A Linear-Time Algorithm For Covering Simple Polygons with Similar Rectangles

Reuven Bar-Yehuda*
Eyal Ben-Chanoch
Computer Science Department, Technion - IIT, Haifa 32000, Israel

December 4, 1994

Abstract

We study the problem of covering a simple orthogonal polygon with a minimum number of (possibly overlapping) squares, all internal to the polygon. The problem has applications in VLSI mask generation, incremental update of raster displays, and image compression. We give a linear time algorithm for covering a simple polygon, specified by its vertices, with squares. Covering with similar rectangles (having a given x/y ratio) is an equivalent problem.

Key words: Computational Geometry, Orthogonal Polygons, Square Cover.

*The research was supported by the Technion V.P.R. Fund - New York Metropolitan R. Fund.

1 Introduction

1.1 Overview

We consider the problem of finding a minimum square cover of an orthogonal polygon. An *orthogonal polygon* is a polygon whose edges are either horizontal or vertical. A *square cover* of a polygon P is a collection of squares, all contained in P, whose union exactly covers P. A *minimum square cover* is a square cover with the minimum number of squares (see Figure 1.1). Obviously, we can concentrate only on maximal squares. A square in the polygon is said to be *maximal* if it is not contained by any larger square contained in the polygon.

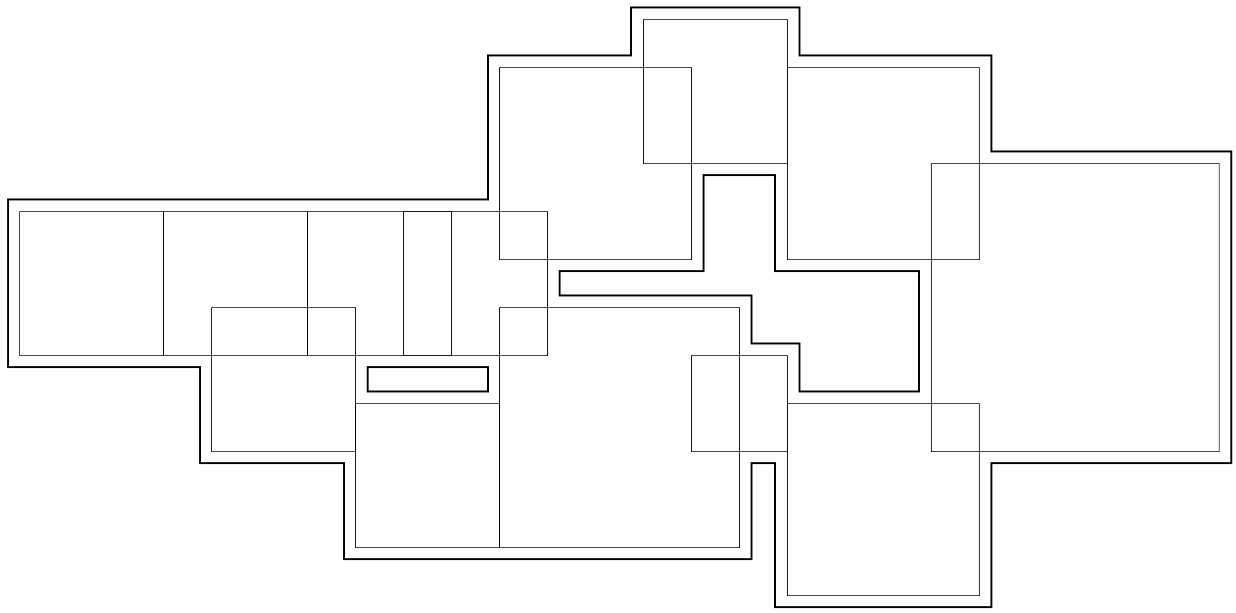


Figure 1.1: A Minimum Square Cover

Our research was motivated by the paper of Scott and Iyenger [SI86] where they argued that square covering is an efficient method to store images, mainly in comparison to the known quad-tree approach [SXL87, HS79]. Moreover, they revealed some of the characteristics of this method such as simplicity, small space demands and invariance in the plane under shifts and rotations. In their work they developed a practical heuristic algorithm to find a square cover (not necessarily minimal), empirically illustrating a better performance than all of the quad-tree methods they tested.

Aupperle et al. [ACKO88] noted another practical application which we quote. The *medial axis* (also called the *symmetric axis* or *skeleton*) of a polygon is the locus of centers of maximal disks contained in the polygon. When specialized to the L_∞ metric for applications to digital images, the medial axis is the locus of centers of maximal squares of odd side length [RK82]. The digital medial axis transform *(MAT)* is used for picture compression: simple images may be covered by few squares, and easily reconstructed from the *MAT* [WBR86].

Additional applications are mentioned by Morita [Mor88], and they include remote sensing, VLSI mask generation, incremental update of raster displays, and object representation for sequential frames of a dynamic polygonal scene.

1.2 Related Work

The previous work on the square cover problem used a bit-map representation for the input polygon (see [AO81, SI86, ACKO88, Mor88]). A *bit-map* representation of a polygon, sometimes referred to as a *digitized image*, is a boolean (zero-one) matrix, where an one represents a point inside the polygon and a zero—a point outside it (see Figure 1.2). When using this representation, complexity is measured in terms of the number of points in the matrix, denoted by p.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	0
0	0	0	0	0	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	0
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	1	1	1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	1	1	1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 1.2: A Bit-Map

Recently Aupperle, Conn, Keil and O'Rourke [ACKO88] showed that the minimum square cover problem is NP-Hard for images containing holes. In the case where the image is hole-free, they provided an $O(p^{1.5})$ time algorithm. They translate the problem into covering chordal graphs by cliques (equivalent to finding a minimum vertex cover). Each node in the graph is associated with a pixel in the bit-map, and two nodes are connected by an edge iff the corresponding pixels are within the same maximal square. Clearly, the nodes of a maximal clique correspond to the pixels of a maximal square. Since the image is hole-free, all cycles in the graph are triangulated, and therefore the algorithm of Gavril [Gav72] for covering chordal graphs by a minimum number of cliques can be used. To decrease the size of the graph, they generate a reduced graph, where each maximal clique (square) is transformed into a single node.

A square cover is called *minimal* if it has no smaller subset that forms a cover. Morita [Mor88] recently developed a parallel algorithm, which finds a minimal square cover for bit-maps which may contain holes. The sequential running time of this algorithm is $O(p)$.

1.3 Bit-Maps vs. Segment Representation

It seems that the original motivation to study the square cover problem came from the field of image processing and other related fields. This may explain why all the previous work concerns only polygons with integer coordinates. Furthermore, the time complexity was measured in terms of the input bit-map size p.

For many problems concerning polygonal shapes, the input is chosen to be in segment representation, i.e., the polygon is specified by a cyclically ordered sequence of its n vertices. We argue that this representation should be preferred, not only theoretically, but also for all of the mentioned practical applications.

In order to illustrate that a bit-map based algorithm may, in the worst case, be exponential (or at the most pseudo-polynomial) in the segment representation input size, consider the following example: The input polygon $((0,0),(0,y),(1,y),(1,0))$, which consumes only $O(\log y)$ space, requires at least $O(y)$ space for the bit-map representation. On the other hand, it may be worthwhile to spend an $O(p)$ time in translating a bit-map representation into a segment representation, since $p > \Omega(n)$, and for most practical applications $p \gg n$. This translation can be implemented on-line, in reading image files. This method requires only $O(n \log p)$ bits, rather than $O(p)$.

The use of the chordal graph method is elegant and natural for bit-maps. In the segment representation the associated chordal graph, and even the reduced one, may have infinite cardinality. In our method, we use the local optimization approach; i.e., selecting, iteratively, the next square "essential" to the cover. Local optimization algorithms differ in the policy adopted for the next local optimization step. The core of our algorithm is the choice of a specific policy, derived by additional topological properties.

1.4 Outline of The Algorithm

In this section we give an intuitive description of our linear-time algorithm.

Imagine that in the beginning of the process you have a piece of black paper cut in the shape of the given input polygon. As the algorithm proceeds, we paint the covered areas with a grey chalk.

Let s be a maximal square in the polygon. Denote by $Black(s)$ the set of the black (uncovered) points in s. A square s is *maximal black* if there is no square s_1 such that $Black(s) \subset Black(s_1)$. A maximal black square s is *essential* if there exist a point such that s is the only square containing it.

```
While there exists an essential square s
    Paint s grey.
```

Algorithm A: A first attempt

Assume the paper is now all painted grey; then the set of the selected squares yields a minimum square cover. This fact is derived from the following local optimization invariant: there exists a minimum square cover for the polygon which contains all of the previously selected squares.

A grey point is *reducible* if it is not included in any maximal black square. Now, for Algorithm B, we get a pair of scissors ready.

```
While there exists an essential square s
    Paint s grey and cut all reducible regions.
```

Algorithm B: Starting to cut

Cutting reducible regions is harmless, and therefore the local optimization invariant remains valid.

The approach: select "essential" and eliminate "reducible", is the same for all special cases of the set-covering problem, including covering chordal graphs by cliques [Gav72] and vertex cover [NT75, BYE85]. Unlike those problems, in our problem both the number of squares (sets) and the number of points (set elements) are infinite. This fact should not affect the correctness of the local optimization approach, as long as an essential square can be found and in each iteration the problem size (input+output) can be reduced. In order to achieve this goal, we concentrate our efforts on a special type of essential squares, called continuators. A *continuator* is a maximal square having a continuous intersection with the polygons' border (for a wider family of squares containing a *knob* see [AO81, ACKO88]). The main idea is that if the residual piece of paper does not contain reducible regions, then any continuator is essential.

```
While there exists a continuator s
    If s is essential then paint s grey.
    Cut all reducible regions contained in s.
```

Algorithm C: Using continuators

We prove that any simple polygon contains a continuator. This, together with the local optimization invariant, implies partial correctness for Algorithm C. By charging each operation to either input (vertices) or output (cover squares) we show that the number of iterations is $O(n+k)$. This algorithm can be implemented so that after a preprocessing phase, consisting mainly of triangulating the polygon, each output operation costs $O(1)$ time, while each input operation costs $O(\log n)$ time. This leads us to $O(n \log n + k)$ time complexity. We then restrict ourselves to continuators with an additional topological property, which leads to an implementation with a time complexity of $O(1)$ for each input and output operation. Since the best known algorithm for triangulation (preprocessing) of a simple polygon has $O(n)$ time complexity [Cha90], the total time is reduced to $O(n+k)$.

As presented, the algorithm is output sensitive. Note that the output size may be even exponential in the input size (see, for example, a polygon which is a long rectangle). O'Rourke [O'R87] suggested a way to overcome this problem by changing the output representation: a set of output squares that form a partitioning of a rectangle are replaced by this rectangle. This causes the output size to also be $O(n)$. Fortunately, our algorithm can be implemented so that such rectangles are used at no extra cost.

1.5 Covering with Similar Rectangles

An *r-ratio rectangle* is a rectangle having ratio r between its x-dimension and its y-dimension. A square is, therefore, a 1-ratio rectangle. The extended covering problem is: "Given r and a polygon P, find a cover for P with a minimum number of r-ratio rectangles". This extended problem can be reduced to the square cover problem by dividing each x coordinate of the vertices of P by r, covering the new polygon by squares and finally multiplying the x coordinates of the squares' vertices by r.

In the rest of the paper we consider only squares. It is obvious, however, that all topological claims can be modified to hold for r-ratio rectangles instead.

1.6 Paper Organization

The rest of this paper is organized as follows:

Section 2 introduces some topological properties which are used in our algorithm. Section 3 shows how the local optimization approach is applied to the problem of square covering. Section 4 implements a basic algorithm using this approach, running in $O(n^2 + k)$ time and using $O(n)$ space. Finally, Section 5, shows how this algorithm can be improved to run in $O(n + k)$ amortized time, still using $O(n)$ space.

2 Topological Properties

Relevant topological properties can be found in [AO81, ACKO88, Mor88]. These properties have been proved only for orthogonal polygons with integer coordinates. In this section we extend these properties to orthogonal polygons with real coordinates. Furthermore, new topological properties are presented, which the reader may find interesting for their own sake.

2.1 Maximal Squares and Square Cover

A square s is *maximal* in a polygon P if no other square contained in P contains s. A *square cover* C of a polygon P is a set of (possibly overlapping) maximal squares in P, covering the area of P. A square cover C is called *minimal* if no subset of C covers P (i.e., C is a local minima), and *minimum* if there is no other square cover of P with smaller cardinality. Obviously, it is sufficient to treat only maximal squares, since each non-maximal square in a cover can be replaced by one of the maximal squares containing it. From this point on, we shall be concerned only with maximal squares. For distinction, we define a polygon by its *segments*, and a square by its *edges*.

2.2 Squares and the Polygons Border

In respect to a polygon P and a square s, a point in s is called *exposed* if it is on one of the polygons' segments (border). The set of all its exposed points is called the *exposed outline* of s (see Figure 2.1). Clearly, the maximality of a square is strongly related to the configuration of its exposed outline.

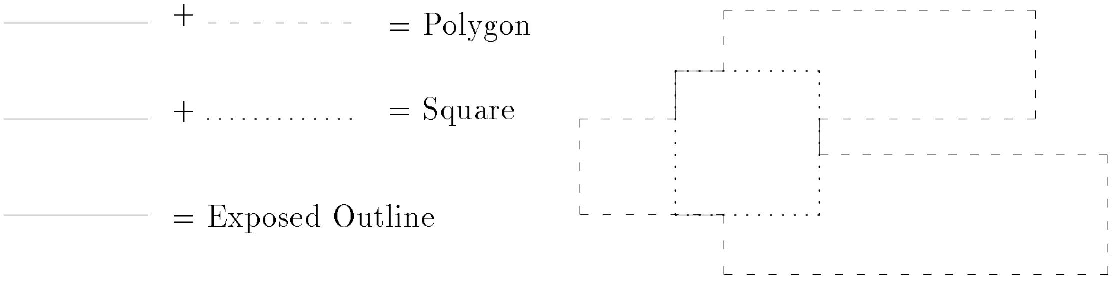


Figure 2.1: Exposed outline of a square in a polygon

Let $s = (A, B, C, D)$ be the vertices of a square, specified in clockwise order where A is the top-left corner. A *corner set* of a square corner $X \in \{A, B, C, D\}$, denoted by $CornerSet(X)$, is the collection of points defined by the corner X and its two attached edges, not including their other extremities (see Figure 2.2).

Lemma 2.1 *A square is maximal iff each of its corner sets contains at least one exposed point.*

Proof: Let s be a square in a polygon P, defined by its corner points $A = (0, a)$, $B = (a, a)$, $C = (a, 0)$, and $D = (0, 0)$.

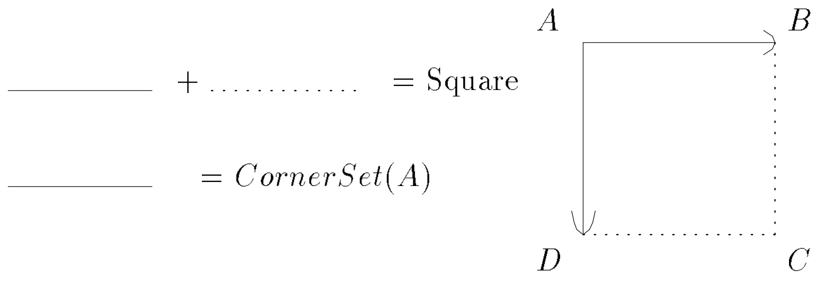


Figure 2.2: $CornerSet(A) = [A, B) + [A, D)$

Assume $CornerSet(B)$ does not contain any exposed point; let E be the nearest exposed point to $CornerSet(B)$ located in the upper right quadrant (not including the axes), and let ϵ be the shortest orthogonal distance from E to $CornerSet(B)$. Observe the square s' with the corners $(0, a+\epsilon)$, $(a+\epsilon, a+\epsilon)$, $(a+\epsilon, 0)$, and $(0, 0)$. Since $\epsilon > 0$, it is obvious that $s \subset s' \subseteq P$, thus, s is not maximal in P. This completes the "if" direction of the lemma. For the other direction, assume that s is not maximal. This implies that there is a larger square s' in P containing s. Therefore, at least one corner of s, say A, is in the interior of s' (this is not necessarily true with general rectangles). This implies that $CornerSet(A)$ has no exposed point, since it is internal to s'. □

Corollary 2.2 *Every maximal square has at least two exposed points on opposite segments.*

2.3 Continuators and Separators

A *corner* in a polygon is the attachment point of two segments. A corner is called *concave* if the small angle (90 degrees) is on the outer side of the polygon, and *convex* otherwise. A *knob* is a segment whose two endpoints are convex corners. A square s in a polygon P is called a *separator* if $P - s$ is not connected. Albertson and O'Keefe [AO81] showed that if s is a non-separating maximal square in a simple polygon P, then s contains a knob. The opposite, however, is not true; i.e., a square containing a knob may be a separator. We need a sufficient and necessary condition on the exposed outline of a square for it to be non-separating.

The exposed outline of a square s in a polygon P is the intersection of two orthogonal polygonal curves. Therefore, the exposed outline consists of a finite collection of orthogonal curves. A square s is called a *continuator* if its exposed outline is continuous (see Figure 2.3).

The continuator and the separator play the main role in this paper. Therefore, understanding the definition and the qualities of these types is essential for understanding the rest of this paper. In addition, we demonstrate an analogy between polygons, continuators, and separators to trees, leaves, and internal vertices. This analogy is shown in the following two lemmas, which are variations on familiar theorems on trees. Lemma 2.3 corresponds to "In a tree, every vertex is either a leaf or internal (disjoint) vertex", and Lemma 2.4 to "In a non trivial tree, there are at least two leaves".

Lemma 2.3 *In a simple polygon every maximal square is either a separator or continuator.*

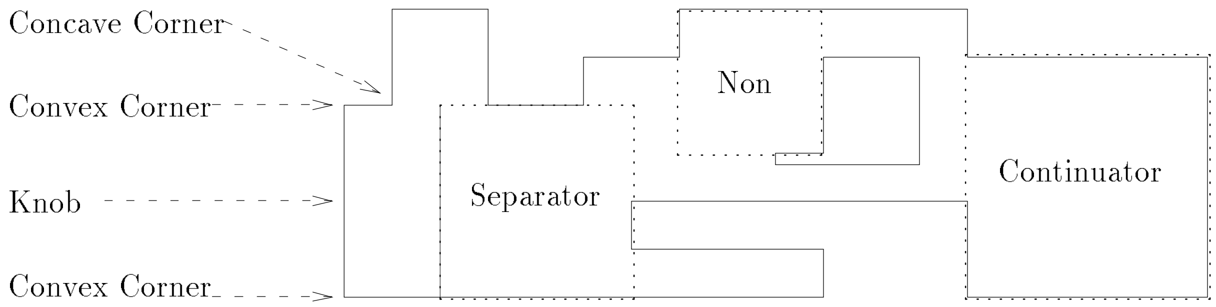


Figure 2.3: Square and corner types

Proof: It is obvious that no square is a continuator and separator, thus, it is sufficient to prove that only the two types exist in a simple polygon. Let s be a maximal square in a polygon P. If s is not a continuator, it has at least two disjointed paths along its exposed-border, say α_1 and α_2. Since P is simple (its contour is continuous), there are two additional disjointed paths, β_1 and β_2, connecting α_1 with α_2. Extract s from P, would delete α_1 and α_2, therefore, disconnecting β_1 from β_2. Thus, s is a separator. □

Lemma 2.4 *If a simple polygon P is not a square, then P has at least two continuators.*

Proof: For this proof we define another square type; A maximal square s in a polygon P is called a *corner-square* if at least one of its corners is a convex corner of P. Note that a continuator is always a corner-square, but a separator is not necessarily one. Moreover, while every convex corner is covered exactly by one corner-square, a continuator contains at least two convex corners of the polygon.

The proof is by induction on k, the number of corner-squares in P.

Base ($k = 2$): A polygon with two corner-squares is either a rectangle or the union of two intersecting squares. In both cases, there exist two continuators.

Step ($k > 2$): Observe the polygon P having $k > 2$ corner-squares, and assume for the contrary that there are less than two continuators in P. By Lemma 2.3, all of the $k-1 > 1$ corner-squares, that are not continuators, are separators; let s be one of them. Let $\{P_i\}_{i=1}^{n\geq 2}$ be the disjointed components of $P - s$, and let P_i^+ be the polygon resulting from joining s (in its original location within P) to P_i (see Figure 2.4). Each P_i^+ has at least one corner-square in addition to s, but it has less corner-squares than P, since no P_i^+ is empty. Therefore, by the induction hypothesis, every such P_i^+ has at least two continuators, one of whom can be s, but the other was inherited from P. Combining the n components P_i, we get the polygon P, having at least $n \geq 2$ (original) continuators.

□

We conclude this section with a new topological property stated in Lemma 2.5. This property is used for speeding up the main algorithm. A point p is *visible* to a segment A in a polygon P, if an

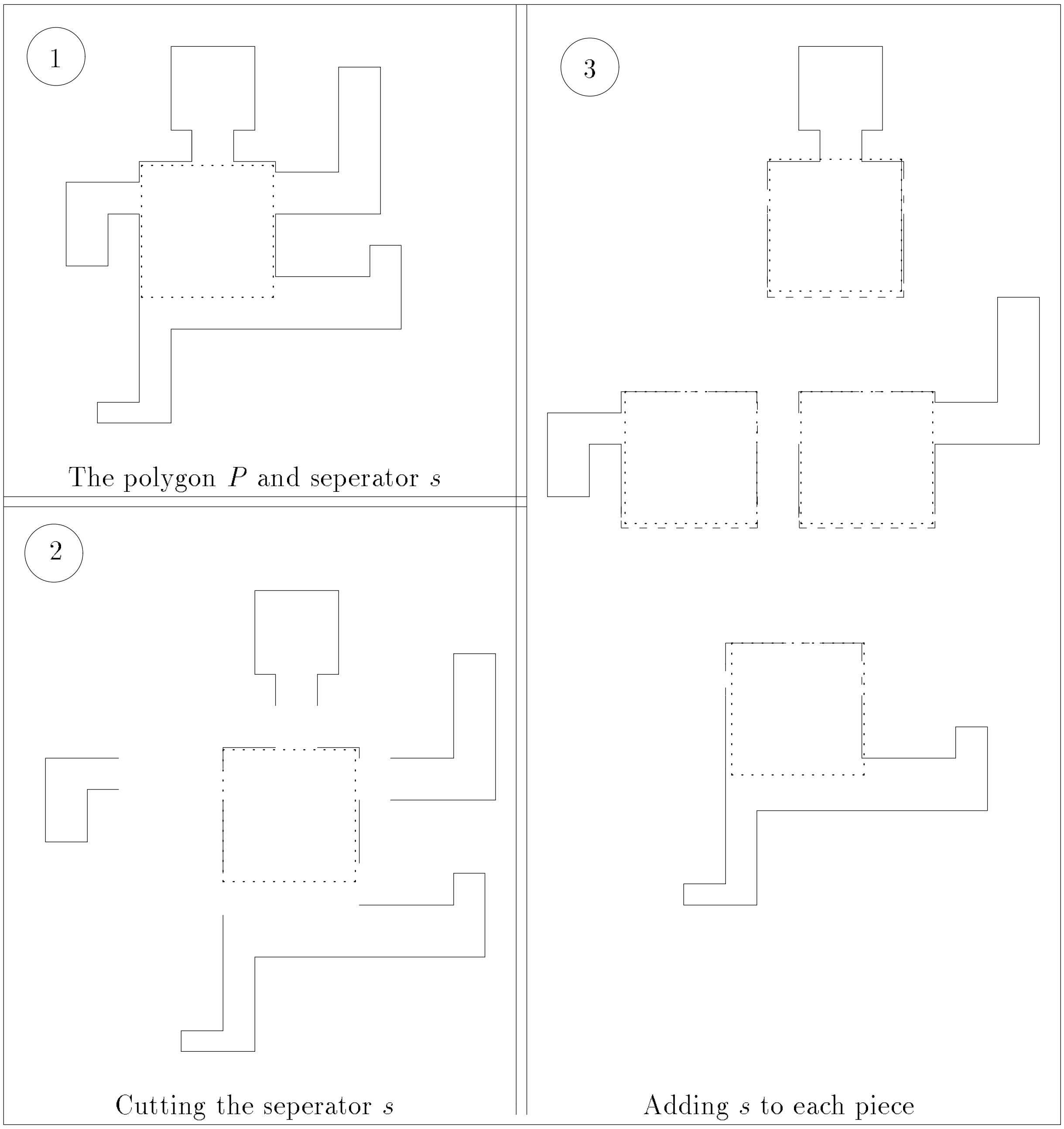


Figure 2.4: Induction step for Lemma 2.4

orthogonal ray (vertical or horizontal), starting at p, hits A before it intersects any other segment of P. In addition, we say that A *sees* a segment B, if all of B's points are visible to A. Note that in a simple polygon, it is sufficient that B's ends would be visible to A for B to be seen by A. Also, we define an *antiknob* to be a segment with two concave endpoints.

Lemma 2.5 *In a simple polygon, there exists a continuator such that each of its knobs sees at most one antiknob.*

Proof: First, we prove two claims on the ratio between the number of the continuators and the number of the antiknobs within a polygon. Letting K be the number of continuators, G_H be the number of horizontal antiknobs that are seen by a horizontal knob, and G_V be the number of vertical antiknobs that are seen by a vertical knob, the following properties both hold:

1. $K \geq G_H + 2$
2. $K \geq G_V + 2$

We prove (1) by induction on the number of horizontal knobs that see an antiknob.

Base (0 antiknobs): By Lemma 2.4, there are at least two continuators in P; therefore, there exist at least two knobs.

Step: Observe the polygon P, having n knobs that see an antiknob. Lemma 2.4 implies that there are at least two knobs in P. Let H be a knob which sees an antiknob V. Add a perpendicular segment connecting H with V (see Figure 2.5). This construction divides P into two smaller polygons, P_1 and P_2. But it also eliminates one antiknob and one original continuator, and adds two new continuators— one for each of the new polygons. P_1 and P_2 each have at least one antiknob less than P, and therefore the induction hypothesis holds so that $K_1 \geq G_{H1} + 2$ and $K_2 \geq G_{H2} + 2$. Adding them we get $K + 1 = K_1 + K_2 \geq G_{H1} + G_{H2} + 4 = (G_H - 1) + 4$. Thus, $K \geq G_H + 2$.

We prove (2), the vertical version, in a symmetrical way. Finally, adding the results of (1) and (2), we get: $G = G_H + G_V + 4 \leq 2 \cdot K$. The average number of antiknobs per knob is $G/K < 2$, and therefore, there is at least one knob that sees at most one antiknob. $\square$

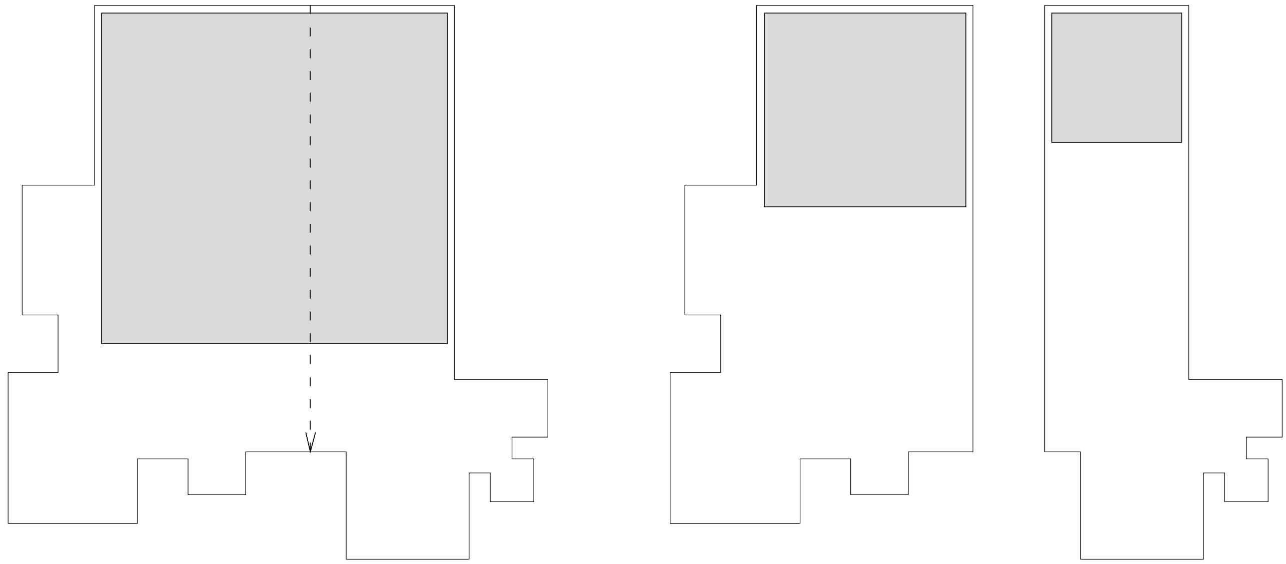


Figure 2.5: Induction step for Lemma 2.5

3 A Local Optimization Approach

The minimum square cover problem is a special case of the well-known problem of set covering. The set covering problem is conventionally given as: "given a universe P of elements, and a set S of subsets of P covering the universe, find a set $C^* \subset S$ of minimum cardinality covering P". In our case, the universe P is the polygon, and S is the set of all maximal squares within P. In order to solve the problem, we adopt the widely known *local optimization* approach. In this approach, we build the cover by iteratively selecting subsets "essential" to the cover, and deleting from our universe the parts which become unneccessary or "erasable".

For the sake of inductive arguments, we shall define (P, C) as the following generalized problem: "given a polygon P and a set of squares C, extend C to a square cover of P". In these terms, the original problem is (P, ϕ). Let us now generalize the square and maximal square definitions in (P, C). A point $p \in P$ is called *covered* if it is contained in some square of C. A *sub-square* of (P, C) is a non-empty set of uncovered points, all in some square in P. A sub-square is *maximal* in (P, C) if there is no other sub-square in (P, C) containing it. Letting p be an uncovered point in (P, C), we denote by $MaxSubSquares(p)$ the set of all maximal sub-squares containing p. We call p *forcing* if $MaxSubSquares(p)$ is a singleton. We call a maximal sub-square $\overline{s}$ *forcing* if it contains a forcing point. A square is called *essential* if it contains a forcing sub-square (see Figure 3.1).

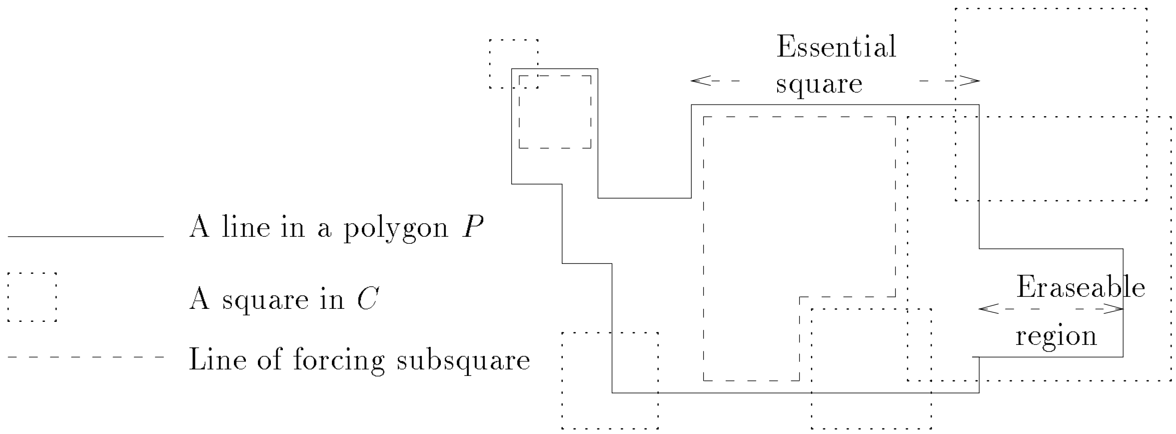


Figure 3.1: Essential Square and Erasable Region in (P, C)

3.1 Essential Squares

Claim 3.1 (The Essential Claim) *If s is an essential square in (P, C), then there exists a minimum cover C^* for (P, C) such that $s \in C^*$.*

Proof: Let C' be a minimum square cover for (P, C), and choose $p \in \overline{s} \subset s$ such that $\overline{s}$ is a forcing maximal sub-square in C and p is a forcing point. Let s' be a square in C' covering p. It is sufficient to show that $C^* = C' - s' + s$ is a cover for P, and for this we show that $\overline{s}$ contains all the uncovered points in $(P, C' - \{s'\})$. Let p' be such an uncovered point. Since p and p' are both in the same square (s'), and since $\overline{s}$ is the only sub-square covering p, it follows that $\overline{s}$ covers p' as well. $\square$

The Essential Claim implies the partial correctness of Algorithm A.

```
(P, C) ← (P, φ)
While C does not cover P

      Select s, an essential square in (P, C)
      (P, C) ← (P, C + {s})
```

Algorithm A: Covering Using Essential Squares

3.2 Erasable Regions

A point p is called *erasable* in (P, C) if every square containing p does not contain a maximal sub-square (see Figure 3.1).

Claim 3.2 (The Erasure Claim) *If E is the set of all erasable points in (P, C), there exists a minimum cover C^* for (P, C), such that no square of C^* intersects E.*

Proof: Let C^* be a minimum cover for (P, C), such that the number of squares of C^* intersecting E is minimal (since P can be partitioned into a finite number of rectangles, and each of them can be covered by a finite number of squares, $|C^*|$ is finite). Let us assume to the contrary that there exists a square $s \in C^*$ such that $s \cap E$ is not empty. Let $\overline{s}$ be a maximal sub-square containing all the uncovered points of s in $(P, C - \{s\})$, and let s' be a square containing $\overline{s}$. Obviously, $C^* - \{s\} + \{s'\}$ is a minimum cover for (P, C). s' clearly does not intersect E, and therefore C^* does not have the minimum number of squares intersecting E — a contradiction. $\square$

The Essential Claim together with the Erasure Claim imply the partial correctness of Algorithm B.

```
(P, C) ← (P, φ)
While P ≠ φ

      Select s, an essential square in (P, C)
      (P, C) ← (P, C + {s})
      Find a set E ⊂ P of erasable points in (P, C)
      (P, C) ← (P − E, C)
```

Algorithm B: Essential Squares and Erasable Regions

In our algorithm, we consider only continuators. Letting s be a continuator in P, we define $balcony(s)$ to be the set of all points in P such that s is the only maximal square containing them.

There are four types of continuators, distinguished by the number of knobs they contain. Figure 3.2 illustrates the balcony shape in each continuator type.

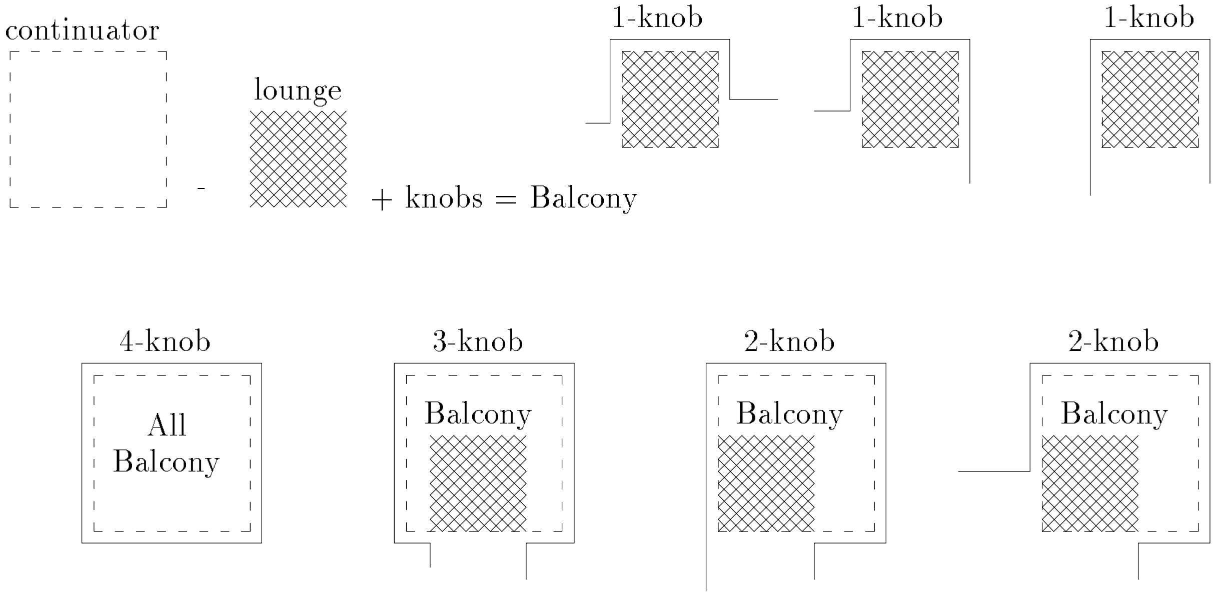


Figure 3.2: Continuator types and their internal structure

It is clear that if *balcony*(s) contains an uncovered point p in (P, C), then p is a forcing point, and hence, s is essential. Otherwise, *balcony*(s) is erasable. After erasing *balcony*(s), the remaining part of s — *lounge*(s) — is a square, and if this square is maximal then it has to be an one-knob continuator in $P - balcony(s)$.

This, together with the Essential Claim and the Erasure Claim, imply the partial correctness of Algorithm C.

In the next section we are going to show how to select a continuator and how to find the erasable region. This is done using visibility.

3.3 Visibility — The Main Tool

3.3.1 Clockwise Visibility

Let p be a concave corner of P. Suppose we traverse the segments of P clockwise (i.e., with the interior always on our right). When we reach p, we have to turn left. Before doing so, we shoot a ray forward (into the interior of P), and mark by p^+ the point at which this ray first hits another segment of P.

Let s be the segment containing p^+ and orthogonal to the ray. We say that p is *positively-visible* to

$(P, C) \leftarrow (P, \phi)$
While $P \neq \phi$

  Select a continuator s in P
  If *balcony*(s) is not covered by C (s is essential)

    $(P, C) \leftarrow (P, C + \{s\})$

  $(P, C) \leftarrow (P - \textit{balcony}(s), C)$
  If *lounge*(s) is an one-knob continuator

    Find a set $E \subset s$ of erasable points in (P, C)
    $(P, C) \leftarrow (P - E, C)$

Algorithm C: Using balconys to determine Essential/Erasable

s. Finally, if the corner p', appearing just before p, is convex, then p^+ is called the *positive semi-visible* point of p' (see Figure 3.3).

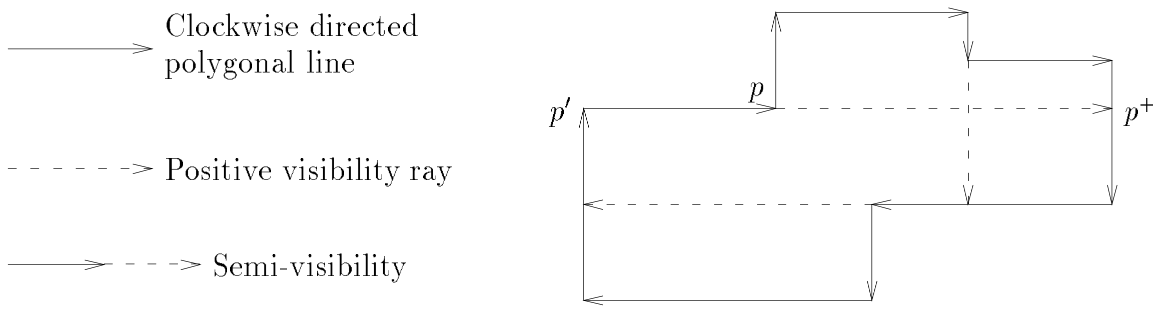


Figure 3.3: Clockwise visibility and semi-visibility

3.3.2 Counterclockwise Visibility

All definitions are similar, replacing left with right, right with left, p^+ with p^-, positively-visible with *negatively-visible*, and positive semi-visible with *negative semi-visible*.

3.3.3 Visibility Distances

We say that a point p is *visible* to a segment s if s contains a point p' such that pp' is orthogonal to s and internal to the polygon. Obviously, a concave corner is visible to a segment s iff it is either positively or negatively visible to s. The *visibility distance* of p from s is $|pp'|$.

3.4 Verifying Continuators

During execution of our algorithm, the polygon changes dynamically. Existing continuators are erased, and others may appear. Therefore, we need to test whether a given knob K is part of a continuator. This is done in terms of visibility.

To check whether a knob K is part of a continuator, we first find the longest chain of knobs containing K and having the same length. If one of the segments adjacent to the chain is a knob shorter than $|K|$, the answer is obviously negative. If the square containing the given knob is indeed a continuator, then all the knobs on the chain must be edges of the square, and the number of those defines the *continuator type*.

In the case of a four-knob continuator, verification is trivial, since the entire polygon is a square. The answer is obviously positive.

For the case of three-knob continuator verification (see Figure 3.4), let $ABCD$ be the square containing the given knob, s.t. AB, BC, and CD are its knobs. Let A_{prev} be the corner just before A, and let D_{next} be the one just after D. The answer is positive iff $A_{prev}^{-} = D_{next}$ (i.e., A_{prev} sees D_{next}.)

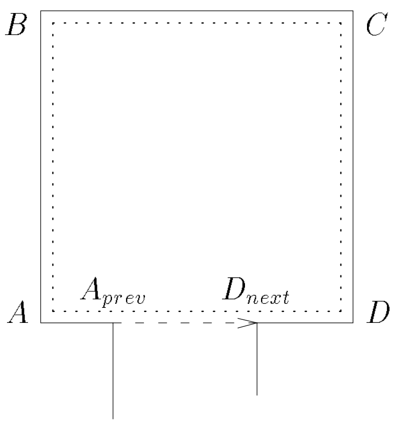


Figure 3.4: Three-knob continuator verification: $A_{prev}^{-} = D_{next}$

For the case of two-knob continuator verification (see Figure 3.5), let $ABCD$ be the square containing the given knob, s.t. AB and BC are its knobs. Let A_{prev} be the corner just before A, and let C_{next} be the one just after C. Finally let a be the length of AB. The answer is positive iff

- $(|AA_{prev}^{-}| > a)$ and $(|CC_{next}^{+}| > a)$,
- $(|AA_{prev}^{-}| = a)$ and $(|CC_{next}| \geq a)$, or
- $(|AA_{prev}| \geq a)$ and $(|CC_{next}^{+}| = a)$.

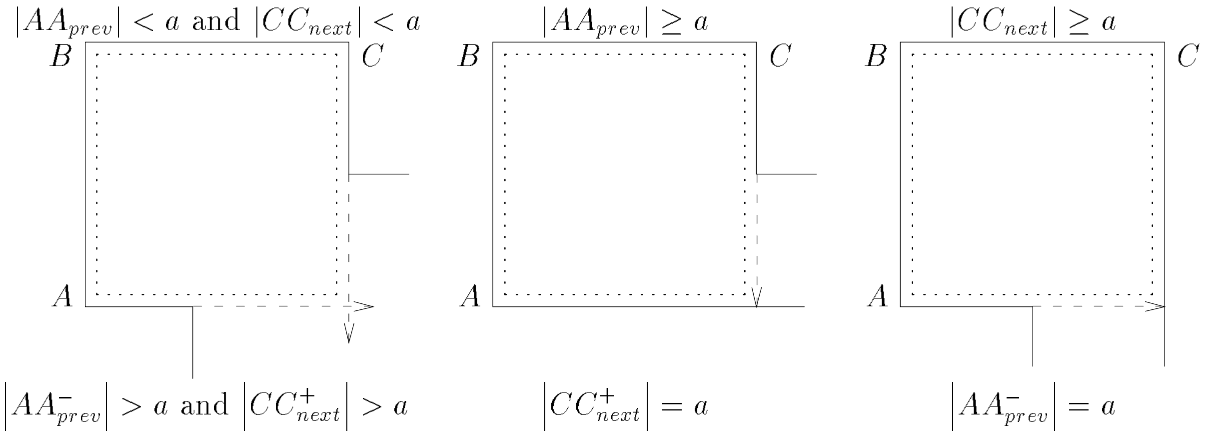


Figure 3.5: Two-knob continuator verification

Finally, for the case of one-knob continuator verification (see Figure 3.6), let AB be the given knob, and let d be the distance to the closest point among the points visible to AB, the point semi-visible to A, and the point semi-visible to B. The answer is positive iff $d > |AB|$.

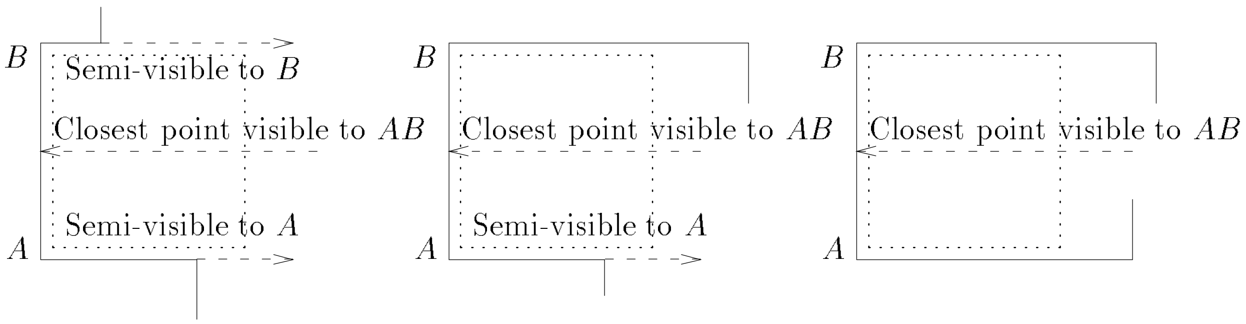


Figure 3.6: One-knob continuator verification

3.5 The Erasable Region in a One-Knob Continuator

Recall that the erasure part of our algorithm consists of two stages, the first erasing the balcony of the selected square and the second erasing points in its lounge, assuming it is an one-knob continuator. Erasing the balcony is straightforward. What is still needed is a criterion for erasing points an one-knob continuator.

Let AB be the knob of a non-essential one knob continuator. We need to find a set E of erasable points. The set we select is a rectangle $ABCD$ satisfying the following conditions (see Figure 3.7):

1. All the points in $ABCD$ are covered.
2. The edges BC and DA are entirely exposed (i.e., contained in the polygon border).
3. Let d be the distance of the closest point visible to AB, and denote the distance $d - |AB|$ by $SecurityDistance(AB)$. The rectangle width, $|BC|$, must be within the security distance, i.e., $|BC| \leq d - |AB|$.

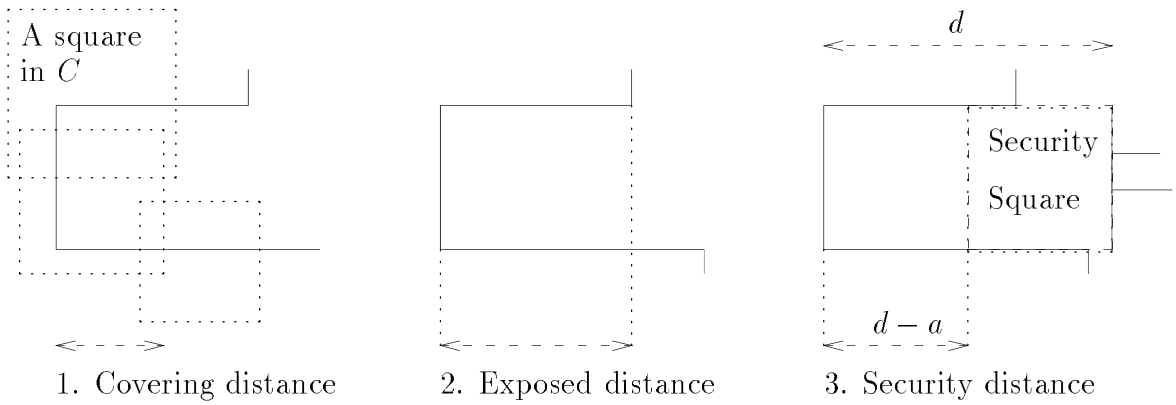


Figure 3.7: One-knob erasable distance = min{Covered distance, Exposed distance, Security distance}

Claim 3.3 *All the points in $ABCD - CD$ are erasable.*

Proof: Consider a maximal square s containing CD and not intersecting $ABCD - CD$. By condition (3), such a square exists. Let s' be a square intersecting $ABCD - CD$. By condition (1), $s' \cap ABCD$ is covered, and by condition (2), $s' - ABCD \subset s$. This implies that $ABCD - CD$ is erasable. □

3.6 Termination in a Linear Number of Iterations

In order to prove the termination of Algorithm C, we use a charging (amortization) method. We charge every iteration either to an input vertex or an output square. Since the input and output size are both finite (an orthogonal polygon can be covered by a finite number of rectangles, each coverable by a finite number of squares), it is sufficient to show that each item, input or output, is charged for a constant number of iterations.

By Lemma 2.4, a continuator s always exists. If s has more than one knob (2, 3 or 4), after erasing its balcony, at least one (original) concave point disappears. This is an input point, and therefore we may charge it for this iteration. Otherwise, let s be the one knob continuator, and let AB be its knob. Following the definitions of the previous section, recall that we erase a rectangular region $E = ABCD$ satisfying conditions (1), (2), and (3). If the security distance (from condition (3)) is larger than the square dimension $|AB|$, then conditions (1) or (2) bound the width of E. In the first, erasing E causes an output square not to intersect the residual polygon anymore; this square won't be referred to anymore, and so we charge it. In the second, when condition (2) bounded E, a concave corner in the input was erased as well, and it is charged for this iteration.

We are left with the case where $SecurityDistance(AB) \leq |AB|$. After the erasure, CD becomes a knob in the residual polygon. Let $s' = CDEF$ be the maximal rectangle containing the knob CD; since $|DE| = SecurityDistance(AB)$, s' is a square. Unless $s' = P$ (which only occurs in the last iteration), the exposed border of s' contains at least one concave point. This is an input point, and it is temporarily charged for the iteration. This point can not appear in more than two such "security squares" (one for each visibility direction). A security square can not change due to one-knob continuator erasure, and therefore it can only change after erasure of a balcony if it becomes a continuator. When this happens, an input concave vertex disappears. This vertex will pay for the current iteration.

The charging approach described above shows that the total number of iterations made during the algorithm's execution is bounded by $O(n + k)$, where k is the output size.

4 A Basic Implementation

The previous section outlined the basics of our algorithm, which can be completed in $O(n + k)$ iterations. However, to fully implement it, we need a data structure capable of performing all the neccessary operations. This data structure will be described in this section.

The queries posed by the algorithm are of two types: visibility distance queries, and covered area measurement. Our data structure will also have to adapt to deletions of rectangular regions from the edge of the polygon.

Our final goal is to describe an algorithm using $O(1)$ amortized time for each iteration. We shall start, however, with a simplistic data structure which will provide answers to visibility queries in $O(n)$ time, and to covered area queries in $O(1)$ time, thus solving the problem in quadratic time.

This algorithm is demonstrated in Figure 4.3.

4.1 Basic Visibility Data Structure

The basis of the data structure is the residual polygon information, which consists of a list of vertices connected by segments. These are stored in a doubly-linked list of segments, each pointing to its two endpoints. In addition, each segment keeps a pointer to a doubly-linked list of all vertices visible to it (the *projection list*). Every vertex "knows" its location and its position in the two projection lists it participates in (see Figure 4.1). For simplicity, we shall refer to the residual polygon as P, and to its cover information as C.

There are two types of distance queries in the algorithm:

- Given a vertex of P, what is the distance to the nearest segment visible in a certain (orthogonal) direction?
- Given a segment of P, what is the distance to the nearest vertex visible to it?

Both queries can be answered using a simple scan of a single projection list. The projection lists themselves are only changed on erasing parts of the polygon, and the only changes possible are union (concatenation), which occurs when a segment is deleted after a knob is moved into the polygon, and deletion of points. Since the lists are doubly linked, and since every point has a link to its appearences in the projection lists, both operations can be performed in $O(1)$ time.

4.2 Covered Regions Information

There are two steps where the algorithm utilizes the covered region information; these are (1) checking if the balcony of a given square is covered, and (2) determining the erasable region in an one-knob continuator. Note that both query types only concern those cover squares which are adjacent to (or partially overlapping) the uncovered part of the polygon.

In order to answer these queries, the data structure holds, for each segment of the residual polygon, a list of all squares s selected for the cover and having the following properties (see Figure 4.2):

- s intersects the segment;
- The two edges of s which are orthogonal to the segment are exposed;

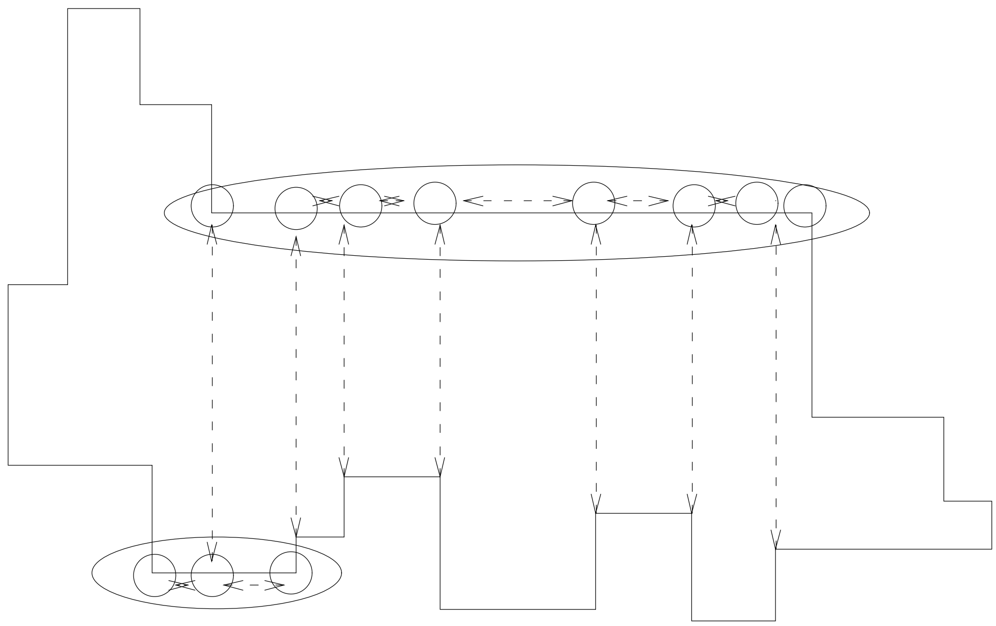


Figure 4.1: A visibility data structure for two segments

- There is a point in P covered only by s (and not by any other square selected so far).

The squares for each segment are kept sorted by their appearence order. Since every intersection of a square with the segment is on an original concave point, the squares on each list are mutually disjoint.

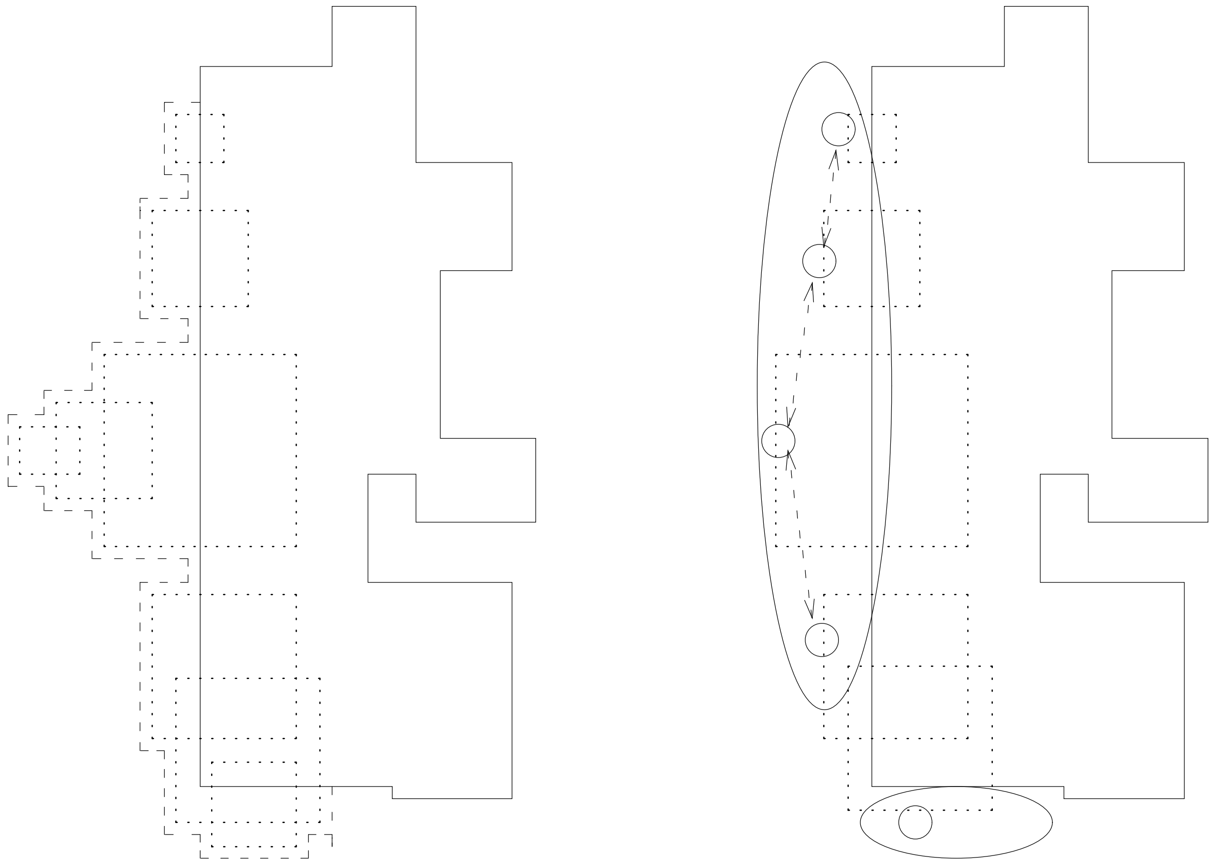
Initialy, this list is empty. When a new square is added to the cover, the list attached to its knob is replaced by the square, and the squares on the neighboring segments' lists which intersect it are removed from the lists. the knob direction and out. The scan stops at the first square that does not intersect the new one, while every square on the way is removed.

4.3 The Simple Algorithm

We are now in a position to present the basic algorithm, in a way that can be understood by a computer.

Initially, our data structure will hold the original polygon's vertices and their visibility information. We shall then test each knob to see whether it is part of a continuator, and keep a list of all the continuators found so far.

At each iteration of the algorithm, we will take a single candidate continuator from our list. We will then test if the square's balcony is covered — this can be done in $O(1)$ time, as no knob can contribute more than a single square to the balcony cover (if a knob has more than one square on its



Cover Squares in the original polygon

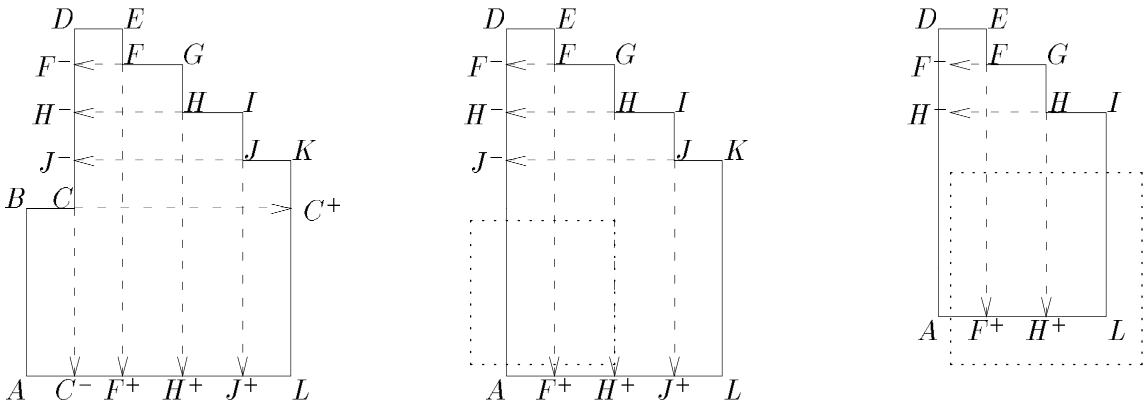
Linked list of cover squares

Figure 4.2: Covered areas data structure

list, the balcony is obviously not covered). The square will be added to the cover iff its balcony is not covered as yet.

Next, the square's balcony will be erased, possibly leaving a new one-knob continuator (if the continuator was one-knob to begin with, this is the original continuator). We will now have to find and erase a region of this continuator; this can be done in linear time using the results of Section 3.5. During erasure, it is possible that original concave points will be deleted; each segment effected by this will be checked again to see if it has become part of a continuator.

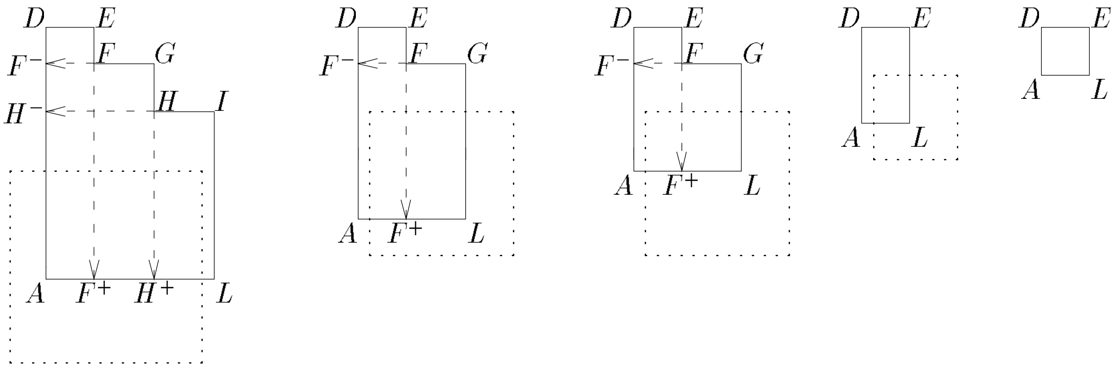
Since we have shown (see Section 3.6) that the algorithm terminates in a linear number of iterations, and since each iteration as described here can be done in linear time, this algorithm solves the problem in $O(n^2)$ time.



1.AB's projection list is empty
Semi visibility: $|BC| + |CC^+| > |AB|$
AB is a 1-knob continuator

2.Add square 1 to the cover
Erase region $ABCC^-$
(Exposed distance)
Concatenate lists of AB and CD
AL, LK are a 2-knob continuator

3.Add square 2 to the cover
Remove square 1 from AD's list
Erase the balcony
Update the projection lists



4.Erase security distance
AL and LI are a 2-knob continuator
...and so on...

Figure 4.3: The simple algorithm

5 An Efficient Visibility Structure

As noted in the previous section, both kinds of visibility queries posed by the algorithm need a linear amount of time to answer. In this section, we aim to reduce the time taken for these operations.

5.1 Efficient Segment Visibility

We shall first try to reduce the time needed for queries of the second kind introduced in the previous section; that is, for a given segment of P, find the vertex having the smallest visibility distance to it. To do this, we employ the result of Lemma 2.5: "There always exists a continuator in P which sees at most one antiknob."

Our improvement will therefore effect both the algorithm and its supporting data structure. First, we shall define a knob as *good* if it is part of a continuator *and* it sees no more than one antiknob. The selection step of the algorithm is now restricted to select only good knobs at each iteration, and replace the simple check for continuators by a check of "goodness". We also add to the data structure a new list for each segment; this is the list of all anti-knobs seen by this segment. Each vertex in the visibility list, which is part of an antiknob, will point to its associated entry in this list. Deletion of a vertex will now update both the projection lists of the segments seeing it and their antiknob lists; this can still be done in constant time. Also, whenever the projection lists of two segments are concatenated, their antiknob lists are also concatenated.

The purpose of restricting our focus to these so-called "good" knobs is that the nearest visible vertex for these can be found very easily; it can be either one of the vertices in the antiknob, or it can be one of the vertices at the ends of this knob's projection list (see Figure 5.1). Therefore, we can now answer one type of distance queries posed by the algorithm in constant time.

5.2 Efficient Vertex Visibility

We are now left with the other kind of visibility queries; these take the form, "given a vertex of P, what is the distance to the nearest segment in a given direction?" Solving this problem is a little more difficult than the previous one.

As a starting point, we can keep, for each visibility point, the distance it originally had from the segment opposing it. However, it is impossible to keep these distances updated throughout execution of the algorithm, as whenever a region is erased and a knob is moved into the polygon, we may need to update up to $O(n)$ visibility points' distances. This causes the saved value to become inaccurate as the algorithm progresses.

A closer look at the algorithm shows that these queries are only posed when calculating semi-visibility distances; we shall now see what the inaccuracy means for each possible query. For four- and three-knob continuators, there are no semi-visibility queries at all. For two-knob continuators (see Figure 3.5, there are three cases:

1. $(|AA^{-}_{prev}| > a)$ and $(|CC^{+}_{next}| > a)$. An error in the semi-visibility distance here cannot matter much, since the segment containing (say) A^{-}_{prev} cannot move into the square $ABCD$ without causing A_{prev} to be erased and violating the conditions for this continuator type.

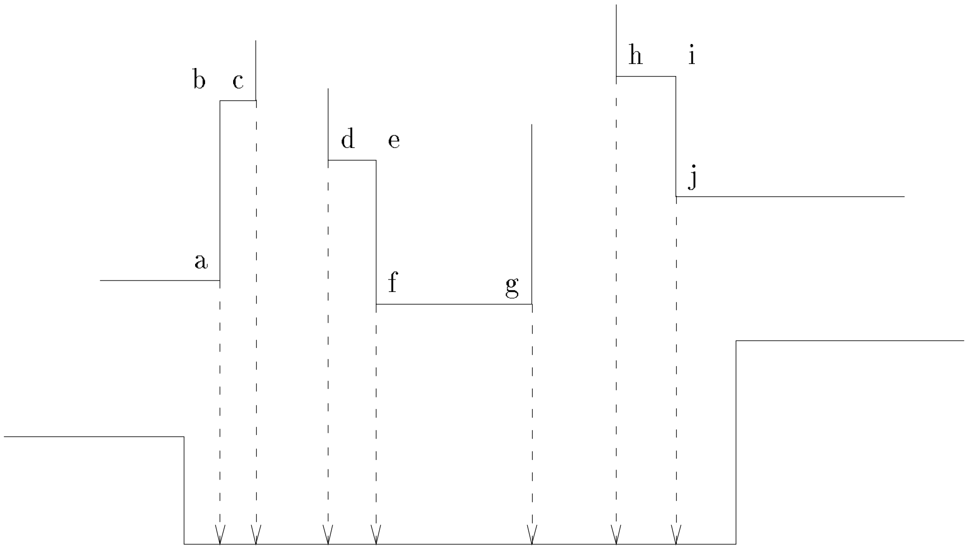


Figure 5.1: Segment visibility with a single antiknob

2. ($|AA^-_{prev}| = a$) and ($|CC_{next}| \geq a$). Here, we can simply replace the semi-visibility distance test by a test that C_{next} is the first vertex on the projection list of AA_{prev}.

3. ($|AA_{prev}| \geq a$) and ($|CC^+_{next}| = a$). This case is symmetric to the previous one.

With one-knob continuators (see Figure 3.6), however, the situation becomes more complex. The only case where semi-visibility is required while testing an one-knob continuator is in when it has an 'omega shape', i.e., both A_{prev} and B_{next} are concave. In this case, it is possible that there are no vertices visible to the knob, and also the saved value does not reflect the current distance of the visible segment; see Figure 5.2.

The only reliable information we could have had about the true distance were the projection list entries for p_1 and p_2, which are now deleted. To achieve this, we add to each segment's data a list of 'shadows' of erased vertices which had been visible to this segment; this list is interleaved with the list of visible vertices (see Figure 5.3), so that deletions still take $O(1)$ time. When we merge two segments, we will also merge their shadow lists; when we erase a knob, we will delete shadows which are now outside the polygon (see Figure 5.4).

These shadow points will come into use when we need to verify an omega-shaped one-knob, and we have no direct visibility information. On such a case, we will scan all of the shadows, and take the nearest of them as the distance to the nearest segment, erasing all the rest of them (note that this is a segment-to-segment distance calculation, and not segment-to-vertex like the visibility distances). Since each vertex can contribute at most two shadows, and each shadow is only used once, we can make the vertex pay for uses of its shadow and leave the amortized time complexity of the algorithm at $O(n)$.

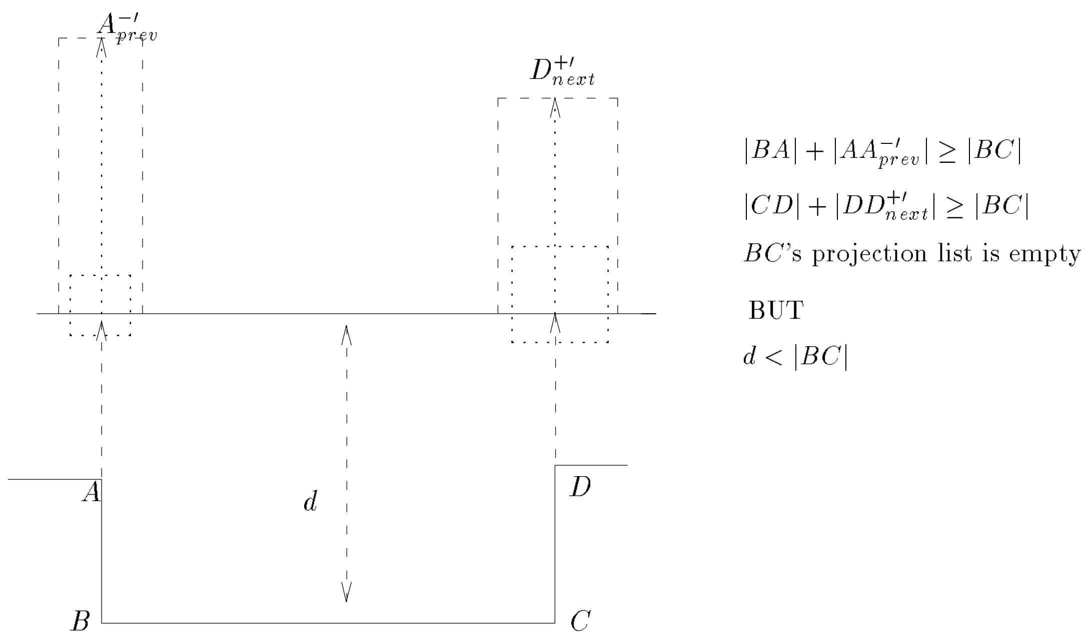


Figure 5.2: The Omega-shape problem

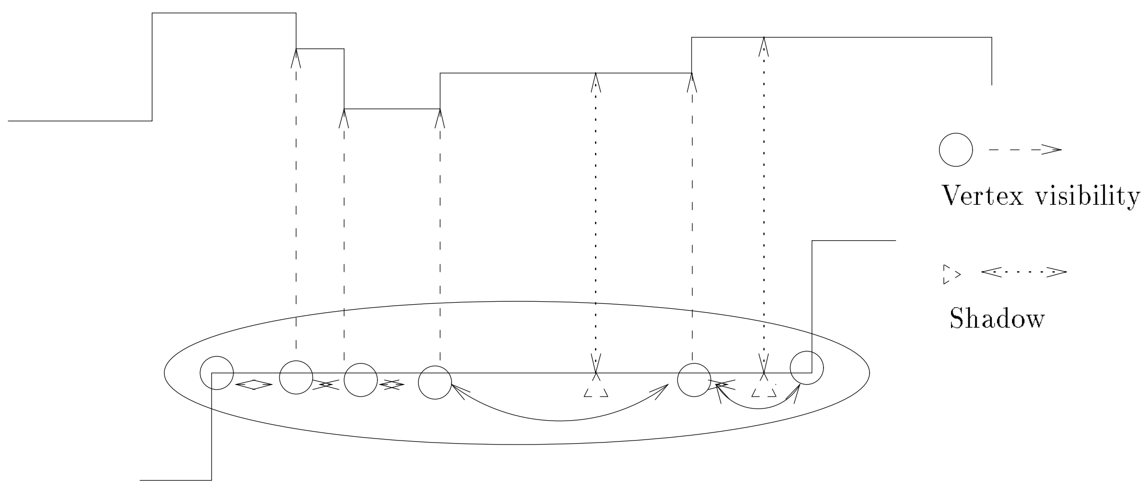


Figure 5.3: A segment with visible vertices and shadows

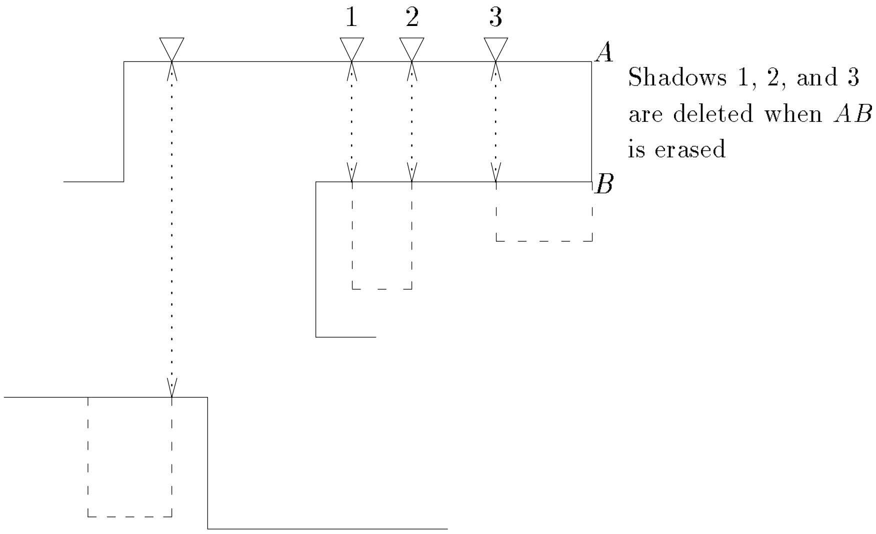


Figure 5.4: Deletion of shadows on knob erasure

The reason for our saving all the shadows and deleting only when we test can be seen from Figure 5.4; had we kept only the nearest shadows, after erasing knob ??, we would be in the same situation we were without shadow information. It is valid to erase the shadows when we do because when there are no more vertices visible to a segment, its shadow list will remain static, so only the nearest matters.

6 Acknowledgments

We would like to thank Prof. Alon Itai for introducing us to the problem, and Mr. Ziv Alon for the actual implementation of the program and for his aid in writing the article.

References

[ACKO88] L. J. Aupperle, H. E. Conn, J. M. Keil, and J. O'Rourke. Covering orthogonal polygons with squares. In 26^{th} *Annual Allerton Conf. on Comm., Control and Computing*, Sept. 1988.

[AO81] O. M. Alberton and C. J. O'Keefe. Covering regions with squares. *SIAM J. Disc. Math.*, 2:240–243, 1981.

[BYE85] R. Bar-Yehuda and S. Even. A local ratio theorem for approximating the weighted vertex cover problem. *Annals Disc. Math.*, 25:27–46, 1985.

[Cha90] B. Chazelle. Efficient polygon triangulation. Technical Report CS-TR-240-90, Dept. of Comput. Sci., Princeton Univ., Feb. 1990.

[Gav72] F. Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques and maximum independent set of chordal graphs. *SIAM J. Comput.*, 1:180–187, 1972.

[HS79] G. M. Hunter and K. Steiglitz. Operations on images using quad-trees. *IEEE Trans. Pattern Matching Machine Intell.*, PAMI-1,2, 1979.

[Mor88] D. Morita. Finding a minimal cover for binary images: an optimal parallel algorithm. Technical Report TR #88-946, Department of Computer Science, Cornell Univ., Ithaca, NY 14853, Nov. 1988. Information System Lab. 89CRD149, Aug. 1989. To appear in Algorithmica.

[NT75] G. L. Nemhauser and L. E. Trotter, Jr. Vertex packing: Structural properties and algorithms. *Mathematical Programming*, 8:232–248, 1975.

[O'R87] J. O'Rourke. *Art Gallery Theorems and algorithms.* Oxford University press, New York, 1987.

[RK82] A. Rosenfeld and A. C. Kak. *Digital Picture Processing.* Academic Press, New York, 1982.

[SI86] D. S. Scott and S. Iyenger. Tid - translation invariant data structure of storing images. *Commun. ACM*, 29:418–428, 1986.

[SXL87] L. Shu-Xiang and M. H. Loew. The quad-codes and its arithmetic. *Commun. ACM*, 30:621–625, 1987.

[WBR86] A. Y. Wu, S. K. Bhaskar, and A. Rosenfeld. Computation of geometric properties from the medial axis transform in $o(n \log n)$ time. *Comput. Vision, Graphics, and Image Processing*, 34:76–92, 1986.