Finding Shortest Triangular Path in a Digital Object

Apurba Sarkar[1(✉)], Arindam Biswas[2], Shouvick Mondal[1], and Mousumi Dutt[3]

[1] Department of Computer Science and Technology, Indian Institute of Engineering Science and Technology, Shibpur, India
as.besu@gmail.com, shouvick.mondal.cemk@gmail.com
[2] Department of Information Technology, Indian Institute of Engineering Science and Technology, Shibpur, India
barindam@gmail.com
[3] Department of Computer Science and Engineering, International Institute of Information Technology, Naya Raipur, India
duttmousumi@gmail.com

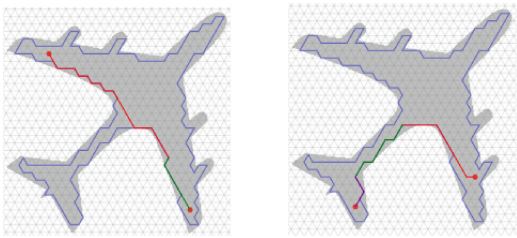
Abstract. A combinatorial algorithm to find a shortest triangular path (STP) between two points inside a digital object imposed on triangular grid is designed having $O(\frac{n}{g} \log \frac{n}{g})$ time complexity, n being the number of pixels on the contour of the object and g being the grid size. First the inner triangular cover of the given digital object is constructed which maximally inscribes the object. Certain combinatorial rules are formulated based on the properties of triangular grid and are applied whenever necessary to obtain a shortest triangular path, where the path lies entirely in the digital object and moves only along the grid edges. The length of STP and number of monotonicity may be two useful parameters to determine shape complexity of the object. Experimental results show the effectiveness of the algorithm.

Keywords: Shortest path · Shortest triangular path · Monotone path · Shape analysis · Shape complexity · Digital geometry

1 Introduction

The shortest path problem is a well-studied problem in graphs (directed and undirected). It enquires the shortest path between two vertices in a graph such that the sum of the weights of its constituent edges is minimized. The weights of the edges may vary depending on the problem being studied. Shortest path problem is a broadly useful problem solving model in robot navigation, texture mapping, typesetting in TEX, urban traffic planning, optimal pipe-lining of VLSI chip, subroutine in advanced algorithms, telemarketer operator scheduling, routing of telecommunications messages, approximating piecewise linear functions, network routing protocols, and optimal truck routing through given traffic congestion pattern [1]. The complete history of shortest path problem can

N. Normand et al. (Eds.): DGCI 2016, LNCS 9647, pp. 206–218, 2016.
DOI: 10.1007/978-3-319-32360-2_16



(a) (b)

Fig. 1. A digital object with inner cover (blue), and shortest paths (a) in (red, green) and (b) in (red, green, purple) (Color figure online)

be found in [13]. The first reported algorithm on shortest path is by Shimbel in 1954 [14], where he had reported few observations on calculating distance and proposed a method which later became known as the Bellman-Ford method.

In 1958, Bellman proposed a dynamic programming based approach for solving shortest path problem in [3] which runs in $O(n^3)$ time, where n is the number of vertices in the graph. In 1957, Moore proposed another algorithm on shortest paths [11]. In 1959, Dijkstra presented a simpler algorithm which runs in $O(n^2)$ time in [5]. Most of the works on shortest paths reported so far mainly find paths between two vertices in a graph. However, we are proposing a combinatorial algorithm on shortest paths which finds a path between two points inside a digital object in triangular grid. M. Dutt et al. [6,7] proposed a combinatorial algorithm to find a shortest isothetic path between two grid points inside a digital object without any holes. In [12], B. Nagy analysed some properties of hexagonal and triangular grid (considering cell model), where distance based neighborhood sequence is defined and an algorithm to calculate shortest distance between two arbitrary points is proposed. In [2], chain-code representation in triangular grid is discussed. There exists another work on finding shortest distance on the hexagonal grid [10], based on the distance function and the neighboring relations. A shortest path in triangular grid is a series of digital straight lines [8].

This paper focuses on finding a shortest path between two grid points inside a digital object imposed on a background triangular grid. First, the inner triangular cover of the given object is computed using the algorithm presented in [4]. This is done to ensure that the computed path does not go outside the inner cover and hence outside the object. An appropriate parallelogram is considered keeping the points at diagonally opposite corners. Combinatorial rules are applied on the intersection points generated on intersection of the inner cover with the parallelogram. Two triangular shortest paths are shown in Fig. 1(a) and (b).

The rest of the paper is organized as follows. All the required definitions and preliminaries are presented in Sect. 2. The method to obtain the shortest path is elaborated in Sect. 3. Estimation of running time of the proposed algorithm is explained in Sect. 4. Section 5 presents the experimental results with analysis and the conclusion is presented in Sect. 6.

2 Definitions and Preliminaries

A *digital object* (henceforth referred as an object A) is a finite subset of $\mathbb{Z}^2$, which consists of one or more 8-connected components. In this paper, a connected hole-free object is considered. A *triangular grid* (henceforth simply referred as grid) $\mathbb{T} := (\mathbb{L}_{60}, \mathbb{L}_0, \mathbb{L}_{120})$ consists of three sets of parallel grid lines, which are inclined at 60°, 0°, and 120° (w.l.o.g) w.r.t. x-axis [9].The grid lines in $\mathbb{L}_{60}$, $\mathbb{L}_0$, $\mathbb{L}_{120}$ correspond to three distinct coordinates, namely α, β, γ. Three grid lines, one each from $\mathbb{L}_{60}$, $\mathbb{L}_0$, $\mathbb{L}_{120}$, 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 $\mathbb{L}_{60}$, $\mathbb{L}_0$, and $\mathbb{L}_{120}$, is called *unit grid triangle* (UGT). For a given grid point, p, there are six neighboring UGTs, given by $\{T_i : i = 0, 1, \ldots, 5\}$ as shown in Fig. 2. A portion of the triangular grid is shown in Fig. 2 along with direction codes. 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 $\alpha_+ \cap \beta_+$, Sextant 2 is defined by the region $\alpha_- \cap \gamma_-$, and so on.

The *triangular distance* (d_t) between two points $p(\alpha_p, \beta_p, \gamma_p)$ and $q(\alpha_q, \beta_q, \gamma_q)$ is defined by $d_t(p, q) = max(|\alpha_p\ \alpha_q|, |\beta_p\ \beta_q|, |\gamma_p\ \gamma_q|)$.

The 6-neighborhood of a point (α, β, γ) is given by $N_6(\alpha, \beta, \gamma) = \{(\alpha', \beta', \gamma') : max(|\alpha - \alpha'|, |\beta - \beta'|, |\gamma - \gamma'|) = 1\}$.

A (finite) polygon P imposed on the grid $\mathbb{T}$ is termed as a *triangular polygon* if its sides are collinear with lines in $\mathbb{L}_{60}$, $\mathbb{L}_0$, 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. Its interior is defined as the set of points with integer coordinates lying inside it. An *inner triangular polygon* (or simply *inner polygon*) tightly inscribes

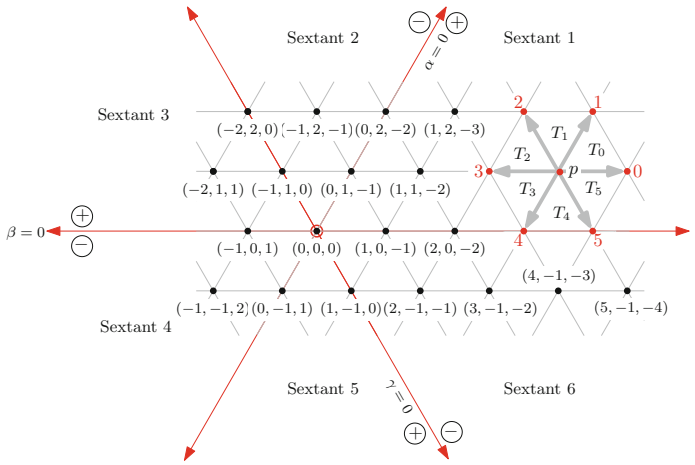


Fig. 2. Portion of a triangular canvas, the UGTs $\{T_0, T_1, \ldots, T_5\}$ incident at a grid point p, and the direction codes $\{0, 1, \ldots, 5\}$ of neighboring grid points of p.

A such that its border is a subset of A and the number of its constituting UGTs is maximum. An *inner triangular hole polygon* (or *inner hole polygon*) tightly circumscribes a hole and its border is a subset of A. 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 of the interiors of the inner hole polygons, contains a UGT if and only if it is a subset of A. In this paper, ITC containing one inner polygon is considered.

A (simple) *triangular path* π from a grid point p to a grid point q is a sequence of n distinct points $p_1, p_2, \ldots, p_n$ with $p_1 = p$ and $p_n = q$ such that $p_i \in N_6(p_{i+1})$, for $1 < i < n$. The length of a given triangular path is the sum of distances traversed along each axis. A triangular path π is said to be shortest if it is of minimum length. A path π in triangular grid is *monotone* if it consists of only one direction or two consecutive directions. Otherwise π is said to be non-monotone. In Fig. 1(a), the triangular path contains two monotone sub-paths whereas in Fig. 1(b), the triangular path contains three monotone sub-paths.

Deriving the Inner Triangular Cover (ITC): The inner triangular cover of A, $\underline{P}$, is constructed using the same method as outer triangular cover as explained in [4], but in the reverse manner. A grid point q is classified as a vertex of the inner cover, if and only if at least one (and at most five) of the six UGTs incident at q is fully occupied by the object A i.e., $T_i^q \cap A = T_i^q$ where $i \in \{0, 1, 2, 3, 4, 5\}$. The *object occupancy vector*, $A_q = \langle a_0 a_1 \ldots a_5 \rangle$, where $a_i = 1$ if T_i^q is fully occupied else it is 0, is used to determine the type of the vertex. Let k denote the number of fully occupied UGTs, then for $k = 0$: q is an exterior point of $\underline{P}$, $k = 6$: interior point, $k = 1$: a 60^0 vertex (included angle is 60^0), and $k = 5$: 300^0 vertex. For other cases, $k = 2, 3, 4$, **Type** of q is derived based on the incoming (d) and outgoing direction (d') at q. If the incoming direction is d, then $a_j = 1$ and $a_{(j+1) \mod 6} = 1$ where $j = (d + 2) \mod 6$. Now, j is incremented until the next 1-bit in A_q, say at j', $a_{j'} = 1$, then the outgoing direction, $d' = j'$. A **Type 3** vertex is considered as edge point. **Type 1, 2** vertices are considered as convex vertices and **Type 4, 5** vertices as concave vertices. The construction of $\underline{P}$ keeps A' (background) to the right during the traversal. The polygon is traced to the next grid point q_n, type of q_n is determined and the direction of traversal from q_n is computed. The traversal continues until the start vertex, v_s is reached. During the construction of $\underline{P}$, 4 lists are maintained L, L_α, L_β, and L_γ, where, L is a doubly linked list of vertices (corner points) of $\underline{P}$ and L_α, L_β, and L_γ simultaneously contain vertices as well as edge points of $\underline{P}$ in lexicographically sorted order with their respective primary and secondary keys. The primary key for L_α is α and secondary key is β, similarly the primary and secondary keys for L_β and L_γ can be defined. An index (in increasing order) is assigned to each vertex of $\underline{P}$ in order of their occurrence in $\underline{P}$.

3 Finding Shortest Path

To find a shortest path between two points p and q, an appropriate parallelogram, B, is constructed keeping p and q at diagonally opposite corners and then a traversal is made along one of the semi-perimeters. Throughout this work, the

object is assumed to lie in sextant 5 and 6, point p is assumed to be above q and the left semi-perimeter is traversed to find shortest path. Three different types of parallelogram are considered depending on the positions of point $q(\alpha_q, \beta_q, \gamma_q)$. Regions are separated by red lines as shown in Fig. 3 and are determined as follows, q is in *Region 1* if $\beta_q < \beta_p$ *and* $\alpha_q < \alpha_p$; q is in *Region 2* if $\alpha_q > \alpha_p$ *and* $\gamma_q > \gamma_p$ and finally q is in *Region 3* if $\beta_q < \beta_p$ *and* $\gamma_q < \gamma_p$. If q lies on the region separator, then there will be only one shortest path (as object does not have holes) and determining the shortest path is straight-forward. The reason behind the construction of parallelogram is that the semi-perimeters are the shortest distance between the two points (if semi-perimeters lie completely within the object). So, to construct a shortest path the traversal is made along one of the semi-perimeters. During this traversal if the semi-perimeter does not lie completely inside the object, intersection points between the semi-perimeter and the inner cover of the object are determined. The traversal is then guided through those intersection points possibly applying the reduction rules to shorten the path in such a way that the path lies inside the object. It is to be noted that not all intersection points will be important to guide the traversal and are eliminated using few combinatorial rules as explained in Sect. 3.1. The traversal continues this way applying the reduction rules whenever necessary until it reaches q. The reduction rules are explained in Sect. 3.2.

3.1 Finding Intersection Points

W.l.o.g, let p be always above q in the bounding parallelogram B and c_1, q, c_2 are the vertices in order, to the left of p in anti-clockwise direction. Then the left semi-perimeter of B is defined by $\overline{pc_1}$ and $\overline{c_1q}$ (Fig. 3). The procedure CONTROL-POINTS in (Algorithm 1) finds out the points which guide the traversal via the semi-perimeter. If the semi-perimeter $\overline{pc_1}$,$\overline{c_1q}$ lies entirely within the object, then the semi-perimeter itself will be a shortest path. Otherwise, the intersection points of $\overline{pc_1}$ with the inner cover of the object are found out by searching $L_x, x \in \{\alpha, \beta, \gamma\}$ depending on the orientation of $\overline{pc_1}$ and stored in M_1 and those of $\overline{c_1q}$ stored in M_2. If an intersection point w_i lies on the edge v_jv_{j+1} of $\underline{P}$, then its index is set to $j + 0.5$ to maintain the order that it appears after v_j. The lists are further examined and some of the points are removed as they will not be important to find the shortest path. The points in

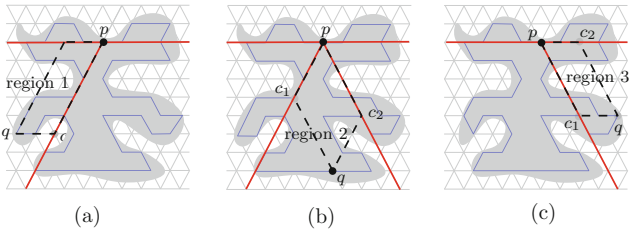
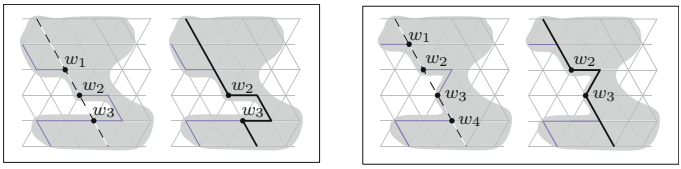


Fig. 3. Three regions and corresponding orientations of bounding parallelogram (a, b, c) (Color figure online)



(a) (b)

Fig. 4. Illustration of removal of unimportant intersection points. Dashed line represents one side of the parallelogram. (a) A concave vertex, w_1, is followed by a convex vertex w_2, w_1 is removed because it has lower index. (b) w_4, a concave vertex with higher index, is removed as it is preceded by a convex vertex, w_3.

two appropriate lists among $M_\alpha, M_\beta, M_\gamma$ are considered in pairs and if a convex (concave) vertex is followed by a concave (convex) vertex, then the vertex with greater (lower) index is discarded using REMOVE-POINTS (Steps 1 and 2) (as shown in Algorithm 1). In Steps 3–9, the final list of intersection points M, is formed by concatenating p, M_1, c_1 (if it is inside A), M_2 and q. In steps 16–22, the indices of the pairs of intersection points are checked to find whether they are in increasing or decreasing order and whether the index falls within indices of the extreme two points in M, namely $M[1]$ and $M[k+k']$, where k is the total number of intersection points. The value of k' indicates whether c_1 has been included in $M(k' = 1$(Step 5)) or not ($k' = 0$ (Step 8)) (Fig. 4).

If the test in Step 16 succeeds, then three consecutive intersection points in M are removed when c_1 is the second next point from $M[i]$ (Step 18); else next two consecutive points are removed (Step 20). Finally M is the required list. Let $M = \langle p, w_1, w_2, \ldots, w_k, q\rangle$. When the left semi-perimeter of B lies inside $\underline{P}$, (semi-perimeter of)B is traversed; otherwise, $\underline{P}$. So pw_1 is traversed along B, then w_1w_2 along $\underline{P}$, next w_2w_3 along B, again w_3w_4 along $\underline{P}$ and so on. Such an alternate traversal is made possible by reordering the vertices in Steps 16–22 if the index ordering does not hold.

Algorithm 1. CONTROL-POINTS

```
   Input: M_1, M_2, p, q, c_1
   Output: M
 1 REMOVE-POINTS(M_1);
 2 REMOVE-POINTS(M_2);
 3 if c_1 ∈ A then
       /* c_1 is the corner point on left
          semi-perimeter
 4     M ← CONCAT(p, M_1, c_1, M_2, q);
 5     k' ← 1;
 6 else
 7     M ← CONCAT(p, M_1, M_2, q);
 8     k' ← 0;
 9 end
10 i ← 1;
11 while M[i] ≠ q do
12     if M[i] = c_1 then
13         i ← i + 1;
14         continue
15     else
16         while ∼ (((index[M[i]] <
   */        index[M[i + 1]]) ∧ (index[M[i + 1]] ⩽
           index[M[k + k']])) ∨ ((index[M[i]] >
           index[M[i + 1]]) ∧ (index[M[i + 1]] ⩾
           index[M[k + k']]))) do
17             if M[i + 2] = c_1 then
18                 DELETE(M[i + 1], M[i +
                   2], M[i + 3])
19             else
20                 DELETE(M[i + 1], M[i + 2])
21             end
22         end
23     end
24     i ← i + 2;
25 end
26 return M
```

3.2 Reduction Rules

While traversing from p via the semi-perimeter of B, the intersection points in M are also encountered to reach q. This path may include convexities which are to be removed to shorten the path. The rules are discussed here. The convexities are detected when the turn at a vertex or the sum of the turns at two consecutive vertices is equal to or greater than 120°. A clockwise (anticlockwise) change in direction at a vertex is considered as a positive (negative) turn by the corresponding angle. All possible cases are depicted in Fig. 5. A **Type 1** vertex makes a turn of 120° so it is a convex vertex. Similarly turn at two consecutive vertices of types **22**, **21** creates a turn of 120° or more and hence create convexities. Pattern **12** and **11** also create convexity.

It is to be noted that although a **Type 2** vertex is treated as a convex vertex, unlike **Type 1** vertex, it alone cannot create a convexity. The proposed algorithm maintains with each vertex, its **Type** *(t)*, length *(l)*, and the outgoing direction *(d)*. Removal of convexity sometimes requires removal of some or all vertices that are involved in the convexity and the deletion of vertex needs adjustment of those information with vertex that precedes or follows convexity. If the convexity is created by a **Type 1** vertex, four consecutive vertices, $v_0v_1v_2v_3$, where v_2 is the vertex of **Type 1** and v_3 is the most recently visited vertex, are considered to apply the rule to remove convexity. On the other hand, if the convexity is created by two consecutive convex vertices (of **Type 22** or **21**) then five consecutive vertices $v_0v_1v_2v_3v_4$, where v_1 and v_2 are convex vertices and v_4 is the most recently visited vertex; are considered to apply the rule to remove convexity. The type of the start (p) and end (q) vertices are set to **6** since in general the path is found between two points that lie inside the cover. The rules are explained as follows.

Pattern $< t_1\mathbf{1}t_3 > t_1, t_3 \in \{4, 5, 6\}$

This pattern implies a convex region created by a single **Type 1** vertex and it is preceded or followed by concave vertices. We consider four most recently traversed vertices, $v_0(t_0, l_0)v_1(t_1, l_1)v_2(t_2, l_2)v_3(t_3, l_3), v_3$ being the most recent. Depending on the lengths l_1 and l_2, three rules are as follows.
R11 $(l_1 < l_2)$**:** $\langle v_0(t_0, l_0)v_1(t_1, l_1)v_2(t_2, l_2)v_3(t_3, l_3)\rangle \rightarrow$
$\langle v_0(t_0, l_0)v_1(t_1 - 1, l_1)v_2(t_2 + 1, l_2 - l_1)v_3(t_3, l_3)\rangle$
R12 $(l_1 = l_2)$**:** $\langle v_0(t_0, l_0)v_1(t_1, l_1)v_2(t_2, l_2)v_3(t_3, l_3)\rangle \rightarrow$
$\langle v_0(t_0, l_0)v_1(t_1 - 1, l_1)v_3(t_3 - 1, l_3)\rangle$
R13$(l_1 > l_2)$**:** $\langle v_0(t_0, l_0)v_1(t_1, l_1)v_2(t_2, l_2)v_3(t_3, l_3)\rangle \rightarrow$

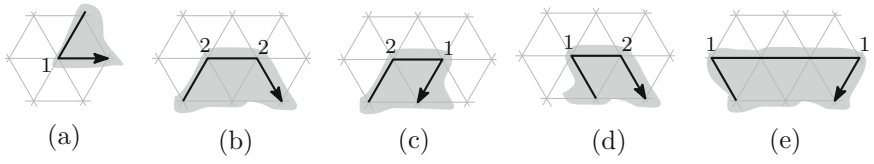


Fig. 5. Types of convexities present in $\underline{P}$

$\langle v_0(t_0, l_0)v_1(t_1, l_1 - l_2)v_2(t_2 + 1, l_2)v_3(t_3 - 1, l_3)\rangle$. After application of the rule whichever necessary, if type of any of the vertices is **3** then its length is added to the vertex that precedes it and the vertex is deleted. For example, if the type of vertex v_3 becomes **3** then l_3 is added to length of v_2 and v_3 is deleted. The illustration of rule **R1** is shown in Fig. 6. **Pattern** $< t_1t_2t_3t_4 >$ where $t_2, t_3 \in \{1, 2\}$ and $t_1, t_4 \in \{4, 5, 6\}$.

This pattern implies two consecutive convex vertices followed and preceded by concave vertices. There will be three possible cases depending on the length of v_2 and v_3 as explained in the following rules (illustrated in Fig. 7).

R21: $(l_1 < l_3)$ $\langle v_0(t_0, l_0)v_1(t_1, l_1)v_2(t_2, l_2)v_3(t_3, l_3)v_4(t_4, l_4)\rangle \rightarrow$
$\langle v_0(t_0, l_0)v_1(t_1 - 1, (t_2 + t_3 - 3)l_1 + l_2)v_3(t_3, l_3 - l_1)v_4(t_4, l_4)\rangle$
R22: $(l_1 = l_3)$ $\langle v_0(t_0, l_0)v_1(t_1, l_1)v_2(t_2, l_2)v_3(t_3, l_3)v_4(t_4, l_4)\rangle \rightarrow$
$\langle v_0(t_0, l_0)v_1(t_1 - 1, (t_2 + t_3 - 3)l_1 + l_2)v_4(t_4 - 1, l_4)\rangle$
R23: $(l_1 > l_3)$ $\langle v_0(t_0, l_0)v_1(t_1, l_1)v_2(t_2, l_2)v_3(t_3, l_3)v_4(t_4, l_4)\rangle \rightarrow$
$\langle v_0(t_0, l_0)v_1(t_1, l_1 - l_3)v_2(t_2, (t_2 + t_3 - 3)l_3 + l_2)v_4(t_4 - 1, l_4)\rangle$

Pattern $< t_1t_2t_3t_4 >$ where $t_2, t_3, t_4 \in \{1, 2\}$ and $t_1 \in \{4, 5, 6\}$.

This pattern implies a concave vertex (t_1) is followed by 3 consecutive convex t_2, t_3, t_4 vertices. The total turn at these three consecutive convex vertices may be more than 180° and if $l_1 > l_3$ the traversal may enter into a convoluted region which should be avoided to obtain shorter path. This is explained with the help of a sample case shown in Fig. 8. Consider the line h along v_2v_1 and the line h' in the direction of v_2v_3 projected at $v'v_4$ (h and h' meet at v'). To avoid the convoluted region, a traversal is made from v_4 and every time it reaches a new vertex, a check is made to determine whether it has entered the free region

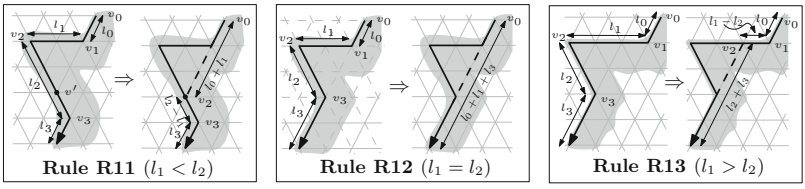


Fig. 6. Illustration of rules: (a) **R11**, (b) **R12**, (c) **R13**

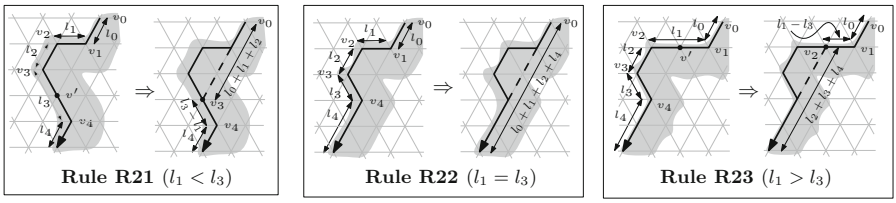


Fig. 7. Illustration of rules (a) **R21**, (b) **R22**, (c) **R23**

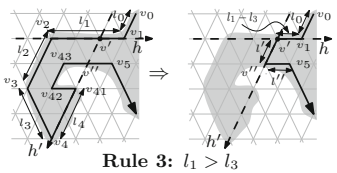


Fig. 8. Illustration of Rule 3

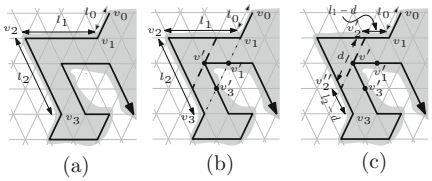


Fig. 9. Nearest concavity line

defined by v_1, v', v_4. For example when the traversal is at vertex v_{41} or v_{42} or v_{43}, it is on or to the right of the line h' and below (right of) the line h and hence they are within the convoluted region. When the traversal reaches v_5, it is on the left of h' and below h which is the free region. The vertices starting from v_2 to v_{43} are deleted, length l' from $v'v''$ and l'' from v'' to v_5 are determined and set accordingly. The type of v'' i.e. t'' is calculated from incoming and outgoing direction at v'' by the formula $(d_{in} - complement(d_{out}) + 6) \mod 6$. The rule is given below.

R3: $\langle v_0(t_0, l_0)v_1(t_1, l_1)v_2(t_2, l_2)v_3(t_3, l_3)v_4(t_4, l_4)\rangle \rightarrow$
$\langle v_0(t_0, l_0)v_1(t_1, l_1 - l_3)v'(t_2, l')v''(t'', l'')\rangle$

However, if $l_1 \leq l_3$, the convexities can be avoided using **Rule 2**. One or more concavity can be intruded inside a convex region. In a convex region, we have to find the concavity which is mostly intruded. This checking has to be performed while applying reduction rules. For example, the convexity created by vertices v_1, v_2, v_3, shown in Fig. 9(a) is of type $< t_1 1 t_3 >$ with $l_1 = l_2$. Within the convex region a concave portion is there. To keep the shortest path inside the object the path should pass along the concavity line which is mostly intruded in the convex region (dashed line via v' as shown in Fig. 9(b)). To locate the required concavity line, the intersection points with line v_1v_3 (dotted line in Fig. 9(b)) are found out by searching the appropriate list $(L_\alpha, L_\beta, L_\gamma)$. If there is no intersection point, then appropriate reduction rule is applied directly. On the other hand, if there are intersection points then a traversal is made in the portion of $\underline{P}$ starting from one intersection point to the next intersection point to find out the nearest concavity line. For example, in Fig. 9(b) line v_1v_3 intersects $\underline{P}$ at v'_1 and v'_3. A traversal is made from v'_3 to v'_1, finding distance from every new vertex it meets to v_2 and choosing the one with minimum distance, v' in this case. So the reduction is made upto v' via the dashed line. This introduces two new vertices v'_2 and v''_2 and their length and types are adjusted as shown in Fig. 9(c). If the distance of v_2 from v' is d, then length of v_1 and v''_2 is set to $l_1 - d$ and $l_2 - d$ respectively. Length of v'_2 is set to d and v_2 is deleted.

3.3 Algorithms

The algorithm FIND-STP (Algorithm 2) takes the inner cover $\underline{P}$, the lists L_α, L_β, and L_γ, source and destination points, p and q, as input. The point of intersections of B with the semi-perimeter $\overline{pc_1}$, $\overline{c_1q}$ are obtained (Steps 3–4)

and non-essential points are removed using the procedure CONTROL-POINTS (Algorithm 1) (Step 5). p is appended to the shortest path, π. In the while loop (Steps 8–23), each point in M is considered until it reaches q. If $M[i]$ is the corner point c_1, then it is appended to π and then reduced (Steps 10–11). The procedure REDUCE uses the reduction rules to remove the convexity in π. In Steps 15–16, $M[i]$ is added and reduction rules are applied if needed. In Step 17, the portion of $\underline{P}$ between $M[i]$ a 19, $M[i+1]$ is added to π and reduced if needed.

In the procedure TRAVERSE in Algorithm 3, if the index of $M[i]$ is less than that of $M[i+1]$, then $\underline{P}$ is traversed in an anticlockwise manner (Steps 1–11); otherwise, $\underline{P}$ is traversed clockwise (Steps 12–22). In Steps 2–3, l' and l'' indicate the pointers to the neighbor vertices of $M[i]$ and $M[i+1]$ in $\underline{P}$, taken appropriately. After adding $\underline{P}[l']$ to path π (Step 4 or 15), each vertex on the path is appended to π in the while loop (Steps 7–11 or 18-22) until the vertex $\underline{P}[l'']$ is reached. Appropriate reduction rules are applied by calling REDUCE in Steps 5 and 16, 9 and 20 as and when necessary. Procedure REDUCE is explained in Sect. 3.2 with reduction rules, and procedure SEARCH is used to search intersection points of the boundary of $\underline{P}$ with the semi-perimeter of the bounding parallelogram.

Algorithm 2. FIND-STP

Input: $\underline{P}, L_\alpha, L_\beta, L_\gamma, p, q$
Output: π
1 $c_1 \leftarrow$ corner point on left semi-perimeter;
2 $\theta_1, \theta_2 \leftarrow$ Orientation of segment $\overline{pc_1}$, $\overline{c_1q}$;
3 $M_1 \leftarrow$ SEARCH(p, c_1, θ_1);
4 $M_2 \leftarrow$ SEARCH(c_1, q, θ_2);
5 $M \leftarrow$ CONTROL-POINTS(M_1, M_2, p, q, c_1);
6 $i \leftarrow 1$, $\pi \leftarrow \phi$;
7 APPEND(π, p);
8 **while** $M[i] \neq q$ **do**
9 **if** $M[i] = c_1$ **then**
10 APPEND(π, c_1);
11 REDUCE(π);
12 $i \leftarrow i + 1$;
13 **continue**
14 **end**
15 APPEND$(\pi, M[i])$;
16 REDUCE(π);
17 TRAVERSE$(\underline{P}, M[i], M[i+1], \pi)$;
18 APPEND$(\pi, M[i+1])$;
19 REDUCE(π);
20 $i \leftarrow i + 2$;
21 APPEND(π, q);
22 REDUCE(π);
23 **end**
24 **return** π

Algorithm 3. TRAVERSE

Input: $(\underline{P}, M[i], M[i+1], \pi)$
1 **if** $index[M[i]] < index[M[i+1]]$ **then**
2 $l' \leftarrow \lfloor (index[M[i]] + 1) \rfloor$;
3 $l'' \leftarrow \lceil (index[M[i+1]] - 1) \rceil$;
4 APPEND$(\pi[m], \underline{P}[l'])$;
5 REDUCE(π);
6 $j \leftarrow l' + 1$;
7 **while** $j \leqslant l''$ **do**
8 APPEND$(\pi[m], \underline{P}[j])$;
9 REDUCE(π);
10 $j \leftarrow j + 1$;
11 **end**
12 **else**
13 $l' \leftarrow \lceil (index[M[i]] - 1) \rceil$;
14 $l'' \leftarrow \lfloor (index[M[i+1]] + 1) \rfloor$;
15 APPEND$(\pi[m], \underline{P}[l'])$;
16 REDUCE(π);
17 $j \leftarrow l' - 1$;
18 **while** $j \geqslant l''$ **do**
19 APPEND$(\pi[m], \underline{P}[j])$;
20 REDUCE(π);
21 $j \leftarrow j - 1$;
22 **end**
23 **end**

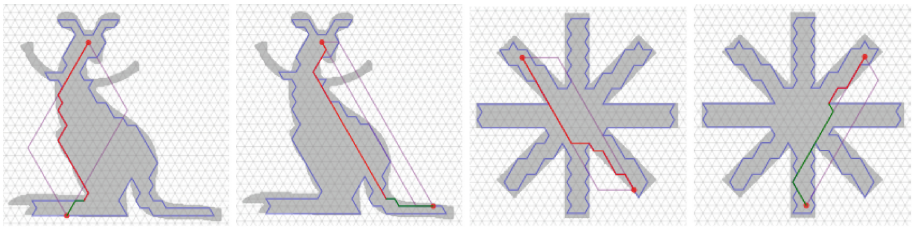
4 Time Complexity

To compute the running time of the proposed algorithm let us look at the steps involved and the cost of each step. Initially the inner cover of the object is

computed by the algorithm presented in [4] which costs $O(n/g)$ time, n being the number pixels in the perimeter of the object and g is the grid size. During the construction of inner cover three sorted lists L_α, L_β and L_γ are also constructed in $O(n/g \log n/g)$ time. The intersection points on the inner cover of the object with the semi-perimeter of the parallelogram are found by searching L_α or L_β or L_γ in $O(\log n/g)$ time. The algorithm to find shortest path uses control points to reach the destination and applies reduction rules whenever necessary. Reductions can be performed in $O(1)$ time. So, the overall running time of the algorithm amounts to $O(n/g) + O(n/g \log n/g) + O(\log n/g) + O(1) \simeq O(n/g \log n/g)$.

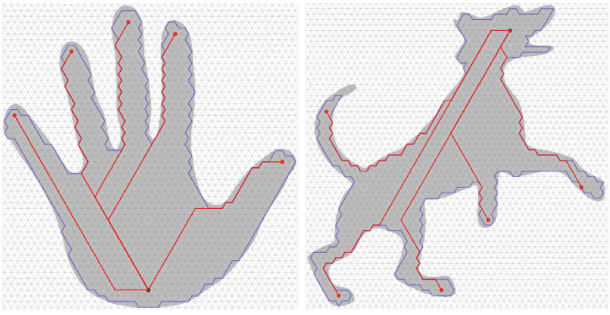
5 Experimental Results and Analysis

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 and tested exhaustively to show the efficacy and correctness of the algorithm. Two instances of shortest paths for two different objects along with the bounding parallelogram (purple) through which the shortest path is calculated are shown with $g = 8$ in Fig. 10. The number of monotone sub-paths (m) with different



(a) $m = 2$ (b) $m = 2$ (c) $m = 1$ (d) $m = 2$

Fig. 10. Shortest Path of three different objects with $g = 8$ and # monotone paths, m, {(a), (b)} Kangaroo, {(c), (d)} Device



(a) (b)

Fig. 11. Shortest Paths of two different objects with $g = 8$ from single source (black) to multiple destinations (red) (Color figure online)

colors are also shown under each object in the results. Figure 11 also shows shortest paths from a single source to multiple destinations for two different objects. It is evident from the results that the reported shortest path is not only the shortest path between the two points but also there exists a set of shortest paths having same path-length and our algorithm reports one of the paths between the two points.

6 Conclusions

A combinatorial algorithm to find a shortest triangular path between two points inside a digital object is presented here, which is not unique. Our algorithm reports one of the shortest triangular paths. Thus in future, this work can be extended to determine all shortest paths between two points. The number of monotone triangular sub-paths depends on the position of the two points inside the digital object and also on the shape of the object. The number of monotone triangular sub-paths and other related properties, e.g., length of the path, distance between two points, can be used to determine shape complexity of the object. These metrics will also be useful for determining shape signatures.

References

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms and Applications. Prentice Hall, Upper Saddle River (1993)
2. Balint, G.T., Nagy, B.: Finiteness of chain-code picture languages on the triangular grid. In: Image and Signal Processing and Analysis (ISPA), pp. 310–315 (2015)
3. Bellman, R.: On a routing problem. Q. Appl. Math. **16**, 87–90 (1958)
4. Das, B., Dutt, M., Biswas, A., Bhowmick, P., Bhattacharya, B.B.: A combinatorial technique for construction of triangular covers of digital objects. In: Barneva, R.P., Brimkov, V.E., Šlapal, J. (eds.) IWCIA 2014. LNCS, vol. 8466, pp. 76–90. Springer, Heidelberg (2014)
5. Dijkstra, E.: A note on two problems in connexion with graphs. Numer. Math. **1**, 269–271 (1959)
6. Dutt, M., Biswas, A., Bhowmick, P., Bhattacharya, B.B.: On finding a shortest isothetic path and its monotonicity inside a digital object. Ann. Math. Artif. Intell. **75**, 27–51 (2015)
7. Dutt, M., Biswas, A., Bhowmick, P., Bhattacharya, B.B.: On finding shortest isothetic path inside a digital object. In: Barneva, R.P., Brimkov, V.E., Aggarwal, J.K. (eds.) IWCIA 2012. LNCS, vol. 7655, pp. 1–15. Springer, Heidelberg (2012)
8. Freeman, H.: Algorithm for generating a digital straight line on a triangular grid. IEEE Trans. Comput. **28**, 150–152 (1979)
9. Her, I.: Geometric transformation on the hexagonal grid. IEEE Trans. Image Process. **4**, 1213–1222 (1995)
10. Luczak, E., Rosenfeld, A.: Distance on a hexagonal grid. IEEE Trans. Comput. **25**(5), 532–533 (1976)
11. Moore, E.: The shortest path through a maze. In: Proceedings of an International Symposium on the Theory of Switching, 25 April 1957, pp. 285–292. Harvard University Press, Cambridge (1959)

12. Nagy, B.: Shortest paths in triangular grids with neighbourhood sequences. J. Comput. Inf. Technol. **11**(2), 111–122 (2003)
13. Schrijver, A.: On the history of the shortest path problem. Doc. Math. 155–167 (2012)
14. Shimbel, A.: Structural parameters of communication networks. Bull. Math. Biophys. **15**(4), 501–507 (1953)