonical traversal of the double boundary of C. Also, the endpoints of C are identified by appropriate pointers into the input table as well as by pointers to the arc-structures whose corresponding arcs pass through the endpoints.

(iv) If the submap is conformal, then its tree decomposition should be available.

We choose what may seem to be a contrived representation of a submap in order to use storage proportionally not to the number of edges in the submap but rather to the number of regions (which is of the same order of magnitude as the number of chords and arcs). It is essential to avoid excessive duplication of information because we need to encode a collection of submaps whose number of distinct features is only \(O(n)\), but whose combined size, counting redundancy, is \(O(n \log n)\). Note that our representation is powerful enough to let us perform canonical vertex/region enumerations in optimal time. If we wish to, we can also enumerate all the vertices of \(\partial C\) in clockwise order directly from a canonical vertex enumeration of the submap, since any arc can be reconstructed explicitly from the succinct information given by the arc-structure: it suffices to explore the input table between the locations indicated by the two pointers of the arc-structure. Note that caution must be used since an arc might wrap around both sides of \(\partial C\), something we call double-heading. This can be detected when we traverse the arc as soon as we reach an edge of \(\partial C\) incident upon an endpoint of \(C\).

Perhaps a less obvious task is to retrieve the arc-structure corresponding to an arc, given one of its edges. More specifically, suppose that we are given an edge \(e\) of \(C\) and a point \(q\) on it. The question is to find the arc-structures of all those arcs in the submap that pass through the point \(q\). By passing through, we do not care whether the arc is on any particular side of the double boundary, so, for example, if \(q\) is not an endpoint of any chord in the submap, then there are at most two distinct arcs to be found. Otherwise, there are at most six of them, two of which are of zero length: this worst case occurs when \(q\) coincides with a vertex of \(C\) that is a local extremum in the \(y\)-direction. Since we know the location of the two endpoints of \(C\) in the arc-sequence table (i.e., which arcs pass through them) we can conceptually break up the circular arc sequence into two linear sequences and perform in each of them a binary search, using the name of the containing edge \(e\) as a query. Either search might take us to a unique arc-structure, in which case we are done, or else to a contiguous interval of arc-structures: this might happen if \(e\) contributes several arcs. We can disambiguate by pursuing the

binary search, now using, say, the \(y\)-coordinate of \(q\) as a query. The time is logarithmic in the number of arcs. This operation is necessary if we want to navigate in a submap across its arcs: we call it \(N\) of a point of \(C\).

We have said repeatedly that a submap has a tree structure, our purpose for a moment and look at a submap subdivision, without distinguishing between chords and arc-structure representations of planar graphs [2], [15], [21]. If we navigate through such a subdivision along the edges in \(\partial G\), we find that there is a unique path from one side of an arc to the other by following a straight line. In order to find which region we are about to enter, we use a double identification. The difficulty here is that unlike walks on standard graph representations do not keep adjacent regions and edges (except for chords). More important, the explicit correspondence between the features on the two sides of an arc makes the region graph more suitable for our purposes.

For reasons which will become clear later, it would be a waste of time to do so.

2.5. A Topological Lemma

We close this discussion of submap and chords, which we use to establish the correct merging algorithm. Let \(D\) be a closed disk in the sphere boundary, and let \(ab\) denote a diametrical chord of \(D\). Pick \(c\) on \(\partial D\) such that the circumcircle of \(a, b, c\) have in clockwise order (with respect to \(c\), a simple curve lying inside \(D\) and running from \(c\) to \(a\) to \(b\) to \(c\)). The circle \(C\) that runs clockwise from \(d\) to \(c\) and let \(B_i\) (\(i = 1, 2\)) be the intersections with the two circular arcs of \(\partial D\) of \(\{a, b\}\) (Fig. 2.7) if either \(B_i\) is empty or else no point of \(\beta\) can be connected to a curve (understood here as a closed set) that lies entirely inside \(\partial D\) and intersects \(ab\) or \(a, b\). In Fig. 2.8, for example, the piece

![Diagram](image-url)

Fig. 2.8
A submap (or map) is given in normal form if the following hold:

1. A submap (or map) is represented in standard edge/node form.
2. The tree contains a record describing the corresponding chord between the arc-structures of the two, three, or four arcs. Conversely, each arc-structure has a pointer to the node of corresponding region incident upon the arc in question.
3. The are stored in a table (the arc-sequence table) in the order of a canonical traversal of the double boundary $BC$. Also, if $C$ is identified by appropriate pointers into the input by pointers to the arc-structures whose corresponding arcs are endpoints conformal, then its tree decomposition should be available.

We refer to the contrived representation of a submap in order to the number of edges in the submap but rather to (which is of the same order of magnitude as the number of $n$ is essential) to avoid excessive duplication of information some a collection of submaps whose number of distinct $n$ whose combined size, counting redundancy, is $O(n \log n)$.

Notation is powerful enough to let us perform canonical

2.5 A Topological Lemma

We close this discussion of visibility submaps by proving a result on the topology of regions and chords, which we use to establish the correctness of our submap merging algorithm. Let $D$ be a closed disk in the spherical plane, let $B$ be its boundary, and let $ab$ denote a diametrical chord of $D$. Pick two distinct points $c$ and $d$ on $B$ such that $a, b, c, d$ occur in clockwise order (with respect to $D$). Let $A$ be a simple curve lying inside $D$ and running from $c$ to $d$. Consider the circular arc that runs clockwise from $c$ to $c$ and let $B_i$ ($i = 1, 2$) be the closures of its intersections with the two circular arcs of $D$ and $D$ (Fig. 2.8).

Note that each $B_i$ consists of $0, 1$, or $2$ circular arcs. We say that a subset $A$ of $B_i$ if either $B_j$ is empty or else no point of $B_j$ can be connected to any point of $B_i$ by a curve (understood here as a closed set) that lies entirely inside $D$ and does not intersect either $ab$ or $A$. In Fig. 2.8, for example, the piece of $A$ running from $c$ to $d$
Lemma 2.4. If \( a \in B_1 \cup B_2 \) (resp. \( b \in B_1 \cup B_2 \)), let \( a' \) (resp. \( b' \)) be the first (resp. last) point of \( ab \) not encountered when traversing the diametrical chord \( ab \) from \( a \) to \( b \). The points \( a' \) and \( b' \) (which might not exist) subdivide \( A \) into a total of one, two, or three connected curves, each of which is shown from some \( B_i \) (not necessarily the same) for all curves). Furthermore, an appropriate \( B_i \) can be identified simply on the basis of \( a, b, c, d, a', b' \).

Proof. We can assume that both \( B_1 \) and \( B_2 \) are nonempty and that \( A \) intersects \( ab \) (else the lemma is trivially correct). By attaching \( B_1 \cup B_2 \) to \( A \), we obtain a simple closed curve within \( D \), which is, therefore, the boundary of a subset of \( D \) homeomorphic to a disk. If \( a \) (resp. \( b \)) belongs to \( B_1 \cup B_2 \), then the segment \( ab \) (resp. \( ab' \)) lies within \( R \) and thus, acting like a chord, subdivides \( R \) into two regions. Since \( a \) and \( b \) cannot intersect, together they subdivide \( R \) into two or three disk-like regions. The boundary of each region intersects the boundary of \( D \) in a single connected arc and therefore avoids one \( B_i \) (outside of \( a \) and \( b \)) entirely. Figure 2.8 illustrates the two possible cases; note that the third case, where the counter-clockwise traversal from \( c \) to \( d \) avoids both \( a \) and \( b \), was eliminated earlier, since it corresponds to a situation where only one of the \( B_i \)'s is empty. None of the curves obtained by removing \( a' \) and \( b' \) (if they exist) from \( A \) can belong to more than one of the subdividing regions, so each of them is fixed in a \( B_i \) which one can be determined by simple examination of the relative order of the points \( a, b, c, d, a', b' \) around the boundary of \( R \).

3. Merging Two Submaps

The inner loop of the visibility algorithm involves merging two conformal submaps. Everything else in the algorithm is part of a control mechanism for deciding what gets to be merged with what, at what time, and with what desired granularity. Let \( C_1 \) and \( C_2 \) be two polyhedral curves of \( n_1 \) and \( n_2 \) vertices respectively, whose union \( C \) forms a connected vertex-to-vertex piece of the input (simple and nonclosed) polygonal curve \( P \). We assume that \( C_1 \cap C_2 \) is a vertex of \( P \). Let \( S_i \) (\( i = 1 \), \( 2 \)) be a \( y \)-granular conformal submap of \( V(C_i) \) with \( y_1 \leq y_2 \). Given any integer \( y \geq y_2 \), to merge \( S_1 \) and \( S_2 \) (where \( y \) is understood) means to compute a normal-form \( y \)-granular conformal submap of \( V(C) \).

To facilitate the exposition we assume that we have at our disposal two primitives: one is a ray-shooting oracle, which allows us to shoot a horizontal ray toward any subarc of \( S_i \) or \( S_j \) and discover which point, if any, is first hit by the ray; this gives us a way to discover new chords. The other primitive is an arc-cutting oracle, which enables us to cut up any subarc into a small number of pieces for which conformal submaps of the appropriate granularity have already been computed. This is to be used for restoring the conformal submaps.

The merge proceeds in three stages. First, we find which of the end chords of \( S_i \) (\( i = 1 \), \( 2 \)) vertices resulting from the duplication of \( C_i \cap C_2 \); this gives us a map to construct a submap \( S \) of \( V(C) \), called the fusion of \( S_1 \) and \( S_2 \). The second stage we ensure that the submap is conformal by adding new chords to cut up regions with more than four; calling upon the arc-cutting oracle, which allows us to draw conformal submaps and their tree decompositions are chords to cut up big regions is carried out by using another appropriate tree decompositions, using the ray-shooting oracle (Section 3.2). In the third stage, finally, we bring the arc-cutting oracle by removing chords if necessary (Section 3.3); the oracles are discussed in Section 3.4.

We need to be able to distinguish between an arc of \( C \) from which it originates. For this purpose we introduce the notion of an endpoint of \( C \) to which an arc \( a \) of \( C \) corresponds. Double-back around an endpoint of \( C \), so \( a \) may not always be.

We assume that each \( S_i \) is given in normal form and the primitives are available. For each region arc \( e \) of \( S_i \) (\( i = 1 \), \( 2 \)) to its arc-structure:

(i) There exists a ray-shooting which given any point \( p \) direction (left or right) and any subarc \( x' \) of \( x \) specified (along with two pointers to the input table to incident edges of \( P \) that contain these two endpoints as well as the side of \( \partial P \) is to be understood), reports the the \( x \) ray of light shot from \( p \) in the given direction \( x' \) of any obstacle except \( x' \). In addition to this point also include the name of the edge of \( P \) that contains \( x' \) and take \( O(f(y)) \) time, so that at most \( (y + 1) \) subarcs \( x_1, x_2, \ldots \) such that \( (y + 1) \) endpoints and a pair of pointers into the input edges of \( P \) contain these endpoints, the pair should be clockwise traversal along \( \partial P \) and the two flags should be on which sides of \( \partial P \) these endpoints fall; (ii) the region should contain a point of \( \partial C \) that corresponds to \( e \), such that each of its vertices must lie entirely on one side of \( \partial C \) (or \( \partial P \) if 

(3) except for \( x_1 \) and \( x_2 \), as shown in the case where these are attached to the points of \( C \) corresponding to the edge \( e \) itself, are vertex-to-vertex subchains of \( C_i \) (i.e., they do not lie on an edge) and, for each of them, an \( N_{y+1} \)-granular \( V(x) \) is available in normal form. Again, \( g \) and nondecreasing functions.
been computed. This is to be used for restoring the conformality of merged submaps.

The merge proceeds in three stages. First we find which points of $\mathcal{C}$ can be seen by the endpoints of the exit chords of $S_i (i = 1, 2)$ and by the companion vertices resulting from the duplication of $C_i \cup C_j$; this gives us chords which we use to create a submap $S$ of $(C, \mathcal{C})$, called the fusion of $S_1$ and $S_2$ (Section 3.1). In the second stage we ensure that the submap is conformal, which might involve adding new chords to cut up regions with more than four arcs. This is done by calling upon the arc-cutting oracle, which allows us to deal with submaps for which conformal submaps and their tree decompositions are available. Finding new chords to cut up big regions is carried out by binary search through the appropriate tree decompositions, using the ray-shooting oracles along the way (Section 3.2). In the third stage, finally, we bring the submap $S$ to the desired granularity by removing chords if necessary (Section 3.3). The implementation of the oracles is discussed in Section 3.4.

We need to be able to distinguish between an arc of $\mathcal{C}$ and the piece of $C$ from which it originates. For this purpose we introduce the notation $\mathcal{S}$ to refer to the portion of $C$ to which an arc $a$ of $\mathcal{C}$ corresponds. Recall that an arc may double-back around an endpoint of $C$, so it may not always be as "long" as $a$.

We assume that each $S_j$ is given in normal form and that the following set of primitives is available. For each region arc of $S_j (j = 1, 2)$ specified by a pointer to its arc-structure:

(i) There exists a ray-shooting which, given any point $p$ along with a horizontal direction (left or right) and any subarc $a'$ of $a$ specified by its two endpoints (along with two pointers to the input table to indicate the names of the edges of $P$ that contain these two endpoints as well as two flags indicating which side of $P$ is to be understood), reports the single point of $a'$ (if any) that a ray of light shot from $p$ in the given direction would hit in the absence of any obstacle except $a'$. In addition to the point hit, the report should also include the name of the edge of $P$ that contains it. The report should take $O(t_1)$ time, where $f$ is a nondecreasing function.

(ii) There exists an arc-cutter which, in $O(g(x))$ time, subdivides the subarc $a'$ into at most $g(x)$ subarcs $a_{x1}, a_{x2}, \ldots$, such that (1) each $a_{xi}$ is specified by its two endpoints and a pair of pointers into the input table to indicate which edges of $P$ contain these endpoints; the pair should be ordered to reflect a clockwise traversal along $P$ and two flags should be included to indicate on which sides of $P$ these endpoints fall; (2) the relative interior of no $a_{xi}$ should contain a point of $\mathcal{C}$ that corresponds to an endpoint of $C_i$, that is, each subarc must lie entirely on one side of $\mathcal{C}$ (no double-backing); and (3) except for $a_{x1}$ and $a_{xj}$ (in the case where these are single-edge pieces attached to the points of $\mathcal{C}$ corresponding to the endpoints of $a'$), all the $a_{xi}$'s are vertex-to-vertex subchains of $C_i$ (i.e., they do not stop in the middle of an edge) and, for each of them, an $\mathcal{H}_2$-granular conformal submap of $V(a)$ is available in normal form. Again, $g$ and $h$ are assumed to be nondecreasing functions.
Given these oracles we show how to merge $S_1$ and $S_2$ in

$$O(n_1/j_1 + n_2/j_2 + 1)f(y_1g(y_2)\log n_1 + \log n_2 + n_3))$$

time. Ideally, we would like the extra factor $f(y_1g(y_2)\log n_1 + \log n_2 + n_3))$ to be constant. This would mean that the merge could be done in time proportional to the total size of the submaps but to the number of chords in them. We cannot achieve this, but we can find functions $f, g, h$ which, although nonconstant, are small enough for our purposes. Specifically, we have $f(x) = O(x^{1/4} \log )$, $g(x) = O(\log x)$, and $h(x) = O(x^{1/6})$. This allows us to carry out a merge in time almost proportional to the total number of chords. Note that to achieve $f(x) = g(x) = x$ is trivial but, for our purposes, completely useless.

### 3.1. Fusion of Two Submaps

By symmetry, we may limit our discussion to the problem of fusing $S_1$ into $S_2$, that is, determining the points of $\partial C$ that are seen by the endpoints of the exit chords of $S_1$ and by the companion vertices resulting from the duplication of $C_1 \cap C_2$. The idea is then to repeat the work described below with respect to $S_2$ (i.e., fusing $S_2$ into $S_1$), and set up a new submap $S$ based on the information collected. Let $a_{i+1}$ and $a_i$ be the companion vertices, as they appear next to each other clockwise around $\partial C$, resulting from the duplication of the vertex $C_1 \cap C_2$ in $\partial C_1$. Let $a_0, a_1, \ldots, a_{m+1}$ be the canonical vertex enumeration of $S$. Recall that this enumerates the exit chord endpoints in $S$ as we encounter them going clockwise around $\partial C$. Since the sequence is circular, we can assume that $a_0$ precedes $a_1$ and $a_{m+1}$ follows $a_0$. Note that it could happen that $a_0$ and $a_{m+1}$ are already part of the sequence, but this need not be the case because of chord removals which might have occurred during previous merges, so, for the sake of generality, we assume that they are not and therefore add them to the sequence. A clockwise tour around $\partial C_1$ that begins at $a_0$ thus ends at $a_{m+1}$. We compute the points of $\partial C$ seen by $a_0, a_1, \ldots, a_{m+1}$ in that order.

We begin with a simple observation. Given any point $p$ of $\partial C_1$ and the arc to which it belongs (specified by its arc structure), we can determine which point of $\partial C$ it sees (with respect to $C_2$) in $O(f(y_1))$ time. This operation is called local shooting. Recall that because of the double boundary the shooting direction is always uniquely defined. If $p$ is an endpoint of an exit chord we can easily do that (even in constant time). If not, then $p$ belongs to a unique region of $S$, which we can determine in constant time (i.e., via its node in the submap tree), and the point of $\partial C$ that it sees lies on one of the region's arcs. This is because regions are closed under visibility, which is a corollary of Lemma 2.1. Using the appropriate ray-shooters, we can find that point by checking each arc in turn and finding the nearest hit. The claim on the time follows from the conformality of $S$, which ensures that at most four arcs need to be checked. Note that local shooting is still possible even if $p$ does not lie on $\partial C_1$; it can lie anywhere in the spherical plane as long as a horizontal direction (left or right) has been properly determined which region of $S$, the point lies. We still call this operation local shooting.

To fuse $S_1$ into $S_2$, we let a variable $p$ run through all the endpoints of $C_2$ encoded in the normal-form representation of $S_2$, in some order. We find its counterpart $p'$ in $S_1$, if it exists. Depending on whether $p'$ is to the left or right of $p$, we adjust the global direction (left or right) and repeat the process until we have found the complete visible region of $S_2$ around $p$. If we then find the point $p_0$ of $\partial C_1$ that $p'$ sees, we can infer all the visible region of $S_2$ around $p_0$ by repeating the process described above. Thus, at each stage, we search the current visible region of $S_2$ for the point of $\partial C_1$ that it sees. If it is found, it becomes the current point $p$ of $S_2$, and if not, we continue searching for the next point. This process is repeated until the current visible region is empty.

**Start-Up.** Using local shooting, we find the point of $\partial C_1$ to $C_1$. Although $a_0$ is at worst infinitesimally close to $\partial C$ on it, as we shall see in the next paragraph. However, using the endpoints of $C_1$ encoded in the normal-form representation points to incident arcs), we can find, in constant time, the point $a_0$ lies. This allows us to do local shooting and find $a_0$ sees with respect to $C_1$. These two pieces of information contain the unique point $a_0$ of $\partial C$ that $a_0$ sees with respect to $C$. In two cases:

1. If $a_0$ belongs to $\partial C_2$, then we set $p = a_0$ and we call by $a_0a_c$ current: the start-up phase is over (Figure 3.1).
2. If $a_0$ belongs to $\partial C_1$, from Lemma 2.1, the chord curve, each closed under visibility. One of these from $a_0$ to $a_c$ clockwise, is a piece of $\partial C_1$ (Fig. 3.1) that its exit chord endpoints see all belong to $\partial C_1$ directly from $S$. We can therefore skip all the way sees a point of $\partial C_1$, namely $a_0$, so we set $p = a_0$ containing $a_0$ current.

Technically, it is not quite true that $a_0$ is always a point of $S$ one most often, but when it sits at a local extremum (in one because of duplication. What is true, however, is that the point of $\partial C_1$, an infinitesimal deformation of $\partial C_1$ locally see one. This is a minor technicality which will not affect fusion algorithm, so for simplicity we still go on saying the with respect to $C$. Another minor problem is that $a_0a_c$ m of $S_2$ and thus there might be more than one candidate

![Fig. 3.1](image-url)
show how to merge $S_1$ and $S_2$ in

$$n_1/2 + 1)(f(1/2) + \log(n_1 + n_2))$$

like the extra factor $f(1/2)\log(n_1 + n_2)$ to be mean that the merge could be done in time proportional the submap $S$ to the number of chords in them. We can find functions $f(s, h)$ which, although nonconstant, or purposes. Specifically, we have $f(s) = \theta(s^{1/3})$, $g(s) = s^{1/2}$. This allows us to carry out a merge in time almost linear number of chords. Note that to achieve $f(s) = g(s) = s$ purposes, completely useless.

maps

limit our discussion to the problem of merging $S_1$ into $S_2$, points of $\mathcal{C}$ that are seen by the endpoints of the exit companion vertices resulting from the duplication of an in to repeat the work described below with respect to $S_1$ and set up a new submap $S$ based on the information $a_0$ be the companion vertices, as they appear next to round $\mathcal{C}_2$, resulting from the duplication of the vertex $a_1, \ldots, a_n$ be the canonical vertex enumeration of $S_1$, test the exit chord endpoints in $S_1$ as we encounter them $\mathcal{C}_2$. Since the sequence is circular we can assume that follows $a_n$. Note that it could happen that $a_0$ and $a_{n+1}$ sequence, but this need not be the case because of chord wise occurred during previous merges so, for the sake of as they are not and therefore add them to the sequence. $\mathcal{C}_2$, that begins at $a_0$ thus ends at $a_{n+1}$. We compute $a_0, a_1, \ldots, a_{n+1}$ in that order.

We observe. Given any point $p$ of $\mathcal{C}_2$ and the arc to the end of its circular structure, we can determine which point of to $C_1$ is $\mathcal{O}(|y|/y)$ time. This operation is called local cause of the double boundary the shooting direction is if $p$ is an endpoint of an exit chord we can easily do that if not, then $p$ belongs to a unique region of $S_1$, which we time (i.e., via its node in the submap tree), and the point time follows from the conformity of $S_1$, which arc need to be checked. Note that local shooting is still the in an $\mathcal{O}(1)$ it can lie anywhere in the spherical plane

as long as a horizontal direction (left or right) has been specified and we know in which region of $S$, the point lies. We still call this operation local shooting.

To fuse $S_1$ into $S_2$ we let a variable $p$ run through $\mathcal{C}_1$ in clockwise order, stopping at $a_1, \ldots, a_{n+1}$, as well as at some other places to be specified. We determine what $p$ sees along the way, while keeping track of the current region of $S_1$ in which $p$ lies. We use a start-up phase to initialize $p$ and launch the fusion.

Start-Up Using local shooting, we find the point of $\mathcal{C}'$, that $a_0$ sees with respect to $C_1$. Although $a_0$ is at worst infinitesimally close to $\mathcal{C}_1$ it does not always lie on it, as we shall see in the next paragraph. However, using the information about the endpoints of $C_1$ encoded in the normal-form representation of $S_1$ (namely, pointers to incident arcs), we can find, in constant time, in which region of $S_1$ the point $a_0$ lies. This allows us to do local shooting and find the point of $\mathcal{C}_2$, that $a_0$ sees with respect to $C_1$. These two pieces of information combine to give us the unique point $C_2$ of $\mathcal{C}$ that $a_0$ sees with respect to $C$. We distinguish between two cases:

1. If $c_0$ belongs to $\mathcal{C}_2$, then we set $p = a_0$ and we call the region of $\mathcal{C}_2$ crossed by $a_0$ current: the start-up phase is over (Figure 3.1.1).
2. If $c_0$ belongs to $\mathcal{C}_1$, from Lemma 2.1, the chord $a_0 c_0$ splits $\mathcal{C}_2$ into two curves, each closed under visibility. One of these curves, the one running from $a_0$ to $c_0$ clockwise, is a piece of $\mathcal{C}_1$ (Fig. 3.1.2), so the points of $\mathcal{C}$ that its exit chord endpoints see all belong to $\mathcal{C}_1$, and thus are available directly from $S_1$. We can therefore skip all the way to $c_0$. Now, however, $c_0$ sees a point of $\mathcal{C}_2$, namely $a_0$, so we set $p = c_0$ and call the region of $S_2$ containing $a_0$ current.

Technically, it is not quite true that $a_0$ is always a point of $\mathcal{C}_1$. It coincides with one most often but when it sits at a local extremum (in the $y$-direction) it is not one because of duplication. What is true, however, is that when $c_0$ cannot see a point of $\mathcal{C}_1$, an infinitesimal deformation of $\mathcal{C}_1$ locally around $a_0$ can make $c_0$ see one. This is a minor technicality which will not affect the remainder of the fusion algorithm, so for simplicity we still go on saying that $c_0$ sees a point of $\mathcal{C}_2$ with respect to $C$. Another minor problem is that $a_0 c_0$ might lie on an exit chord of $S_2$ and thus there might be more than one candidate for the status of current

![Fig 3.1](image-url)
region. We break ties by electing the region that we locally enter as we leave $p$ in a clockwise traversal of $\partial C$. This concludes the start-up phase. At this point we have the following situation (all visibility being understood with respect to $C$):

A. The points of $\partial C$ that are seen by the exit chord endpoints of $S_1$ on the portion of $\partial C_1$ running clockwise from $a_2$ to the point $p$ in its current position have all been determined already.

B. The point $q$ of $\partial C$ that is seen by $p$ belongs to $\partial C_2$ and the chord $pq$ lies in the region of $S_2$ called current. If $p$ lies on a chord between two regions of $S_2$, the current region should be the one that we enter as we locally leave $p$ clockwise around $\partial C_2$.

These two conditions form our loop invariant, that is, they hold prior to every iteration of the process which we now describe.

**Main Loop.** Let $A_j$ denote the oriented arc of $S_j$ running from $a_{j-1}$ to $a_j$ (in clockwise order around $\partial C_j$); by extension $A_j$ stands for the subarc extending from $a_{j-1}$ to $a_j$ (resp. $a_j$ to $a_{j-1}$). Let $A$ be the arc containing $p$. In the likely event that $p$ is an endpoint of a chord of $S_1$, and thus belongs to two arcs, we must choose the starting (and not ending) at $p$, i.e., we set the condition $p \neq a_1$. When $p$ is set to $a_{j-1}$, however, the algorithm simply terminates and no $A_j$ need be defined. Let $R$ denote the current region prior to entering the following loop: iterate through $j = k, k+1, \ldots$ until

(i) $a_j$ lies in $R$ and the point of $\partial C$ that $a_j$ sees belongs to $\partial C_j$ (Figure 3.2.1), or

(ii) the previous condition (i) does not hold, but $R$ has at least one exit chord such that the point of $\partial C$ seen by one of its endpoints belongs to $A_j$ but strictly follows $p$ (Figure 3.2.2), or

(iii) $j = m + 2$.

If case (i) occurs, find which point of $\partial C$ is seen by $a_j$, declare that all $a_k$ ($k \leq j$, $j < k$) see points of $\partial C_j$ (with respect to $C_j$), set $p = a_j$, let the current region still be $R$, and iterate through the loop, resetting $k$ so as to comply with its definition. If case (ii) occurs, then all the candidate endpoints satisfying (ii), determine the point $q$ that is the last one encountered as we traverse $\partial C_j$ clockwise starting from $p$. In Figure 3.2, for example, $p$ is the point labeled $p_2$ and the chosen endpoint is labeled $q_1$.

Next, declare that $a_j$'s ($k \leq i < j$) see points of $\partial C_i$, set $p = p_j$, make our entry, we enter as we locally cross the exit chord at $p'$ also updating $k$ and $R$ according to their definitions. In case $k = m + 2$, we declare that all $a_j$'s ($k \leq i \leq m + 1$) see points of $\partial C_i$, and the chord $pq$ lies in the region of $S_j$ called current. If $p$ lies on a chord between two regions of $S_j$, the current region is the one that we enter as we locally leave $p$ clockwise around $\partial C_j$.

**Lemma 3.1.** It is possible to compute the fusion $S$ of $S_1$, $S_2$, $S_3$, $S_4$ in $O(n_1/t_1 + n_2/t_2 + 1/(f_1 + f_2) + log n_1)$ time.

**Proof.** We restrict our attention to the task of fusion being similar. We have already shown that the start-up which satisfies the loop invariant, so it suffices to establish inner loop past $a_j$. In case (i) we know that $a_j$ lies in and sees a point of $\partial C_j$, so invariant (B) is satisfied. If the claim that $a_{j-1}, a_{j-1}$ see points of $\partial C_j$. But a for $a_{j-1}, a_{j-1}$ not strong enough to reach the necessary $a_{j-1}, a_{j-1}$ must see. Any of these points if they exist outside of $R$ should lie outside $R$ imply seeing some $l (k \leq j)$, $a_j$ lies in region $R$ distinct from $R$ but also sees $\partial C_j$. We derive a contradiction. Let $a_j$ of $\partial C_j$, as we go from $p$ to $a_j$ clockwise, and let $q$ (resp. $p_j$ by $p$. The union of $a_j$, the chords $pq$ and $a_j$, running clockwise (with respect to $C_j$) from $a_j$ to $q$ forms a polygon (Figure 3.3). Since the dual graph of a submap exit chord of $R$ that leads to $R$ (note that $ab$ need not since there might be one or even several regions separate $\partial R$ runs from $R$ to $R$ it passes through one of the chord be the point of $ab \in \partial$ first encountered as we go from. Note that $ab$ cuts through $\partial R$, so $ab$ is well defined. The other with respect to $\partial C_j$ and $a_j$ lies in $A_j$, for some $h
Triangulating a Simple Polygon in Linear Time

all $a_i$ (i ≤ j) see points of $C^*$, set $p = p'$, make current the region of $S_j$ which we enter as we locally cross the exit chord at $p'$ along $C^*$, and iterate after updating $k$ and R according to their definitions. In case (iii) we stop and, unless $k = m + 2$, we declare that all $a_i$'s (i ≤ j ≤ m + 1) see points of $C^*$. We have made several claims and skipped over important implementation issues in order to get the main idea of the algorithm across. Next, we fill in the missing parts and substantiate our claims.

**Lemma 3.1.** It is possible to compute the fusion $S$ of $S_1$ and $S_2$ in

$$\Theta(n_1/\tau_1 + n_2/\tau_2 + 1/k\tau_2 + \log n_1 + n_2)$$

time.

**Proof.** We restrict our attention to the task of fusing $S_1$ into $S_2$, the other case being similar. We have already shown that the start-up phase leads to a situation which satisfies the loop invariant, so it suffices to establish the correctness of the inner loop past $a_0$. In case (i) we know that $a_i$ lies in R (actually is its interior) and sees a point of $C^*$, so invariant (B) is satisfied. How about (A)? We made the claim that $a_0$, $a_1$, $a_2$, all see points of $C^*$. But actually the negation of (i) for $a_0$, $a_1$, $a_2$, is not strong enough to reach the necessary conclusion about what $a_0$, $a_1$, $a_2$, must see. Any of these points (if they exist) either sees $C^*$ or lies outside of R. Why should lying outside R imply seeing $C^*$? Suppose that, for some l, $l < i < j$, $a_i$ lies in region R distinct from R (like $a_{i+1}$ in Figure 3.2.1) but also sees $C^*$. We derive a contradiction. Let $\mathcal{A}$ denote the directed portion of $C^*$, as we go from p to $a_i$ clockwise, and let $\mathcal{A}$ be the point of $C^*$ seen by $\mathcal{A}$. The union of $\mathcal{A}$, the chords $p_j$ and $a_j$, and the portion $\mathcal{B}$ of $C^*$ running clockwise (with respect to C) from $p_j$ to $\mathcal{A}$ forms the boundary of a simple polygon $\mathcal{B}$ (Figure 3.3). Since the dual graph of a submap is a tree, there is a unique exit chord ab of $\mathcal{B}$ (that leads to $\mathcal{B}$' (note that ab need not be an exit chord of R, since there might be one or even several regions separating R from $\mathcal{B}$). Because $\mathcal{A}$ runs from $\mathcal{B}$ to $\mathcal{B}$ it passes through one of the chord endpoints, say, a. Let $\mathcal{A}$ be the point of $\mathcal{A}$ first encountered as we go from a to b along the chord. Note that ab cuts through $\mathcal{A}$, so $\mathcal{A}$ is well defined. The points a and $\mathcal{A}$ see each other with respect to $C^*$, and $\mathcal{A}$ lies in $a_j$, for some h between k and l inclusive.
Because, in clockwise order around $\partial C$, the point $a'$ is leaving $R$ locally, it cannot be equal to $p$. Therefore, the inequality $i < j$ implies that case (ii) must have already occurred when the running variable $j$ was assigned to some integer between $k$ and $l$, which is impossible.

Having shown that the loop invariant remains satisfied through case (i), we must do the same with case (ii). Let $a'$ now denote the directed portion of $\partial C_i$ as we go from $p$ to $a'$ clockwise. The new assignment of $p$ is the last point of $a'$, distinct from $p$, that sees an endpoint of an exit chord of $R$. Certainly, the new assignment of the current region satisfies invariant (B). Figure 3.2.2 shows three candidate endpoints, with $a_1$ winning the contest. Turning now to invariant (A), we must prove our claim that the points of $\partial C$ seen by $a_0, a_1, \ldots, a_{n-1}$ all belong to $\partial C_i$. We omit the proof since it is identical to the previous one, with the role of $a$ now played by $p$.

What about termination? Obviously, the three cases rule out an infinite loop. Every time we fall in either of the two cases (i) or (ii) we determine more visibility information, so that all visibility relations are known from $a_0$ all the way to the current position of $p$. How about the last iteration, the one leading to case (iii)? We claim that all $a'_i (0 \leq i \leq m + 1)$ see points of $\partial C_i$. This follows from the proof of the last paragraph, which showed that if $a_i$ sees $\partial C_i$, then either $a_i$ lies in $R$ (case (i)) or it does not, but then, we must fall in case (ii) after leaving $p$ but upon or prior to getting to $a$. The proof of correctness is now complete.

Let us now analyze the complexity of the algorithm. To test whether $a_i$ lies in $\mathbb{Z}$ can be done in $O(f(\sqrt{y}))$ time by using the ray-shooters for each arc that bounds $R$: first we find which point of an arc is hit by a ray of light shot from $a_j$ in its assigned chord direction. If there is no hit on any arc, $a_j$ is not in $\mathbb{R}$. Else, let $s$ be the first point hit by the ray over all the arcs of $R$. Whether $a_j$ lies in $R$ or not can be directly inferred from the local orientation of the hit at $s$ and which side of the double boundary is hit. This is because, as we should recall, arcs structures encode on which side(s) of the double boundary the arcs lie. If $a_j$ lies in $R$, then $s$ is the point of $\partial C_i$ seen by $a_j$ with respect to $C_i$. Next we use local shooting within $S_i$ to determine the point $t$ of $\partial C_i$ hit by a ray of light shot from $a_j$ in its assigned direction. Note that most often (i.e., when $0 < j \leq m$) $a_j$ is the endpoint of a chord of $S_i$, so $t$ is just the other endpoint of the chord. Now that we know which points of $\partial C_i$ and $\partial C_j$, the point $a_j$ can see, we can immediately derive the point of $\partial C$ that it sees and, from there, decide whether we are in case (i). The test takes $O(f(y)/\sqrt{y}) + O(f(\sqrt{y}))$ time (since $f$ is nondecreasing and $\sqrt{y} < y$). This cost also includes the start-up phase.

Regarding (ii), we must assess how fast we can find the point of $\partial C$ that is seen by an endpoint $a_i$ of a given exit chord $a_i$ of $R$. Actually, we must find that point only if it belongs to $A_i$. So, we can shoot a ray of light from $a_i$ toward $A_i$ in the appropriate direction and see what happens, which takes $O(f(y))$ time. If we do not get a hit, or if the hit does not lie on $a_i$, or if it occurs before $p$ along $A_i$, or if $a_i$ does not have the proper orientation which lets us see $A_i$ without the other side of the double boundary of $A_i$ interfering (a constant-time test), then the endpoint can be disqualified. Otherwise, we must find whether the point $s$ of $A_i$ hit by the ray of light can see $a$ with respect to $C$: the problem here is that other arcs $A_j (i \neq j)$ might get in the way. Using local shooting, shoot a ray of light from $s$ toward $a$. We aim at the $s$-ray from $s$ since we have already ruled out the "cones" prevent it from seeing $a$. The point $t$ hit by the ray is it shooting from $s$ to $t$ passes through $a$, then $s$ and $a$ see $C$ and we fall in case (ii), else we know that case (ii) exists $A_j$ and the endpoint $a_j$ of the chord $a_i$ (although it may other endpoints of exit chords in $R$). This shows that testi es.

We thus have shown that every elementary test $i$, $O(f(y))$ time. At each such test we advance through the a pair of visible points in $\partial C$, one of which is an endpoint. These events are never duplicated because we always $\pi_i$. Therefore, to discover all the chords of the fusion $S_i$, $O(mf(y))$ time, where $m$ is the total number of arcs and $C_i$ by Lemma 2.3, is in $O(n^{1/2} + m^{1/2} + 1)$. Notice that $C_i$ included in the fusion $S_i$ we not only have the newly discover $\partial C_i$ and $\partial C_j$ as well as the old chords of $S_i$ and $S_j$ that points with respect to $C$, but we also have all the null-ler as well as the chords incident upon the vertices of $\partial C$ result of the vertex $C_i \cap C_j$, this last category is where null-ler.

After fusing $S_i$ (resp. $S_j$) into $S_i$ (resp. $S_j$), we have all the $S$ and we can set it up in normal form quite easily. In order vertex enumerations, let us sort the endpoints of these chords done by sorting the names of the edges of $R$ on which time sorting the endpoints falling within the same edges by co.

This allows us to set up the required arc-sequence table also be used instead of sorting but this step is not the 
With this information it is now straightforward to set up $S_j$, along with all the necessary arc-structures and their re-
submap might not be conformal we dispense with the tree 
point. Very conservatively, all this work can be done in $O(n^{1/2} + n^{1/4} + 1) O(n^{1/2} + n^{1/4} + 1)$. 
Putting everything together, we derive the upper bound

$$O(n^{1/2} + n^{1/4} + 1) + \log(n) + n$$

on the time needed to complete the fusion of $S_i$ and $S_j$.

**Remarks.**

1. Since the chords of $S$ reflect the visibility of $S_i$ and $S_j$, they need not be incident upon any vertex of augmented maps and submaps.

2. It is possible to simplify the fusion procedures in various expense of a slightly more complicated proof of correctness.
under around $\partial C$, the point $a$ is leaving $R$ locally, it cannot
be the inequality $i < j$. We imply that case (ii) must have already
been variable $j$ was assigned to some integer between $k$ and

the loop invariant remains satisfied through case (i), we

case (ii). Let $A'$ now denote the directed portion of $A$, 

lookwise. The new assignment of $p$ is the last point of $A'$, 

end point of an exit chord of $R$. Certainly, the new

region satisfies invariant (B). Figure 3.2.2 shows three

the point $q$, winning the contest. Turning now to invariant (A), we

the points of $C$ seen by $a_0, \ldots, a_{j-1}$ all belong to $C$. It is
to the previous one, with the role of $a_j$ now

Evidently, the three cases rule out an infinite

in either of the two cases (i) or (ii) we determine more

that all visibility relations are known from $a_0$ all

point of an arc is hit by a ray of light shot from $a_j$

section. If there is no hit on any arc, $a_j$ is not in R. Else,

by the ray over all the arcs of $R$. Whether $a_j$ lies in $R$

seed from the local orientation of the hit at $a$ and which

hit is this. Because, as we should recall, arc-

side(s) of the double boundary the arcs lie. If $a$ lies

of $C$ seen by $a_j$ with respect to $C_j$. $a_j$ we use local

test. We see if the point $t$ of $C$, hit by a ray of light shot

Note that most often (i.e., when $0 < j \leq m$) $a_j$ is the

it seen and, from there, decide whether we are in case

case (ii) of (i + $f(t_j)$) = $O(f(t_j))$ time (since $f$ is nonincreasing

include the start-up phase.

we can find the point of $C$ that is seen

exit chord $ab$ of $R$. Actually, we must find that point

we can shoot a ray of light from a toward $A_j$ in the

see what happens, which takes $O(f(t_j))$ time. If we do

it does not lie on $ab$ or, if it occurs before $p$ along $A_j$, or

operating orientation which lets $a$ see $A_j$ without the other

of $A_j$ interfering (a constant-time test), then the

end. Otherwise, we must find whether the point $s$ of $A_j$

see $a$ with respect to $C$: the problem here is that other

Triangulating a Simple Polygon in Linear Time

arcs $A_i$ ($i \neq j$) might get in the way. Using local shooting in $S_i$, however, we can

shoot a ray of light from $s$ toward $a$. We aim in the natural shooting direction

from $s$ since we have already ruled out that the "companion" point of $s$ should prevent it from seeing $a$. The point $t$ hit by the ray is found in $O(f(t_j))$ time. If shooting from $s$ to $t$ passes through $a$, then $s$ and $a$ see each other with respect to $C$ and we fall in case (ii), else we know that case (ii) cannot occur with respect to $A_j$ and the endpoint $a$ of the chord $ab$ (although it might occur with respect to other endpoints of exit chords in $R$). This shows that testing case (ii) takes $O(f(t_j))$ time.

We thus have shown that every elementary test (i), (ii) can be performed in $O(f(t_j))$ time. At each such test we advance through the list of $A_j$'s or we report a pair of visible points in $C$, one of which is an endpoint of an exit chord of $S_i$. These reports are never duplicated because we always move forward among the $A_j$'s. Therefore, to discover all the chords of the fusion $S$ of $S_i$ and $S_j$ takes time $O(nf(t_j))$ time, where $n$ is the total number of arcs and exit chords in $S_i$ and $S_j$. By Lemma 2.3, $m \leq O(nf(t_j) + n_j^2 + 1)$. Note that among the chords to be included in the fusion $S$, we not only have the newly discovered chords that connect $C_i$ and $C_j$ as well as the old chords of $S_i$ and $S_j$ that still form visible pairs of points with respect to $C$, but we also have all the null-length chords of $S_i$ and $S_j$ as well as the chords incident upon the vertices of $C$, resulting from the duplication of the vertex $C_i \cap C_j$; this last category is where null-length chords originate.

After fusing $S_j$ (resp. $S_j$) into $S_i$ (resp. $S_j$), we have all the chords of the submap $S$ and we can set it up in normal form quite easily. In order to allow for canonical vertex enumerations, let us sort the endpoints of these chords along $C$, which is done by sorting the names of the edges of $P$ on which these arcs abut, and then sorting the endpoints falling within the same edges by considering $y$-coordinates. This allows us to set up the required arc-sequence table. Note that merging can also be used instead of sorting but this step is not the dominant cost, anyway. With this information it is now straightforward to set up the tree of the submap $S$ along with all the necessary arc-structures and their relevant pointers. Since the submap might not be conformal we dispense with the tree decomposition at this point. Very conservatively, all this work can be done in time

\[ O(n_i f(t_j) + n_j^2 + 1 + \log(n_j) + n_j)) \]

Putting everything together, we derive the upper bound of

\[ O(n_i f(t_j) + n_j^2 + 1 + \log(n_i) + n_j)) \]

on the time needed to complete the fusion of $S_i$ and $S_j$.}

\[ \square \]

Remarks. 1. Since the chords of $S$ reflect the visibility of the chord endpoints of $S_i$ and $S_j$, they need not be incident upon any vertex of $C$, hence the notion of augmented maps and submaps.

2. It is possible to simplify the fusion procedure in various ways, albeit at the expense of a slightly more complicated proof of correctness. For one thing, the
start-up phase is not strictly required and with a bit of care can be integrated within the main loop. Also, the two fusing passes can be unified into a single one (toward that end, we might want to catch the first occurrence of case (ii) and not the last one, for example).

3.2. Restoring Conformality

As we said earlier, there is no reason to believe that the fusion S should be conformal. Things can never be too bad, however. Indeed, let A₁, A₂, ... be the arcs of a region R of S in counterclockwise order. It is clear that each Aᵢ belongs to ∂C₁ or ∂C₂ but not both. So, we can partition the sequence of arcs into runs, B₁, B₂, ..., meaning that B₁ = A₁, A₂, ..., Aᵣ₋₁ is a maximal subsequence of arcs from either ∂C₁ or ∂C₂ (but not both). In the definition of maximal, we regard Aᵣ, Aᵣ₊₁, ..., as a circular sequence. Because any exit chord endpoint of Sᵢ is still an endpoint of a chord in S and, with the possible exception of the chords incident upon αᵢ₀ or αᵢ₋₁, every chord of S that connects two points of ∂C must also be a chord of Sᵢ, it follows by conformity that a run associated with ∂Cᵢ cannot have more than four exit chords in its midst, not counting the new chords incident upon αᵢ₀ or αᵢ₋₁. Therefore, a run cannot have more than a constant number of arcs. On the other hand, it follows from Lemma 2.1 that there are at most two runs. Why is that so? The lemma says that if we walk along ∂C clockwise we will in effect traverse, among other things, the boundary of R counterclockwise (except for the exit chords). If we begin our walk at one of the two points of ∂C corresponding to the vertex C₁ ∩ C₂, we first exhaust, say, ∂C₁, and then ∂C₂. Therefore, the counterclockwise traversal of the boundary of R must exhaust first the runs Bᵢ contributed by Sᵢ and then the runs contributed by Sᵢ₊₁. Obviously, this leaves only the possibility of having at most one run of each type, and hence a total of at most two runs. The conclusion to draw is that, although not necessarily conformal, the submap S has no region with more than a bounded number of arcs. If S is not conformal we must now reduce the number of arcs per region to four or less by adding new chords into S. To discover these chords we need the ability to check whether two arcs or subarcs of the same region can "see" each other (Lemma 3.2). We also need to show that the desired chords do exist (Lemma 3.3).

Lemma 3.2. Given two arcs A₁, A₂ of the same region of S, assume that they have a pair of mutually visible points, one of which is a vertex of ∂C meaning that, say, A₁ contains a vertex v which is also a vertex of ∂C and is such that the point of ∂C seen by v lies in A₂. Then we can find a point of A₁ (not necessarily a vertex of ∂C) that sees A₁ in time O(f(βlogγ₁ + log γ₂)).

Proof. To begin, observe that A₁ and A₂ are arcs or subarcs of either S₁ or S₂, but cannot overlap both ∂C₁ and ∂C₂. The reason is that all chord endpoints in S₁ and S₂ are still chord endpoints in S (perhaps with different chords) and that we added chords incident upon the vertices of ∂C resulting from the duplication of the vertex C₁ ∩ C₂. Therefore, because of the bounded region, it is still possible to do local shooting within any r and f is nondecreasing this takes O(f(β)) time. Thus, we have time O(f(γ)) whether a given vertex of A₁ qualifies as the point it is on the desired arc or is the companion of already discussed how local shooting can decide which we so we will not make further mention of that minor details do not check all the vertices of A₁ because there might just instead, we need to do some kind of binary search among

For that purpose we invoke the arc-cutter associated containing A₁, which allows us to subdivide A₁ into a sequence of subarcs Sᵢ, where l = 1 if A₁ ⊂ ∂C₁ and l = 2 if A₁ ⊂ ∂C₂. Except for at most one double attached to the endpoints of A₁ (which we ignore), for each normal-form m(γ) granular conformal submap S of T, in turn, stopping as soon as we find a good vertex of the normal-form representation of S, we provide us with T of the submap, we are able to check the candidate in O(f(γ)log(γ) + log γ₂) time, provided that the following in O(f(γ)) time: given a chord ab of S, either determine a and b, or has no point that sees A₁, where aᵢ and bᵢ denote the k of a and b. Note that δ₁ is a superset of a with twice the closer because the arc-cutter oracle guarantees that a does an equivalent. First we show how such a test can be used as a test. Then we explain how to implement the test and why cases. We begin by applying this test with respect to the chord root of the tree T (corresponding to the hierarchical decomposition, as claimed, either we terminate with a positive answer or a₁ or a₂, say, a₁, such that a₁ = a₂ is empty or has no positive answer, we find the child of the root that corresponds to a₁ or possibly takes us to the bottom of the tree. Note that the branch to at each step is trivial once we have identified So, we can perform the test just as stated above without reflect the current status of the ever-shrinking set of canonical vertices in the region of the tree is O(log(γ) + log γ₂), as claimed. Again, γ₁ ≤ γ₂ and that h is nondecreasing. Whenever we discover a successful candidate point, the be stopped right before. What remains to be seen is why the corresponding region is the only one which can still process. Let us assume that the search ends up at a leaf. At the very
Triangles are required and with a bit of care can be integrated so, the two fusion phases can be unified into a single one the want to catch the first occurrence of case (ii) and not the.

There is no reason to believe that the fusion $S$ should be ever be too bad, however. Indeed, let $A_1, A_2, \ldots$ be the counterclockwise order. It is clear that each $A_i$ belongs both. So we can partition the sequence of arcs into runs, $B_1 = A_1, A_2, \ldots, A_{i-1}, A_i = A_{i+1}, \ldots, A_{20}$ is a maximal subsequence of $C$ (but not both). In the definition of maximal, we regard sequence. Because any exit chord endpoint of $S_i$ is still an and, with the possible exception of the chords incident to $S$ that connects two points of $\partial R$ it also a chordal boundary that a run associated with $\partial R$ cannot have more of its midpoints counting the new chords incident upon $a_0$ cannot have more than a constant number of arcs. On the other hand, the boundary of $R$ counterclockwise (except for the walk at one of the two points of $\partial C$ corresponding to go, say, $C_1$ and then $C_2$. Therefore, the $A_i$s of the boundary of $R$ must exhaust first the runs $B_i$ in the runs contributed by $S$. Obviously, this leaves only at most one run of each type, and hence a total of at least one of $\partial R$ to draw is that, although not necessarily conformal, $C_2$. Obvious, this leaves only at most one run of each type, and a total of at least one of $\partial R$ to draw is that, although not necessarily conformal, $C_2$. Obvious, this leaves only at most one run of each type, and a total of at least one of $\partial R$ to draw is that, although not necessarily conformal, $C_2$. Obvious, this leaves only at most one run of each type, and a total of at least one of $\partial R$ to draw is that, although not necessarily conformal, $C_2$. Obvious, this leaves only at most one run of each type, and a total of at least one of $\partial R$ to draw is that, although not necessarily conformal, $C_2$. Obvious, this leaves only at most one run of each type, and a total of at least one of $\partial R$ to draw is that, although not necessarily conformal, $C_2$.

Let $A_1, A_2, \ldots, A_n$ be of the same region of $S$, assume that they are points, one of which is a vertex of $C$ meaning that, $C$ which is also a vertex of $C$ and is such that the point $x$. Then we find a point of $A_1$ (not necessarily a vertex of $C$).

Note that $A_1, A_2$ are arcs or subarcs of either $S_1$ or $S_2$, $\partial C_1$, and $\partial C_2$. The reason is that all chord endpoints in $\partial C$ resulting from the duplication of the vertex $C_1 \cap C_2$. Therefore, because of the bounded number of arcs per region, it is still possible to do local shooting within any region of $S$. Since $\gamma_1 \leq \gamma_2$ and $f$ is nondecreasing this takes $O(1)$ time. Thus, we can efficiently check, in time $O(f(\gamma_2))$, whether a given vertex of $A$ qualifies as the point $\gamma_2$ sought. (Again, we must be careful that local shooting reports edges of $\partial D$ and does not tell us if the point hit is on the desired arc or is the companion of a point of the arc. We already discussed how local checking can decide which way it is in constant time, so we will not make further mention of that minor difficulty.) Of course, we should not check all the vertices of $A_i$ because there might just be too many of them. Instead we need to do some kind of binary search among the vertices of $A_i$.

For that purpose we invoke the arc-cutter associated with the arc of $S_1$ or $S_2$ containing $A_i$, which allows us to subdivide $A_i$ into at most $\log_2 f(\gamma_2)$ subarcs, with $l = 1$ if $A_i \subseteq \partial C_1$ and $l = 2 f(\gamma_2) \leq \partial C_2$. Except for at most two single-edge subarcs attached to the endpoints of $A_i$ (which we ignore), for each subarc $a$ we have a normal-form $\mathcal{H}(a)$-grahal conformal submap $S_a$ of $V_i$. We search each subarc in turn, stopping as soon as we find a good vertex or point, if ever. Since the normal-form representation of $S_a$ provides us with the tree decomposition $T$ of the submap, we are able to check the candidacy of $a$ in its entirety in $O(1) + \log_2 f(\gamma_2)$ time, provided that the following test can be performed in $O(1)$ time: given a chord $ab$ of $S_a$, either determine that $a$ or $b$ is a point of $a$ and see $A_i$ with respect to $C$, or find some $i \in [1, 2]$ such that $a \cap a_i$ is empty or has no point that sees $A_i$, where $a_i$ and $a_{i-1}$ denote the two points of $a$ between $a$ and $b$. Note that $a$ is a superset of $a$ twice the number of vertices (not fewer because the arc-cutter or-cutter guarantees that $a$ does not double-back around an endpoint). First we show how such a test can be used to check the candidacy of $a$. Then we explain how to implement the test and why it covers all possible cases.

We begin by applying this test with respect to the chord corresponding to the root of the tree $T$ (corresponding to the hierarchical decomposition of $S_i$). Then, as claimed, either we terminate with a positive answer or else we identify one of $a_1$ or $a_2$, say, $a_1$, such that $a_i \cap a_1$ is empty or has no point that sees $A_i$. In that case, we find the child of the root that corresponds to $a_1$ and we iterate on this process from that node. This leads us to termination at some internal node of $T$ or perhaps takes us to the bottom of the tree. Note that determining which node to branch to at each step is trivial once we have identified the $a_i$ to be rejected. (So, we can perform the test just as stated above without having to "resize" $a$ to reflect the current status of the over-shrinking set of candidates.) If we reach a leaf, we examine each vertex of the region associated with it and, among those belonging to $a$, we check whether any of them can see $a_i$. Since there are only $O(\log_2 f)$ vertices in the region and the depth of the tree is $O(\log_2 f)$ the running time of the algorithm is $O(\log_2 f + \log_2 f)$, that is claimed. Again we use the fact that $\gamma_1 \leq \gamma_2$ and that $f$ is nondecreasing.

Whenever we discover a successful candidate point, the search can obviously be stopped right there. What remains to be seen is why upon reaching a leaf the corresponding region is the only one which can still provide the desired answer. Let us assume that the search ends up at a leaf. At the very beginning, let us say
that each point of \( \mathcal{B} \) is a candidate. Every time we branch down the tree we limit the candidacy to those points of \( \mathcal{B} \) in the regions of \( S \) associated with the leaves of the subtree which we enter. At the end, the remaining candidates are the vertices in the region associated with the leaf where the tree search ends. This proves the correctness of the procedure. So, to summarize, if the basic test stated earlier can be performed in \( O(1/\epsilon^2) \) time, then we can solve the entire problem in \( O(1/\epsilon^2) \) time, which proves the lemma.

We now show how to perform the test and why it is sound, i.e., covers all cases. Removing \( \mathcal{B} \) from the spherical plane leaves two open regions, each polygonal and homeomorphic to a disk. One of them is infinitesimally small; let \( D \) be the closure of the other one. It is important that \( D \) should be homeomorphic to a closed disk and not to a 2-sphere, so the interior of \( D \), and, more generally, of \( C \), should be understood as being very small but nonempty. Let \( c \) and \( d \) be the endpoints of \( a \) on \( \mathcal{B} \). Removing \( c \) and \( d \) from \( \mathcal{C} \) leaves \( a \) and a curve \( A \), both lying in \( D \), so we have set the stage for Lemma 2.4.

Figure 3.4 illustrates the correspondence: the snake on the left represents \( C \); the disk \( D \) corresponds to the outside of the portion of the snake between \( c \) and \( d \), while it is the inside of the circle on the right. The curve \( A \) runs along the snake clockwise from \( d \) to \( c \) and corresponds to the boundary of \( D \) part of the way. The subarc \( a \) runs clockwise from \( c \) to \( d \) and corresponds to \( B_1 \cup B_2 \).

Figure 3.4 shows the case where only one endpoint of \( a \) lies in \( B_1 \cup B_2 \), which corresponds to Figure 2.8.2. The reader will easily draw an example matching the case of Figure 2.8.1, where both \( a \) and \( b \) lie in \( a = B_1 \cup B_2 \).

To compute \( a \) and \( b \) (if defined) can be done by local shooting in the region of \( S \) incident upon \( A_1 \) and \( A_2 \), which takes \( O(1/\epsilon)^2 \) time. Note that no shooting is needed for \( a \) or \( b \) if the point in question does not lie in \( a \). If ever \( a \) (resp. \( b \)) is a point of \( a \) and \( d \) (resp. \( b \)) belongs to \( A_2 \), then obviously we are done and successful in our search, so we can assume that neither conjunction holds. But, in that case, \( A_2 \) lies entirely within one of the connected components of the curve \( A \) after it has been cut up by removing \( a \) and \( b \) (whichever exists). Therefore, by Lemma 2.4, \( A_2 \) must be shielded from some \( B_2 \), which means that it cannot be connected to \( B_1 \) without crossing \( a \) or \( b \). Furthermore, we know that \( B_1 \) can be identified in constant time. The key observation now is that \( B_2 \) coincides precisely with one of \( a \cup a_1 \) or \( a \cup a_2 \), say, the first one. It follows that no point of \( a \cup a_1 \) can see \( A_2 \), and the test is completed.

**Lemma 3.3.** Let \( A_1, \ldots, A_k \) be the clockwise circular region of \( S \), if \( k > 4 \), then there exist \( i, j \), such that

\[
(i) \quad i - j \neq -1, 0, 1 (\text{mod} \; k) \quad \text{and} \quad (ii) \quad A_i \text{ has a vertex of } C \text{ that sees } A_j \text{ with respect to } C.
\]

**Proof.** Recall that the region is associated with a subtree \( T \). If \( S \) contains chords incident upon no vertex of \( C \), these chords as well and speak of the tree of the augmented tree. Let us add to the subtree in question the edges that \( C \) has. With respect to this augmented subtree, \( c \) is associated with a distinct node of degree 1 (but the case where some of these exit chords might be of zero le minimal tree of these particular degree-1 nodes (i.e., the edges that join these nodes together), and for simplicity to it by ignoring nodes of degree 2. Now embed this curve by a simple closed curve that connects together (Figure 3.5). By using an embedding that preserves the local area around the nodes, the area inside that closed curve is preserved and corresponds to a distinct arc \( A_i \). Because there are \( k \) arcs and the maximum node-degree is \( 4 \), it is immediate that some edge must be incident upon two faces associated with \( A_i \), \( i \neq j \), \( i - j \neq -1, 0, 1 \) (mod \( k \)). (For a simple proof, try all possible pairs of adjacent nodes of degree 1 and observe that the property remains true.) Of all the chords in the region only the exit chord upon at least one vertex of \( C \). It follows that the edge to one or several chords of the original, nonaugmented \( A_i \).

Equipped with the two previous lemmas, making easy. Recall that although \( S \) might not be conformal in regions has more than a bounded number of arcs. For

![Figure 3.4](image)

![Figure 3.5](image)
a candidate. Every time we branch down the tree we limit points of $\partial A$ in the regions of $S_i$ associated with the leaves to enter, at the end, the remaining candidates are the vertices $A_i$ with the leaf where the tree search ends. This proves the lemma. So, to summarize, if the basic test stated earlier can perform the test and why it is sound, i.e., covers all cases. Spherical plane leaves two open regions, each polygonal a disk. One of them is infinitesimally small; let $D$ be the 2-sphere, so the interior of $A$ and, more generally, of $C$, as being very small but nonempty. Let $c$ and $d$ be the two curves from $C$ and $A$ outside of $D$ part of the way. The subarc $y$ runs clockwise around $D_1$ or $D_2$. Figure 3.3 shows the case where only $D_1$ or $D_2$, which corresponds to Figure 2.8.2. The reader note matching the case of Figure 2.8.1, where both $c$ and $y$ (if defined) can be done by local shooting in the region end $A_2$. which takes $O(t/12)$ time. Note that no shooting point in question does not lie in $S$. If ever a (resp. b) is $S$, $b'$ belongs to $A_1$, then obviously we are done and so we can assume that neither conjunction holds. But, in $\partial A$ within one of the connected components of the curve $A$ by removing $a'$ and $b'$ (whichever exists). Therefore, by $A_i$, which means that it cannot be crossing $ab$ or $A$. Furthermore, we know that $B_i$ can be $A$. The key observation now is that $B_i$ coincides precisely $S_2$, say, the first one. It follows that no point of $\partial A_i$ is completed.

**Lemma 3.3.** Let $A_1, \ldots, A_k$ be the clockwise circular sequence of arcs around a region of $S$. If $k > 4$, then there exist $i, j$, such that

(i) $i-j \not= -1, 0, 1 \pmod {k}$ and

(ii) $A_i$ has a vertex of $C_i$ that sees $A_j$ (with respect to $C$).

**Proof.** Recall that the region is associated with a subtree of the visibility tree of $C$. If $S$ contains chords incident upon no vertices of $C$, then we must include these chords as well and speak of the tree of the augmented visibility map of $C$. Let us add to the subtree in question the edges that connect it to the rest of the visibility tree. With respect to this augmented subtree, each exit chord of the region is associated with a distinct node of degree 1 (but the converse may not be true). Note that some of these exit chords might be of zero length. Consider the Steiner minimal tree of these particular degree-1 nodes (i.e., the smallest connected set of edges that connect these nodes together), and for simplicity form a tree homomorphic to it by ignoring nodes of degree 2. Now embed this new tree in the plane and enclose it by a simple closed curve that connects together all its degree-1 nodes. (Fig. 3.5) By using an embedding that preserves the local orientation of the edges around the nodes, the area inside that closed curve is partitioned into $k$ faces, each corresponding to a distinct arc $A_i$. Because there are at least five degree-1 nodes and the maximum node-degree is 4, it is immediate that at least one edge of the theorem must be incident upon two faces associated with $A_i$ and $A_j$, respectively, where $i-j \not= -1, 0, 1 \pmod {k}$ (For a simple proof, try all possible cases with five nodes of degree 1 and observe that the property remains true with the addition of more nodes.) Of all the chords in the region only the exit-chords may fail to be incident upon at least one vertex of $C$. It follows that the edge in question corresponds to one or several chords of the original, nonaugmented $V(C)$ that connect $A_i$ and $A_j$.

Equipped with the two previous lemmas, making $S$ conformal is now quite easy. Recall that although $S$ might not be conformal right after fusion, none of its regions has more than a bounded number of arcs. For any region with more than
four arcs, let us apply Lemma 3.2 to every pair of nonconsecutive arcs until we find a chord which we can add to S. We iterate on this process until no region has more than four arcs. Note that although S keeps changing, Lemma 3.2 always holds since region arcs can only become smaller. Lemma 3.3 tells us that this chord addition process will not stop until S becomes conformal. Since the total number of arcs in S is \(O(n_1/\gamma_1 + n_2/\gamma_2 + 1)\), we conclude:

**Lemma 3.4.** The submap S can be made conformal in time

\[
O(n_1/\gamma_1 + n_2/\gamma_2 + 1) f(\gamma_1, \gamma_2, \gamma_3) \log \gamma_3.
\]

### 3.3. Maintaining Granularity

Since by making S conformal we did not remove any exit chord, it is still the case that, as observed in the proof of Lemma 3.2, no arc has more than \(\gamma_2\) edges. Therefore, S is conformal and \(\gamma_2\)-semigranular. We must now check whether the second criterion for \(\gamma_2\)-granularity holds. This criterion says that contracting any edge of the submap tree that is incident upon at least one node of degree less than 3 produces a new node whose weight exceeds \(\gamma_2\). This is very easy to enforce: if an edge does not pass the test, we just contract it by removing its corresponding exit chord (and those endpoints that are not vertices of \(\delta C\)). Note that this will not cause a violation of the first criterion, since the size of all the arcs will always remain within \(\gamma_2\). Similarly, the removal keeps the submap conformal. We process each exit chord in turn and check whether it is removable. Chords need be processed only once since the removals cannot make any chord removable if it was not already so before. Therefore, \(\gamma_2\)-granularity, and more generally \(\gamma\)-granularity, for any \(\gamma \geq \gamma_2\), can be enforced in this nondeterministic fashion in time linear in the size of the submap tree, that is, \(O(n_1 + n_2 + 1)\). We can now put S in normal form, which includes computing its tree decomposition. As we discussed earlier, this can be done very simply in time \(O(n_1/\gamma_1 + n_2/\gamma_2 + 1) \log n_1 + n_2)\). With the inconsequential assumption that \(\gamma_1 = O(n_1 + n_2)\), we derive the following result from Lemmas 3.1 and 3.4:

**Lemma 3.5.** Let \(C_1\) and \(C_2\) be two polygonal curves of \(n_1\) and \(n_2\) vertices, respectively, whose union forms a connected vertex-to-vertex piece of the input (simple and nonclosed) polygonal curve P. Suppose that we are given a normal-form \(\gamma\)-granular conformal submap of each V(C), where \(\gamma \leq \gamma_2\), along with the ray-shooting and arc-cutting oracles necessary for merging. Then, for any \(\gamma \geq \gamma_2\), it is possible to compute a normal-form \(\gamma\)-granular conformal submap of V(C) in time

\[
O(n_1/\gamma_1 + n_2/\gamma_2 + 1) f(\gamma_1, \gamma_2, \gamma_3) \log(n_1 + n_2)).
\]

### 3.4. Implementing the Oracles

Of the two oracles defined earlier, the ray-shooter is the more challenging to implement, the reason being that it addresses the key issue in the triangulation business, which is the discovery of new chords. The arc-oracle, using the divide-and-conquer structure of the up-phase oracle, works. Since a better understanding of the up-phase is necessary, we postpone the discussion of its implementation till a later section.
Lemma 3.2 to every pair of nonconsecutive arcs until we can add to $S$. We iterate this process until no region $R$ can be made conformal in time $n_1+y_1+1)$, we conclude: 

$$n_1+y_1+1) + (O(\log n_1 + \log_2 n_2))$$

To avoid removing any existing chord, it is still the case that removing any arc results in at least one node of degree less than $\gamma_2$. This is very easy to enforce: if $\gamma_2$ is too low, just add it by removing its corresponding point at most vertices of $\gamma_2$. Note that this will be the first criterion, since the size of all the arcs is always $\gamma_2$. The removal keeps the subtree conformal. We proceed and check whether it is removable. Chords need be removed that are not removable, although the removal cannot change any chord. Therefore, $\gamma_2$-granularity, and more generally $\gamma$-granularity, can be enforced in this nondeterministic fashion of the subtree tree, that is, $O(n_1+y_1+1)$. We now see form, which includes computing its tree decomposition, that this can be done very simply in time $O(n_1+y_1+1)$. With the inconsequential assumption that the following result from Lemma 3.1 and 3.4:

Suppose that we are given a normal-form map of each $V(C)_i$, where $\gamma_1 \leq \gamma_2$, along with the ray-shooters necessary for merging. Then, for any $\gamma \geq \gamma_1$, it is $O(\log n_1 + n_2))$.

Not earlier, the ray-shooters are the more challenging to implement that it addresses the key issue in the triangulation business, which is the discovery of new chords. The arc-cutter is implemented by using the divide-and-conquer structure of the up-phase of the visibility algorithm.

Since a better understanding of the up-phase is necessary to understand how oracle works, we postpone the discussion of its implementation a little. Turning our attention to the ray-shooting oracle, it might appear at first that fast planar point location should be the answer. But traditional methods, e.g., [9] and [19], are inadequate for several reasons, the most crucial of which is their inability to support merging in sublinear time. We turn this problem around by exploiting further the approximation scheme provided by the concept of granularity.

Let $C$ be a connected vertex-to-vertex piece of the input polygonal curve $P$ and let $m$ be its number of vertices. Let $S$ be a normal-form $\gamma$-granular conformal submap of $V(C)$. So far, we have focused mostly on the tree structure of $S$. But let us now regard $S$ as a planar graph. For this purpose, we must temporarily forget the fact that $C$ has been given a double boundary. We define $S^*$ to be the planar subdivision obtained by taking $S$ and making every vertex (vertices) of $\gamma$-granularity) a vertex on both sides of the double boundary, whose thickness is now null. As a result, the edges of $S^*$ might be smaller than those of $S$, but unlike $S$, no edge of $S^*$ is of zero length. More important, each face of $S^*$ coincides exactly with a distinct region of $S$, except for the fact that it might have many more vertices incident upon it. Indeed, a region's only vertices are the endpoints of its own exit chords along with some vertices of $\gamma$-granularity, whereas the vertices of a face include all of the above plus all the chord endpoints that abut on the corresponding region from the outside. Notice that since the notion of double boundary is lost, a face might have dangling edges or edges incident upon it on both sides. There are several examples of this in Fig. 3.8, which shows the subdivision $S^*$ corresponding to the submap of Fig. 2.8. Note also that the correspondence face-regions is not surjective because empty regions have no associated faces. Besides being planar, the graph $S^*$ has two remarkable properties:

(i) From Lemma 2.3, we know that it has $O(m^2)$ faces, which is much simpler than the number of vertices (when $\gamma$ is large).
(ii) Although a given face might be very complex (i.e., incident upon many edges) its number of non-collinear edges is small, i.e., $O(1)$.

These two features allow us to implement an efficient ray-shooting oracle.

Fig. 3.6
Let $G$ be the dual graph of $S^*$, that is, the graph obtained by associating a distinct node with each face of $S^*$ and connecting two nodes if and only if they are distinct and their corresponding faces share a common edge. It is a classical result of graph theory that $G$ is planar. How hard is it to compute $G$, say, in the form of adjacency lists? Two faces are adjacent if and only if either they share a chord or one of them has a chord endpoint that sits on a nonnull length arc of the region associated with the other face. The first type of adjacencies can be detected immediately from $S$. The latter can be done by double identification, as discussed in Section 2.4, followed by sorting along $C$, which takes $O(\mu \log \mu)$ time, where $\mu = O(m^{1/2} + 1)$ is the number of nodes in $G$. It can also be done faster by merging chord endpoints along both sides of $IC$.

If $\mu = 1$, then ray-shooting can be done trivially in $O(\mu)$ time, so let us assume that $\mu > 1$. We show that after $O(\mu \log \mu)$ preprocessing we can do ray-shooting in $O(\mu^{1/2})$ time. The planarity of $G$ works wonders for us. The first payoff is that the number of edges is at most $3\mu - 6$. The second reward is that we can apply the linear-time algorithm of Lipton and Tarjan [21] to find a good separator. This partitions the nodes of $G$ into three subsets $A$, $B$, $D$, such that

(i) no edge joins a node of $A$ with a node of $B$,
(ii) neither $A$ nor $B$ contains more than $2\mu/3$ nodes, and
(iii) $D$ contains at most $\sqrt{2\mu}$ nodes.

Let $G_A$ (resp. $G_B$) be the graph obtained by keeping only the nodes of $A$ (resp. $B$) and the edges of $G$ that join only nodes of $A$ (resp. $B$). We repeat the procedure over with respect to each of $G_A$ and $G_B$ and iterate in this fashion until none of the graphs have more than $\mu$ nodes, for some fixed $\delta (0 < \delta < 1)$. Let $D^*$ be the set of all separators, i.e., the union of all the $D$’s. We easily verify that $|D^*| = O(\mu^{1/2})$, provided that $\delta$ is chosen large enough; for example, $\delta = 1/2$ [22]. In $O(\mu \log \mu)$ time we can compute $D^*$ and partition the remaining nodes into subsets $D_1, D_2, \ldots$, each of which at most $\mu^{1/2}$, such that no path of $G$ can join two nodes in distinct subsets without passing through a node of $D^*$.

What is the utility of $D^*$ for ray-shooting? Take a vertical line passing to the right of all the vertices of $P$, and intersect it with the chords of the regions in $S$. This breaks up the line into segments, every one of which falls entirely within some region; to split up the line and identify the regions cut by each segment can be done by traversing $G$ and checking each chord for intersection with the vertical line. Since the regions cut correspond to nodes of $G$ lying on a path, sorting the intersections comes for free, and all the work can be done in $O(\mu)$ time. We now claim that ray-shooting toward $IC$ from any point can be done in $O(\mu^{1/2})$ time. Our first task is to shoot within each region that is dual to a node of $D^*$, using a naive algorithm which involves checking all the $O(\mu)$ edges of the region (and not the edges of the face, which might be much more numerous). Assume that the ray of light hits a point among the edges of the regions dual to the nodes of $D^*$. Let $R$ be the last region of $S$ traversed before the first hit. To identify $R$ can be done by double identification, followed by checking the local orientation of the hit. If $R$ is a region dual to a node of $D^*$, then the starting point of the ray lies in $R$ (otherwise an earlier hit would have been detected).

So, assume now that $R$ is dual to a node $s$ not in $D^*$. The double identification needed to find $R$ might require a large collection of collinear edges. Let $R'$ be the region in edge containing the point of $IC$ actually hit by the ray-shoot that we are looking for (Fig. 3.7). If $R$ and $R'$ are not the same, we can be connected by a horizontal line segment that avoids $D^*$. It follows that the node $s$ associated with $R'$ can be found from $s$ that avoids $D^*$. Consequently, $s$ and $w$ both lie in $R$, and from there, answer the ray-shooting query, but takes constant time since we know $R$, and then naively dual to nodes in $D_1$, which takes $O(\mu^{1/2})$ time. Return analysis, assume now that the ray of light hits no region. Then the ray-shooting takes place entirely within the region of a single $D_i$. To find out which one, we shoot toward $R$ which segment of the line is hit. This takes $O(\log \mu)$ time and can now identify the region $R$ immediately. The remaining unchanged. We conclude that ray-shooting can be done in $O(\mu^{1/2})$.
B. Chouette

graph of $S^*$, that is, the graph obtained by associating a
face of $S^*$ and connecting two nodes if and only if they
corresponding faces share a common edge. It is a classical
that $G$ is planar. How hard is it to compute $G$, say, in the
two cases are adjacent if and only if either they share a
as a chord endpoint that abuts on a nonnull length arc of
with the other face. The first type of adjacencies can be
taken from $S$. The latter can be done by double
identification, as
followed by sorting along $C$, which takes $O(\log n)$ time.
the number of nodes in $G$. It can also be done faster by
is along both sides of $IC$.
noting can be done trivially in $O(m)$ time, so let us assume
at after $O(\log m)$ preprocessing we can do ray-shooting
in a $D^*$ of $G$ works wonders for us. The first payoff is that
is that all $m_k = \delta$. The second reward is that we can apply
$\delta$ to Lipton and Tarjan [21] to find a good separator. This
$G$ into three subsets $A$, $B$, $D$, such that
node of $A$ with a node of $B$, contains more than $2\mu/3$ nodes, and
are $\delta/6$ nodes.

graph obtained by keeping only the nodes of $A$ (resp. $B$)
join only nodes of $A$ (resp. $B$). We repeat the procedure
$a$ of $G_a$ and $G_b$ and iterate in this fashion until none of
some fixed $\delta < \delta < 1$. Let $D^*$ be the
the graph of all the $D_s$. We easily verify that $|D^*| = O(\mu^3)$,
large enough, for example, $\delta = \frac{1}{4}$ [22]. In $O(\mu \log \mu)$ time
partition the remaining nodes into subsets $D_1$, $D_2$, etc.,
, such that no path of $G$ can join two nodes in distinct
through a node of $D^*$.

for ray-shooting? Take a vertical line passing to the
of $R$, and intersect it with the chords of the regions in $S$.
into segments, every one of which falls entirely within the
line and identify the regions cut by each segment can
and checking each chord for intersection with the vertical
not correspond to nodes of $G$ lying on a path, sorting
for free, and all the work can be done in $O(\mu \log \mu)$ time. We
saging toward $IC$ from any point can be done in $O(\mu \log \mu)$
shoot within each region that is dual to a node of $D^*$,
which involves checking all the $(\delta^3)$ edges of the region
face, which might be much more numerous). Assume
a point among the edges of the regions dual to the nodes
region of $S$ traversed before the first hit. To identify $R$
identification, followed by checking the local orientation
in dual to a node $v$ of $D^*$, then the starting point of the

Triangulating a Simple Polygon in Linear Time

513

ray lies in $R$ (otherwise an earlier hit would have been detected) and we are trivially
done.

So, assume now that $R$ is dual to a node $v$ not in $D^*$. Incidentally, note that
the double identification needed to find $R$ might require a binary search among
a large collection of collinear edges. Let $R'$ be the region incident upon the (region)
edge containing the point of $IC$ actually hit by the ray-shooting; this is the region
that we are looking for (Fig. 37). If $R$ and $R'$ are not the same then the two regions
can be connected by a horizontal line segment that avoids all the regions dual to
$D^*$. It follows that the node $v$ associated with $R'$ can be reached by a path in $G$
from $v$ that avoids $D^*$. Consequently, $v$ and $w$ both lie in the same $A$. We can
find $w$, and, from there, answer the ray-shooting query, by first finding $D_s$, which
takes constant time since we know $R$, and then naively checking all the regions
dual to nodes in $D_s$, which takes $O(\mu^2)$ time. Returning to our earlier case-
analysis, assume now that the ray of light hits no region dual to a node in $D^*$. Then
the ray-shooting takes place entirely within the regions dual to the nodes of a single $D_s$. To find out which one, we shoot toward the vertical line and find
which segment of the line is hit. This takes $O(\mu \log \mu)$ time by binary search. We
can now identify the region $R$ immediately. The remainder of the algorithm is
unchanged. We conclude that ray-shooting can be done in $O(\mu \log \mu)$ time if $\mu = 1$, and

Fig. 37
Lemma 3.6. Let \( C \) be a connected vertex-to-vertex piece of the input polygonal curve \( P \) and let \( m \) be its number of vertices. Let \( S \) be a normal-form \( \gamma \)-granular conformal submap of \( V(C) \). Then it is possible to preprocess \( S \) in \( O(m \log m \gamma + 1) \) time so that ray-shooting within \( S \) can be done in time \( O(m^3 \log m) \).

4. The Visibility Algorithm

Let \( P \) be a simple nonclosed polygonal curve with \( n \) vertices \( v_1, \ldots, v_n \). By padding the curve with additional vertices, if necessary, we can assume that \( n = 2^k + 1 \). Any subcurve of \( P \) of the form \( v_a, \ldots, v_b \), where \( a < b \) is a multiple of \( 2^k \) and \( b-a = 2^k \) is called a chain in grade \( \lambda \). Obviously,

(i) a grade-\( \lambda \) chain has \( 2^k + 1 \) vertices,
(ii) there are \( 2^n - 2 \) chains in grade \( \lambda \), and
(iii) there are \( p + 1 \) grades: \( 0, 1, \ldots, p \).

We begin our work bottom-up, computing conformal submaps of granularity roughly \( m^p \), where \( m \) is the size of the underlying curve and \( \beta \) is some small enough positive constant; we set \( \beta = \frac{1}{4} \), but to make the complexity analysis more explicit we leave \( \beta \) as a parameter in most of the calculations. We pursue the computation until the submap for the whole polygon has been obtained, which completes the up-phase. Then we reverse the process and work top-down until the submap has been completely refined into its full-fledged visibility map. The down-phase does not work by calling the visibility algorithm recursively on the regions of the top submap, but rather it uses data structures left behind during the top-phase (the submaps for the chains and their ray-shooting structures) to speed up the refinement process.

4.1. The Up-Phase

We begin with a piece of terminology: given a curve \( C \), consisting of \( m \) contiguous edges of \( P \), we say that a submap of \( V(C) \) is canonical if it is \( 2^{n+\tilde{m}} \)-granular, conformal, and represented in normal form. Note that a canonical submap for a chain in grade \( \lambda \) is \( 2^{2^\lambda} \)-granular. For \( \lambda = 0, 1, \ldots, p \) in that order, we process grade \( \lambda \), which means:

(i) We compute a canonical submap of \( V(C) \) for each chain \( C \) in that grade.
(ii) We preprocess each canonical submap for ray-shooting along the lines of Lemma 3.6, setting \( \gamma \) to the value \( 2^{\lambda+1} \).

This work can be done naively for the early grades, so let us pick up the action at a grade \( \lambda \) larger than some appropriate constant, assuming that all grades less than \( \lambda \) have been processed already. We need the following result.

Lemma 41. Suppose that all grades less than \( \lambda \) have been processed, and \( \lambda \) has been set to be any portion \( D \) of the form \( \gamma_1, \ldots, \gamma_p \), where \( 2^{-1} < b \) is a canonical submap of \( V(D) \) in time proportional to \( 2^{\lambda+1} \).

Proof. In \( O(\lambda) \) time we can partition \( D \) into \( j < 2 \lambda \) chains, each less than \( \lambda \), with at most two chains per grade. Then for \( \lambda \) submaps grows monotonically with the size of the union, we can trivially reset the granularity of each chain as \( j \) is proportional to the total number of vertices. Lemma 2.3, on the order of \( \log_2 n \), \( 2^{\lambda+1} \); that is, \( O(\lambda) \).

Let us now merge these submaps two-by-two \( D_i \), \( D_j \). More generally, we consider a perfectly balanced binary bijection with the \( D_i \)'s and we merge submaps bottom-up. Application of Lemma 3.5 results in a canonical submap of \( P \) as a consequence, that the required oracles are available. But any merge amount to either of the two input submaps. Therefore, any subclass \( \alpha \leq \gamma \) can be subdivided in \( 2^\lambda \) contiguous pieces (with no double-backing) whose cores consist of single line segments (at most two of them) and \( \gamma \) of \( P \), each with at most \( 2^\lambda \) edges. Each of these pieces collection of \( \alpha \) chains in grades at most \( \beta \lambda \). Our size ensures that we have ray-shooting structures for the grade these chains. Thus, to shoot a ray toward \( \alpha \), we use \( \alpha \) submaps of its decomposition and determine the closest to a single-edge subarc by using the shooting structure of a canonical submap for \( \alpha \). Assuming that \( \beta \lambda < \gamma \) (which is true for \( \lambda \) large enough), the structures have been computed and therefore, by Lemma 2.3, in time \( O(2^{\lambda+1}) \), which is \( O(2^{\lambda+1}) \). Since it follows that the ray-shooting oracle can be implemented by

\[
 f(y) = 2^{2^{\lambda+1}} + 2^{2^{\lambda+1}},
\]

As we mentioned, the subclass \( \alpha \) is decomposed into \( 2^\lambda \) subarc along with \( \alpha \) pieces for which we have conformal submaps. We verify that all the requirements of \( \alpha \) are satisfied by this decomposition, so that we can set

\[
 g(y) = O(\lambda)
\]

and

\[
 h(y) \leq 2^{2^{2\lambda+1}}.
\]
Lemma 4.1. Suppose that all grades less than \( k \) have been processed. Then, given any portion \( D \) of the form \( x_{\nu_0} \ldots x_{\nu_\nu} \), where \( 2^{k-1} < b - a \leq 2^k \), we can compute a canonical submap of \( V(D) \) in time proportional to \( \lambda(\log \lambda) \beta^{\nu_\nu - \nu_0 + 1} \).

Proof. In \( O(k) \) time we can partition \( D \) into \( j \leq 2^{\nu_0} \) chains, \( D_1, \ldots, D_j \), in grades less than \( k \), with at most two chains per grade. This implies that, for each \( i = 1, \ldots, j \), a canonical submap \( S_i \) of \( V(D_i) \) is available. Let \( \gamma \) be the granularity of a canonical submap of \( V(D_i) \); we have \( \gamma \leq 2^{\nu_i} \). Since the granularity of canonical submaps grows monotonically with the size of the underlying polygonal curve, we can trivially reset the granularity of each \( S_i \) to \( \gamma \) (Section 3.3). The time to do that is proportional to the total number of chordls in all the \( S_i \)'s which, from Lemma 2.3, is on the order of \( \sum_{i \leq \nu_i} \beta^{\nu_{i-1}} \), that is, \( O(2^{\nu_0} \gamma^{\nu_{i-1}}) \).

Let us now merge these submaps two-by-two (\( D_i \) with \( D_j \), \( D_2 \) with \( D_3 \), etc.). More generally, we consider a perfectly balanced binary tree whose leaves are in bijection with the \( S_i \)'s and we merge submaps bottom-up by following the tree pattern. Application of Lemma 3.5 results in a canonical submap of \( V(D) \), provided, of course, that the required grades are available. But are they? Notice that during any merge any arc \( x \) is either of the two input submaps consists of at most \( \gamma \) edges. Therefore, any subarc \( x' \subseteq x \) can be subdivided into a constant number of contiguous pieces (with no double-backing whose corresponding portions of \( P \) consist of a single line segments (at most two of them) and vertex-to-vertex pieces of \( P \), each with at most \( 2^{\nu_0} \) edges. Each of these pieces can be partitioned into a collection of \( O(k) \), of chains at most \( \beta \). Our work at previous grades ensures that we have ray-shooting structures for the canonical submaps associated with these chains. Thus, to shoot a ray toward \( x' \), we shoot toward each of the \( O(k) \) subarcs of its decomposition and determine the closest hit (if any). Shooting toward a single-edge subarc is trivial. Shooting toward any other subarc makes use of the shooting structure of a canonical submap for a chain in grade \( k \leq \beta \). Assuming that \( \beta \) is large enough (which is true for \( k \) large enough) all these shooting structures have been computed and therefore, by Lemma 3.6, ray-shooting can be done in time \( O(2^{\nu_0} \beta^{\nu_{i-1}}) \), which is \( O(2^{\nu_0} \gamma^{\nu_{i-1}}) \). Since there are \( O(k) \) subarcs, it follows that the ray-shooting oracle can be implemented so that

\[
 f(x) = \left\lfloor 2^{\nu_0} \beta^{\nu_{i-1}} \cdot x \right\rfloor.
\]

As we mentioned, the subarc \( x \) is decomposed into at most two single-edge pieces, along with \( O(k) \) pieces for which we have conformal submaps of granularity at most \( 2^{\nu_{i-1}} \). We verify that all the requirements of the arc-cutting oracle are satisfied by this decomposition, so that we can set

\[
g(x) = O(k)
\]

and

\[
h(x) \leq 2^{\nu_{i-1}}.
\]
To appreciate the connection between the left- and right-hand sides of these relations, recall that $\gamma$ and $\lambda$ are related by the identity $\gamma = 2^{4\lambda}$. By Lemma 3.5, if a merge takes input curves with a total of $m$ vertices, then the time to carry it out is at most proportional to

$$2^{\left(\log \lambda\right) 2^{2^{4\lambda} \lambda + 2^{3\lambda} + \log \eta \lambda}}.$$ 

There are $O(\log \lambda)$ levels of merging to be performed, each involving a total of $b - a \leq 2^\lambda$ edges, therefore the time to merge the submaps for all the $D_j$'s into one is at most (up to within a constant factor)

$$2^{\left(\log \lambda\right) 2^{2^{4\lambda} \lambda + 2^{3\lambda} + \log \eta \lambda}}.$$ 

Since the initial cost of resetting the granularity is only $O(2^{4\lambda} \cdot \Phi)$, the lemma follows readily.

Let us now turn to the processing of grade $\lambda$. Lemma 4.1 can be called upon to compute a canonical submap of the visibility map of each chain in grade $\lambda$. Preprocessing each chain for ray-shooting is done by using Lemma 3.6. Since there are $(n - 1)/2^\lambda$ chains in grade $\lambda$, we conclude that processing grade $\lambda$ requires time at most proportional to

$$n 2^{\left(\log \lambda\right) 2^{2^{4\lambda} \lambda + 2^{3\lambda} + \log \eta \lambda}}.$$ 

From our choice of $\beta = \frac{1}{2}$, it follows that preprocessing grade $\lambda$ takes $O(n 2^{4\lambda} \cdot \Phi)$ time, therefore processing all $p + 1$ grades, and thereby completing the up-phase, takes linear time.

4.2. The Down-Phase

Now that we have canonical submaps for each chain in each grade, along with their oracle structures and tree decompositions, we are ready to refine the canonical submap of $V(P)$. This is done incrementally by going down the tree.

The following lemma provides the key to the algorithms:

**Lemma 4.2.** Let $\lambda$ be any positive grade and let $C$ be an arbitrary chain in any grade $\geq \lambda$. If a $2^{2\lambda}$-granular conformal submap of $V(C)$ is available in normal form, then it is possible to compute $V(C)$ in time at most $(c - 1)/2^\lambda$, where $c$ is some constant large enough.

**Proof.** We proceed by induction on $\lambda$. Let $S$ be the $2^{2\lambda}$-granular conformal submap of $V(C)$. The case where $\lambda$ is a constant is trivial since the regions of $S$ have bounded size, and therefore the missing chords can be provided in constant time per region. So, let us switch directly to the inductive case, assuming that $\lambda$ is large enough. Let $R$ be a region of $S$. Because of conformality, the union of all the arcs of $R$ can be partitioned into a constant number of vertex-to-vertex pieces of $2\lambda$ with at most $2^{2\lambda}$ edges. We can compute a canonical submap for each connected partition in time at most proportional to

$$2^{\left(\log \lambda\right) 2^{2^{4\lambda} \lambda + 2^{3\lambda} + \log \eta \lambda}}.$$ 

Each of these submaps has granularity at most $2^{2\lambda}$ by merging by putting together all these submaps and thus forming a $2^{2\lambda}$-granular conformal submap of $V(R)$, where $\lambda$ is a vertex (to ensure that it is bounded). For constant $R^*$ as a standard polygonal curve and $\lambda$ as part of a operation requires a constant number of merges, so we can by merging submaps two-by-two like in Lemma 4.1.

There is a small subtlety in this last round of merges, viz. taking a example, suppose that $R^*$ has two arcs $a_1b_1, a_2b_2$, in cyclic order. It could be that the edge vertices of $2\lambda$, so we deal with the most general case, as $b_1b_2, b_2a_1, a_1a_2$, and $a_2$ consists of $b_1b_2, a_1a_2$, where $a_1b_1, a_2b_2$ of $2\lambda$ (Fig. 4.1). Let $S_1$ (resp. $S_2$) be the canonical submap piece of $S$ corresponding to $b_1$ (resp. $b_2$). Let $T_1$ (resp. $T_2$) for the 3-edge polygonal curve $a_1a_2b_1$ (resp. $a_1b_2a_2$). The application of Lemma 4.1, while $T_1$ and $T_2$ are computed $a_1b_1$ and $a_2b_2$ symbolically to keep the merging algorithm.

![Fig 4.1](image-url)
the arcs of $R$ can be partitioned into a constant number of single edges and vertex-to-vertex pieces of $C$ with at most $2^m$. Applying Lemma 4.1, we can compute a canonical submap for each connected polygonal piece in the partition in time at most proportional to
\[ n \log \log n + 2^{O(n^{1-\epsilon})}. \]

Each of these submaps has granularity at most $2^{m^2}$, so we can pursue the merging by putting together all these submaps and thus create a single normal-form $2^{m^2}$-granular conformal submap of $V(R^*)$, where $R^*$ is the boundary of $R$ minus a vertex (to ensure that it is nonclosed). For consistency, we should regard $R^*$ as a standard polygonal curve and not as part of a double boundary. The operation requires a constant number of merges, so we can carry it out effectively by merging submaps two-by-two like in Lemma 4.1.

There is a small subtlety in this last round of merges, which we should explain. To take a simple example, suppose that $R^*$ has two arcs and two exit chords: $a_1, a_2, b_1, b_2$, in cyclic order. It could be that the endpoints of $a_1$ or $a_2$ are not vertices of $C$, so to deal with the most general case, assume that $a_1$ consists of $b_1, b_2, a_0, a_1$, and $a_2$ consists of $b_1, b_2, a_2, a_3$, where $a_0, b_1, b_2$ are all vertices of $C$ (Fig. 4.1). Let $S_1$ (resp. $S_2$) be the canonical submap for the vertex-to-vertex piece of $P$ corresponding to $b_1$ (resp. $b_2$) and let $T_1$ (resp. $T_2$) be canonical submaps for the 3-edge polygonal curve $a_0, a_1, b_1$ (resp. $a_2, a_3, b_2$). We obtain $S_1$ and $S_2$ by application of Lemma 4.1, while $T_1$ and $T_2$ are computed directly (tilting the edge $a_0, b_1$ and $a_1, b_2$ symbolically to keep the merging algorithm from complaining later).

---

**Fig. 4.1**
We are now ready to merge $S_1$ with $T_1$, then merge the resulting submap with $S_2$, and finally merge the result with $T_2$. Note that we treat the edges $a_1b_1$ and $a_2b_2$ as part of the input curve although they are not part of $P$. As a result, ceasing (temporarily) to be chords, these edges cannot be removed during the merges. Since we add at most a constant number of new edges to the input curve, all the oracle machinery needed for the merges is still available, i.e., the new edges create only constant-time multiplicative overhead. Although the final submap is conformal it might no longer be so if we now reinterpret $a_1b_1$ and $a_2b_2$ as chords, which we do once the last round of merges is completed. To remedy this we apply the conformationally restoring procedure of Section 3.2 to each region that might have more than four chords with this new interpretation. Again, it is immediate to see that all the required oracles are still available.

The time taken by this last round of merges is dominated by the cost of the earlier merges, so computing the $2^{\Omega(\log \log \log n)}$-granular conformal submap of all the $V(R')s$ takes time at most proportional to

$$2^{\Omega(\log \log \log \log \log n)}.$$  

We can now extract the relevant information, i.e., the exit chords failing entirely within each region $R$. This involves checking the exit chords of the computed submap of $V(R')$ and keeping only those both of whose endpoints lie on the arcs (in the double boundary sense) of the region $R$. This leads to a new map $S^*$ of $V(C)$ which is a refinement of $S$: all its arcs originate from the previous merges, therefore $S^*$ is a $2^{\Omega(\log \log \log \log \log n)}$-granular conformal submap of $V(C)$. We can only speak of semigranularity because some of the chords connecting the $R'$'s might be removable now. We can check each of the exit chords directly, which as we saw in Section 3.3, takes a total amount of time linear in the number of exit chords in $S^*$.

Now that we have a $2^{\Omega(\log \log \log \log \log n)}$-granular conformal submap of $V(C)$ at our disposal we observe that $|\beta| \leq \lambda - 1$ for $\lambda$ large enough, so that we can apply the induction hypothesis and derive $V(C)$ in time at most $(c - 1)(\lambda - 1)b^*$. Putting everything together, the total running time for the construction of $V(C)$ is at most

$$a2^{\Omega(\log \log \log \log \log n)} + \left( c - \frac{1}{\lambda - 1} \right)^b,$$

for some constant $a > 0$. With the setting $\beta = \frac{1}{2}$, this is no more than

$$a2^{\Omega(\log \log \log \log \log n)} + \left( c - \frac{1}{\lambda - 1} \right)^b \leq \left( c - \frac{1}{2} \right)^b,$$

for $\lambda$ large enough.

During the up-phase we built a normal-form $2^{\Omega(\log \log \log \log \log n)}$-granular conformal submap of $V(P)$ in linear time. By Lemma 4.2, therefore, $V(P)$ can be obtained also in linear time. As demonstrated in [6] and [11], a triangulation from the visibility map in linear time, so our main goal is achieved.

**Theorem 4.3.** It is possible to compute the visibility map of a curve, and, hence, a triangulation of a simple polygon, in $\mathcal{O}(n)$ time.

5. Some Applications and Open Problems

There are many uses for a fast triangulation algorithm. For example, when the input is a simple polygon, we can compute the visibility map of the polygon in $\mathcal{O}(n)$ time. This is done by computing the visibility map of the polygon's boundary, which can be done in $\mathcal{O}(n)$ time. Then, we can compute the visibility map of the polygon's interior by using the visibility map of the polygon's boundary and the visibility map of the polygon's boundary to compute the visibility map of the polygon's interior. This can be done in $\mathcal{O}(n)$ time. 

Next, we can compute the visibility map of the polygon's interior by using the visibility map of the polygon's boundary and the visibility map of the polygon's boundary to compute the visibility map of the polygon's interior. This can be done in $\mathcal{O}(n)$ time. Then, we can compute the visibility map of the polygon's interior by using the visibility map of the polygon's boundary and the visibility map of the polygon's boundary to compute the visibility map of the polygon's interior. This can be done in $\mathcal{O}(n)$ time. Finally, we can compute the visibility map of the polygon's interior by using the visibility map of the polygon's boundary and the visibility map of the polygon's boundary to compute the visibility map of the polygon's interior. This can be done in $\mathcal{O}(n)$ time.

*From this we easily derive a new result: testing whether a point is in the interior of a polygon in $\mathcal{O}(n)$ time.*
merge $S_i$ with $T_i$, then merge the resulting submap with $S_i$, result in $T_j$. Note that we treat the edges $a_i b_i$ and $a_i b_i$, although they are not part of $P$. At a result, ceasing each edge, these edges cannot be removed during the merges. Constant number of new edges to the input curve, all the edges for the merges is still available, i.e., the new edges create a complex overhead. Although the final submap is conceptually so maybe we now reinterpret $a_i b_i$ and $a_i b_i$ as chords, at round of merges is completed. To remedy this we apply the procedure of Section 3.2 to each region that might have with this new interpretation. Again, it is immediate to see that the tiles are still available.

The last round of merges is dominated by the cost of the running the $2^{\text{granular}}$ conformal submap of all the tiles proportional to

$$2^{1/2 \log 2} 2^{P(\log \log 3 - 1/3)}$$

relevant information, i.e., the exit chords failing entirely. This involves checking the exit chords of the computed graph (e.g., only those of whose endpoints lie on the arc sense) of the region $R$. This leads to a new map $S$ of count of $S$, all its arcs originate from the previous merges, semigranular conformal submap of $P(C)$. We can only because some of the chords connecting the $R$'s might can check each of the exit chords directly, as we can get a total amount of time linear in the number of exit chords

$$2^{P(\log \log 3 - 1/3)} \leq \left( \frac{c - 1}{c - 1/3} \right)^{2^n}$$

With the setting $\beta = 1/3$, this is no more than

$$2^{-3/3} \leq \left( \frac{c - 1}{c - 1/3} \right)^{2^n} \leq \left( \frac{c - 1}{c - 1/3} \right)^{2^n}$$

We built a normal-form $2^{\text{granular}}$ conformal submap by Lemma 4.2, therefore, $V(P)$ can be obtained also in linear time. As demonstrated in [6] and [11], a triangulation can be easily derived from the visibility map in linear time, so our main goal has been reached.

**Theorem 4.3.** It is possible to compute the visibility map of a simple polygonal curve, and, hence, a triangulation of a simple polygon, in linear time.

5. Some Applications and Open Problems

There are many uses for a fast triangulation algorithm. We mention only two of them and refer the reader to [1], [13], [14], [16], [20], [24], [26], and [28] for some pointers to other applications. Kirkpatrick [9] and Edelsbrunner et al. [9] have given optimal planar point-location algorithms which require linear preprocessing time provided that the planar subdivision has triangulated or monotone faces. With our algorithm the preprocessing can be made linear as long as the graph is connected.

The application follows from the observation that the triangulation algorithm can be adapted to check whether a polygonal curve is simple. We briefly outline the method. We begin by computing the visibility map. In all likelihood the program will crash if the polygonal curve is not simple. But if it does not, then we are handed over a would-be visibility map with its corresponding visibility tree. In linear time we can verify that the map is locally sound by checking that all regions and adjacencies are topologically and geometrically satisfactory. If the curve has a self-intersection $p$, then two regions must strictly overlap around that point, but this can be shown to contradict the local soundness of the map. Indeed, draw a line connecting $p$ to any point $q$ that provably belongs to only one region (if an arbitrary choice of $q$ does not work, then obviously the curve is not simple). By a continuity argument, the segment $pq$ must cross an edge at a point where the number of overlapping regions differs locally around that point. But, occurring on an edge, this must be detected by the local tests. So, to summarize, a linear number of local geometric tests suffice for testing simplicity, once the output of the visibility algorithm is available.

From this we easily derive a new result: testing whether two simple polygons intersect in linear time. The reduction of this problem to simplicity-testing was observed by Dobkin et al. [8]. Slightly simplified, their method consists of taking the highest points of both polygons and shoot horizontally from the lower point $p$ toward the other polygon in both directions. If there is no hit, then no intersection can occur. Otherwise, we can immediately infer from the local orientation of the hits whether $p$ lies inside or outside the polygon. In the first case we conclude with a positive answer, else we connect the two polygonal boundaries into a single one by adding one of the connecting segments and duplicating it, thus reducing the problem to that of testing simplicity.

It is interesting to notice that our algorithm can be used to perform Jordan sorting in linear time. This provides a completely different alternative to Hoffman et al.'s algorithm [18]. Recall that the problem is to sort a sequence of numbers $a_1, \ldots, a_n$ which correspond to the intersection points of a Jordan curve with the
Assuming that we do not know anything else about the curve, we construct the polygon $P$ with vertices $A_1, A_2, A_3, \ldots, A_n, A_1$, where $A_i = (a_i, 0)$ and

$$B_i = \left( \frac{a_i + a_{i+1}}{2}, -\frac{a_{i+1} - a_i}{2} \right)$$

with $a_{n+1} = a_1$. It is immediate that $P$ is simple and that its visibility map gives us the $a_i$'s in sorted order. (Recall that to have many vertices with the same $y$-coordinate is not a serious problem.) It is instructive to compare the two methods: Hoffman et al.'s algorithm is on-line and corresponds to an asymptotically optimal search scheme for Jordan permutations. Our method works off-line and uses divide-and-conquer. It is an intriguing open question whether triangulation can be done on-line in a manner similar to [18]. Tarjan and Van Wyk's method (almost) fails in that category but it is not optimal. Can their algorithm be made optimal? More generally, is it possible to maintain the visibility map optimally under on-line insertion of new edges? Obviously not in an explicit fashion, since a new edge can cut through a linear number of diagonals, hence creating a quadratic blowup. Even implicitly, however, it can be trivially shown that identifying the order types of the visibility maps of all the prefixes of a simple $n$-vertex polygonal curve requires $\Theta(n \log n)$ bits, which is bad news. Thus, something with less information content should be maintained by an optimal on-line algorithm. But what? This question might seem rhetorical in light of our linear-time algorithm, but the underlying issue is whether a fundamentally different optimal algorithm exists, in particular, one that works on-line. A close look at the up-phase of our visibility algorithm shows that constructing the computation tree in symmetric order is tantamount to inserting the edges one at a time along the boundary and maintaining a small collection of distinct visibility submaps. Since these maps provide only partial visibility information this can be looked at as a partial answer to our question.

Also, we might wonder whether there exists a simple optimal probabilistic triangulation algorithm, say, one as straightforward as Clarkson et al.'s [7] or Seidel's [25]? Our algorithm can be modified by replacing the planar separator step by a randomized construction, but whether the resulting algorithm is actually simpler is debatable. We close by mentioning what is probably the most interesting open question left on the subject of polygonal curves. This is the problem, previously posed by Tarjan and Van Wyk [27], of computing all the self-intersections of a non-simple polygon in time linear in the sum of the input and output sizes.

Acknowledgments

I wish to thank the referees for reading this paper carefully and making numerous suggestions to improve its presentation.
B. Chandle

do not know anything else about the curve, we construct
the metric $d_1, d_2, d_3, \ldots, d_n$ where $d_i = \phi_i, \theta_i$ and

\[ d_i = \left( \frac{\phi_i + \theta_i}{2} \right) - \left( -\frac{\phi_i - \theta_i}{2} \right) \]

mediate that $F$ is simple and that its visibility map gives
sider. (Recall that to have many vertices with the same
serious problem.) It is instructive to compare the two
algorithms: one is in-line and corresponds to an asymptotic
for Jordan permutations. Our method works off-line
better. It is an intriguing open question whether triangula-
tion is in a manner similar to [18]. Tarjan and Van Wyk’s
algorithm is in that category but it is not optimal. Can their algorithm
in general, is it possible to maintain the visibility map
and insertion of new edges? Obviously not in an explicit
way to construct a linear number of diagonals, hence
skew. Even implicitly, however, it can be trivially
the order types of the visibility maps of all the prefixes of
all curves require $O(n \log n)$ bits, which is bad news. Thus, the
information content should be maintained by an optimal
algorithm. Is this question theoretical in light of our
observation? The underlying issue is whether a fundamen-
tally different, in particular, one that works on-line. A close look at
the Tarjan algorithm shows that constructing the computa-
tion tree is tantamount to inserting the edges one at a time along the
edges of a small collection of distinct visibility submaps. Since
the partial visibility information this can be looked at as a
subproblem.

Consider whether there exists a simple optimal probabilistic
algorithm, one as straightforward as Clarkson et al.’s [7] or
which can be modified by replacing the planar separator
construction, but whether the resulting algorithm is really
close by mentioning what is probably the most interesting
aspect of polygonal curves. This is the problem.

Tarjan and Van Wyk [27] of computing all the self-

References

1. A. Aggarwal and J. Wein, Computational Geometry, Technical Report MIT/ LCS/ R3, MIT,


3. T. Booth, Some fast algorithms on graphs and trees, Ph.D. Thesis, Technological Report CS-TR-

4. B. Chandle, A theorem on polygon cutting with applications, Proc. 23rd Ann. IEEE Sympos. on

5. B. Chandle and J. C. Gubra, Visibility and intersection problems in plane geometry, Discrete

135-152.

7. K. Clarkson, R. E. Tarjan, and C. J. Van Wyk, A fast Las Vegas algorithm for triangulating a


9. H. Edelsbrunner, L. J. Guibas, and J. Stolf, Optimal point location in a monotone subdivision,

10. H. Edelsbrunner and J. P. Mckeon, Simulations of simplification: A technique to cope with degenerate

11. A. Fournier and D. Y. Mouton, Triangulating simple polygons and equivalent problems, ACM


System Sci. 32 (1986), 126-152.

visibility and shortest path problems inside triangulated simple polygons, Algorithmica 2 (1987),
308-333.

15. L. J. Guibas and J. Stolf, Primitives for the triangulation of general subdivisions and the

16. J. Hershberger, Finding the visibility graph of a simple polygon in time proportional to its size,

218.

time using level-linked skip trees, Inform. and Control 60 (1985), 170-194.


20. D. G. Kirkpatrick, M. M. Klawe, and R. E. Tarjan, On $O(n \log n)$ polygon triangulation with

177-189.

(1980), 615-627.


25. R. Seidel, A simple and fast incremental randomized algorithm for constructing trapezoidal
decompositions and for triangulating polygons, Manuscript, 1990.

Discrete & Computational Geometry

Instructions for Authors

General Information
Manuscripts for this journal should be typed on one side of the paper, double spaced, with 1-inch margins all around. All elements of formulae, etc., should be typewritten whenever possible.

Each figure (line drawings and graphs) should be submitted on a separate sheet of paper; each diagram should be drawn precisely in India ink, using lines of uniform width. The figures should be drawn at twice the size desired; the maximum area for the reduced figures is 4" x 7" (10.2 x 17.8 cm).

All diagrams should be numbered consecutively, and their first mention in the text should be noted in the left-hand margin. Legends should be fixed consecutively on a single sheet of paper.

An abstract of the article should be included on the title page. An abbreviated version (80 characters or less) of the article title should be provided for use as a running head.

Footnotes other than those referring to the title should be avoided. If they are essential, they should be numbered consecutively and placed at the foot of the page to which they refer.

References to the literature should be listed alphabetically at the end of the manuscript, in the following form:


Marking the Manuscript
In text, words or phrases requiring italic type should be underlined. Only the end of a proof should be indicated by □.

All letters in formulae, as well as single letters in the text, are automatically printed italic and therefore require no underlining. Please use the following color code for marking:

Greek letters: red Gothic letters: blue Script letters: green