Final Report of the Major Research Project

Generation, Decomposition and Analysis of the Isothetic Polygon in Digital Geometric Paradigm

Dr. Arindam Biswas

(Principal Investigator)

Dr. Hafizur Rahaman

(Co-Principal Investigator)

Department of Information Technology
Indian Institute of Engineering Science and Technology, Shibpur

ज्ञान - विज्ञानं विमुक्तये

University Grant Commission
Bahadurshah Zafar Marg
New Delhi – 110002

Contents

Chapter 1

Introduction

Isothetic polygon has an important role in the field of image processing and shape analysis. The generation of algorithms for construction of isothetic polygon is significant in the domain of digital geometry. The isothetic polygon has applications in different fields, which leads to development of various algorithms for analysis of the isothetic polygon. It is an emerging area of interest among the researchers and engineers from Computer Science and Information Technology. Study of isothetic polygons may find a special role in Robotics for path planning, object gripping etc. The algorithms discussed here are applied on digital objects for its shape analysis.

The *family of shortest isothetic paths* (FSIP) between two grid points in a digital object A is the union of all possible *shortest isothetic paths* (SIPs) between them. A novel algorithm to compute the FSIP between two grid points of the *inner isothetic cover* P that inscribes the object A, is proposed here. The algorithm is based on certain combinatorial rules derived from the characterization of FSIP. A preprocessing is done at first with the grid points comprising the border B_P of P in $O(n \log n)$ time, n being the number of points in B_P. Then, for any pair of grid points in P, the algorithm computes their FSIP in $O(n)$ time. As shown in this chapter, the FSIP can be useful for different shape-related applications, such as finding a *critical region* inside an object. Experimental results show the effectiveness of the algorithm and its further prospects in shape analysis and related applications.

The construction of a minimum-area geometric cover of a digital object is important in many fields of image analysis and computer vision. We propose here the first algorithm for constructing a minimum-area polygonal cover of a 2D digital object as perceived on a uniform triangular grid. The polygonal cover is triangular in the sense that its boundary consists of a sequence of edges on the underlying grid. The proposed algorithm is based on certain combinatorial properties of a digital object on a grid, and it computes the tightest cover in time linear in perimeter of the object. We present experimental results to demonstrate the efficacy, robustness, and versatility of the algorithm, and they indicate that the runtime varies inversely with the grid size.

This report is organized as follows. Chapter 2 contains the algorithm to construct family of shortest isothetic paths between two grid points inside a digital object. The family of shortest isothetic paths is unique, which contains the region through which all possible SIPs can pass. The construction of outer triangular cover is discussed in chapter 3. First the object is imposed on background triangular grid. By using some combinatorial rules outer triangular cover of the object is obtained, which tightly covers the object. The application of triangular cover in shape analysis is presented in 4. The algorithm to determine largest rectangle and its corresponding rectangularization is proposed in 5. Concluding remarks are given in chapter 6.

Chapter 2

On Finding the Family of Shortest Isothetic Paths inside a Digital Object

[1]Dutt, M., A. Biswas, P. Bhowmick, and B. B. Bhattacharya (2014). On the Family of Shortest Isothetic Paths in a Digital ObjectAn Algorithm with Applications. Computer Vision and Image Understanding 129, 75-88.

2.1 Introduction

Shortest path algorithms and their numerous applications in various fields have been studied extensively over several decades in different algorithmic paradigms, such as graph theory, computational geometry, and operations research. Some of the worth-mentioning applications of shortest paths are networking, robotics, GIS, VLSI design, resource allocation and collection, TSP with neighborhoods, etc. [2, 17, 27, 34, 53, 60, 61]. Due to such a vast range of applications, the variation in constraint formulation and customization has also become quite engrossing, as evidenced in the related literature [23, 24, 25, 18, 32, 40, 41, 42, 55]. Shortest paths are also found to be useful in image processing and computer vision. For example, in [26], different deformed templates, such as deforming contours, articulated objects, and smooth contours have been tracked in simple and complicated backgrounds using the idea of shortest path. Nevertheless, the design of such algorithms still poses newer challenges in various emerging areas of image analysis and computer vision. This mandates further investigation of application-specific shortest-path problems in the framework of digital geometry.

The work in [28] is focused on finding a *shortest isothetic path* (SIP) inside a digital object A (without any hole) laid on a grid $\mathbb{G}$ (Sec. 3.2), such that the SIP lies entirely inside A and consists of moves along grid edges only. Although the problem has some similarity with certain problems of computational geometry, no efficient algorithmic solution is available for the shortest isothetic path problem till date. For a given grid size, the mentioned algorithm runs in $O(n \log n)$ time, n being the number of grid points on the border of the inner isothetic cover P, which is the maximum-area isothetic polygon inscribing A [7, 8].

[1]*

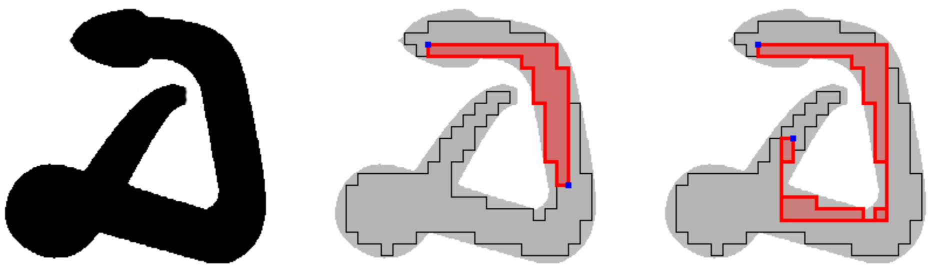


Figure 2.1: Family of SIPs (shown in red) between two grid points (in blue color). **Left:** A digital object, A. **Middle:** All possible SIPs in the FSIP are monotone. **Right:** All possible SIPs in the FSIP are non-monotone. The boundary B_P of the inner isothetic cover P of A is shown by black lines. The infimum and the supremum SIPs defining the boundary of each FSIP are shown by thick red lines, which sometimes overlap with some portions of B_P.

A shortest isothetic path—whether monotone or not—between two points is usually not unique. Hence, the focus of the work proposed here is to find the (unique) *family of shortest isothetic paths* (FSIP) between two points. The FSIP between two points is the region containing all possible SIPs between them. The definitions and terminologies related to FSIP are given in Sec. 3.2. An FSIP possesses some interesting topological properties, which are derived and explained in Sec. 2.3. Based on these properties, the algorithm for computing the FSIP is designed, which has been described and demonstrated in Sec. 2.4. The nature of an FSIP depends on the complexity of its SIPs. As an introductory example, see Fig. 3.1. In Fig. 3.1(left), a digital object, A, is shown, and two FSIPs for two pairs of points in A are shown in Fig. 3.1(middle, right). All SIPs between the two points in Fig. 3.1(middle) are monotone, and those in Fig. 3.1(right) are non-monotone. Further experimentation and related results with detailed explanations are given in Sec. 5.5.

2.2 Definitions and Preliminaries

A *digital object*, A (henceforth referred to as an object), is a finite subset of $\mathbb{Z}^2$, which consists of one or more $k(= 4$ *or* $8)$*-connected components* [38]. In this chapter, the considered object is a single 8-connected component. The following definitions are required in the context of this work.

Definition 1 (Grid)**.** *The* background grid *is given by* $\mathbb{G} = (\mathbb{H}, \mathbb{V})$*, where* $\mathbb{H}$ *and* $\mathbb{V}$ *represent the respective sets of (equi-spaced) horizontal grid lines and vertical grid lines. The* grid size*,* g*, is defined as the distance between two consecutive horizontal/vertical grid lines. A* grid point *is the point between intersection of a horizontal and a vertical grid line.*

Definition 2 (Inner Isothetic Cover)**.** *An* isothetic polygon *has its vertices as grid points and its edges lying on grid lines. The* inner isothetic cover *of* A *is the maximum-area isothetic polygon* P *that inscribes* A *(Fig. 3.1).*

Definition 3 (SIP)**.** *An (simple)* isothetic path $\pi_{(p,q)}$ *from a grid point* $p \in P$ *to a grid point* $q \in P$ *is a sequence of 4-connected grid points such that all the constituent points of* $\pi_{(p,q)}$ *are distinct and lie on or inside* P*. For two given points in* P*, a/the* shortest isothetic path *(SIP) has the minimum*

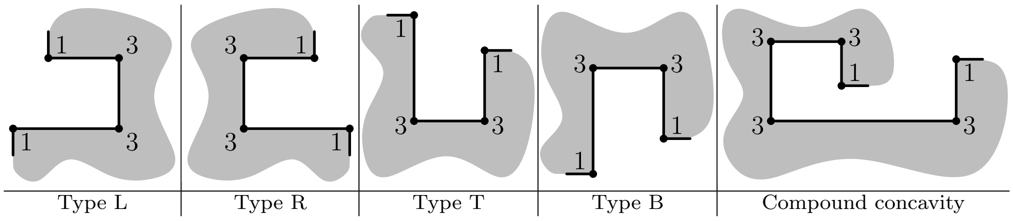


Figure 2.2: Different types of concavity (polygon interior shown in gray).

length over all isothetic paths between them. A SIP is said to be monotone *if it consists of moves only in one or two (orthogonal) directions.*

Definition 4 (FSIP)**.** *The union of all possible SIPs between two given points $p \in P$ and $q \in P$ defines their* family of SIPs *(FSIP), denoted by $\mathcal{F}_{(p,q)}$. The set $\mathcal{F}_{(p,q)}$ is equivalent to a (isothetic) region bordered by two SIPs between p and q, which are said to be its* extremum SIPs*. If the border of $\mathcal{F}_{(p,q)}$ is traversed from p to q such that each other SIP in $\mathcal{F}_{(p,q)}$ lies to the left during the traversal, then that path of traversal is one of the two* extremum *SIPs, which is termed as the* infimum SIP *and denoted by $\pi^{\inf}_{(p,q)}$. Similarly, on traversing the border of $\mathcal{F}_{(p,q)}$ from q to p with each other SIP lying to the left during the traversal, the other* extremum *SIP is obtained, termed as the* supremum SIP *and denoted by $\pi^{\sup}_{(p,q)}$.*

The set $\mathcal{F}_{(p,q)}$ is, therefore, bounded and well-defined by its two extremum SIPs, namely $\pi^{\inf}_{(p,q)}$ and $\pi^{\sup}_{(p,q)}$. Figure 3.1 shows a couple of typical examples. The crux of this algorithm is to find $\pi^{\inf}_{(p,q)}$ and $\pi^{\sup}_{(p,q)}$, which, in turn, defines the region that contains all the SIPs comprising $\mathcal{F}_{(p,q)}$. The numbers of grid points for all SIPs in $\mathcal{F}_{(p,q)}$ are same, as they all have the minimum length. As a degeneracy, if there is exactly one SIP between p and q, then $\mathcal{F}_{(p,q)}$ contains only that SIP, which is also $\pi^{\inf}_{(p,q)}$ and $\pi^{\sup}_{(p,q)}$ coincident with each other. Otherwise, $\mathcal{F}_{(p,q)}$ will have distinct $\pi^{\inf}_{(p,q)}$ and $\pi^{\sup}_{(p,q)}$, which although can have partial overlaps. The following definitions are needed to characterize such overlaps between $\pi^{\inf}_{(p,q)}$ and $\pi^{\sup}_{(p,q)}$, and hence to characterize $\mathcal{F}_{(p,q)}$ defined by them.

Definition 5 (Convexity and Concavity)**.** *Since P is an isothetic polygon, it would have 90^0 and 270^0 vertices, which are referred to as vertices of type* **1** *and type* **3***, respectively. The sequence of vertices of P is such that P always lies left of each edge during its traversal. In this vertex sequence, a pair of consecutive vertices of type* **1** *gives rise to a* convexity*, whereas a pair of consecutive vertices of type* **3** *gives rise to a* concavity*. The straight line passing through two consecutive vertices of type* **1** *(type* **3***) is called the* convexity line *(*concavity line*).*

A vertex pattern of **1331** *may occur in four possible ways w.r.t. orientation: Type L (left), Type R (right), Type T (top), and Type B (bottom) (Fig. 2.2). If there are more than two consecutive vertices of type* **3***, then the concavity is termed as* compound concavity*.*

Definition 6 (Convex Region)**.** *A* convex region *C corresponding to a convexity line l is the maximum-area rectangle fully contained in P, having its one side as l, and the endpoints of the side opposite to l (i.e., its* open side*) lying on the border of P.*

For example, in Fig. 2.3(left), the convex region is closed on three sides by the edges (v_{i-2}, v_{i-1}), (v_{i-1}, v_i), and (v_i, v_{i+1}), and open on the side defined by the horizontal line passing through v_{i-2}. In Fig. 2.3(right), the convex region is closed similarly on three sides by the edges as in Fig. 2.3(left), and it is open on the side defined by the nearest concavity line.

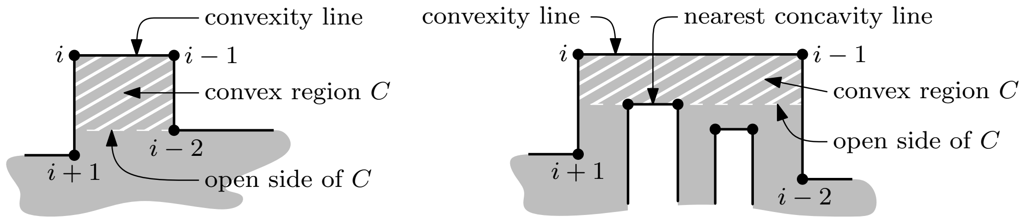


Figure 2.3: Instances of convex region (hatched in white) defined by convexity line v_iv_{i-1} on one side and nearest vertex v_{i-2} or v_{i+1} (left) or nearest concavity line (right) on the opposite side. Here, a vertex v_i is denoted by i for simplicity.

Definition 7 (Convex Isothetic Polygon)**.** *A convex isothetic polygon is an isothetic polygon whose intersection with any grid line is either empty or a single line segment [10].*

As evident from Definitions 4 and 7, and from examples shown in Figure 3.1, an FSIP in general can also be perceived as a *set of convex isothetic polygons*, which are connected by *grid edges.* For example, in Figure 3.1(right), the FSIP has four convex isothetic polygons joined by grid edges in succession. The following definitions are used to characterize an FSIP.

Definition 8 (Convexity Index)**.** *The* convexity index*, α, of $\mathcal{F}_{(p,q)}$ is defined as the number of convex isothetic polygons that it consists of.*

For example, for the FSIP shown in Fig. 3.1(middle), $\alpha = 1$; and for the one in Fig. 3.1(right), $\alpha = 4$.

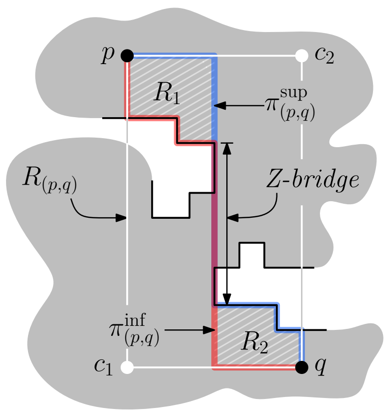
Definition 9 (Bridge)**.** *If $\mathcal{F}_{(p,q)}$ consists of two or more convex isothetic polygons, then every two consecutive polygons are connected by a sub-path common to $\pi^{\inf}_{(p,q)}$ and $\pi^{\sup}_{(p,q)}$, which is called a* bridge*.*

Each *bridge* in $\mathcal{F}_{(p,q)}$ lies on one or more (collinear) lines of concavity, with its two endpoints coinciding with two of the endpoints (type **3** vertices) of those concavity line(s). Each SIP in $\mathcal{F}_{(p,q)}$ passes through all the bridges in $\mathcal{F}_{(p,q)}$.

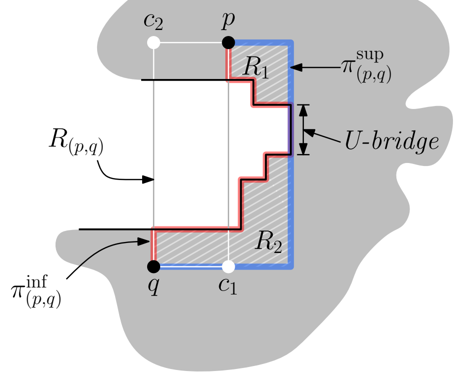
Bridges are of two types, namely *Z-bridge* and *U-bridge*, as shown in Fig. 2.4, which have the following characterization. W.l.o.g., let a bridge be vertical. If the directions of horizontal movements of a SIP preceding and following this bridge are same, then the corresponding bridge is termed as *Z-bridge* (Fig. 2.4(left)). But if the aforesaid directions are opposite, then the corresponding bridge is termed as *U-bridge* (Fig. 2.4(right)). A *Z-bridge* does not break the monotonicity of a SIP, whereas a *U-bridge* always breaks the monotonicity of a SIP. In other words, the presence of a *U-bridge* in a SIP implies that the SIP is non-monotone.

Definition 10 (Monotone and Non-Monotone FSIP)**.** *If $\mathcal{F}_{(p,q)}$ consists of monotone SIPs, then it is said be a* monotone FSIP*; otherwise, it is a* non-monotone FSIP*.*

It is easy to observe that $\mathcal{F}_{(p,q)}$ cannot contain both monotone and non-monotone SIPs, since the length of a non-monotone SIP between p and q is always greater than that of a monotone SIP between p and q.



A *Z-bridge* that connects R_1 and R_2, lying on its two opposite sides.



A *U-bridge* that connects R_1 and R_2, lying on the same side of the bridge.

Figure 2.4: Two types of *bridges*. Note that in both cases, $\mathcal{F}_{(p,q)}$ consists of R_1, R_2 (which are two convex isothetic polygons), and the bridge joining them. The bridge is the common sub-path between $\pi^{\inf}_{(p,q)}$ and $\pi^{\sup}_{(p,q)}$.

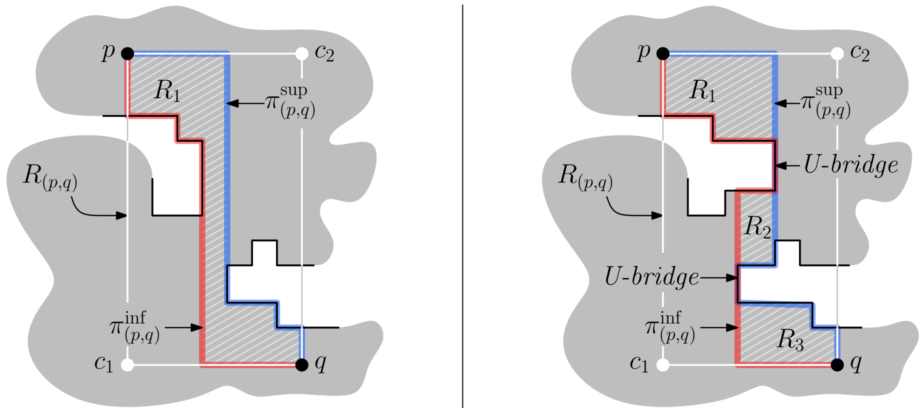
2.3 Characterization of FSIP

It is evident from Definition 4 that $\mathcal{F}_{(p,q)}$ may be conceived as an isothetic polygon, which is either *simple* or *non-simple* in the sense that it may have intersecting edges. The intersecting edges, in particular, give rise to bridges. Let $R_{(p,q)}$ denote the rectangle having p and q as two of its diagonally opposite corners. Then the location of the aforesaid isothetic polygon corresponding to $\mathcal{F}_{(p,q)}$ has two possible topological cases, having a few combinatorial sub-cases, as explained below.

1. $\mathcal{F}_{(p,q)} \subseteq R_{(p,q)}$.
   (a) $\mathcal{F}_{(p,q)}$ is monotone.
      i. If $\mathcal{F}_{(p,q)}$ is simple, then it has no bridge (Fig. 2.5(left)).
      ii. If $\mathcal{F}_{(p,q)}$ is non-simple, then it has one or more Z-bridges but no U-bridge (Fig. 2.4(left)).

      These are proved in Lemma 1 and Lemma 2.
   (b) If $\mathcal{F}_{(p,q)}$ is non-monotone, then it is always non-simple, has one or more U-bridges (and may or may not contain any Z-bridge). This is proved in Lemma 2.
2. If $\mathcal{F}_{(p,q)} \nsubseteq R_{(p,q)}$, then it is non-monotone, non-simple, and has one or more U-bridges (may or may not contain any Z-bridge). This is evident from Fig. 2.4(right) and proved in Theorem 2.

When a SIP lies outside $R_{(p,q)}$, it passes through at least one U-bridge that lies outside $R_{(p,q)}$. The algorithm proposed in Sec. 2.4, while computing the *extremum* SIPs by traversing along the two consecutive sides of $R_{(p,q)}$, determines each such U-bridge lying outside $R_{(p,q)}$ and the corresponding isothetic convex polygons of $\mathcal{F}_{(p,q)}$ connected by it. See, for example, the illustration in Fig. 2.4(right). Here the line of concavity is outside $R_{(p,q)}$. Hence, for any $\pi_{(p,q)}$, there is a sub-path to the right side of $R_{(p,q)}$. The union of all these sub-paths over all SIPs from p to q defines the portion of $\mathcal{F}_{(p,q)}$ lying outside $R_{(p,q)}$. And all the sub-paths in this portion passes through the



$\mathcal{F}_{(p,q)}$ is monotone (here simple, and hence no bridge; non-simple case shown in Fig. 2.4(left)).	$\mathcal{F}_{(p,q)}$ is non-monotone, and hence non-simple, containing U-bridges.

Figure 2.5: Two possible cases when $\mathcal{F}_{(p,q)} \subseteq R$.

U-bridge lying outside $R_{(p,q)}$. Detailed explanation is given in Sec. 2.4. First the following lemmas on a monotone SIP are considered.

Lemma 1. *If a SIP $\pi_{(p,q)}$ is monotone, then it lies in $R_{(p,q)}$, and hence its length is given by the semi-perimeter of $R_{(p,q)}$.*

Proof. As $\pi_{(p,q)}$ is monotone, its length is $|\pi_{(p,q)}| = \text{abs}(x_p^{(g)} - x_q^{(g)}) + \text{abs}(y_p^{(g)} - y_q^{(g)})$, where $(x_p^{(g)}, y_p^{(g)})$ and $(x_q^{(g)}, y_q^{(g)})$ denote the respective coordinates of p and q, measured in grid units. If $\pi_{(p,q)}$ goes outside $R_{(p,q)}$, then the sum of lengths of its horizontal (or vertical) sub-paths would be greater than $\text{abs}(x_p^{(g)} - x_q^{(g)})$ (or $\text{abs}(y_p^{(g)} - y_q^{(g)})$), wherein lies the contradiction. □

Lemma 2. *$\mathcal{F}_{(p,q)}$ is monotone if and only if any SIP between p and q is monotone.*

Proof. From Lemma 1, it is clear that if a SIP (between p and q) is monotone and another SIP is non-monotone, then the length of the latter would be greater than that of the former, which is a contradiction. Hence, either all SIPs in $\mathcal{F}_{(p,q)}$ are monotone or all of them are non-monotone. □

The theorem on monotonicity of $\mathcal{F}_{(p,q)}$, considering its containment in $R_{(p,q)}$ is as follows.

Theorem 1 (FSIP monotonicity)**.** *If $\mathcal{F}_{(p,q)}$ is monotone, then it lies in $R_{(p,q)}$. However, the condition $\mathcal{F}_{(p,q)} \subseteq R_{(p,q)}$ does not guarantee the monotonicity of $\mathcal{F}_{(p,q)}$.*

Proof. By Lemma 2, it is clear that $\mathcal{F}_{(p,q)}$ is monotone if and only if all SIPs between p and q are monotone. By Lemma 1, all monotone SIPs always lie in $R_{(p,q)}$, which implies $\mathcal{F}_{(p,q)}$ lies in $R_{(p,q)}$.

But the converse is not true, since a non-monotone FSIP may lie in $R_{(p,q)}$, as evident from the illustration in Fig. 2.5(right). □

The theorem that explicates the necessary and sufficient condition of a non-monotone FSIP is as follows.

Theorem 2 (FSIP non-monotonicity)**.** *$\mathcal{F}_{(p,q)}$ is non-monotone if and only if it has a U-bridge.*

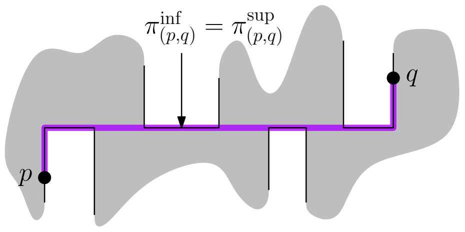


Figure 2.6: The SIP between p and q is unique.

Proof. A monotone FSIP is either an isothetic convex polygon without any bridge (Fig. 2.5(left)) or a sequence of isothetic regions (i.e., sub-polygons of P) connected by one or more Z-bridges (Fig. 2.4(left)). Consider a bridge, which is, w.l.o.g., a vertical line segment. If the horizontal movements preceding and following this bridge are in same direction, then it is a Z-bridge (Fig. 2.4(left)); and if they are in opposite directions, then it is a U-bridge (Fig. 2.4(right)). Thus, a SIP with a U-bridge contains three different movements, which implies that it is non-monotone (Definition 3). Hence, by Lemma 2, if a SIP from p to q has a U-bridge, then $\mathcal{F}_{(p,q)}$ is non-monotone.

The converse proof is similar. If $\mathcal{F}_{(p,q)}$ is non-monotone, then each $\pi_{(p,q)} \in \mathcal{F}_{(p,q)}$ is non-monotone (Lemma 2) and hence consists of more than two directions. As explained above, the horizontal movements preceding and following a Z-bridge are in same direction. So, the monotonicity breaks only due to U-bridge. □

From Theorem 1 and Theorem 2, the following can be inferred.

Corollary 1 (FSIP uniqueness)**.** *For two given points p and q, the family of their shortest isothetic paths in P, i.e., $\mathcal{F}_{(p,q)}$, is unique.*

Further, when there exists exactly one SIP between p and q, then $\pi^{\inf}_{(p,q)}$ and $\pi^{\sup}_{(p,q)}$ entirely coincide, and $\mathcal{F}_{(p,q)}$ is just a single isothetic path. This is a very typical case, which occurs when p and q lie on the perimeter of P, and there is a Z-bridge between them, on which two or more lines of concavity are incident (Fig. 2.6).

To characterize the FSIP now, the sequence of its constituent isothetic sub-polygons and bridges are used. The theorems are as follows.

Theorem 3 (FSIP sub-polygons)**.** *All the sub-polygons in $\mathcal{F}_{(p,q)}$ are simple isothetic convex polygons.*

Proof. $\mathcal{F}_{(p,q)}$ consists of sub-polygons, connected by bridges (Theorem 2). It is easy to observe that each of these sub-polygons is simple, since $\mathcal{F}_{(p,q)}$ is not simple only at bridges. To prove that a sub-polygon R_i is an isothetic convex polygon, observe that the part of $\pi^{\inf}_{(p,q)}$ and the part of $\pi^{\sup}_{(p,q)}$ defining the border of R_i are both monotone, since the sub-polygon R_i does not contain any bridge in general, and any U-bridge in particular. Hence, none of those parts of $\pi^{\inf}_{(p,q)}$ and $\pi^{\sup}_{(p,q)}$ has two consecutive vertices of type **3** [10]. This implies R_i has no two consecutive vertices of type **3**, wherefore R_i is an isothetic convex polygon. □

Theorem 4 (FSIP bridges)**.** *Each $\pi_{(p,q)} \in \mathcal{F}_{(p,q)}$ has the same sequence of bridges, $\mathcal{B}_{(p,q)}$, and hence passes through the same sequence of (simple) convex isothetic polygons, $\mathcal{R}_{(p,q)}$.*

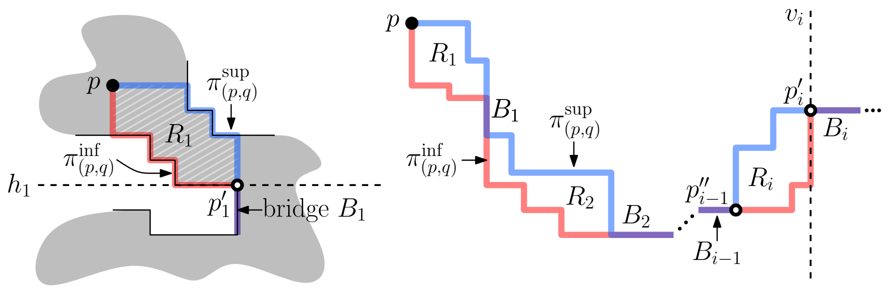


Figure 2.7: Induction basis (left) and induction step (right) of the proof of Theorem 4.

Proof. The proof is done by induction. From Corollary 1 and Theorem 3, $\mathcal{F}_{(p,q)}$ can be represented by a unique sequence of convex isothetic polygons, namely $\mathcal{R}_{(p,q)}$, intertwined by a unique sequence of bridges. Let B_1 be the first bridge in $\mathcal{B}_{(p,q)}$. Then by Definition 9, B_1 connects the first two polygons R_1 and R_2 in $\mathcal{R}_{(p,q)}$. Let p'_1 be the point where R_1 is incident on B_1 and h_1 be the horizontal line passing through p'_1. Let, w.lo.g., q be lying somewhere below h_1, as shown in Fig. 2.7. Then, in order to reach q, any SIP from p has to pass through p'_1 and hence contain B_1.

For the induction hypothesis, assume that each $\pi_{(p,q)}$ from p has passed through $R_1, B_1, R_2, B_2, \ldots, R_i$. The induction step is proved that $\pi_{(p,q)}$ enters R_{i+1}.

Let p''_{i-1} and p'_i be the respective points at which R_i is incident on B_{i-1} and B_i. As shown in Fig. 2.7, if v_i be the vertical line perpendicular to B_i and passing through p'_i, then after reaching p'_i from p''_{i-1}, $\pi_{(p,q)}$ has to move along B_i, since any other direction would result in more than two types of moves inside R_i, thereby contradicting the minimum path length of a SIP. It cannot also move upwards along v_i, since the bridge B_i is not along that direction. So, it enters R_{i+1} through B_i, whence the proof. $\square$

The above theorem leads to a relation between the number of bridges in $\mathcal{F}_{(p,q)}$ and its convexity index (Definition 8). The theorem is as follows.

Theorem 5 (Convexity index)**.** *The convexity index of $\mathcal{F}_{(p,q)}$ is $\alpha = \beta + 1$, where β is the number of bridges in $\mathcal{F}_{(p,q)}$.*

Proof. From Theorem 4, each $\pi_{(p,q)} \in \mathcal{F}_{(p,q)}$ passes through β bridges. Each bridge B_i is either a U-bridge or a Z-bridge, and connects two convex isothetic polygons, namely R_i and R_{i+1}. As the sequence of these polygons and the sequence of bridges are both unique (Corollary 1), the proof follows. $\square$

2.4 Algorithm to Compute FSIP

Algorithm Find-FSIP computes $\mathcal{F}_{(p,q)}$ by finding its extremum SIPs. As shown in Algorithm 1, its input is the (ordered) list L containing vertices of P, two lexicographically sorted lists consisting of vertices and edge points (i.e., all grid points on the boundary of P), which are denoted as L_x (x as primary key and y as secondary key) and L_y (y as primary key and x as secondary key), and p, q as the start and the end points. The procedure Find-U-Bridges is used to find the sequence of U-bridges between p and q, and the pair of endpoints defining each bridge is stored in the array B (Step 1). Note that B is initialized with p and appended with q in Find-U-Bridges (Steps 2

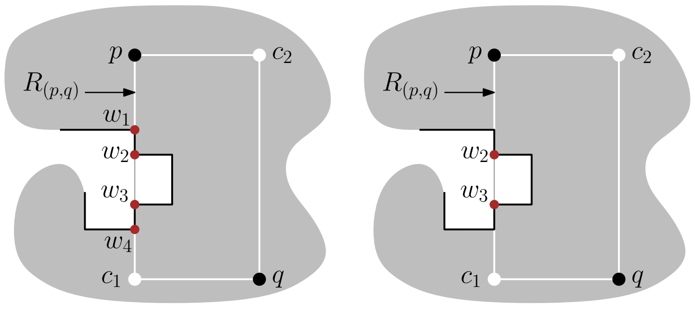


Figure 2.8: Removing intersection points based on vertex indices.

Algorithm 1: Find-FSIP

```
Input: $L, L_x, L_y, p, q$
Output: $\mathcal{F}_{(p,q)}$
$B \leftarrow$ Find-U-Bridges$(p, q)$
$\mathcal{F}_{(p,q)} \leftarrow \emptyset, i \leftarrow 0$
do
    $\pi^{\inf}_{(p,q)} \leftarrow$ Find-Extremum$(B[i], B[i+1])$
    $\pi^{\sup}_{(p,q)} \leftarrow$ Find-Extremum$(B[i+1], B[i])$
    $\mathcal{F}_{(p,q)} \leftarrow \mathcal{F}_{(p,q)} \cup$ Find-Region$(\pi^{\inf}_{(p,q)}, \pi^{\sup}_{(p,q)})$
    if $B[i+1] \neq q$ then
        $\mathcal{F}_{(p,q)} \leftarrow \mathcal{F}_{(p,q)} \cup$ Append-Bridge$(B[i+1], B[i+2])$
        $i \leftarrow i + 2$
while $B[i+1] \neq q$ ;
return $\mathcal{F}_{(p,q)}$
```

and 2). For each ordered point pair $(B[i], B[i+1])$ in B—excepting the first and the last pairs, the respective points $B[i]$ and $B[i+1]$ represent the end of a bridge and the start of the next bridge in $\pi^{\inf}_{(p,q)}$ (Step 1); they also represent the same two bridges in $\pi^{\sup}_{(p,q)}$ in reverse order (Step 1). The first pair has p and the start point of the first bridge, and the last pair has the end point of the last bridge and q. This is used to find the extremum SIPs in the **do-while** loop of Find-FSIP (Steps 1–1).

Step 1 of Find-FSIP uses the procedure Find-Region to compute the isothetic convex polygons bounded by the extremum SIPs (Theorem 3). It also finds the Z-bridges. While traversing the vertices of $\pi^{\inf}_{(p,q)}$ and $\pi^{\sup}_{(p,q)}$ in succession, it identifies the overlapping line segments of $\pi^{\inf}_{(p,q)}$ and $\pi^{\sup}_{(p,q)}$, which correspond to Z-bridges.

A U-bridge appearing after a isothetic convex polygon is appended by the procedure Append-Bridge, in the set $\mathcal{F}_{(p,q)}$ (Step 1). The first entry in B is p. So, the first U-bridge, B_1, will appear after the first region, R_1. The last entry in B is q. Thus, at the last iteration of the **do-while** loop of Find-FSIP, there will be no U-bridge after the last isothetic convex polygon in the sequence, i.e., Steps 1–1 will not be executed. Otherwise, the U-bridge after the isothetic convex polygon, generated in Step 1, is appended by the procedure Append-Bridge in Step 1. If the U-bridge consists of more than two type **3** vertices, then all the vertices belonging to the U-bridge are appended in $\mathcal{F}_{(p,q)}$ by Append-Bridge. The pointer gets incremented by two, and so it points to the end vertex of the last U-bridge (Step 1).

In the procedure Find-U-Bridges, the point c_1 is determined as the corner point of $R_{(p,q)}$,

Procedure `Find-U-Bridges`(p, q)

```
$c_1 \leftarrow$ RECT$(p, q)$
if $x_{c_1} = x_p$ then
    $M_y \leftarrow$ SEARCHVERT$(p, c_1, L_y)$, $M_x \leftarrow$ SEARCHHORZ$(c_1, q, L_x)$
else
    $M_x \leftarrow$ SEARCHHORZ$(p, c_1, L_x)$, $M_y \leftarrow$ SEARCHVERT$(c_1, q, L_y)$
$M \leftarrow$ REMOVE-POINTS$(M_y, M_x, p, q)$
$C_{(p,q)} \leftarrow$ FIND-POLYCHAIN$(M, p, q)$
$i \leftarrow 1, j \leftarrow 1, B[0] \leftarrow p$
while $C_{(p,q)}[i] \neq q$ do
    if $type[C_{(p,q)}[i]] = 3 \wedge type[C_{(p,q)}[i+1]] = 3$ then
        $B[j] \leftarrow C_{(p,q)}[i]$
        $i \leftarrow i+1, j \leftarrow j+1$
        while $type[C_{(p,q)}[i+1]] = 3$ do
            $i \leftarrow i+1$
        $B[j] \leftarrow C_{(p,q)}[i]$
        $j \leftarrow j+1$
    $i \leftarrow i+1$
$B[j] \leftarrow q$
return $B$
```

which appears in the anticlockwise traversal of $R_{(p,q)}$ from p to q (Step 2). $R_{(p,q)}$ intersects P at different points. These intersection points on $\overline{pc_1}$ and $\overline{c_1q}$ are determined in Step 2 or in Step 2, depending on the location of c_1 in $R_{(p,q)}$. In Step 2, the procedure REMOVE-POINTS is used to remove some of the intersection points based on some combinatorial rules discussed in [28]. The vertices of P are indexed in increasing order, starting from p. If there are two consecutive vertices of type **13** (**31**) in the list of intersection points, then the vertex with greater (lower) index is discarded, since the line segment joining these two intersection points lies entirely outside P. For example, in Fig. 2.8, the vertices w_1, w_2, w_3, w_4 are the four intersection points, with type$(w_1, w_2) = \mathbf{31}$ and type$(w_3, w_4) = \mathbf{13}$. According to the aforesaid rules, here w_1 and w_4 are discarded. After applying this rule, all intersection points are considered in pair. The indices of pairs of intersection points are then checked to find whether they are in increasing or in decreasing order and whether the index falls within the indices of extreme two points in M, namely $M[1]$ and $M[k]$, where k is the total number of intersection points. If it is not so, then the last point of first pair and the first point of next pair are discarded. Now, M contains only the intersection points such that its traversal produces movements along the borders of $R_{(p,q)}$ and P alternately. That is, if $M[i]$ to $M[i+1]$ describes a segment on the border of $R_{(p,q)}$, then $M[i+1]$ to $M[i+2]$ indicates a sequence of vertices along the boundary of P. The latter sequence may be reduced by using combinatorial rules (Procedure APPLY-RULE in [28]) to get a tight polychain from $M[i+1]$ to $M[i+2]$, if the sequence creates a convexity. Let $M = \langle p, w_1, w_2, \ldots w_k, q\rangle$. Then (p, w_1) is traversed along $R_{(p,q)}$, (w_1, w_2) along P, (w_2, w_3) along $R_{(p,q)}$, (w_3, w_4) along P, and so on. During traversal along P, when the polychain $C_{(p,q)}$ enters a convex region, then it is reduced and made to pass along the open side of the convex region (Fig. 2.3(left)). If there are one or more concavities inside the convex region, then though the reduced polychain passes along the open side (i.e., along the line of concavity which is nearest to the convexity line), it would store the two concave vertices of the nearest concavity (Fig. 2.3(right)).

Once the polychain $C_{(p,q)}$ is obtained, U-bridges are extracted from it (FIND-U-BRIDGES: Steps 2–2). i and j are initialized to '1', and p is appended in B (Step 2). The two endpoints corresponding to each U-bridge are stored in B in sequence by traversing $C_{(p,q)}$. Whenever there are two or more consecutive vertices of type **3** in $C_{(p,q)}$ (Steps 2-2), then the two extreme type **3** vertices of the

Procedure `Find-Extremum`(r, s)

```
$c \leftarrow$ RECT$(r, s)$
$M_y \leftarrow$ SEARCHVERT$(r, c, L_y)$
$M_x \leftarrow$ SEARCHHORZ$(c, s, L_x)$
$M \leftarrow$ REMOVE-POINTS$(M_y, M_x, r, s)$
$i \leftarrow 1, m \leftarrow 1, \pi[0] \leftarrow r$
while $M[i] \neq s$ do
    $\pi[m] \leftarrow M[i]$
    if $M[i] = c$ then
        $i \leftarrow i + 1, m \leftarrow m + 1$
        $\pi[m] \leftarrow M[i]$
        if $type[\pi[m-2]] = 1 \wedge type[\pi[m-1]] = 1$ then
            APPLY-RULE$(\pi[m-2], \pi[m-1])$
        if $M[i] = s$ then
            continue
    TRAVERSE$(M[i], M[i+1], \pi, m)$
    $\pi[m+1] \leftarrow M[i+1]$
    $i \leftarrow i + 2, m \leftarrow m + 2$
$\pi[m] \leftarrow s$
return $\pi$
```

sequence are stored in B as the endpoints of the particular bridge. At the end of B, q is appended (Step 2), and finally B is returned in Step 2.

The procedure FIND-EXTREMUM is applied on each pair of points in B to find the isothetic convex polygon between every two consecutive bridges, which becomes a part of $\mathcal{F}_{(p,q)}$ (Theorem 3). The first four lines of FIND-EXTREMUM are similar to those in FIND-U-BRIDGES. The extremum SIPs obtained at the two sides of a U-bridge are monotone (Theorem 3). The list of intersection points, M, is traversed and each extremum SIP moves alternately along the sides of $R_{(p,q)}$ and along the boundary of P (Steps 3–3). Initially, i and m (indices for M and π respectively) are initialized to '1', and p is appended in π (Step 3). The first movement is along the side of $R_{(p,q)}$; so, the first intersection point is appended in π (Step 3). If there is a corner point, then the corner point and the intersection point next to it are appended in π, and reduction rules are applied (by APPLY-RULE) if a convex region is created (Steps 3–3). The loop terminates on reaching the destination point (Steps 3–3).

The second movement is along the boundary of P between the two intersection points by the procedure TRAVERSE (Step 3). The SIP moves along the boundary of P, and if it enters any convex region, then corresponding reduction rules are applied. The next intersection point is appended in π in Step 3. The variables i and m are incremented by 2 in Step 3, q is appended at the end of π in Step 3, and π is returned by the procedure in Step 3.

2.4.1 Demonstration

Figure 2.9 shows a demonstration of Algorithm 1. Two points p and q, and their corresponding rectangle, $R_{(p,q)}$, are shown in Fig. 2.9(top-left). First, $C_{(p,q)}$ is determined using FIND-POLYCHAIN (Fig. 2.9: top-middle). From $C_{(p,q)}$, U-bridges are detected. Here, two U-bridges are there, u_1 and u_2 being the endpoints of the first U-bridge, and u_3 and u_6 of the second (Fig. 2.9: top-right). Two extremum SIPs are obtained from p to u_1, which are shown in red (infimum) and in blue (supremum).

The procedure FIND-REGION extracts the Z-bridge $\overline{z_1z_2}$ from the corresponding region and then the U-bridge $\overline{u_1u_2}$ is appended. So, up to this point, $\mathcal{F}_{(p,q)} = R_1 \cup \overline{z_1z_2} \cup R_2 \cup \overline{u_1u_2}$ (Fig. 2.9: bottom-left). Next, the two extremum SIPs between u_2 and u_3 are determined (Fig. 2.9: bottom-middle).

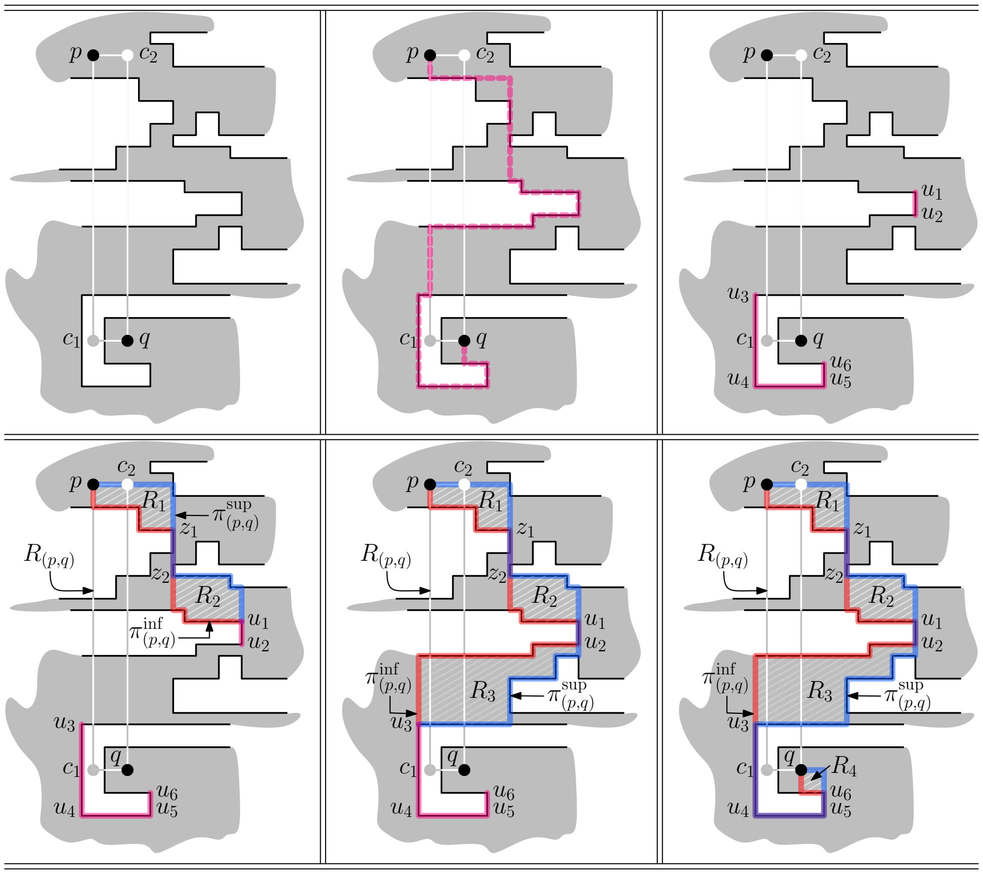


Figure 2.9: A demonstration of computing $\mathcal{F}_{(p,q)}$ by FIND-FSIP.

The corresponding region does not contain any Z-bridge, and $\mathcal{F}_{(p,q)} = R_1 \cup \overline{z_1 z_2} \cup R_2 \cup \overline{u_1 u_2} \cup R_3$ is obtained. The next U-bridge consists of more than two vertices of type **3**. All the segments between the two extreme points of the U-bridge are appended, and so $\mathcal{F}_{(p,q)}$ gets enhanced to $R_1 \cup \overline{z_1 z_2} \cup R_2 \cup \overline{u_1 u_2} \cup R_3 \cup \overline{u_3 u_4} \cup \overline{u_4 u_5} \cup \overline{u_5 u_6}$. The concluding portions of the extremum SIPs between u_6 and q are obtained in a similar manner (Fig. 2.9: bottom-right). There is no more U-bridge, since q is the destination. Hence, finally, $\mathcal{F}_{(p,q)} = R_1 \cup \overline{z_1 z_2} \cup R_2 \cup \overline{u_1 u_2} \cup R_3 \cup \overline{u_3 u_4} \cup \overline{u_4 u_5} \cup \overline{u_5 u_6} \cup R_4$.

2.4.2 Time Complexity

The inner isothetic cover P, which tightly inscribes the object A, is obtained as an ordered sequence of n grid points (vertices and edge points) and stored in L. This needs $O(ng)$ time, since the intersection of each grid edge with A is checked in $O(g)$ time and the number of grid points visited is bounded by $\Theta(n)$. Note that ng is linear in the number of pixels constituting the contour of A. Subsequently, the lists L_x and L_y are constructed from L in $O(n \log n)$ time. Thus, the total preprocessing time before applying FIND-FSIP algorithm is $O(n \log n)$.

The algorithm FIND-FSIP finds the U-bridges from the intersection points between P and $R_{(p,q)}$, which are less in number than the grid points comprising the border of P. To find the intersection points, L_x and L_y are accessed (FIND-U-BRIDGES: Steps 2–2). To find the positions of p, q, and c_1 in these lists, binary search is applied, which takes $O(\log n)$ time in total. Other intersection points lie consecutively from p to c_1 in L_y, and from c_1 to q in L_x, and hence obtained by appropriate linear search in $O(n)$ time. The procedure REMOVE-POINTS discards some of the intersection points from M_y and M_x based on combinatorial rules, which takes $O(k)$ time, where $k(< n)$ is the number of intersection points. Note that g times k does not exceed the semi-perimeter of $R_{(p,q)}$.

The procedure FIND-POLYCHAIN computes the polychain $C_{(p,q)}$ by traversing along (the borders of) $R_{(p,q)}$ and P. Traversal along $R_{(p,q)}$ is done to add intersection points in $C_{(p,q)}$, which requires $O(k)$ time. Traversal along P—from one intersection point to the next—yields all the vertices of P lying among the intersection points. These vertices are added linearly in $C_{(p,q)}$ in a maximum $O(n)$ time. During the traversal along P, reduction rules are applied by APPLY-RULE on the corresponding parts of $C_{(p,q)}$. The total time complexity for this is linear in the number of vertices of $C_{(p,q)}$, which is at most $O(n)$. Hence, all U-bridges are computed and added in the FSIP in $O(n)$ time in the worst case.

The time taken by FIND-EXTREMUM is upper-bounded by the maximum possible length of an extremum SIP, which is $O(n)$. The procedure FIND-REGION extracts the isothetic convex polygons and the Z-bridges by traversing $\pi^{\text{inf}}_{(p,q)}$ and $\pi^{\text{sup}}_{(p,q)}$. As explained in Sec. 2.4, the Z-bridges are obtained from the overlapping line segments in $\pi^{\text{inf}}_{(p,q)}$ and $\pi^{\text{sup}}_{(p,q)}$, using a linear pass on each, which needs $O(n)$ time in worst case. The APPEND-BRIDGE procedure adds the isothetic convex polygons and the bridges in $\mathcal{F}_{(p,q)}$, which also takes $O(n)$ time in worst case. Thus, the total worst-case time complexity to compute the FSIP is $O(n)$.

2.5 Experimental Results

The proposed algorithm is implemented in C in Ubuntu 10.4, Kernel version 2.6.32-21-generic, Dual Intel Xeon Processor 2.8 GHz, 800 MHz FSB. It is tested on several datasets containing various digital images of different shapes and forms. As an example, the resultant FSIP between two given points in `octopus` image is shown in Fig. 2.10. Notice that this FSIP consists of three U-bridges and four simple convex isothetic polygons, and so, $\beta = 3$ and $\alpha = 4$. Using this information, FSIPs can be used for shape analysis of digital objects, as discussed next.

2.5.1 FSIP for Shape Analysis

An FSIP encompasses all the shortest isothetic paths for a given pair of points. For different pair of points, FSIPs would be different. All these FSIPs can be used together to analyze the shape of the corresponding object. For example, the region of intersection of different FSIPs for different pairs of *control points* may provide some crucial information about the object shape. In this experimentation, these control points are selected manually. The FSIPs corresponding to all these pairs of control points have varying overlaps. The number of FSIPs overlapped in a particular region signifies its *centrality measure*, from which a *critical region* can be determined. However, the selection of control points is an important task here, which remains open for further exploration.

Figure 2.11 illustrates some critical regions generated by this algorithm on the image `guitarist`. In Fig. 2.11(top-right), there are six control points with $\binom{6}{2} = 15$ FSIPs. The lower part of the left hand (holding the guitar) of the guitarist is reported as the critical region (shown in red), since a maximum number of eight FSIPs overlap there. Four of these eight FSIPs are from the point at the

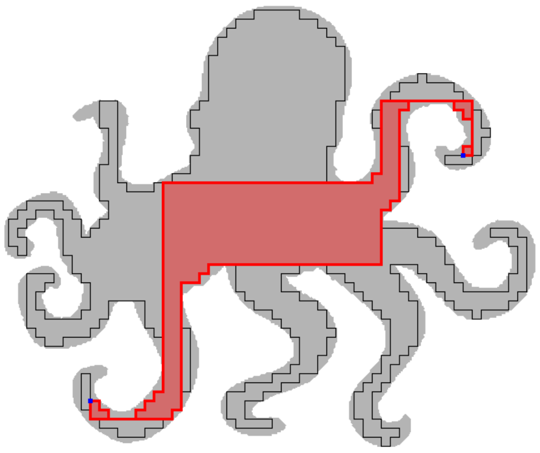


Figure 2.10: FSIP for a pair of points in the image `octopus`.

bottom of the guitar, and remaining four from the point at the top of the guitar—both sets leading to points in head, hand, and two legs. When one point is disregarded (say, the bottom point in the guitar), there arise two critical regions with five control points, as shown in Fig. 2.11(bottom-left). In the critical region shown in red, there are four overlapping FSIPs: two from guitar top to hand and right leg, and two from head to hand and right leg. In the other critical region shown in indigo blue, there are also four such overlapping FSIPs: two from guitar top to hand and right leg, and two from hand and head to left leg. So, it is the "guitar" here that changes the critical region. When there is no control point inside the "guitar", then the position of the critical region remains in the "body", as shown with four control points in Fig. 2.11(bottom-right). The result in this case has six critical regions, which are shown in red, indigo blue, olive green, pink, purple, and sea green.

The critical region, C, in Fig. 2.11(top-right) forms the "bridge" between families (or parts of families) lying left of C and those lying right of C. This is a typicality of `guitarist` image and may not always arise. However, if it arises, then it would give a meaningful partition of the digital shape. Another example on the image `dance` is shown in Fig. 2.12. In this image, the critical region is at the gripping hands. A critical region, however, may not always straightway provide a meaningful partition. For example, in Fig. 2.13, the critical region (red) varies for different poses of the tennis player, and so it cannot partition the tennis bat from the tennis player. However, it is interesting to notice that in each of these poses, the critical region lies near the waist at the side of the hand holding the bat.

FSIP can be applied for shape analysis. Choosing of *control points* in a digital object is important. If proper *control points* are determined, then FSIP between two points will give some important information regarding the shape of the object. The convexity index of a FSIP gives a measure on the complexity of the object. The overlapping portions of multiple FSIP gives critical region.

The data of FSIPs generated for three images, i.e., `guitarist`, `dance`, and `tennis` set, are tabulated in Tables 2.1, 2.2, and 2.3. The source and the destination points are given in the first column of each table. The second column contains the ideal (Manhattan) distance $\|pq\|_1$ between p and q, whereas the third column contains the length of a SIP between them, which is denoted by $\|\pi_{(p,q)}\|$. The *path complexity* of a SIP (and its FSIP, thereof) is estimated as the percentage of $\|pq\|_1$ w.r.t. $\|\pi_{(p,q)}\|$. The rationale is, as a SIP deviates more and more from the sides of $R_{(p,q)}$, its complexity goes on increasing. This complexity can be used to capture the shape of the object part

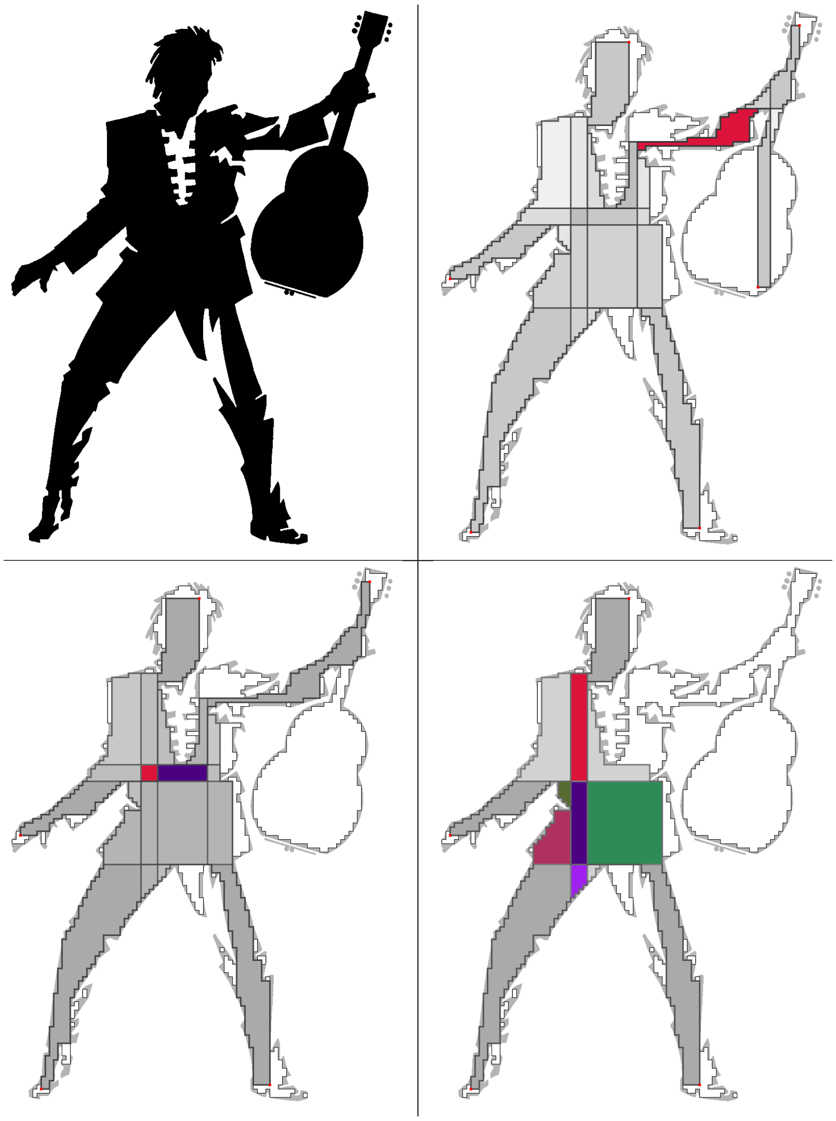


Figure 2.11: Results on `guitarist` (**top-left**) image for varying control points (shown as red dots). FSIPs with six points (**top-right**), five points (**bottom-left**), and four points (**bottom-right**).

lying between the concerned points. In combination with FSIP complexity, the number of U-bridges can also be used to estimate the shape complexity. For, a long SIP may not always signify a complex shape. As an example, in `guitarist` image, the point lying at top of the guitar and that in the right leg are furthest (depicted in Table 2.1), but the corresponding FSIP is monotone. On the contrary, the path length between the top of the guitar and the head of the guitarist is although less, the FSIP is more complex with two U-bridges. Again, the FSIP between the bottom of the guitar and the head has maximum number (three) of U-bridges, but a path in this FSIP is less complex if it is

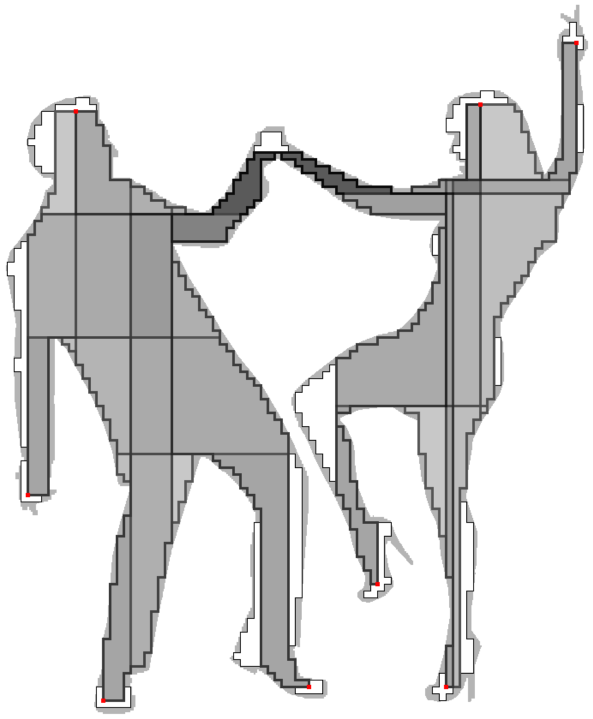


Figure 2.12: FSIPs and the resultant critical region (black) for `dance` image.

considered merely the complexity estimate of the FSIP.

In `dance` image, the FSIP with maximum path complexity is from the man's left leg to the girl's right leg (Table 2.2), but it consists of three U-bridges. The FSIP from man's left leg to girl's left leg has maximum number (four) of U-bridges. The FSIPs from man's head to man's hand and left leg, girl's head to girl's right leg, and girl's hand to girl's right leg are all monotone. Monotone FSIP implies freedom of movement, whereas the presence of a Z-bridge or a U-bridge restricts this freedom. Presence of U-bridges makes a SIP non-monotone, whereas Z-bridges do not break the monotonicity. So, bridges carry a strong signature for the analysis of shape of an object.

The FSIP characteristics for two images of `tennis` set (Fig. 2.13) are provided in Table 2.3. Different poses give rise to different path complexities for the same pair of points, since the nature and the orientation of the concavities change with these poses. Between bat and left leg in the `tennis` set, some poses have monotone FSIPs and some have non-monotone. But, between right leg and left leg, the FSIPs are non-monotone for all the four poses. For Pose 1 tabulated in Table 2.3, there is more than one U-bridge, whereas for each other pose, there is one or none.

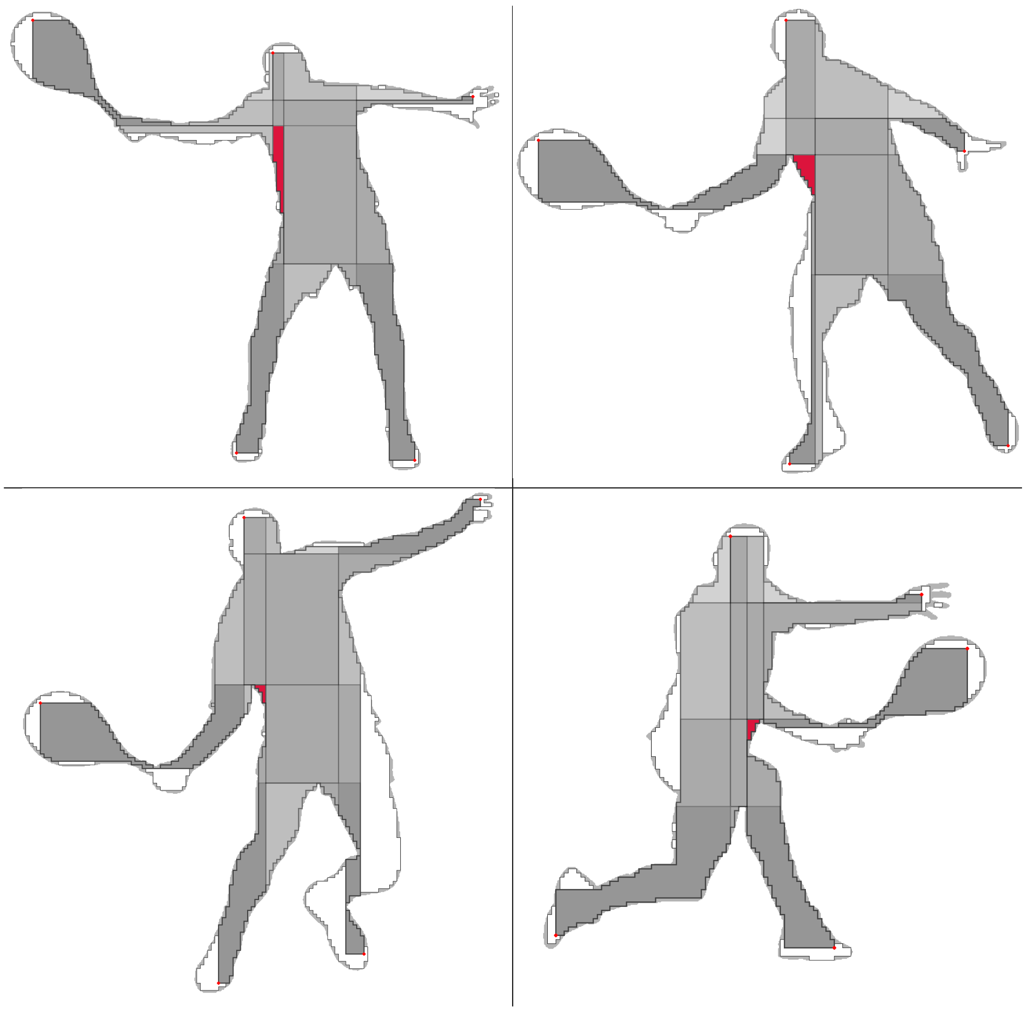


Figure 2.13: FSIPs on `tennis` image set.

Table 2.1: FSIP characteristics for `guitarist` image. p and q are the points in the image; $\|pq\|_1$ =Manhattan distance between p and q; $\|\pi_{(p,q)}\|$ =length of a SIP between p and q; κ =complexity of $\mathcal{F}_{(p,q)}$; β_U =number of U-bridges in $\|\pi_{(p,q)}\|$; α =convexity index of $\mathcal{F}_{(p,q)}$. ($GTop$ = point at guitar top; $GBtm$ = point at guitar bottom; $Head$ = point at head; $Hand$ = point at hand; $LLeg$ = point at left leg; $RLeg$ = point at right leg.)

p, q	$\|pq\|_1$	$\|\pi_{(p,q)}\|$	κ	β_U	α
$GTop$, $GBtm$	584	584	0	0	1
$GTop$, $Head$	360	1160	222.22	2	3
$GTop$, $Hand$	1160	1160	0	0	1
$GTop$, $RLeg$	1608	1608	0	0	1
$GTop$, $LLeg$	1160	1400	20.69	1	2
$GBtm$, $Head$	720	1312	82.22	3	4
$GBtm$, $Hand$	608	1312	115.79	1	2
$GBtmp$, $RLeg$	1024	1760	71.88	1	2
$GBtm$, $LLeg$	576	1552	169.44	2	3
$Head$, $Hand$	800	800	0	0	1
$Head$, $RLeg$	1248	1248	0	0	1
$Head$, $LLeg$	1072	1232	14.92	1	2
$Hand$, $RLeg$	528	1120	112.12	2	3
$Hand$, $LLeg$	960	1168	21.67	1	2
$RLeg$, $LLeg$	448	1298	189.73	1	2

Table 2.2: FSIP characteristics for `dance` image. ($MHead$ = point at man's head; $MHand$ = point at man's hand; $MRLeg$ = point at man's right leg; $MLLeg$ = point at man's left leg; $GHead$ = point at girl's head; $GHand$ = point at girl's hand; $GRLeg$ = point at girl's right leg; $GLLeg$ = point at girl's left leg.)

p, q	$\|pq\|_1$	$\|\pi_{(p,q)}\|$	κ	β_U	α
$MHead$, $MHand$	584	584	0	0	1
$MHead$, $MRLeg$	720	784	8.89	1	2
$MHead$, $MLLeg$	944	944	0	0	1
$MHead$, $GHead$	480	816	70.00	3	4
$MHead$, $GHand$	664	1000	50.60	3	4
$MHead$, $GLLeg$	904	1304	44.25	4	5
$MHead$, $GRLeg$	1104	1264	14.49	3	4
$MHand$, $MRLeg$	328	760	131.71	2	3
$MHand$, $MLLeg$	552	920	66.67	1	2
$MHand$, $GHead$	984	1080	9.76	2	3
$MHand$, $GHand$	1168	1264	8.22	2	3
$MHand$, $GLLeg$	512	1568	206.25	3	4
$MHand$, $GRLeg$	712	1264	8.45	2	3
$MRLeg$, $MLLeg$	256	800	212.50	1	2
$MRLeg$, $GHead$	1136	1232	8.45	2	3
$MRLeg$, $GHand$	1168	1264	7.27	2	3
$MRLeg$, $GLLeg$	512	1568	277.19	3	4
$MRLeg$, $GRLeg$	712	1264	303.85	2	3
$MLLeg$, $GHead$	880	1296	47.27	3	4
$MLLeg$, $GHand$	1064	1480	39.10	3	4
$MLLeg$, $GLLeg$	200	1784	792.00	4	5
$MLLeg$, $GRLeg$	160	1744	990.00	3	4
$GHead$, $GHand$	184	360	95.65	1	2
$GHead$, $GLLeg$	680	776	14.12	1	2
$GHead$, $GRLeg$	792	792	0	0	1
$GHand$, $GLLeg$	864	960	14.12	1	2
$GHand$, $GRLeg$	904	904	0	0	1
$GLLeg$, $GRLeg$	200	728	264	3	4

Table 2.3: FSIP characteristics for `tennis` image. (Bat = point at Bat; $Head$ = point at head; $Hand$ = point at hand; $LLeg$ = point at left leg; $RLeg$ = point at right leg.)

Pose 1 (Fig. 2.13: top-left)

p, q	$\|pq\|_1$	$\|\pi_{(p,q)}\|$	κ	β_U	α
Bat, $Head$	750	1150	53.33	1	2
Bat, $Hand$	1420	1580	11.27	1	2
$Batp$, $RLeg$	1750	2010	14.86	1	2
Bat, $LLeg$	2260	2260	0	0	1
$Head$, $Hand$	670	690	2.98	1	2
$Head$, $RLeg$	1200	1260	5.00	1	2
$Head$, $LLeg$	1510	1510	0	0	1
$Hand$, $RLeg$	1630	1630	0	0	1
$Hand$, $LLeg$	1160	1480	27.59	1	2
$RLeg$, $LLeg$	510	1550	203.92	1	2

Pose 2 (Fig. 2.13: top-right)

p, q	$\|pq\|_1$	$\|\pi_{(p,q)}\|$	κ	β_U	α
Bat, $Head$	1010	1390	37.62	1	2
Bat, $Hand$	1200	1700	41.67	2	3
$Batp$, $RLeg$	1580	2020	27.85	3	4
Bat, $LLeg$	2130	2430	14.08	2	3
$Head$, $Hand$	850	850	0	0	1
$Head$, $RLeg$	1230	1370	11.38	1	2
$Head$, $LLeg$	1780	1780	0	0	1
$Hand$, $RLeg$	380	1520	300.00	1	2
$Hand$, $LLeg$	930	1530	64.52	1	2
$RLeg$, $LLeg$	550	1590	189.09	1	2

Chapter 3

A Combinatorial Technique for Construction of Triangular Covers of Digital Objects

[1]Das, B., M. Dutt, A. Biswas, P. Bhowmick, and B. B. Bhattacharya (2014). A Combinatorial Technique for Construction of Triangular Covers of Digital Objects. In: 16th International Workshop on Combinatorial Image Analysis: IWCIA'14. Vol. 8466. Brno, Czech Republic: LNCS, pp.76-90.

3.1 Introduction

Optimal polygonal covers of digital objects find diverse applications in many fields of image analysis and computer vision. Such covers are often obtained with a certain approximation parameter or with respect to an underlying grid that determines their precision. The grid is usually defined or characterized by the type of its constituent cells, which may be axis-parallel squares [8], equilateral triangles [21], or isosceles right-angled triangles [58]. Whether polygonal covers can efficiently be generated in low-order polynomial time for different grid patterns is a challenging problem, and this chapter is focused on the specific problem when the cells comprising the underlying grid are equilateral triangles.

A *triangular grid*, also called an *isometric grid* [31], is a grid formed by tiling the plane regularly with equilateral triangles. Given a triangular grid, the *triangular cover* of a digital object placed on the grid corresponds to a collection of equilateral triangles. Such a cover captures the structural and topological information of the concerned object, and finds useful applications in many fields, such as digital image processing and image analysis [48], geometric modeling [45], clustering and pattern analysis [8, 45], ecological simulation [5] etc.

Triangular grids and hexagonal grids are duals of each other [48]. There exists a multitude of work in digital geometry in the framework of square grids, as well as triangular and hexagonal grids. Based on algorithmic comparison, it has been shown in [65] that there is a strong degree of similarity between square grids and hexagonal grids. This is also evidenced from the literature today, as several interesting works are seen on characterization and construction of geometric primitives on triangular

[1]*

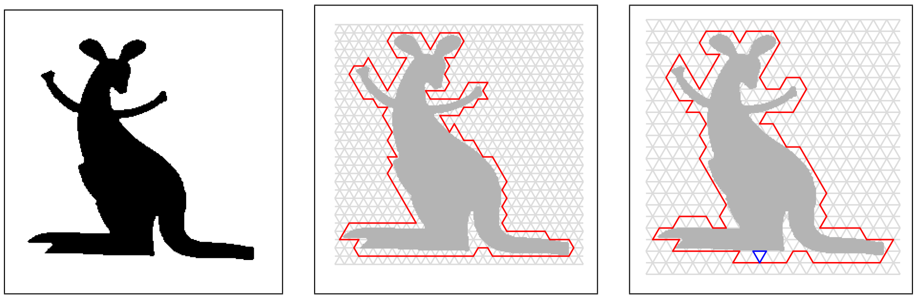


Figure 3.1: A digital object (left) and its outer triangular covers (shown in red) for $g = 12$ (middle) and $g = 14$ (right).

and hexagonal grids. Some of these related to digital lines, digital circles, and polygons, may be found in [30, 35, 47, 51, 57].

The first algorithm for construction of triangular covers of a digital object for different grid sizes, which is based on certain combinatorial rules, is proposed here. See Fig. 3.1 for triangular covers produced by this algorithm on a typical digital object for two different grid sizes. Although there exist similar work in the literature, no definite algorithm is available till date for the aforesaid problem. For example, the work in [58] is on tiling an image by isosceles right-angled triangles of the same size. Some investigation was made on analyzing the cellular topology with triangular grids, e.g., [50], where coordinate triplets are used to address the triangle pixels of both orientations, the edges between them, and the points at the triangle corners. The main advantage of a topological description is that it provides more information compared to the usual description, storing not only the pixels of the two-dimensional image but also the lower dimensional segments, the edges, and the points separating the pixels. To represent the points of a triangular grid and their neighborhood relations, coordinate triplets can be used, as shown in [48].

There also exist algorithms to compute the shortest distance between two given points on the triangular grid [46], as well as on the hexagonal grid [43], based on the distance function and the neighboring relations. In [49], it has been shown how distance computation can be done first with neighborhood sequence in 3D digital space and then by injection of the triangular grid to the cubic grid. For computing isoperimetrically optimal polygons in the triangular grid with Jordan-type neighborhood on the boundary, a method has been proposed in [51].

Some results on tiling in triangular grid have also been reported in recent time. There is no polynomial-time algorithm to solve the finite tiling problem. So, an attempt has been made in [59] to solve the problem using group theory. Such group-theoretic approaches for tiling a given finite region using polyominoes from triangular, square, or hexagonal lattices can be traced back to 1990's [20]. In [33], regular production systems are used as a tool for analyzing triangle tilings. In [4], a solution has been provided for the problem of N-tiling of a triangle ABC by a triangle T (tile) such that ABC can be represented as the union of N triangles congruent to T, overlapping only at their boundaries. In [39], a technique has been proposed for regular and irregular tiling of convex polygons with congruent triangles. Another technique proposed in [12] shows how to compute the locations of the vertices of a polygon, so that it can be perfectly tiled by lattice triangles. Tiling has also been implemented with other techniques such as using trichromatic colored-edged triangles, Golden Triangles, and Penrose Rhombs [11, 19].

For construction of optimum-area outer and inner covers of a digital object against a given isothetic or a square grid, an algorithm is proposed in [8]. The algorithm proposed here for the triangular grid is essentially different from the one in [8] mainly for three reasons. First, the combinatorial cases on the triangular grid are different from those on the square grid. Second, the grid lines are non-rectilinear in a triangular grid, thus posing a higher level of difficulty compared to the square grid. Third, all the computations in the square grid can be restricted to integers only by setting the grid size to an integer value, whereas, the computations are in real domain in the case of triangular grid. Note that triangular mesh has also wide applications in the area of surface representation and construction. However, efficient computation of the tightest cover of a digital object on a triangular grid remains an open problem.

The chapter is organized as follows. Definitions related to the triangular cover are stated in Sec. 3.2. Combinatorial rules that lead to the covering algorithm, with necessary explanations and runtime complexity, are discussed in Sec. 3.3. Experimental results on some datasets are presented in Sec. 5.5.

3.2 Definitions

A *(digital) object* is a finite subset of $\mathbb{Z}^2$ consisting of one or more $k(= 4$ *or* $8)$*-connected components* [38]. Here, the considered object is a single 8-connected component. A *triangular grid* (henceforth simply referred as *grid*) $\mathbb{T} := (\mathbb{L}_0, \mathbb{L}_{60}, \mathbb{L}_{120})$ consists of three sets of parallel *grid lines*, which are inclined at 0^0, 60^0, and 120^0 w.r.t. x-axis. The grid lines in $\mathbb{L}_0, \mathbb{L}_{60}, \mathbb{L}_{120}$ correspond to three distinct coordinates, namely α, β, γ. Three grid lines, one each from $\mathbb{L}_0$, $\mathbb{L}_{60}$, and $\mathbb{L}_{120}$, intersect at a (real) *grid point*. The distance between two consecutive *grid points* along a grid line is termed as *grid size*, g. A line segment of length g connecting two consecutive grid points on a grid line is called *grid edge*. The smallest-area triangle formed by three grid edges, one each from $\mathbb{L}_0$, $\mathbb{L}_{60}$, and $\mathbb{L}_{120}$, is called *unit grid triangle* (UGT). A portion of the triangular grid is shown in Fig. 3.2. It has six distinct regions called *sextants*, each of which is well-defined by two rays starting from $(0, 0, 0)$. For example, Sextant 1 is defined by the region lying between $\{\beta = \gamma = 0, \alpha \geq 0\}$ and $\{\alpha = \gamma = 0, \beta \geq 0\}$. One of α, β, γ is always 0 in a sextant. For example, $\gamma = 0$ in Sextant 1 and Sextant 4.

For a given grid point, p, there are six neighboring UGTs, given by $\{T_i : i = 0, 1, \ldots, 5\}$. The three coordinates of p are given by the corresponding moves along a/the shortest path from $(0, 0, 0)$ to p, measured in grid unit. For example, $(1, 2, 0)$ means a unit move along 0^0 followed by two unit moves along 60^0, starting from $(0, 0, 0)$. The grid point p can have six neighbor grid points, whose direction codes are given by $\{d' : i = 0, 1, \ldots, 5\}$.

A (finite) polygon P imposed on the grid $\mathbb{T}$ is termed as a *triangular polygon* if its sides are collinear with lines in $\mathbb{L}_0$, $\mathbb{L}_{60}$, and $\mathbb{L}_{120}$. It consists of a set of UGTs, and is represented by the (ordered) sequence of its vertices, which are grid points. It can also be represented by a sequence of directions and a start vertex. Its *interior* is defined as the set of points with integer coordinates lying inside it. An *outer triangular polygon* is the minimum-area triangular polygon that tightly covers the digital object A in $\mathbb{T}$. Each UGT having at least one of its sides on the boundary of the outer polygon has object occupancy. An *outer triangular hole polygon* is the maximum-area triangular polygon that inscribes a hole or a concavity of A; that is, A lies outside the outer triangular hole polygon. The *outer triangular cover* (OTC), $\overline{P}$, is the set of outer triangular polygons and outer triangular hole polygons, such that the region given by the union of the outer triangular polygons minus the union of the interiors of the outer triangular hole polygons, contains a UGT if and only if it has object occupancy. The *inner triangular cover* (ITC), $\underline{P}$, is the set of inner polygons and inner hole polygons, such that the region given by the union of the inner polygons minus the union

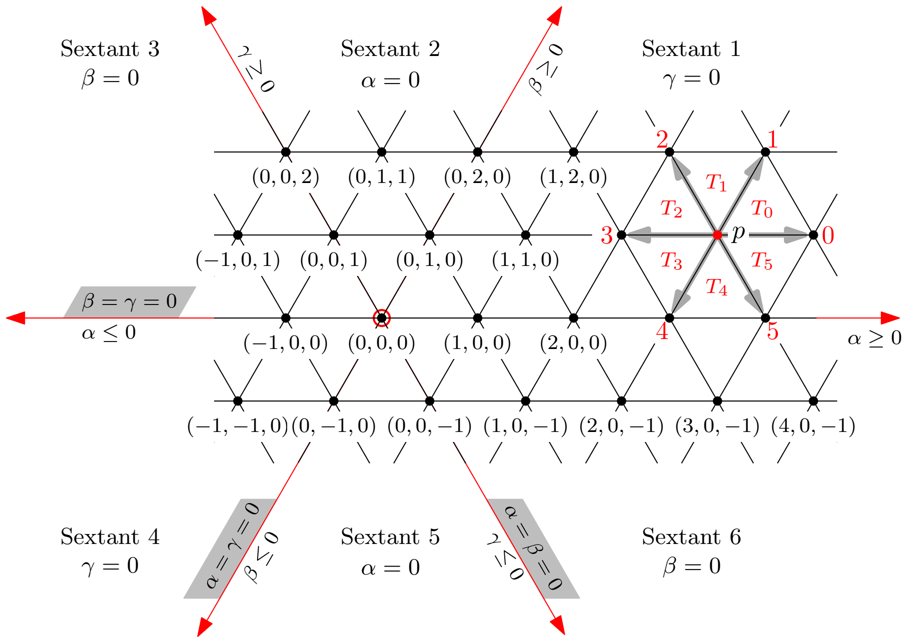


Figure 3.2: Portion of a triangular grid, the UGTs $\{T_0, T_1, \ldots, T_5\}$ incident at a grid point p, and the direction codes $\{0, 1, \ldots, 5\}$ of neighboring grid points of p.

of the interiors of the inner hole polygons, contains a UGT if and only if it is a subset of A.

Hence, the OTC of an object consists of minimum number of UGTs and its ITC consists of maximum number of UGTs. As a consequence, given an object A and a grid imposed on A, the OTC and ITC are, respectively, of minimum and of maximum areas measured in the grid unit.

3.3 Obtaining outer triangular cover

While obtaining the triangular cover $\overline{P}$ of A, this algorithm traverses the boundary of $\overline{P}$ with the invariant that A always lies left w.r.t. the direction of traversal. The top-left pixel on the boundary of the object, p_0, is given as input, from which the start vertex q_0 of $\overline{P}$ is computed, as shown in Fig. 3.3. If A contains any hole, then the top-left pixel of the hole boundary is also given as input. However, if the object contains a large concave region with 'narrow mouth', then the hole polygon inscribing that concave region is detected and traversed later from a UGT containing the border of the narrow mouth.

3.3.1 Classification of vertices

The *object occupancy vector* corresponding to $\langle T_i : i = 0, 1, \ldots, 5\rangle$ incident at a grid point q is given by $A_q = \langle a_0 a_1 \cdots a_5 \rangle$, where, for $i = 0, 1, \ldots, 5$, $a_i = 1$ if T_i incident at q has object occupancy, and 0 otherwise. The object occupancy of each T_i is determined by checking its interior integer points lying along its borders. The algorithm then uses the vector A_q to determine whether q lies on $\overline{P}$ and to compute the edges of $\overline{P}$ incident at q. For example, $A_q = 000011$ implies T_4 and T_5

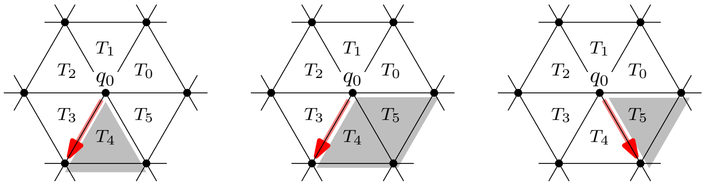


Figure 3.3: Determining the start vertex, q_0, from the top-left object pixel, p_0, which would lie either in T_4 (left or middle configuration) or in T_5 (middle or right). Object-occupied UGTs are shown in gray, and start directions in red.

are object-occupied. So, the direction of the incoming edge at q becomes 3 and that of the outgoing edge becomes 5, since the object lies left during traversal of $\overline{P}$.

A grid point q lies on $\overline{P}$ if A_q contains both 0's and 1's. If $A_q = 0^6$, then q lies outside $\overline{P}$ (Fig. 3.4a); and $A_q = 1^6$ implies q lies strictly inside $\overline{P}$ (Fig. 3.4l). Apart from these two cases, there arise $2^6 - 2 = 62$ combinatorial cases of A_q. These 62 cases can be simplified to following 5 cases only, with their corresponding sub-cases (Fig. 3.4b-k), depending on the number of 1's in A_q, which is denoted by $k(= 1, 2, \ldots, 5)$.

- $k = 1$: There are 6 sub-cases depending on the position of 1-bit in A_q. For each sub-case, the angle at q is $\pi/3$ (Fig. 3.4b). If $a_i = 1$, then the *direction pair* representing the incoming and the outgoing edges at q is $((i + 4) \bmod 6, i)$.

- $k = 2$: Let $a_i = a_j = 1$. If $j = (i + 1) \bmod 6$, then there are 6 sub-cases, each with angle at q as $2\pi/3$ and direction pair $((i + 5) \bmod 6, i)$, as shown in Fig. 3.4c. Otherwise, there are 9 sub-cases, each with q occurring twice in $\overline{P}$—once (as a vertex or a non-vertex edge point) on the outer triangular polygon, and once on a triangular hole polygon (Fig. 3.4d).

- $k = 3$: Following are three possibilities.
  - The positions of three 1-bits in A_q are i, $(i+1) \bmod 6$, and $(i+2) \bmod 6$. There are 6 sub-cases in which q is not a vertex but a grid point on an edge of $\overline{P}$, and with direction pair (i, i); see Fig. 3.4e.
  - Two 1-bits have positions i and $(i + 1) \bmod 6$, and the third 1-bit in any position j other than $(i + 2) \bmod 6$. There are 12 sub-cases, each with q occurring twice in $\overline{P}$—once as a vertex of the outer triangular polygon, and once as a vertex of a triangular hole polygon (Fig. 3.4f).
  - All three 1-bits in A_q are non-consecutive. There are 2 sub-cases, each with q occurring thrice in $\overline{P}$—once as a vertex of the outer triangular polygon, and twice as vertices of two triangular hole polygons (Fig. 3.4g).

- $k = 4$: There are three possibilities as follows.
  - The positions of four 1-bits in A_q are $(i + j) \bmod 6$, for $j = 0, 1, 2, 3$ (Fig. 3.4h). There are 6 sub-cases, each with angle at q as $4\pi/3$ and direction pair $((i + 7) \bmod 6, i)$.
  - The positions of two 1-bits are i and $(i + 1) \bmod 6$, and those of the other two 1-bits are j and $(j + 1) \bmod 6$, where $j = (i + 3) \bmod 6$ (Fig. 3.4i). There are 3 sub-cases, each with q

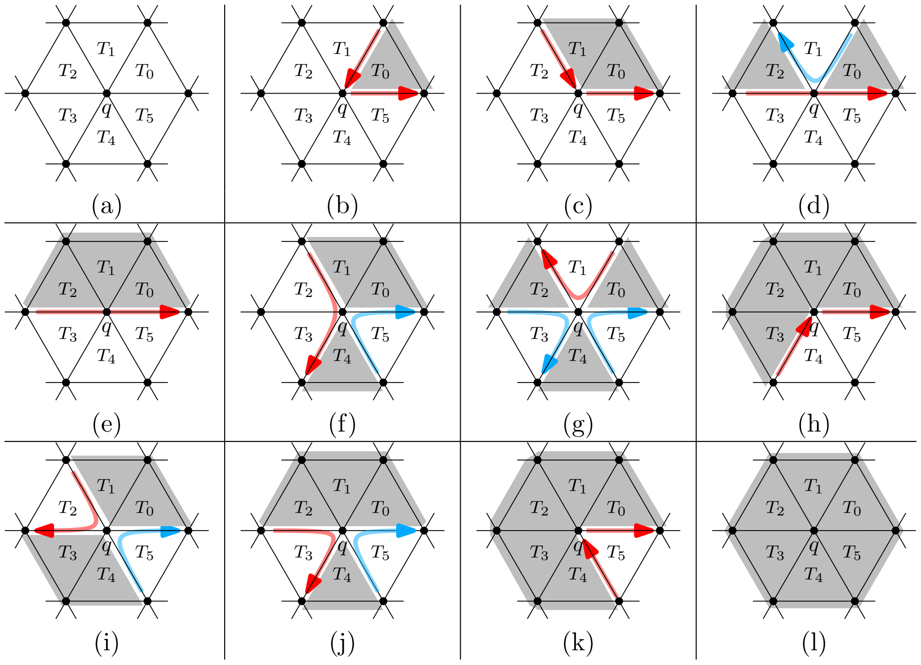


Figure 3.4: Classification of a grid point q for (a) $k = 0$, (b) $k = 1$, (c, d) $k = 2$, (e-g) $k = 3$, (h-j) $k = 4$, (k) $k = 5$, (l) $k = 6$. A red line indicates the traversal direction of the outer triangular polygon, and a blue line indicates that of an outer triangular hole polygon. Detailed explanations are given in Sec. 3.3.1.

occurring twice in $\overline{P}$—once as a vertex of the outer triangular polygon, and once as a vertex of a triangular hole polygon.

- The positions of three 1-bits are i, $(i+1) \bmod 6$, and $(i+2) \bmod 6$, and that of the fourth 1-bit is $(i+4) \bmod 6$ (Fig. 3.4j). There are 6 sub-cases, each with q occurring twice in $\overline{P}$, as in the previous case.

- $k = 5$: There are 6 sub-cases depending on the position of 0-bit in A_q (Fig. 3.4k). For each sub-case, the angle at q is $5\pi/3$. If $a_i = 0$, then the direction pair representing the incoming and the outgoing edges at q is $((i+3) \bmod 6, (i+1) \bmod 6)$.

3.3.2 Determining the next direction

The outgoing direction d' at a grid point q lying on $\overline{P}$ is computed from the incoming direction d and $A_q := \langle a_0 a_1 \cdots a_5 \rangle$. It is based on the following observation: The incoming direction d implies $a_j = 1$ and $a_{(j+1) \bmod 6} = 0$, where $j = (d+2) \bmod 6$. Hence, j is incremented (in modulo 6 arithmetic) until the next 1-bit in A_q is encountered, say at j', so that $a_{j'} = 1$. The outgoing direction is then given by $d' = j'$. For example, in Fig. 3.4f, $A_q = 110010$ and $d = 5$ gives $j = (5+2) \bmod 6 = 1$, implying $a_1 = 1$ and $a_2 = 0$; since $a_3 = 0$ and $a_4 = 1$, $d' = 4$ is obtained for the outer triangular polygon. While traversing the hole polygon in Fig. 3.4f, $d = 2$ gives $j = (2+2) \bmod 6 = 4$, implying $a_4 = 1$ and $a_5 = 0$; since $a_0 = 1$, $d' = 0$ is obtained.

Algorithm 4: Construct-OTC

```
Input  : $A, \mathbb{T}, p_0$
Output: $L$
$q \leftarrow q_0 \leftarrow$ Find-Start($p_0$)
$L \leftarrow \{q_0\}$
$d \leftarrow$ Find-Start-Dir($q_0$)
do
    $d' \leftarrow$ Find-Dir($d, q$)
    $q \leftarrow$ Next-Vert($d', q$)
    $d \leftarrow d'$
    $L \leftarrow L \cup \{q\}$
while $q \neq q_0$ ;
return $L$
```

3.3.3 Algorithm to construct $\overline{P}$

Algorithm 4 (Construct-OTC) shows the important steps to construct the outer triangular polygon of a digital object (single connected component without holes). It takes the object, A, the triangular grid, $\mathbb{T}$, and the top-left pixel of $p_0 \in A$, as input, and generates the sequence of vertices of $\overline{P}$ as output. The same algorithm is also used to construct the other polygons, e.g., outer triangular hole polygons and also other outer triangular polygons in case of a digital object with multiple components. For each component or each hole, the top-left pixel is supplied as an input.

In Step 4 of the algorithm, the top-left grid point (start vertex) of $\overline{P}$, namely q_0, is determined from p_0 using the procedure Find-Start. Its related details have already been explained in Sec. 3.3. The list L stores the sequence of vertices of $\overline{P}$. It is initialized with q_0 in Step 4. In Step 4, the start direction is determined based on the UGT containing p_0, as explained earlier and illustrated in Fig. 3.3. In the **do-while** loop (Steps 4–4), the outgoing direction d' and the next vertex are determined, until the next vertex coincides with the start vertex.

The outgoing direction d' at q is determined by the procedure Find-Dir in Step 4. Its related operations are explained in Sec. 3.3.2. From d', the next vertex is easily computed by Next-Vert in Step 4, using $\mathbb{T}$ and the coordinates of q. The ordered list of vertices, L, is returned finally in Step 4.

The inner cover, $\underline{P}$, can be constructed by the method of generating $\overline{P}$, but in the reverse manner. A grid point $q \in A$ is classified as a vertex of $\underline{P}$, if and only if at least one of the six UGTs incident at q is fully occupied by the object A i.e., $T_i^q \bigcap A = T_i^q$ where $i \in \{0, 1, 2, 3, 4, 5\}$.

3.3.4 Correctness and time complexity

From the combinatorial cases explained in Sec. 3.3.1 and from the strategy of getting outgoing directions explained in Sec. 3.3.2, the following observation can be proposed: For every grid edge e lying on $\overline{P}$, one UGT incident on e has object-occupancy and the other has not. Further, among these two UGTs, the interior of only the former lies inside $\overline{P}$. This implies that none of the UGTs whose edges lie on $\overline{P}$ and interiors lie inside $\overline{P}$, can be removed from $\overline{P}$, which proves the minimum-area criterion of $\overline{P}$ produced by Algorithm 4.

To prove that the start vertex q_0 is finally reached when the algorithm terminates, observe that the algorithm maintains the invariant that the object A always lies left during traversal. Since the incoming direction is left out while commencing the start direction from q_0, the outgoing direction from the last grid point on $\overline{P}$ finally leads to q_0, hence ensuring its successful termination.

For time complexity analysis, let n be the number of pixels comprising the contour (border) of a connected component comprising A. A UGT may contain as few as one contour pixel, but for every seven UGTs through which the contour passes in succession, there would be at least $O(g)$ contour pixels, as g is the grid size of $\mathbb{T}$. Hence, there are at most $O(n/g)$ UGTs containing the object contour, or, the number of grid points on $\overline{P}$ is at most $O(n/g)$.

To find the start vertex q_0 and the corresponding start direction, $O(1)$ time is required. So, Steps 4–4 of Algorithm 4 takes $O(1)$ time. Each UGT is checked in $O(g)$ time, as explained in Sec. 3.3.1. As each grid point has six neighboring UGTs, the time complexity to compute the object occupancy vector at each grid point q lying on $\overline{P}$ is $6 \cdot O(g) = O(g)$. Thus, for q, outgoing direction and next grid point finding are performed in $O(g)$ time. Hence, the *worst-case* time complexity of Algorithm 4 is $O(n/g) \cdot O(g) = O(n)$.

For the *best-case* analysis, observe that there can be $O(g^2)$ contour pixels in a UGT, which implies that there can be $O(n/g^2)$ UGTs containing the object contour, or, the number of grid points on $\overline{P}$ is $O(n/g^2)$. As each UGT can be checked for object occupancy in $O(g)$ time, The time complexity is $O(n/g^2) \cdot O(g) = O(n/g)$.

3.4 Experimental results

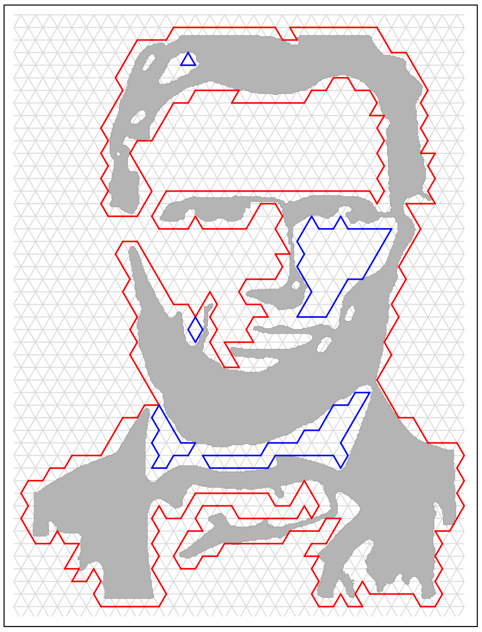
The proposed algorithm is implemented in C in Ubuntu 12.04, 32-bit, kernel version 3.5.0-43-generic, the processor being Intel core i5-3570, 3.40GHz FSB. The algorithm has been used to construct triangular covers of digital objects in various binary images of logo datasets, real-world objects, and geometric shapes, for different grid sizes.

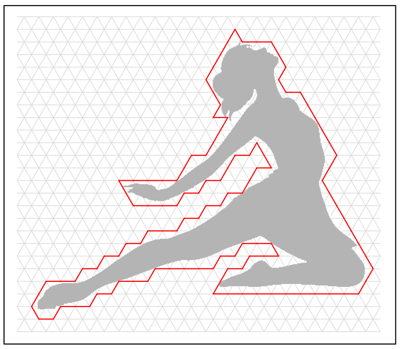
The outer triangular covers (OTCs) of three different objects are shown in Fig. 3.5. The outer triangular polygons are shown in red and outer hole polygons in blue. For `Lincoln` image, there are five outer hole polygons along with the outer triangular polygon. It may be noticed that there are a few small holes in the top-left region of the digital object of `Lincoln` image; however, only one outer hole polygon in that region is reported by this algorithm, as the other holes are not large enough to accommodate any polygon for $g = 20$.

Figure 3.6 shows the outer triangular covers of five objects from the datasets. The second and the fourth ones are geometric shapes, the third one is a logo image, and the first and the fifth are real-world objects. In each row, the input object, the number of pixels constituting its border (n), the OTCs for some grid sizes, and the number of vertices (n_v) of the OTCs are shown. The results indicate that n_v decreases with the increase of g. As g decreases, smaller irregularities on the object border do not leave much impression on $\overline{P}$, but the gross shape of the object gets captured through fewer vertices. The cover $\overline{P}$ captures the structural property of the object for smaller values of g, and it loses information with increasing values of g. For higher values of g, output complexity is low, resulting in less storage, but topological information is not always retained. For example, some of the outer triangular hole polygons may disappear for higher values of grid size. Such a collection of $\overline{P}$ for changing grid size might be useful for multi-scale shape analysis.

Table 3.1 presents a consolidated information of the OTCs for several objects at different grid sizes. In all cases, the number of vertices decreases with g, which is also evident from the plots showing n_v versus g in Fig. 3.7. The CPU times plotted against different grid sizes for four images are shown in this figure. As evident from these plots, the CPU time taken to compute the OTCs falls sharply with the increasing value of g, which indicates that the runtime of the algorithm on a real-world image follows the best-case runtime complexity of $O(n/g)$, as explained in Sec. 3.3.4. The results of inner and outer triangular cover on complex objects for various grid sizes are shown in Fig. 3.9 and Fig. 3.10.

To estimate the effectiveness of triangular covers in capturing the object shape, a comparison of the proposed algorithm has been done with the algorithm for finding isothetic covers proposed in


$n_v = 200$


$n_v = 66$

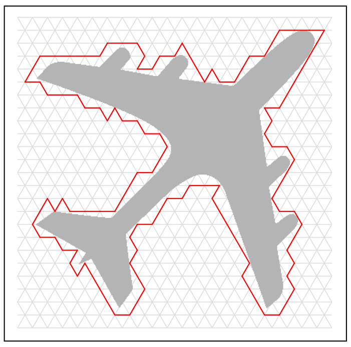

$n_v = 78$

Figure 3.5: Triangular covers produced by the proposed algorithm on `Lincoln`, `dancer`, and `aircraft` images, with $g = 20$.

[8]. Figure 3.8 shows this comparison for a geometric shape (`Letter_T`), a logo image (`Kangaroo`), and a real-world object (`Crane`), all for $g = 16$.

The object in `Letter_T` is perfectly aligned with the horizontal and the vertical lines of the isothetic grid. Hence, its isothetic cover has fewer vertices compared to its triangular cover; however, in area measure, the triangular cover is better compared to the isothetic cover. For the image `Kangaroo`, the number of vertices of the triangular cover is more than that of the isothetic cover; but the area of the triangular cover is comparatively less, indicating that it is tighter than the isothetic cover, and hence more compact. The boundary of the object in `Crane` has some portions aligned roughly along the grid lines of the triangular grid, which results in fewer vertices of the

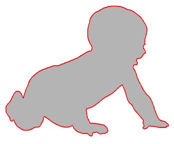
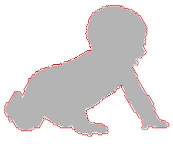
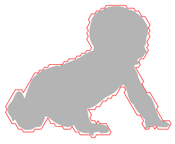
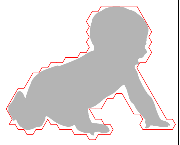
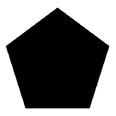
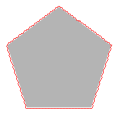
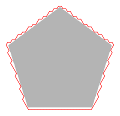
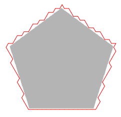
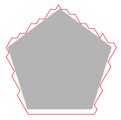
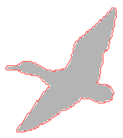
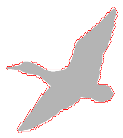
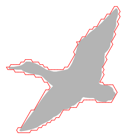
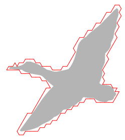
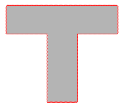
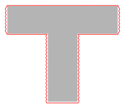
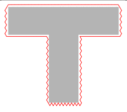
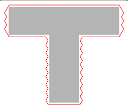
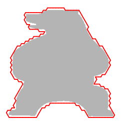
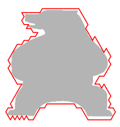
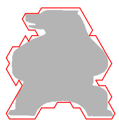
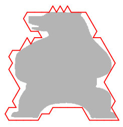
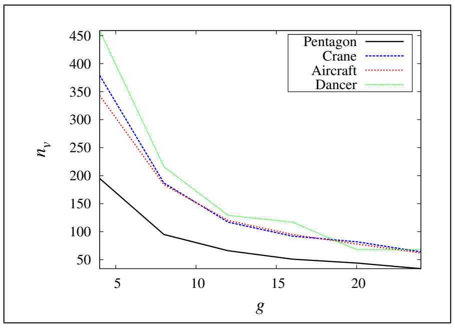
				
Baby $n = 3024$	$g = 1$ $n_v = 2976$	$g = 6$ $n_v = 243$	$g = 12$ $n_v = 129$	$g = 22$ $n_v = 71$
				
Pentagon $n = 1824$	$g = 6$ $n_v = 127$	$g = 12$ $n_v = 66$	$g = 18$ $n_v = 46$	$g = 22$ $n_v = 39$
				
Crane $n = 3178$	$g = 8$ $n_v = 187$	$g = 10$ $n_v = 143$	$g = 14$ $n_v = 97$	$g = 20$ $n_v = 82$
				
Letter_T $n = 2100$	$g = 4$ $n_v = 600$	$g = 10$ $n_v = 108$	$g = 16$ $n_v = 88$	$g = 22$ $n_v = 50$
				
Bear $n = 1240$	$g = 6$ $n_v = 120$	$g = 10$ $n_v = 52$	$g = 14$ $n_v = 32$	$g = 16$ $n_v = 40$

Figure 3.6: Experimental results for different grid sizes. (n = number of pixels on the object contour; n_v = number of vertices of $\overline{P}$.)

triangular cover compared to the isothetic cover; and here also, the area measure of the triangular cover is less than that of the isothetic cover. Thus, the triangular covers are particularly effective, both in terms of vertex count and area measure, when the object boundary has some alignment with the underlying grid. An isothetic cover, in general, may have a smaller set of vertices, and so its computational time may be less. On the other hand, a triangular cover usually requires less area to encompass the object; and hence, it offers more compactness, although it can have more vertices and may require more computational time.



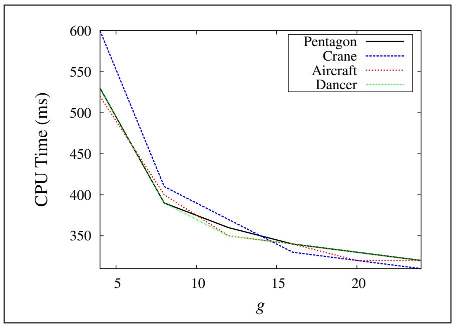
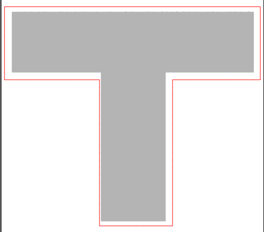
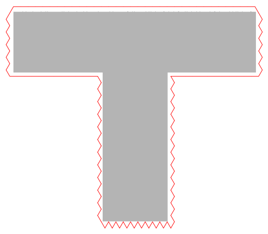
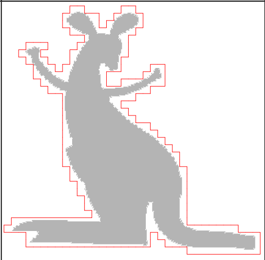
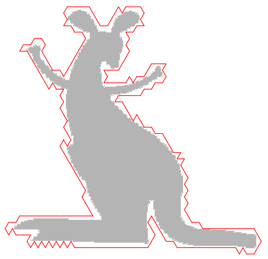
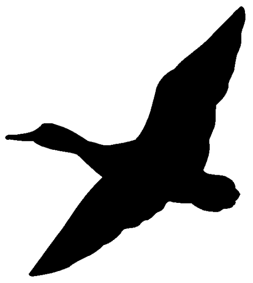
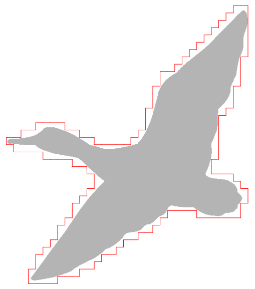
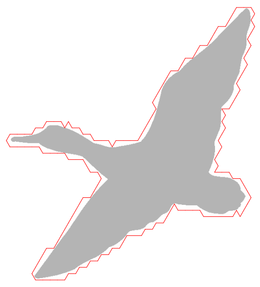


Figure 3.7: Plots on (left) number of vertices and (right) CPU time versus g for `aircraft`, `dancer`, `Crane`, and `pentagon` images.

Object	Isothetic Cover	Triangular Cover
		
		
		

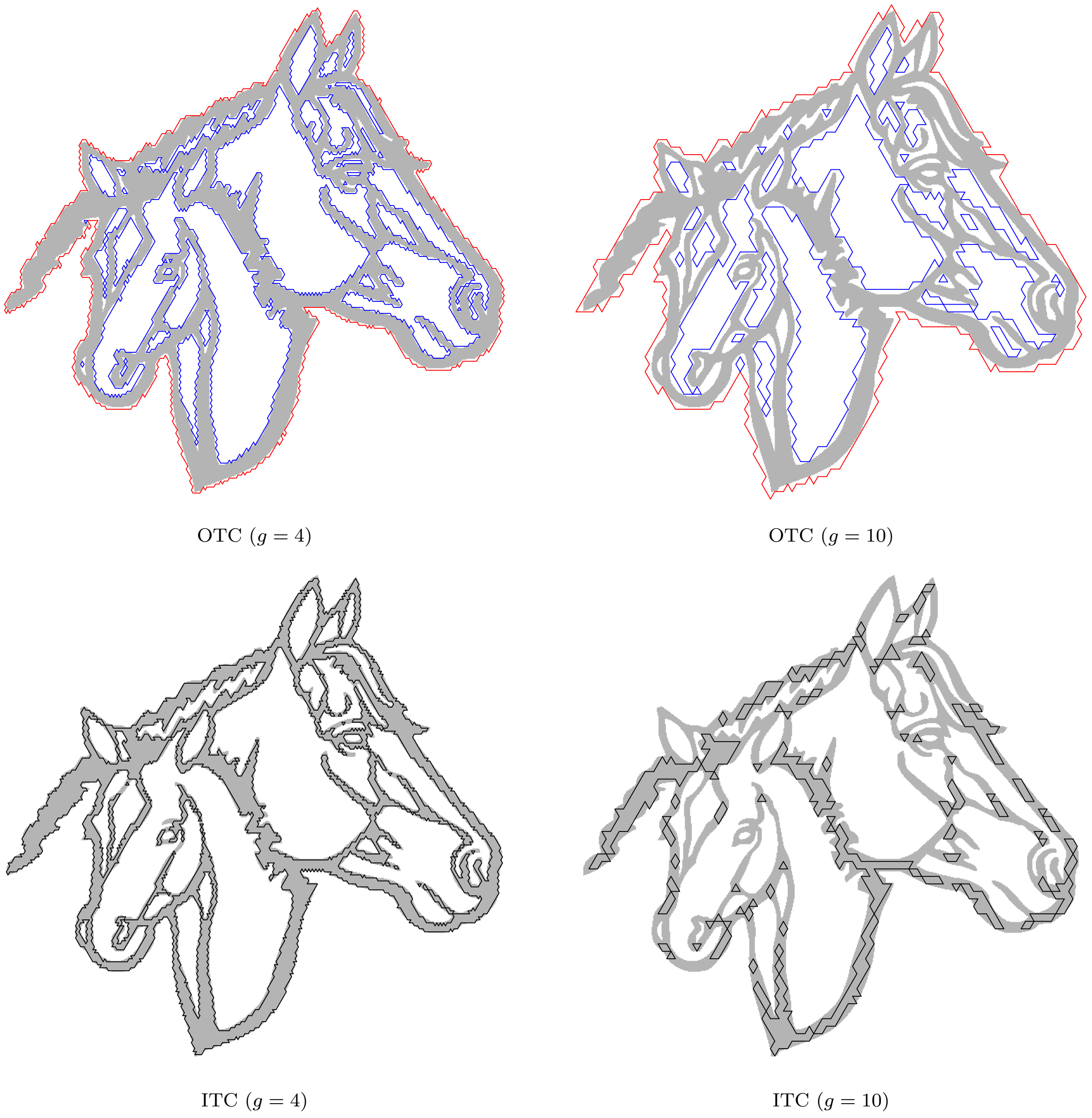
Object	Area	Perimeter	Outer Cover	n_v	Area	Perimeter
`Letter_T`	117966	2100	Triangular	88	136014	2320
			Isothetic	8	140800	2080
`Kangaroo`	92292	4926	Triangular	128	113955	3392
			Isothetic	114	122112	3392
`Crane`	101362	2864	Triangular	92	118057	2512
			Isothetic	110	123904	2848

Figure 3.8: Comparison of the triangular covers produced by the proposed algorithm with the isothetic covers produced by the algorithm in [8], all for $g = 16$. (n_v=number of vertices.)

Object A	$\lvert A\rvert$	n	g	n_{60}	n_{120}	n_{240}	n_{300}	n_v	n_T	$\lvert\overline{P}\rvert$	$\mathrm{p}_{\overline{P}}$	T
`Baby`	132629	3024	4	45	159	149	47	400	19760	136901	2744	600
			8	26	81	69	29	205	5093	141141	2760	410
			12	2	68	52	7	129	2350	146531	2568	360
			16	4	46	30	9	89	1348	149427	2592	340
			20	3	35	21	7	66	887	153633	2540	320
			24	2	31	19	5	57	619	154388	2472	310
`Pentagon`	116711	1636	4	19	81	77	18	195	17185	119061	1516	530
			8	10	40	36	9	95	4401	121964	1528	390
			12	0	36	30	0	66	2004	124957	1464	360
			16	2	26	22	1	51	1152	127701	1504	340
			20	3	22	16	3	44	745	129038	1540	330
			24	2	18	12	2	34	537	133936	1560	320
`Aircraft`	50607	2724	4	44	131	123	45	343	7835	54282	2460	520
			8	23	75	57	29	184	2086	57809	2496	400
			12	5	61	43	11	120	998	62229	2352	350
			16	5	46	38	6	95	593	65735	2320	340
			20	8	34	28	8	78	396	68589	2360	320
			24	4	30	24	4	62	286	71333	2208	320
`Crane`	101362	2864	4	47	147	135	50	379	15199	105302	2652	600
			8	21	77	65	24	187	3951	109493	2648	410
			12	5	57	49	6	117	1828	113983	2520	370
			16	2	49	35	6	92	1065	118057	2512	330
			20	4	40	34	4	82	704	121936	2560	320
			24	3	32	26	3	64	506	126204	2544	310
`Letter_T`	117966	2100	4	168	135	129	168	600	17480	121105	2800	570
			8	20	68	64	19	171	4502	124763	2288	410
			12	45	47	43	44	179	2057	128262	2676	360
			16	10	37	31	10	88	1227	136014	2320	330
			20	0	32	24	1	57	805	139430	2180	320
			24	1	26	20	1	48	551	137428	2184	310
`Dancer`	46280	2997	4	75	161	141	82	459	7295	50541	2772	530
			8	28	84	74	30	216	1967	54511	2608	390
			12	11	56	52	10	129	951	59298	2472	350
			16	16	46	38	17	117	565	62631	2592	340
			20	4	33	27	4	68	387	67030	2280	330
			24	8	29	23	8	68	274	68340	2352	320
`Bear`	36450	1238	4	27	67	53	31	178	5511	38181	1208	130
			8	7	39	25	11	82	1443	39990	1144	90
			12	4	24	12	7	47	690	43024	1116	90
			16	7	15	13	5	40	395	43786	1136	80
			20	1	20	10	3	34	273	47285	1100	70
			24	3	18	8	5	34	189	47139	1176	70

$\lvert A\rvert$ = number of object pixels; n = number of pixels on the boundary of A; g = grid size; n_θ = number of vertices with internal angle θ $(= 60, 120, 240, 300)$; n_v = total number of vertices of $\overline{P}$; n_T = number of UGTs in outer cover; $\lvert\overline{P}\rvert$ = number of pixels in $\overline{P}$; T = CPU time in milliseconds; $\mathrm{p}_{\overline{P}}$ = perimeter of $\overline{P}$.

Table 3.1: Summary of experimental results.


OTC ($g = 4$) OTC ($g = 10$)

ITC ($g = 4$) ITC ($g = 10$)

Figure 3.9: The OTCs and ITCs corresponding to the image `Horse` for $g = 4$ and $g = 10$.

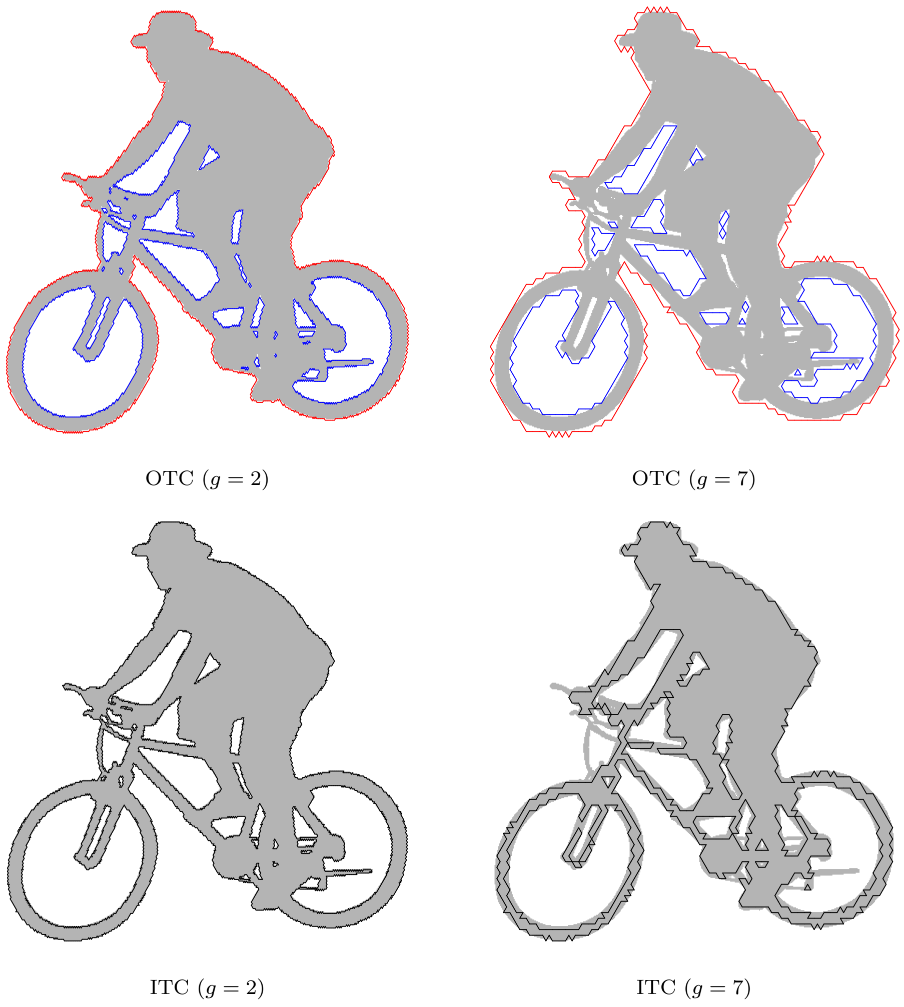


Figure 3.10: The OTCs and ITCs corresponding to the image `Cyclist` for $g = 2$ and $g = 7$.

Chapter 4

Application of Triangular Covers

An approximate shape of an object is captured by the triangular cover from which a chain code representing the shape of the object is derived which can be used for shape analysis. For this purpose we have put to use the Levenshtein Distance which is the minimum number of insertions, deletions or substitutions required to convert one string i.e., chain code into other [64].

- A database consisting of 125 shapes has been created. Some of these shapes are downloaded from the internet and some are taken from MPEG7 database.
- All the images are normalized to 256×256 such that the width and height of the bounding rectangle of all the shapes are same.
- The chain codes of all the shapes are generated using Algo. 3.3.3 and are saved in a file $B.txt$.
- For the query image, the chain code is also generated known as the query code.
- The Levenshtein Distance between the query code and the file containing all the chain codes of the shapes are computed and stored in another file say $C.txt$.
- The values of $C.txt$ which are the dissimilarity measures, are sorted in ascending order.
- Based on the sorted values of $C.txt$, the shapes are retrieved matching with the query image in an increasing order of the Levenshtein distance measure. This is because of the fact that the shapes having less Levenshtein measures are more similar to the query shape.

Several experimental results have been shown in Figures 4.1, 4.2, 4.3 and 4.4 where shapes similar to the query image have been retrieved from a given shape database in the increasing order of their Levenshtein distances.

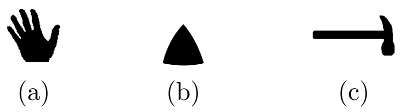


Figure 4.1: Query Images (a) `Hand`, (b) `Geometric Shape` and (c) `Hammer`.

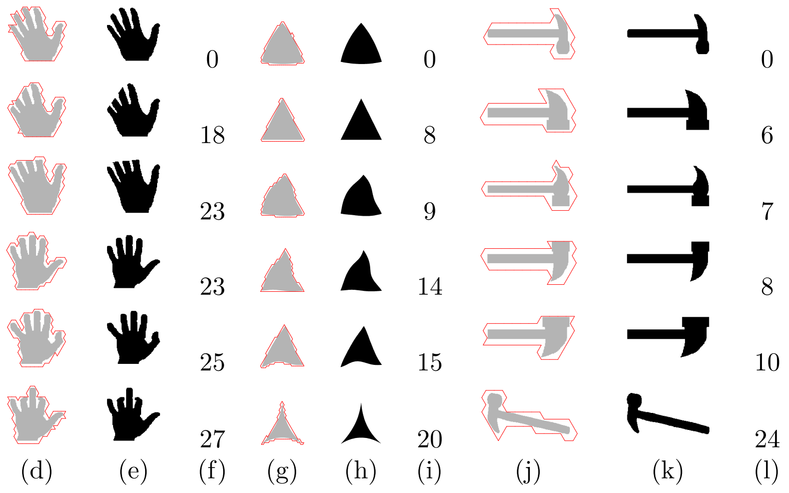


Figure 4.2: (d), (g), (j) Triangular Cover at $g = 20$. (e), (h), (k) Original Image. (f) Levenshtein Distance w.r.t. Fig. 4.1a. (i) Levenshtein Distance w.r.t. Fig. 4.1b. (l) Levenshtein Distance w.r.t. Fig. 4.1c.

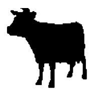


Figure 4.3: Query Image (a) `Cow`.

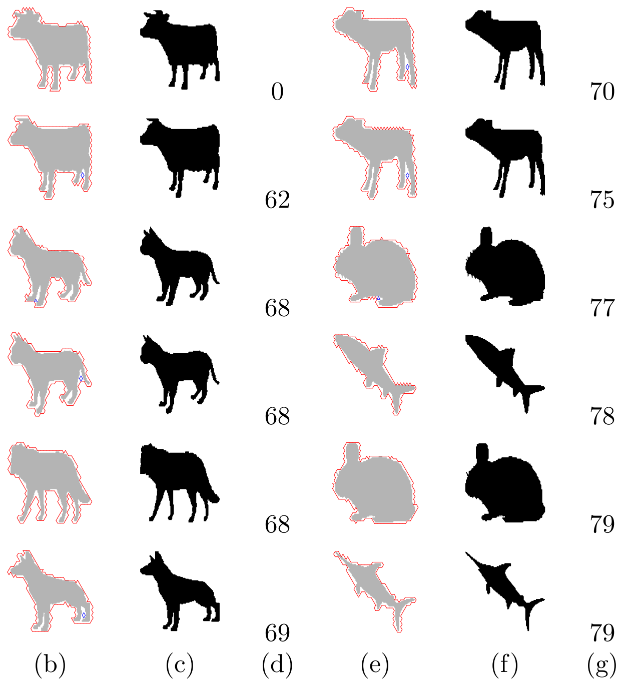


Figure 4.4: (b), (e) Triangular Cover at $g = 10$. (c), (f) Original Image. (d), (g) Levenshtein Distance w.r.t. Fig. 4.3a.

Chapter 5

Finding Largest Rectangle inside a Digital Object and its Rectangularization

5.1 Introduction

One of the geometric optimization problem in the class of polygon inclusion is finding the largest area axis-parallel rectangle (LR) inside a general polygon of n vertices [13]. For several practical importance the problem can be formulated for various scenarios (e.g. in convex polygon, in orthogonal polygon, etc.). The detection of the largest area subrectangle with sides parallel to the given rectangle of N points where the subrectangle does not contain any of the N points is proposed in [15, 16] with $O(N \log^3 N)$ time complexity and $O(N \log N)$ space complexity. Aggarwal et. al. [1] stated an improved algorithm that takes same $O(n \log^3 n)$ time but $O(n)$ space. Another algorithm proposed by them has $O(n \log^2 n)$ time complexity and $O(n)$ space complexity. The geometric optimization problem of finding maximum area axis parallel rectangle from a n-vertex general polygon is presented in [22] where the different cases are considered based on the types of contacts between the rectangle and the polygon. A framework is also proposed that can transform an algorithm for orthogonal polygons into an algorithm for non-orthogonal polygons and showed that the running time of their algorithm for general polygons to be $O(n \log^2 n)$. Also, the lower bound for finding the largest empty rectangles in both self-intersecting polygons and general polygons with holes has been established to be $O(n \log n)$.

McKenna et al. [44] use a divide-and-conquer approach to find the largest rectangle in an orthogonal polygon in $O(n \log^5 n)$ time. For the merge step at the first level of divide-and-conquer, they obtain an orthogonal, vertically separated, horizontally convex polygon. At the second level, their merge step produces an orthogonally convex polygon, for which they solve the LR problem in $O(n \log^3 n)$ time. They also establish a lower bound of time in $\Omega(n \log n)$ for finding the largest rectangle in orthogonal polygons with degenerate holes, which implies the same lower bound for general polygons with degenerate holes.

LR problem has many applications in electronic design automation and design and verification of physical layout of integrated circuits [52, 63]. Largest area rectangle problem has many interesting industrial applications also, e.g., consider a sheet of fabric or a rectangular piece metal with certain number of flaws in it. This problem can be applied to find a maximum area rectangular sheet that

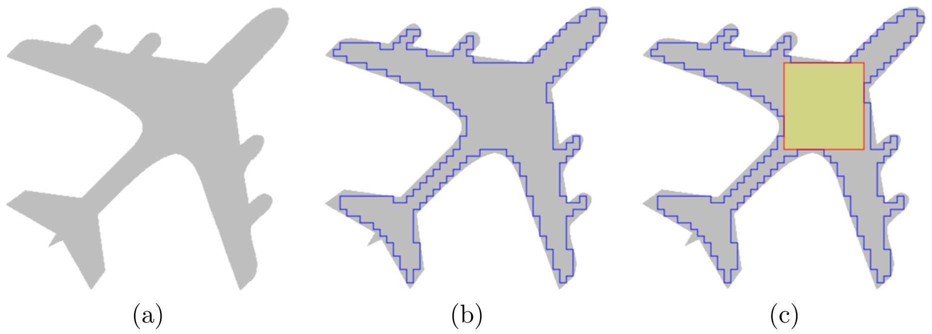


Figure 5.1: (a) The digital object, A, (b) Inner isothetic cover ($g = 8$), (c) Largest Rectangle.

does not contain any flaws.

Another variation of the same problem is presented here— finding largest rectangle in a digital object which is applicable for shape analysis of the object. The inner isothetic cover of the digital object is found which tightly inscribes the digital object. The digital object and the inner isothetic covers are shown in Fig. 5.1(a)-(b). It is to be noted that the grid size may be varied for better shape analysis of the digital object. The boundary of the inner isothetic cover is traversed counter clockwise and the combinatorial rules are applied wherever required. The resulting largest rectangle may not be unique. A largest rectangle for the object shown in Fig. 5.1(a) is given in Fig. 5.1(c) for grid size $g = 8$. The algorithm presented in this chapter runs in $O(n \log n)$ time, where n is the number of pixels on the contour of the digital object.

The chapter is organized as follows. Section 5.2 presents the definitions and preliminaries required to understand the chapter. Section 5.3 describes the rules to determine a largest rectangle inside a digital object. The properties of largest rectangle are discussed in Sec. 5.4. Experimental results and related data are given in Sec. 5.5. Section 5.6 presents the concept of rectangularization using largest rectangle which is a novel concept.

5.2 Definitions and Preliminaries

A *digital object*, A (henceforth referred to as an object), is a finite subset of $\mathbb{Z}^2$, which consists of one or more $k(= 4$ *or* $8)$-*connected components* [37]. In this chapter, we consider the object as a single 8-connected component. We need the following definitions in the context of our work.

Definition 11 (Grid)**.** *The* background grid *is given by* $\mathbb{G} = (\mathbb{H}, \mathbb{V})$*, where* $\mathbb{H}$ *and* $\mathbb{V}$ *represent the respective sets of (equi-spaced) horizontal grid lines and vertical grid lines. The* grid size*,* g*, is defined as the distance between two consecutive horizontal/vertical grid lines. A* grid point *is the point between intersection of a horizontal and a vertical grid line. A* unit grid block *(UGB) is the smallest square having its four vertices as four grid points and edges as grid edges.*

Definition 12 (Isothetic Polygon)**.** *An* isothetic polygon P *is a simple polygon (i.e., with non-intersecting sides) of finite size in* $\mathbb{Z}^2$ *whose alternate sides are subsets of the members of* $\mathbb{H}$ *and* $\mathbb{V}$*. The polygon* P*, hence given by a finite set of UGBs, is represented by the (ordered) sequence of its vertices, which are grid points. The border* B_P *of* P *is the set of points belonging to its sides. The interior of* P *is the set of points in the union of its constituting UGBs excluding the border of* P*.*

An isothetic cover has two type of vertices 90^0 (type **1**) and 270^0 (type **3**).

Definition 13 (Inner Isothetic Cover)**.** *The* inner (isothetic) cover *(IIC), denoted by $P(A)$, is a set of inner polygons and (inner) hole polygons, such that the region, given by the union of the inner polygons minus the union of the interiors of the hole polygons, contains a UGB if and only if it is a subset of A.*

Definition 14 (Convex Edge and Convexity)**.** *An edge of P defined by two consecutive vertices of type* **1** *is termed as a* convex edge*, as it gives rise to a* convexity*.*

Definition 15 (Concave Edge and Concavity)**.** *An edge defined by two consecutive type* **3** *vertices gives rise to a* concavity*, and hence termed as a* concave edge*.*

5.2.1 Deriving the Inner Isothetic Cover

Using the algorithm in [9, 6], we obtain (the ordered set of vertices of) P for A, which is, therefore, the maximum-area isothetic polygon inscribing A. During the construction of P, the vertices and grid points lying on the edges of P are dynamically inserted in a circular doubly-linked list, L, and simultaneously in two lexicographically sorted (in increasing order) lists, L_x and L_y, with respective primary and secondary keys as x- and y-coordinates for L_x, and vice versa for L_y. Each node of the list L has two level pointers, the lower level pointers are used to link both edge and corner points of IIC and the top level pointers are used to link only the corner points of IIC.

5.3 Procedure to Determine a Largest Rectangle

To find a largest rectangle, the inner isothetic cover P (constructed using the algorithm in [9, 6]) is traversed in anti-clockwise direction from its top left corner. During this traversal, whenever a convex edge is encountered, corresponding histogram polygon (i.e., a portion of the main polygon) is constructed. Largest rectangle inscribed in the histogram polygon is determined. The convexity encountered in P is reduced using the appropriate reduction rule. After reduction, it may give rise to a convex edge, corresponding to which the histogram polygon, thereof the largest rectangle inscribed in it, is determined. Thus, the traversal continues till it reaches the start point of P, the largest of all such largest rectangles, inscribed in the histogram polygons of convex edges, is the resulting largest rectangle of P.

5.4 Properties of Largest Rectangle in a Digital Object

The largest rectangle tightly fits into the inner isothetic cover. It exhibits several interesting properties.

Theorem 6. *The four sides of the largest rectangle must coincide with at least four edges of the inner isothetic cover partly or fully.*

Proof. A largest rectangle tightly fits in the inner isothetic cover. Let us consider the smallest possible rectangle inside an isothetic polygon. Let the four sides (left, right, top, and bottom) of the rectangle be s_l, s_r, s_t and s_b, respectively. The rectangle has to be expanded as much as possible. Suppose s_l, s_r, s_t and s_b are moved towards left, right, top, and bottom, until there are some obstructions. These obstructions are nothing but the edges of the inner cover.

Now, let us assume that three sides of the largest rectangle coincides with the edges of the inner isothetic cover partly or fully. There is one side which does not coincide with any of the edges of the isothetic polygon. Thus, there is a scope for expansion upto the obstruction, i.e., upto the edge(s) of the isothetic polygon. Hence, it can be stated here that the four sides of the largest rectangle coincide with the edges of the inner isothetic cover, partly or fully. It is trivial to proof that for four sides at least four obstruction (i.e., edges) are required. □

Area Coefficient:
Largest rectangle is not unique. There may be more than one largest rectangle having same area. In most of the cases, it is seen that the largest rectangle represents the central area of the digital object. The following shape signature can be defined in this context. Let α be the area-coefficient of the largest rectangle, i.e. the portion of the digital object occupied by the largest rectangle. It is defined as follows.

$$\alpha = \frac{A_{LR}.area}{A_{in}.area} \tag{5.1}$$

In Eqn. 5.1, $A.area$ and $A_{LR}.area$ are the area of the inner isothetic cover, A_{in}, inscribing the digital object, A, and the largest rectangle, A_{LR}, respectively. α becomes 1, when the largest rectangle occupies the whole area of A_{in} (where A_{in} must be a rectangle). If the value of α is almost equal to zero, then it implies that the corresponding object is very complicated. When the value of α is high, then it can be said that the object is simple and largest rectangle occupies most of the portion of the object.

5.5 Experimental Results

The proposed algorithm is implemented in C in Ubuntu 12.04, 64-bit, kernel version 3.5.0-43-generic, the processor being Intel i5-3570, 3.4 GHz FSB. Experimental results of different digital objects are shown in Fig. 5.2. It is seen that a largest rectangle occupies approximately one-third area of inner isothetic polygon.

Some important observations on largest rectangle from the experimental results are noted here which make it useful for shape analysis of a digital object. The largest rectangles for various grid sizes inside the same digital object are shown in Fig. 5.3. The areas of the largest rectangles remain almost same for different grid sizes of the object. The position of the largest rectangles may slightly vary which are shown in Fig. 5.3, as the inner isothetic cover changes slightly for different grid sizes.

A largest rectangle for various rotations of the `Hand` image is shown in Fig. 5.4. It is seen that the largest rectangle every time appears almost in the same position— in the plam. The positions for all the rectangles are almost same. After rotation the area of the largest rectangle will change. It can be claimed that the largest rectangles are position invariant for various rotations. Largest rectangle can be used to detect central region of a digital object. This property may be useful for shape analysis of the digital object.

5.6 Rectangularization Using Largest Rectangle

The intersection of A_{in} with the largest rectangle may create a set of sub-polygons, $\mathcal{A}$. The cardinality of this set will be zero only when A_{in} itself a rectangle. Let the cardinality of $\mathcal{A}$ be t and $\mathcal{A} = \{a_1, a_2, \ldots, a_t\}$. For each of the sub-polygons in $\mathcal{A}$, largest rectangles can be obtained. Let a_i^{LR} be the largest rectangle for the sub-polygon a_i. Similarly, a_i^{LR} divides a_i into many sub-polygons. By recursive generation of this procedure, a tree can be formed which is termed as *LR-Tree*. The number of branches at each level is the number of sub-polygons generated on intersection of the

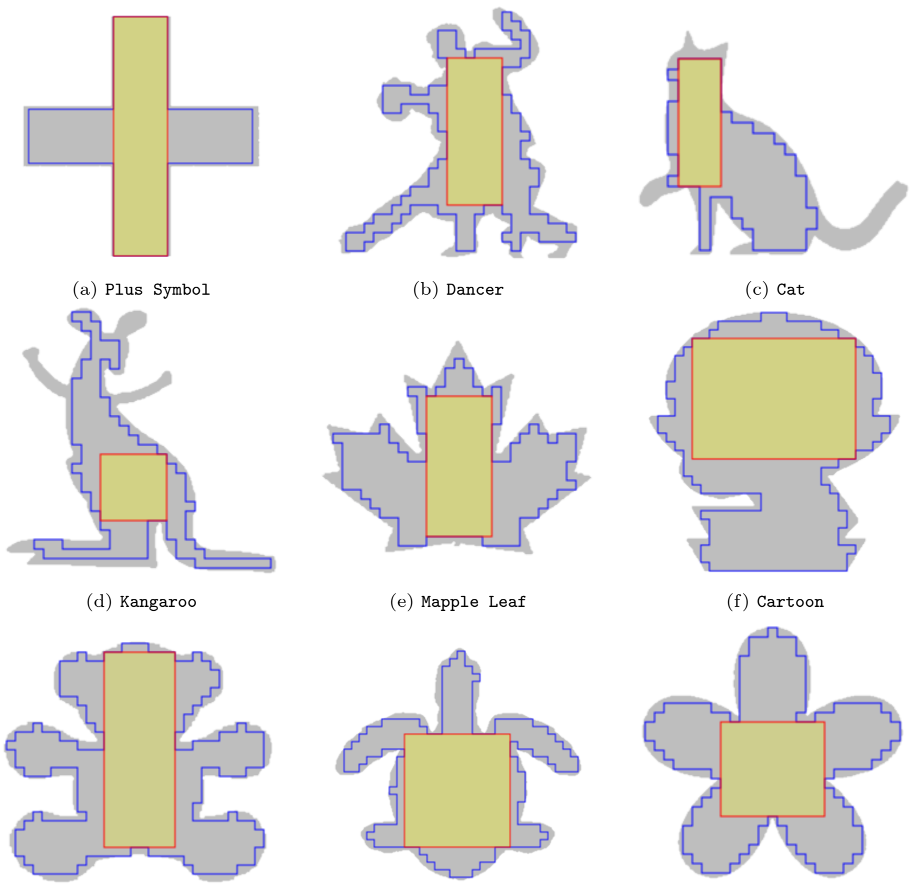
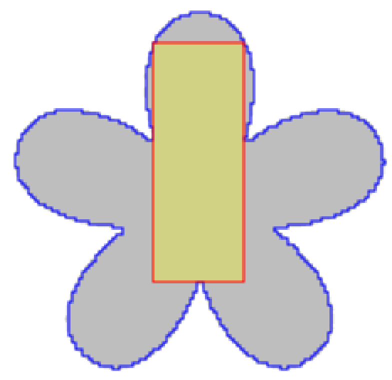
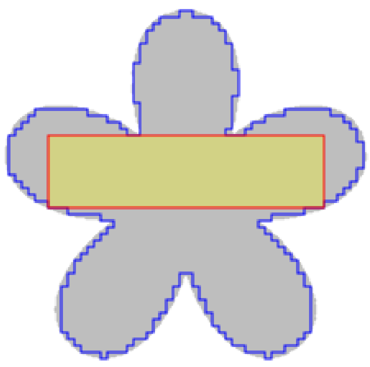

(a) `Plus Symbol` (b) `Dancer` (c) `Cat`
(d) `Kangaroo` (e) `Mapple Leaf` (f) `Cartoon`
(g) `Teddy` (h) `Tortoise` (i) `Flower`

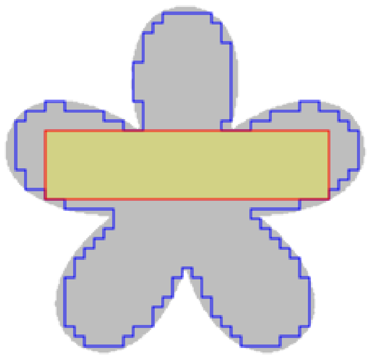
Figure 5.2: Largest area rectangles (shaded in yellow) inside six different objects for $g = 8$.

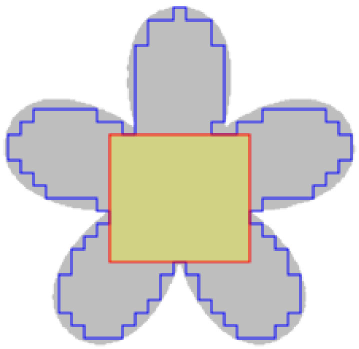
largest rectangle with the polygon or sub-polygon. A node in the LR-Tree will be a leaf node for which $t = 0$ and $\alpha = 1$. Each node contains two information and they are the area-coefficient and the level. All these information are useful for shape analysis of the digital object. This procedure leads to rectangularization of the inner isothetic cover. The inner isothetic cover is decomposed into non-overlapping rectangles (overlapping occurs only at the boundaries) based on largest rectangle criteria. Two types of edges are there in the LR-Tree— *strong edge* (represented by solid and thick line) and *weak edge* (represented by dashed line). A node at level i will be connected to a node at level $i + 1$ by strong edge if the two corresponding rectangles coincide at one side, otherwise they will be connected by weak edge. The LR-Tree can be converted into a *LR-Graph* based on rectangularization by adding one more edge termed as *forward edge*. The two nodes at level i and level j where $|i - j| = 1$, are connected by forward edge (denoted by solid line) if the two corresponding rectangles have overlapping boundaries.

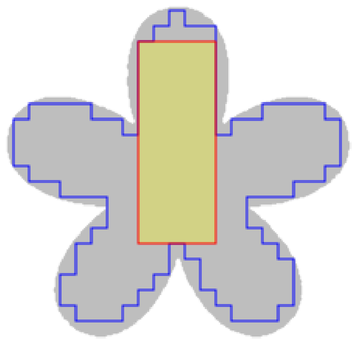
A rich literature survey exists on the polygon decomposition problem. A polygon can be decomposed into convex components, rectangles, star-shaped components, monotone components, etc.


(a) $g = 2$ $area = 7668$


(b) $g = 4$ $area = 7392$


(c) $g = 6$ $area = 7308$


(d) $g = 8$ $area = 7040$


(g) $g = 10$ $area = 6500$

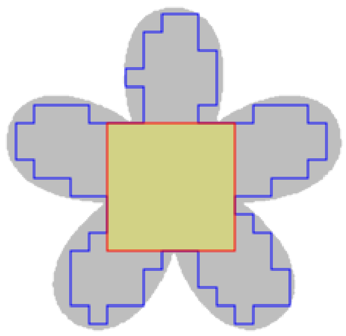

$g = 12$ $area = 7056$

Figure 5.3: Position of largest rectangle of an object for different grid size.

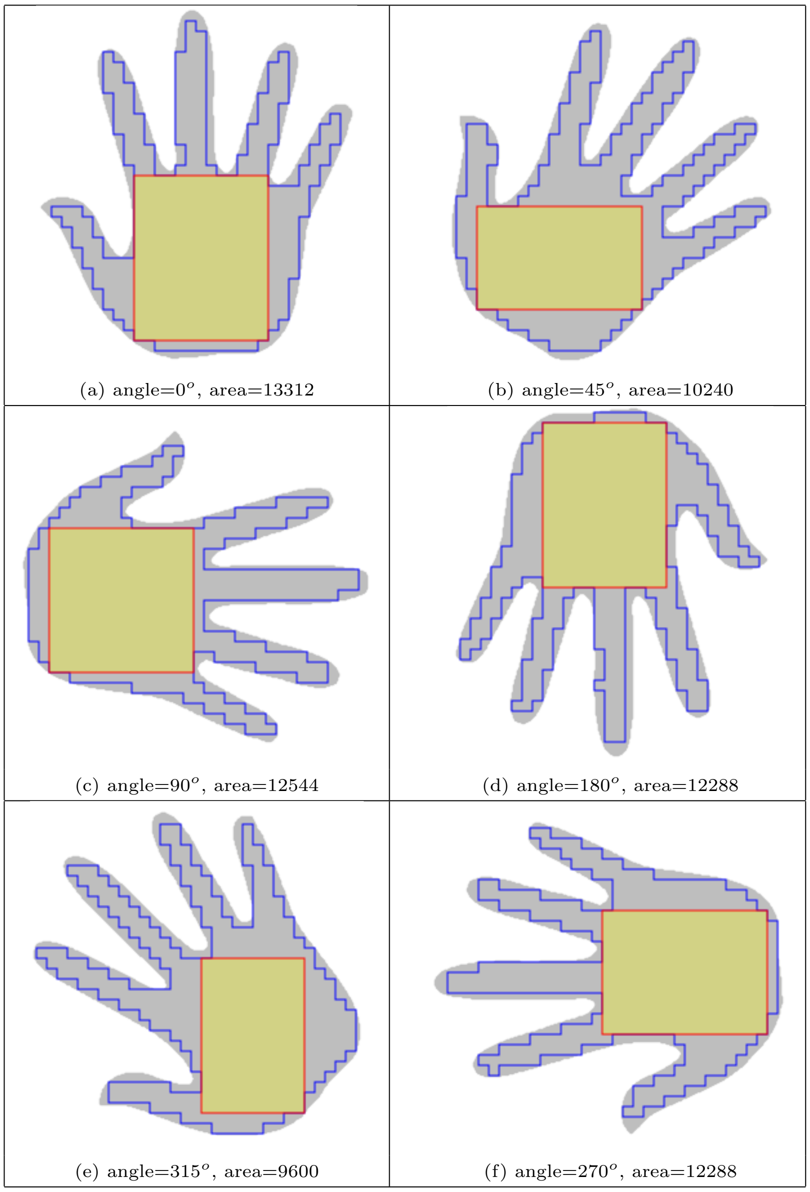
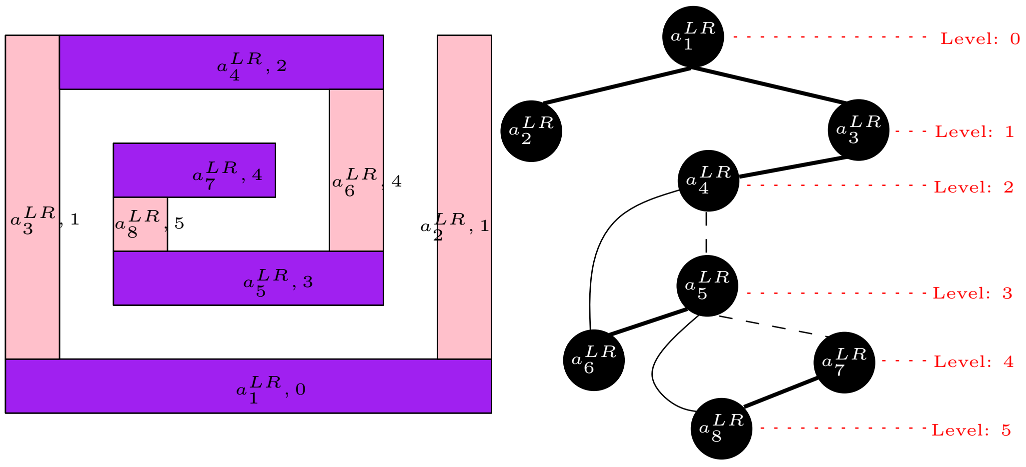
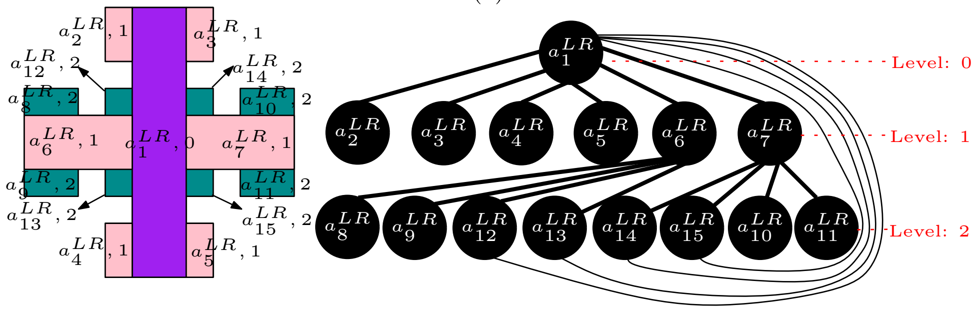

(a) angle=0^o, area=13312
(b) angle=45^o, area=10240
(c) angle=90^o, area=12544
(d) angle=180^o, area=12288
(e) angle=315^o, area=9600
(f) angle=270^o, area=12288

Figure 5.4: Largest area rectangle (shaded in yellow) obtained in six different rotations of same object for $g = 8$.

Here, in this chapter we will only give a brief study on the decomposition of polygons in rectangular components. Polygon decomposition has many applications in theory and practice, especially in visual pattern recognition and image analysis [14, 36, 54, 56, 62]. Minimum rectangular partition of digitized blobs has been performed by using the properties of bipartite graph in [29]. A technique for partitioning of orthogonal polygons into rectangles by extension of edges from reflex vertices is



(a)



(b)

Figure 5.5: Rectangularization using largest rectangle and corresponding LR-Graphs. The levels are mentioned in the rectangles.

shown in [3]; upper and lower bounds for the number of convex components have also been obtained. Hence, rectangularization using largest rectangle is a novel work.

For different types of object, the LR-Graph will have different structures. The LR-Graph for the object shown in Fig. 5.5(b) has no weak edges. The number of levels (if LR-Tree is considered) is less compared to the number of nodes. The structure of this object is symmetric. It indicates less complicated object. For the spiral type object, the number of levels is higher in LR-Tree. The LR-Graph contains all types of edges. It indicates a complex object. For shape analysis, several metrics can be defined based on the types of edges, number of levels, number of nodes, area-coefficient of each nodes, etc.

Chapter 6

Conclusion

The proposed algorithm (FSIP) in Chapter 2 finds the region containing all possible shortest isothetic paths between two given points. The algorithm is efficient, and given the inner isothetic cover P, it runs in a time linear in the number of grid points on the border of P. The algorithm can be applied on digital objects without holes. However, for objects with holes, it requires appropriate modifications. Results for multiple FSIPs are also shown in this chapter, where the control points are chosen manually. An algorithm or heuristic can be designed to find such control points in a given digital object, which may enhance this work and further its scope towards effective shape analysis, such as finding critical regions, as shown in this chapter.

We have proposed an algorithm (Chapter 3) for constructing a minimum-area cover of a digital object on a triangular grid. It can be used to determine all outer triangular polygons including the outer triangular hole polygons in a digital object with multiple components and holes. The minimum-area cover provides a compact geometric information of the object, which is useful in shape characterization along with those obtained from orthogonal or isothetic covers. Further, its runtime can be controlled by tuning the grid size, which offers a mechanism of trading off speed with granularity of the object boundary. Experimental results demonstrate that a triangular cover can serve as an approximate estimate of shape complexity and of similar other signatures used for characterizing a digital object.

There are several applications of triangular covers, one of them have been implemented and analysed as shown in Chapter 4 with results shown. Further analysis of this shape retrieval technique will be exploited in the near future. Triangular polygons can further be studied for digital convexity, medical image processing, developing triangular convex hull etc.

Chapter 5 presents a combinatorial algorithm to find a largest area rectangle inside a digital object in $O(k.n/g + (n/g)\log(n/g))$ time. Experimental results of the algorithm along with data and related explanations are also provided. The usefulness of the largest rectangle in the context of shape analysis is also given. The decomposition of a digital object using largest rectangles is given here. This problem has some industrial applications also which has been stated in Sec. 5.1. In future, some topological information for the digital objects can be derived from above mentioned technique. The shape analysis using LR-Graph and rectangularization can be done later on. All these implies the practical importance of the problem in various shape related applications.

Bibliography

[1] A. Aggarwal and S. Suri. Fast algorithms for computing the largest empty rectangle. In *Proc. 3rd Annu. Symposium on Computational Geometry*, pages 278–290, 1987.

[2] E. Arkin, J. Mitchell, and C. Piatko. Minimum-link watchman tours. *Information Processing Letters*, 86:203–207, 2003.

[3] A. L. Bajuelos, A. P. Tomás, and F. Marques. Partitioning orthogonal polygons by extension of all edges incident to reflex vertices: lower and upper bounds on the number of pieces. In *Proceedings of the International Conference Computational Science And Its Applications (ICCSA'04)*, volume 3045 of *LNCS*, pages 127–136, Assisi, Italy, 2004. Springer-Verlag.

[4] M. Beeson. Triangle tiling I: The tile is similar to ABC or has a right angle. *arXiv preprint arXiv:1206.2231*, 2012.

[5] C. P. D. Birch, S. P. Oom, and J. A. Beecham. Rectangular and hexagonal grids used for observation, experiment and simulation in ecology. *Ecological Modelling*, 206(3):347–359, 2007.

[6] A. Biswas, P. Bhowmick, and B. B. Bhattacharya. **TIPS**: On finding a tightisothetic polygonal shape covering a 2d object. In *14th Scandinavian Conference on Image Analysis (SCIA)*, volume 3540, pages 930–939. Springer, Berlin, 2005.

[7] A. Biswas, P. Bhowmick, and B. B. Bhattacharya. TIPS: On Finding a Tight Isothetic Polygonal Shape Covering a 2D Object. In *Proceedings of the 14th Scandinavian Conference on Image Analysis*, volume 3540 of *LNCS*, pages 930–939, Joensuu, Finland, 2005. Springer-Verlag.

[8] A. Biswas, P. Bhowmick, and B. B. Bhattacharya. Construction of isothetic covers of a digital object: A combinatorial approach. *Journal of Visual Communication and Image Representation*, 21(4):295–310, 2010.

[9] A. Biswas, P. Bhowmick, and B. B. Bhattacharya. Construction of isothetic covers of a digital object: A combinatorial approach. *J. Vis. Comun. Image Represent.*, 21(4):295–310, May 2010.

[10] A. Biswas, P. Bhowmick, M. Sarkar, and B. B. Bhattacharya. A linear-time combinatorial algorithm to find the orthogonal hull of an object on the digital plane. *Information Sciences*, 216:176–195, 2012.

[11] O. Bodini and E. Rémila. Tilings with trichromatic colored-edges triangles. *Theoretical Computer Science*, 319(1):59–70, 2004.

[12] S. Butler, F. Chung, R. Graham, and M. Laczkovich. Tiling polygons with lattice triangles. *Discrete & Computational Geometry*, 44(4):896–903, 2010.

[13] J. Chang and C. Yap. A polynomial solution for the potato-peeling problem. *Discrete Computational Geometry*, 1:155–182, 1986.

[14] B. Chazelle. Approximation and decomposition of shapes. In J. T. Schwartz and C. K. Yap, editors, *Proceedings of the*, pages 145–186, Hillsdale, NJ, 1987. Lawrence Erlbaum Associates.

[15] B. Chazelle, R. L. Drysdale, and D. T. Lee. Computing the largest empty rectangle. In *STACS-1984*, pages 43–54. Springer, 1984.

[16] B. Chazelle, R. D. III, and D. Lee. Computing the largest empty rectangle. *SIAM J. Comput.*, 15:300–315, 1986.

[17] W. P. Chin and S. Ntafos. The zookeeper route problem. *Information Sciences*, 63(3):245–259, 1992.

[18] K. L. Clarkson, S. Kapoor, and P. Vaidya. Rectilinear shortest paths through polygonal obstacles in $o(n(\log n)^2)$ time. In *Proceedings of the 3rd Annual Symposium on Computational Geometry*, SCG '87, pages 251–257, New York, NY, USA, 1987. ACM.

[19] R. G. Clason. Tiling with golden triangles and the penrose rhombs using logo. *Journal of Computers in Mathematics and Science Teaching*, 9(2):41–53, 1989.

[20] J. H. Conway and J. C. Lagarias. Tiling with polyominoes and combinatorial group theory. *Journal of Combinatorial Theory, Series A*, 53(2):183–208, 1990.

[21] H. Daniel, K. Tom, and L. Elmar. Exploring simple triangular and hexagonal grid polygons online. *arXiv preprint arXiv:1012.5253*, 2010.

[22] K. Daniels, V. Milenkovic, and D. Roth. Finding the largest area axis-parallel rectangle in a polygon. *Computational Geometry: Theory and Applications*, 7:125–148, 1997.

[23] M. de Berg. On rectilinear link distance. *Computational Geometry: Theory and Applications*, 1(1):13–34, 1991.

[24] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry Algorithms and Applications*. Springer-Verlag, Heidelberg, 2008.

[25] M. de Berg, M. van Kreveld, B. J. Nilsson, and M. H. Overmars. Finding shortest paths in the presence of orthogonal obstacles using a combined l_1 and link metric. In *Proceedings of the 2nd Scandinavian Workshop on Algorithm Theory*, volume 447 of *LNCS*, pages 213–224, Bergen, Norway, 1990. Springer-Verlag.

[26] M.-P. Dubuisson-Jolly and A. Gupta. Tracking deformable templates using a shortest path algorithm. *Computer Vision and Image Understanding*, 81(1):26–45, 2001.

[27] A. Dumitrescu and J. S. B. Mitchell. Approximation algorithms for tsp with neighborhoods in the plane. *J. Algorithms*, 48(1):135–159, 2003.

[28] M. Dutt, A. Biswas, P. Bhowmick, and B. B. Bhattacharya. On finding shortest isothetic path inside a digital object. In R. P. Barneva, editor, *Proceedings of the 15th International Workshop on Combinatorial Image Analysis: IWCIA'12*, volume 7655 of *LNCS*, pages 1–15, Austin, Texas, November 2012. Springer-Verlag.

[29] L. Ferrari, P. V. Sankar, and J. Sklansky. Minimal rectangular partitions of digitized blobs. *Computer Vision, Graphics, and Image Processing*, 28(1):58–71, 1984.

[30] H. Freeman. Algorithm for generating a digital straight line on a triangular grid. *IEEE Transactions on Computers*, 100(2):150–152, 1979.

[31] M. Gardner. *Knotted Doughnuts and Other Mathematical Entertainments.* Freeman and Company, New York, 1986.

[32] S. K. Ghosh. *Visibility Algorithms in the Plane.* Cambridge University Press, New York, USA, 2007.

[33] C. Goodman-Strauss. Regular production systems and triangle tilings. *Theoretical Computer Science*, 410(16):1534–1549, 2009.

[34] J. Gudmundsson and C. Levcopoulos. A fast approximation algorithm for tsp with neighborhoods and red-blues separation. In *Proceedings of the 5th Annual International Conference on Computing and Combinatorics*, COCOON '99, pages 473–482, Berlin, Heidelberg, 1999. Springer-Verlag.

[35] H. Innchyn. Geometric transformations on the hexagonal grid. *IEEE Transactions on Image Processing*, 4(9):1213–1222, 1995.

[36] J. M. Keil and J. R. Sack. Minimum decompositions of polygonal objects. In J. T. Toussaint, editor, *Computational Geometry*, pages 197–216. North-Holland Publishing, Amsterdam, 1985.

[37] R. Klette and A. Rosenfeld. *Digital Geometry: Geometric Methods for Digital Picture Analysis.* Morgan Kaufmann, San Francisco, 2004.

[38] R. Klette and A. Rosenfeld. *Digital Geometry: Geometric Methods for Picture Analysis.* Morgan Kaufmann, San Francisco, CA, 2004.

[39] M. Laczkovich. Tilings of convex polygons with congruent triangles. *Discrete & Computational Geometry*, 48(2):330–372, 2012.

[40] R. C. Larson and V. O. Li. Finding minimum rectilinear distance paths in the presence of barriers. *Networks*, 11:285–304, 1981.

[41] F. Li and R. Klette. Finding the shortest path between two points in a simple polygon by applying a rubberband algorithm. In *Proceedings of the First Pacific Rim conference on Advances in Image and Video Technology*, volume 4319 of *PSIVT'06*, pages 280–291, Berlin, Heidelberg, 2006. Springer-Verlag.

[42] T. Lozano-Perez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Magazine Communications of the ACM*, 22(10):560–570, 1979.

[43] E. Luczak and A. Rosenfeld. Distance on a hexagonal grid. *IEEE Transactions on Computers*, 25(5):532–533, 1976.

[44] M. McKenna, J. O'Rourke, and S. Suri. Finding the largest rectangle in an orthogonal polygon. In *Proc. 23rd Allerton Conference on Communication, Control and Computing*, pages 486–495, 1985.

[45] B. Nagy. *Neighbourhood sequences in different grids.* PhD thesis, University of Debrecen, 2003.

[46] B. Nagy. Shortest paths in triangular grids with neighbourhood sequences. *Journal of Computing and Information Technology*, 11(2):111–122, 2003.

[47] B. Nagy. Characterization of digital circles in triangular grid. *Pattern Recognition Letters*, 25(11):1231–1242, 2004.

[48] B. Nagy. Generalised triangular grids in digital geometry. *Acta Mathematica Academiae Paedagogicae Nyíregyháziensis*, 20(1):63–78, 2004.

[49] B. Nagy. Distances with neighbourhood sequences in cubic and triangular grids. *Pattern Recognition Letters*, 28(1):99–109, 2007.

[50] B. Nagy. Cellular topology on the triangular grid. In *Combinatorial Image Analaysis*, pages 143–153. 2012.

[51] B. Nagy and K. Barczi. Isoperimetrically optimal polygons in the triangular grid with Jordan-type neighbourhood on the boundary. *International Journal of Computer Mathematics*, 90(8):1–24, 2012.

[52] S. C. Nandy, B. B. Bhattacharya, and S. Ray. Efficient algorithms for identifying all maximal isothetic empty rectangles in vlsi layout design. In *Proc. FST and TCS 10, Lecture Notes in Computer Science*, pages 255–269. Springer, 1990.

[53] S. Ntafos. Watchman routes under limited visibility. *Computational Geometry: Theory and Applications*, 1:149–170, 1992.

[54] J. O'Rourke. *Art Gallery Theorems and Applications.* Oxford University Press, New York, 1987.

[55] M. Sharir and A. Schorr. On shortest paths in polyhedral spaces. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing*, volume 15 of *STOC '84*, pages 193–215, New York, NY, USA, 1986. ACM.

[56] T. C. Shermer. Recent results in art gallery. *Proceedings of IEEE*, 80(9):1384–1399, 1992.

[57] K. Shimizu. Algorithm for generating a digital circle on a triangular grid. *Computer Graphics and Image Processing*, 15(4):401–402, 1981.

[58] K. G. Subramanian and P. Wiederhold. Generative models for pictures tiled by triangles. *Science and Technology*, 15(3):246–265, 2012.

[59] B. Sury. Group theory and tiling problems. *Symmetry: A Multi-Disciplinary Perspective*, 16(16):97–117, 2011.

[60] X. Tan. Approximation algorithms for the watchman route and zookeeper's problems. *Discrete Applied Mathematics*, 136(2–3):363–376, 2004.

[61] X. Tan and T. Hirata. Finding shortest safari routes in simple polygons. *Information Processing Letters*, 87(4):179–186, 2003.

[62] G. T. Toussaint. Pattern recognition and geometrical complexity. In *Proceedings of the 5th International Conference on Pattern Recognition*, ICPR '80, pages 1324–1347, 1980.

[63] J. Ullman. *"Ch.9: Algorithms for VLSI Design Tools". Computational Aspects of VLSI.* Computer Science Press., 1984.

[64] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1):168–173, 1974.

[65] C. A. Wüthrich and P. Stucki. An algorithmic comparison between square-and hexagonal-based grids. *CVGIP: Graphical Models and Image Processing*, 53(4):324–339, 1991.