



A linear-time combinatorial algorithm to find the orthogonal hull of an object on the digital plane[☆]

Arindam Biswas^{a,*}, Partha Bhowmick^b, Moumita Sarkar^a, Bhargab B. Bhattacharya^c

^a Department of Information Technology, Bengal Engineering and Science University, Howrah 711 103, India

^b Computer Science and Engineering Department, Indian Institute of Technology, Kharagpur 721 302, India

^c Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata 700 108, India

ARTICLE INFO

Article history:

Received 5 July 2008

Received in revised form 11 February 2012

Accepted 31 May 2012

Available online 20 June 2012

Keywords:

Combinatorial algorithm

Convex hull

Digital geometry

Image processing

Multiresolution analysis

Shape analysis

ABSTRACT

A combinatorial algorithm to compute the orthogonal hull of a digital object imposed on a background grid is presented in this paper. The resolution and complexity of the orthogonal hull can be controlled by varying the grid size, which may be used for a multiresolution analysis of a given object. Existing algorithms on finding the convex hull are based on divide and conquer strategy, sweepline approach, etc., whereas the proposed algorithm is combinatorial in nature whose time complexity is linear on the object perimeter instead of the object area. For a larger grid size, the perimeter of an object decreases in length in terms of grid units, and hence the runtime of the algorithm reduces significantly. The algorithm uses only comparison and addition in the integer domain, thereby making it amenable to usage in real-world applications where speed is a prime factor. Experimental results including the CPU time demonstrate the elegance and efficacy of the proposed algorithm.

© 2012 Elsevier Inc. All rights reserved.

1. Introduction

The convex hull of an object A , denoted by $CH(A)$, is the smallest convex set that contains A . There exist several algorithms to find the convex hull of a point set or a polygonal object A having arbitrary shape on the real/digital plane [6,13,28]. The time complexities of some of the referred ones are $O(n^3)$ (brute force), $O(n \log n)$ (Graham scan [16]), $O(nh)$ (Jarvis march [18]), and $O(n \log h)$ (Kirkpatrick–Siedel's algorithm [22]), where n is the number of points/vertices constituting A , and h is the number of vertices of $CH(A)$. Some other algorithms are [5,12,32]. A detailed study of the convex hull algorithms may be seen in [3]. However, the execution of these algorithms for a sufficiently large digital object is not as fast as required in a practical application. This is caused by the inherent procedural complexities in these algorithms. For example, in Graham scan, to decide whether there is a left-turn or a right-turn or no turn at a point p_i , considering its previous point p_{i-1} and its next point p_{i+1} , the sign of a 3×3 determinant (which includes the coordinates of the three consecutive points) is used. Computation of this determinant needs multiplication apart from comparison and addition/subtraction. Similarly, in Jarvis march, polar coordinates of the points are computed from their Cartesian coordinates, which involves trigonometric operations. A generic approach to obtain orthogonal convex hull has been discussed in [20], which takes $O(n \log \log u)$ time, where n is the number of input points and u the grid size; it is based on 'maximal layers' formed by scanning points from left

[☆] A preliminary version of the paper appeared in [8].

* Corresponding author. Tel.: +91 33 2668 6151; fax: +91 33 2668 2198.

E-mail address: abiswas@it.becs.ac.in (A. Biswas).

to right. On the contrary, our algorithm is mainly combinatorial in nature and takes $O(n)$ time as it is based on traversal of the object boundary in the digital/integer plane.

Given a 2D digital object S imposed on the background grid \mathcal{G} consisting of a set of equi-spaced horizontal and vertical lines, the problem of computing the orthogonal hull (Definition 4) of S is addressed in this paper. It is evident that the resulting orthogonal hull is dependent upon the registration of S with \mathcal{G} . Also, the precision and the complexity of the hull can be made to change by varying the grid spacing, thereby making it amenable to multi-grid treatment. Thus, the specification of an orthogonal hull covering a digital object may be tuned, depending on the requirement of an application.

Several interesting notions of convexity have been defined in the literature for their usage in various applications. Some of these are orthogonal convexity, finitely oriented convexity, restricted oriented convexity, convex fuzzy sets, etc. [15,25,27,33,36]. Projection onto convex sets have been used for restoration of a continuous-tone image from a given half-tone image using space and frequency domains [34], for error concealment in JPEG 2000-coded images [1], and for artifact reduction in compressed images based on region homogeneity constraints [35]. In [19], a method of deblurring has been presented by merging differently blurred images in the spectrum domain using fuzzy projection onto convex sets.

Applications of orthogonal hulls have, in fact, diversified recently to a greater extent specially in the fields of modern computing and digital imaging. Some of the applications have already been mentioned above. Another important application lies in designing a fault-tolerant algorithm in mesh-connected computers, where it is important to define a faulty region that is convex, and at the same time, to include a minimum number of non-faulty nodes. Recently, an algorithm has been presented by Wu and Jiang [37] to compute the minimum orthogonal convex polygon, which includes the faulty blocks and minimum number of non-faulty nodes. The algorithm is based on a labeling scheme. It grows the region with faulty nodes and finally shrinks to form the orthogonal convex polygon. The algorithm is particularly suitable for a set of nodes representing the processors in an orthogonal grid. However, it is not suitable for image analysis, since it is designed to operate on a small number of nodes.

Some other typical applications involving orthogonal hulls are analysis of land-mark data, shape analysis and classification, measuring the polygonal entropy, and many such areas of computer vision and pattern recognition [9,14,17,26]. Orthogonal convex polygons also find use in models of polymers, cell growth, and percolation [10]. In discrete tomography, the reconstruction of discrete sets is done using the concept of $h\nu$ -convex discrete sets [4]. A properly defined convex polygon describing a real or a digital object is often considered to be the domain of interest of the underlying object. As a result, the subject has received a considerable attention amongst researchers [11,30,31,38].

In [21], an algorithm is proposed to compute the orthogonal (convex) hull for a set of points in \mathbb{Z}^2 . The survey in [2] elaborates several optimization issues while finding an empty convex polygon of maximum area or perimeter amidst a point set in \mathbb{R}^2 . Recently, an algorithm has been proposed in [24] to find the largest empty ortho-convex polygon in a (possibly sparse and scattered) point set in \mathbb{R}^2 . These algorithms use the scan-line strategy after doing a lexicographical sorting of the points in \mathbb{R}^2 . The proposed algorithm, on the contrary, finds the orthogonal hull for a given object in \mathbb{Z}^2 , which is defined as a set of one or more connected components. The algorithm avoids any sorting and finds the hull while traversing tightly around the object contour. The runtime of the proposed algorithm has been shown to be proportional to the perimeter of the object.

We have designed and tested a novel algorithm for finding the orthogonal hull (Definition 4) of a given digital object such that the hull edges lie on a set of equally spaced horizontal and vertical grid lines. The ordered list of hull vertices is obtained by an analysis of the object occupation of the four neighboring quadrants corresponding to a grid point (Definition 2) lying near the boundary of the object. The orthogonal hull consists of fewer vertices with an increase of the grid size (spacing between two consecutive horizontal/vertical grid lines), enabling a multiresolution analysis of the object. The algorithm is based on the fact that a polygon is orthogonally convex if and only if a counterclockwise traversal of its boundary never makes two consecutive right turns. (Alternatively, a clockwise traversal of its boundary never makes two consecutive left turns.) The algorithm involves only comparison and addition/subtraction in the integer domain, and hence runs very fast, as demonstrated by the CPU time in our experiments (Section 5).

The convex hull and the orthogonal hulls for $g = 22$ and $g = 8$, corresponding to a digital object (22,404 pixels), are shown in Fig. 1. The convex hull algorithm (Graham scan) on a digital object shown in this figure takes 2573 ms, whereas the

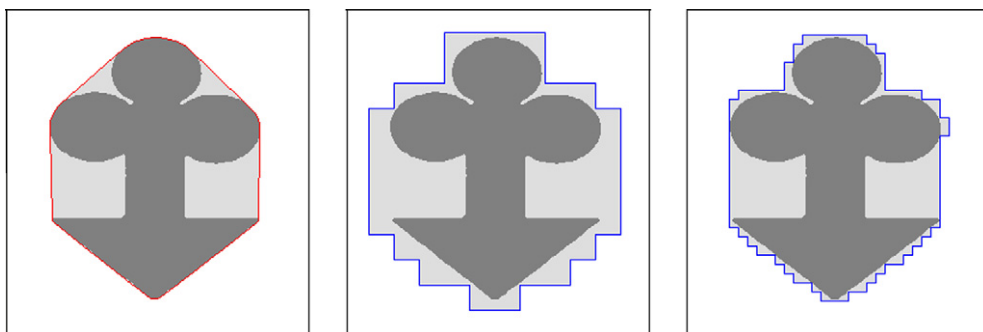


Fig. 1. A sample 2D object, its convex hull (left), and its orthogonal hulls for grid size $g = 22$ (middle) and $g = 8$ (right).

proposed algorithm on finding the orthogonal hull takes only a few milliseconds ($g = 22$: 0.94 ms, $g = 8$: 3.16 ms). The time required by the Graham scan algorithm may be reduced by considering the object contour instead of the entire object as input; but finding the object contour needs an edge extraction algorithm. The edge extraction can be done in linear time, for example using the border-tracing algorithm presented in [29]. On the contrary, given a digital object, the proposed algorithm runs on the object contour without resorting to any edge extraction. The proposed algorithm has also the ability to capture the shape information of an arbitrary object. For example, for $g = 22$, the orthogonal hull of the object shown in Fig. 1 is vertically symmetrical, which conforms to the vertical symmetry of the object; for $g = 8$, the orthogonal hull is also almost symmetrical. The vertices of the orthogonal hull are reported in counterclockwise order in terms of their types (90° and 270°), from which the symmetry can be ascertained. The non-convex regions—detected and removed while deriving the corresponding convex regions using certain semantic rules based on a combinatorial analysis—may be used to capture the shape complexity of the concerned object, which can be used in subsequent applications.

The rest of this paper is organized as follows. Section 2 contains the definitions and preliminaries, and briefs out how a tight orthogonal traversal around the contour of a digital object can be made. Section 3 explains the rules required to find the orthogonal hull from the orthogonal traversal explained in Section 2. Section 4 describes the algorithm to construct the orthogonal hull, presents a detailed demonstration, and justifies the time complexity of the algorithm. The experimental results and their physical analyses are reported in Section 5. Finally, the concluding notes along with some directions for future work, are presented in Section 6.

2. Definitions and preliminaries

Following is the list of symbols used in this paper.

\mathbb{Z}^2	Set of integer points
S	A digital object
\mathcal{G}	Background grid
\mathcal{H}	Set of equispaced horizontal grid-lines
\mathcal{V}	Set of equispaced vertical grid-lines
g	Grid size
P	An orthogonal polygon
$OH(S)$	Orthogonal convex hull of S
p	A point of object S
Q_i	Quadrant i
C_i	Class i of a grid point (on basis of object containment)
v_i	A vertex
t_i	Type of vertex v_i
d_i	Direction of traversal from v_i
l_i	Length of line segment from v_i to v_{i+1}
p_0	Top left point of S with coordinates (i_0, j_0)
v_s	Start vertex with coordinates (i_s, j_s)
R11, R12, R13	Rules for pattern 1331
R21, R22, R23	Rules for pattern 1333
L	List containing the vertices of OH
k	Index of the last vertex appended in L

Definition 1. A subset of \mathbb{Z}^2 in which every pair of points is k -connected,¹ is called a k -connected set. (In this paper, we define a digital object S to be an 8-connected subset of \mathbb{Z}^2 whose complement $\mathbb{Z}^2 \setminus S$ is a 4-connected set [29].)

Definition 2. The background grid is given by $\mathcal{G} = (\mathcal{H}, \mathcal{V})$, where \mathcal{H} and \mathcal{V} represent two sets of equi-spaced horizontal and vertical grid lines respectively. The grid size g is defined as the distance between two consecutive horizontal/vertical grid lines. A grid point is the point of intersection of a horizontal and a vertical grid line.

Definition 3. P is an orthogonal polygon if and only if each of its vertices is a grid point and each of its edges is axis-parallel.

¹ Two points p and q are said to be k -connected ($k = 4$ or 8) in a set S if and only if there exists a sequence $\langle p = p_0, p_1, \dots, p_n = q \rangle \subseteq S$ such that $p_i \in N_k(p_{i-1})$ for $1 \leq i \leq n$, where $N_k(p)$ denotes k -neighbors of p . The four neighborhood of a point $p(x, y) \in \mathbb{Z}^2$ is given by $N_4(p) = \{(x', y') : |x - x'| + |y - y'| = 1\}$ and its 8-neighborhood by $N_8(p) = \{(x', y') : |x - x'| + |y - y'| = 1\}$.

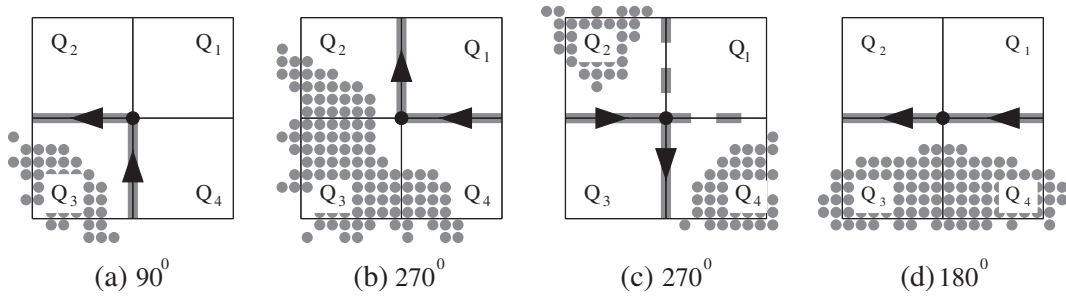


Fig. 2. Different vertex types of an orthogonal polygon are decided using the object (gray dots) occupation in the four cells quadrants incident at a grid point.

Definition 4. The *orthogonal convex hull*, or simply *orthogonal hull*, of a digital object S , denoted by $OH(S)$, is the smallest-area orthogonal polygon such that (i) each point $p \in S$ lies inside $OH(S)$ and (ii) intersection of $OH(S)$ with any horizontal or vertical line is either empty or exactly one line segment.²

In order to detect and remove the concavities, we traverse around the object contour, orthogonally along the grid lines. The nature of traversal is such that we visit the vertices (in order) of the smallest-area orthogonal cover of the digital object using an efficient combinatorial technique based on object containments of the four cells incident to a particular grid point [7]. The classification of a grid point, q , is based on the object containment of the four cells (Q_1 – Q_4) incident at q and their combinatorial arrangement, as shown in Fig. 2. If the number of object-containing cells incident at q is i , then q is classified to class C_i as follows.

C_0 : q is not a vertex, since none of the Q_i s has object containment.

C_1 : Exactly one of the Q_i s is intersected by the object. Hence q is classified as a 90° vertex, as shown in Fig. 2a.

C_2 : Two cells occupied by the object can have two different arrangements:

(i) if adjacent cells are occupied, then q is an edge point (Fig. 2d);

(ii) if diagonally opposite cells are occupied, then q is a 270° vertex (Fig. 2c). It may be noted that, out of the four edges incident at q , exactly two edges will be traversed so that q becomes a 270° vertex. For example, if q is visited from the left, then the outgoing edge from q will be directed downwards (shown in solid lines). Similarly, if q is visited from the right, then the outgoing edge will be directed upwards (shown in dashed lines).

C_3 : Three cells are occupied by the object and hence q is classified as a 270° vertex (Fig. 2b).

C_4 : All four cells are occupied by the object, and so q is not a vertex.

During the traversal, a grid point is determined either as a vertex or as a non-vertex point. Since we traverse orthogonally, a grid point, q , if detected as a vertex, can be a 90° vertex or a 270° vertex. Otherwise, q is simply a point on the edge of the orthogonal cover, or a grid point lying inside the object and also inside the orthogonal cover, or lying outside the object and also outside the orthogonal cover. Henceforth in this paper, a 90° vertex is referred to as a Type ‘1’ vertex ($1 \times 90^\circ$), and a 270° vertex as a Type ‘3’ vertex ($3 \times 90^\circ$) for ease of notation.

An intermediate vertex, v_i , is represented by a three-tuple $\langle t_i, d_i, l_i \rangle$, where t_i ($= 1$ or 3) is the type of the vertex, d_i is the direction of traversal from v_i to v_{i+1} , and l_i is the length of the (horizontal/vertical grid-) line segment from v_i to v_{i+1} . The direction of traversal, d_i , from v_i , can assume the value 0, 1, 2, or 3, indicating the direction towards right, top, left, or bottom respectively. The direction of traversal d_i is derived from the previous direction, d_{i-1} (the direction from v_{i-1} along which v_i has been traversed to), and the type of the vertex v_i , and is given by $d_i = (d_{i-1} + t_i) \bmod 4$. Once d_i is computed, the next grid point is determined and its class is evaluated. Thus the traversal proceeds from v_i to v_{i+1} and finally concludes when it returns back to start vertex. For determination of the start vertex, we assume that the top-left point, $p_0(i_0, j_0)$, of the digital object S is given. The point p_0 is top-left point of S if and only if, for each other point $p(i, j) \in S$ either $j < j_0$, or $j = j_0$ and $i > i_0$. Then, the coordinates of the start vertex v_s is given by, $i_s = (\lceil i_0/g \rceil - 1) \times g$, $j_s = (\lfloor j_0/g \rfloor + 1) \times g$. The type t_s of the start vertex v_s is determined by checking the object occupancy of its four neighboring cells. The starting direction from v_s is given by $d_s = (2 + t_s) \bmod 4$.

3. Rules for finding the orthogonal hull

The concavities present in the orthogonal cover are detected and removed when the object boundary is traversed orthogonally along the grid lines as mentioned in Section 2. More importantly, the proposed algorithm finds the orthogonal hull of a

² It may be noted that, a discrete set is referred as horizontally and vertically convex (shortly, *hv-convex*) in digital tomography [4] if all the rows and columns of the set are 4-connected. We provide the above definition in the perspective of the real plane, since it helps proving the correctness of our algorithm in a simpler way.

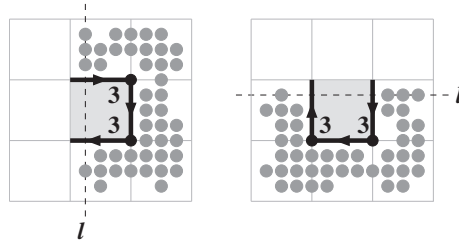


Fig. 3. A concave region possesses two or more consecutive vertices of Type 3, which give rise to multiple intersections with a vertical line (left) or a horizontal line (right).

digital object without any prior knowledge about its orthogonal cover. Deriving the orthogonal hull, therefore, proceeds with the orthogonal traversal around the object boundary.

During the traversal, if two vertices of Type 3 appear consecutively, then it implies a concave region, which defies the property of orthogonal convexity (Definition 4). Illustrated in Fig. 3 are two such patterns for which the intersection of a vertical or a horizontal line, l , with the orthogonal polygon has more than one segment. Our goal is to identify such regions and derive the edges of the orthogonal hull such that the properties of orthogonal convexity are maintained. In this incremental algorithm, the part of the orthogonal hull obtained up to a point does not contain two consecutive vertices of Type 3, which acts as the invariant of the algorithm. Whenever such an occurrence appears, we apply necessary reduction rules to maintain the invariant and to ensure the orthogonal convexity, thereof. However, rest of the patterns, 13, 31, and 11, are in conformance with the algorithm's invariant and hence do not violate the properties of orthogonal convexity. Hence, in effect, our strategy is to remove all patterns of 33 during the orthogonal traversal such that the resulting orthogonal polygon is free of two consecutive vertices of Type 3 and is also area-minimized.

Let v_0, v_1, v_2, v_3 , and v_4 be five consecutive vertices for which the rule has to be applied in order to remove the concavity, if any, where v_4 is the most recently traversed vertex, and $v_0 \dots v_3$ are the previous four vertices already visited. Since a reduction rule is applied only when two consecutive vertices are of Type 3, we have designed two sets of rules depending on whether the type of the vertex following the pattern 33 is 1 or 3. We consider that the two consecutive vertices of Type 3 are designated by v_2 and v_3 , and the vertex v_1 preceding v_2 is always of Type 1. The type of the vertex v_0 can be either 1 or 3. For, in our algorithm, the traversal always starts from a vertex of Type 1, which is verified from the combinatorial arrangement of its four neighboring cells, as explained in Section 2. Adopting this policy of starting the traversal always ensures that there will be at least one vertex of Type 1 preceding two consecutive vertices of Type 3. A dummy vertex is introduced before the start vertex to handle the situation when two consecutive Type 3 vertices appear immediately after the start vertex. It may be observed that only the Rule R12 may have to be applied in such a case.

3.1. Pattern 1331

This pattern signifies a Type 1 vertex followed by two consecutive Type 3 vertices and another Type 1 vertex. Occurrence of two consecutive 3s essentially signifies a concavity in the object, as explained earlier. The rules for removal of the associated concavities are stated in Fig. 4. The concave regions are detected and coalesced to their corresponding convex products using the related edge lengths in the reduction mechanism. There can arise three cases depending on the relation between l_1 and l_3 , which are as follows:

- Rule **R11** (Applied when $l_1 = l_3$): Vertices v_1, v_2, v_3 , and v_4 are removed, and the length of v_0 is modified to $l_0 + l_2 + l_4$.
- Rule **R12** (Applied when $l_1 > l_3$): The lengths l_1 and l_2 are modified as $l_1 - l_3$ and $l_2 + l_4$ respectively. The vertices v_3 and v_4 are removed.
- Rule **R13** (Applied when $l_1 < l_3$): Here, v_1 and v_2 are removed, and l_0 and l_3 are updated to $l_0 + l_2$ and $l_3 - l_1$ respectively.

Note that, in each of the above three cases, the type t_0 of the vertex v_0 remains unaltered, which holds true and causes no problem even if v_0 is the dummy vertex.

3.2. Pattern 1333

Such a pattern signifies a convoluted object boundary. Hence, the traversal is continued until the orthogonal chain comes out of the convoluted (and non-convex, thereof) region. The rules for removal of concavity corresponding to this pattern, shown in Fig. 5, are as follows.

- Rule **R21**: Applied when $l_1 < l_3$. Here, v_1 and v_2 are removed, and l_0 and l_3 are updated to $l_0 + l_2$ and $l_3 - l_1$ respectively.
- Rule **R22**: Applied when $l_1 \geq l_3$ and $d = d_2$. Three Type 3 vertices in succession indicate the beginning of a convoluted region (Fig. 5). The traversal is continued from the vertex v_4 . Let v denote the current vertex up to which the traversal has progressed so far. Let l_h be the horizontal line passing through v_2 and l_v be the vertical line passing through v_4 .

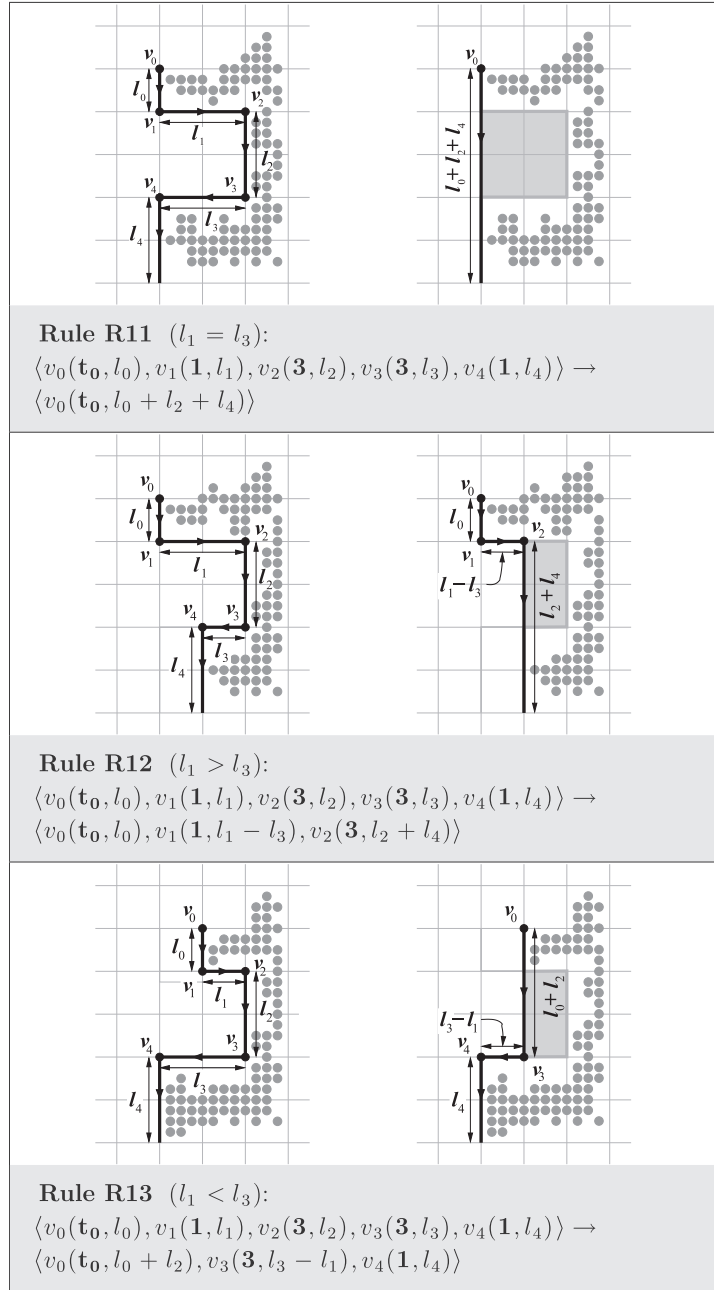


Fig. 4. Concavity (shaded regions) detection and removal rules for pattern **1331**.

The reduction rule is applied only when v lies below the half-plane defined by l_H and to the left of the half-plane defined by l_V simultaneously; otherwise the traversal is continued. As any point above (and inclusive of) l_H or right (and inclusive) of l_V falls within the concave region, no action is taken as long as the current vertex v lies within this region. Two variables, l' and l'' , which are initialized as $l_1 - l_3$ and l_4 respectively, are used to determine whether v has come out of the convoluted-cum-concave region. The lengths l' and l'' are updated depending upon the direction of traversal from v . If the direction of traversal from v , denoted by d , is same as the direction of traversal from v_1 , i.e., d_1 , then l' is updated as $l' = l' + l$, where l is the traversed length from v . On the contrary, if the direction of traversal is same as that of v_3 , then l' is updated as $l' = l' - l$. Similarly, l'' is updated as $l'' = l'' + l$ or $l'' = l'' - l$, depending on whether $d = d_4$ or $d = d_2$. The traversal is continued until the conditions i) $l' < l_1 - l_3$ and ii) $l'' < l_2$ are fulfilled. Once these conditions are reached, the reduction rule is applied.

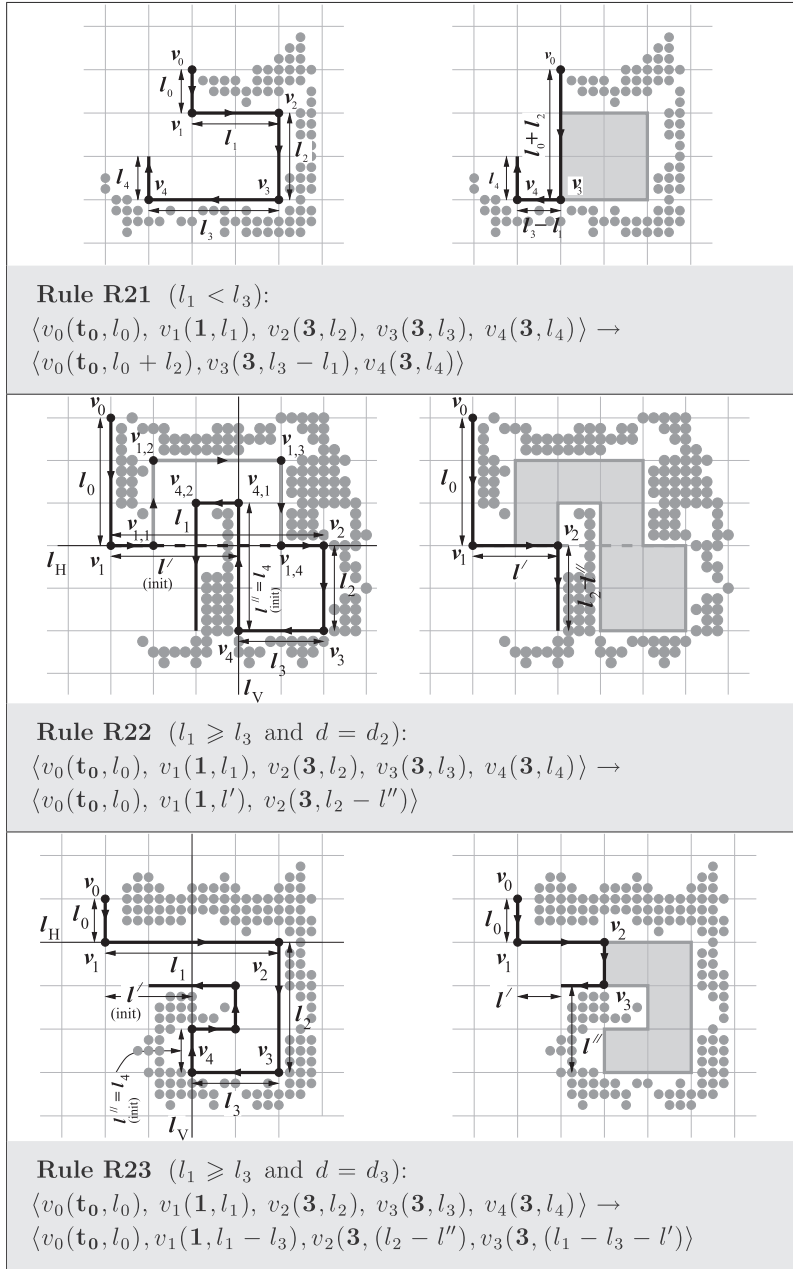


Fig. 5. Concavity detection and removal rules for pattern 1333.

As stated above, the quarter-plane below l_H and left of l_V is the region where, when the traversal reaches, a reduction rule is applied. Entry to this quarter-plane can occur either from the half-plane above l_H or from the half-plane right of l_V . Rule **R22** formulates the reduction for the former case, i.e., when the direction of exit from convoluted region is same as d_2 . The length l_1 is modified as l' and l_2 is modified as $l_2 - l''$. The vertices v_3 and v_4 are removed.

For example, in Fig. 5, the sequence of vertices for reduction is $\langle v_0, v_1, v_2, v_3, v_4 \rangle$. Notice that this chain is obtained after the reduction of $v_1, v_{1,1}, v_{1,2}, v_{1,3}, v_{1,4}$ (Rule **R21**). When $v_{4,1}$ is visited, no reduction is done as $l' < l_1 - l_3$ and $l'' > l_2$, and the traversal is continued to $v_{4,2}$. At $v_{4,2}$, it is found that $l' < l_2$ and $l' < l_1 - l_3$, and hence the reduction rule **R22** is applied.

- Rule **R23**: Applied when $l_1 \geq l_3$ and $d = d_3$, i.e., when an entry to the quarter-plane below l_H and left l_V occurs from the right.

The direction of traversal is d_3 when the conditions (i) $l' < l_1 - l_3$ and (ii) $l'' < l_2$ are fulfilled. The vertex v_4 is removed and l_1, l_2 , and l_3 are modified as follows: $l_1 = l_1 - l_3, l_2 = l_2 - l''$, and $l_3 = l_1 - l_3 - l'$.

To summarize, each rule for the pattern **1331** removes a non-convoluted concave region depending on l_1 and l_3 , whereas each rule for the pattern **1333** finds the start of a convoluted region and reduces when the traversal finally comes out of the convoluted region, using the lengths l_0, \dots, l_4 . It may be noted that the outcome of the Rule **R21** and the Rule **R23** has a pattern **33** at the end. This pattern is removed when the next vertex is visited and the appropriate rule is applied depending on its type.

3.3. Applying the rules

The rules for detection and removal of concavities are applied while traversing around the object contour (without touching the contour) in an orthogonal path. A list L is initialized with a dummy vertex, whose necessity is explained earlier in Section 3. The traversal starts from the start vertex, which is determined from the given top-left point of the digital object. As the traversal is continued, each new vertex visited is appended to the list L . Then, the last five vertices of L are checked for reducibility. If a reduction is done, then again the last five vertices of L are checked for reducibility. When no reduction is done, then the traversal proceeds to the next vertex. The process is continued until the traversal reaches the start vertex, which is the terminating condition of the algorithm. At the termination of the algorithm, the list L contains the list of the vertices of the orthogonal hull in order.

4. The algorithm to find the orthogonal hull

A brief outline of the algorithm is given in Fig. 6, with a demonstration in Fig. 7. The start vertex v_s is computed from p_0 , and in the **do-while** loop, the next vertex v_n is determined and appended to L . If the types of last four vertices of L have the pattern **1331**, then one of the rules of **R11**, **R12**, or **R13** is applied depending on certain conditions (discussed in detail later). Similarly, one of the rules of **R21**, **R22**, or **R23** is applied if the vertex types of the last four vertices have the pattern **1333**. The **do-while** loop continues until v_s is reached.

The algorithm **ORTHO-HULL** that outputs the ordered list of vertices of the orthogonal hull corresponding to a digital object, is shown in Fig. 8. It takes the digital object S , the grid size g , and the start vertex v_s as input parameters. The type (t) and the direction (d) from the start vertex is determined as explained in Section 2. The list L is initialized with a dummy vertex, v_d . The length of the edge from v_d is trivially set to zero, and the type and the direction of traversal from v_d have no significance in our algorithm. The dummy vertex is required because a reduction rule is always applied on a sequence of five vertices. A situation may arise when v_s , which is always of Type **1**, is followed by two consecutive Type **3** vertices, and one Type **1** vertex, leading to a pattern **1331** and $l_1 > l_3$. Then the reduction Rule **R12** has to be applied (on five consecutive vertices), which requires a dummy vertex. In each iteration of the **do-while** loop (Steps 3–14), the next vertex is evaluated and appended to L . If the list L contains more than five vertices (Step 6), then the pattern formed by the types of the last four vertices in L is checked. The vertices v_4, v_3, v_2, v_1 , and v_0 , as mentioned in Section 3.1, correspond to the vertices represented by $L[k], L[k-1], L[k-2], L[k-3]$, and $L[k-4]$ respectively, where $L[k]$ is the most recently visited vertex. If the

1. Find the start vertex, v_s .
2. **do**
3. find the next vertex, v_n (Sec. 2)
4. append it to the list L (initially empty)
5. **if** last four vertex types in L has the pattern **1331**
6. **then** apply rule **R11**, **R12**, or **R13** (Sec. 3)
7. **else if** last four vertex types in L has the pattern **1333**
6. **then** apply rule **R21**, **R22**, or **R23** (Sec. 3)
7. **while** ($v_n \neq v_s$)

Fig. 6. Brief outline of the proposed algorithm.

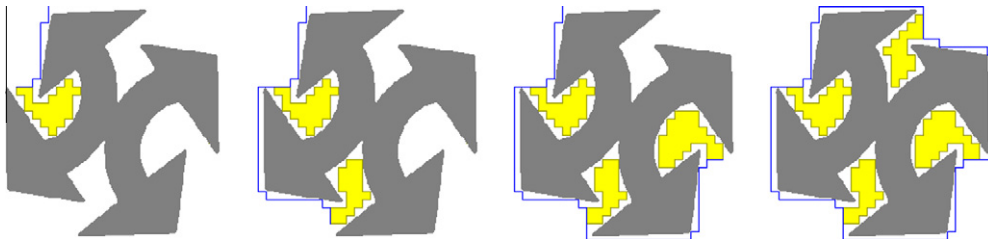


Fig. 7. Demonstration of the algorithm on a sample 2D object. Each image shows the result after removal of the concave parts in successive steps.

Algorithm ORTHO-HULL (S, g, v_s)

Steps:

1. $L[1] \leftarrow v_d, k \leftarrow 2$
2. $L[k].d = d_s, L[k].t = t_s, \langle i, j \rangle \leftarrow \langle i_s, j_s \rangle$
3. **do**
4. $\langle (d, t, L[k].l), i, j \rangle$
 $\leftarrow \text{NEXT-VERTEX}(S, i, j, L[k].d, g)$
5. **while** (TRUE)
6. **if** ($k \geq 5$)
7. **if** ($L[(k-3)..k].t = 1331$)
8. **then** $k \leftarrow \text{APPLY-R1}(L, k)$
9. **else if** ($L[(k-3)..k].t = 1333$)
10. **then** $\langle k, i, j \rangle \leftarrow \text{APPLY-R2}(L, k, i, j, g)$
11. **else break**
12. $k \leftarrow k + 1$
13. $L[k].d \leftarrow d, L[k].t \leftarrow t$
14. **while** ($\langle i, j \rangle \neq \langle i_s, j_s \rangle$)

Procedure NEXT-VERTEX (S, i, j, d, g)

Steps:

1. $m \leftarrow 0, r \leftarrow 0, l \leftarrow 0$
2. **while** (TRUE)
3. $(i, j) \leftarrow d \otimes (i, j)$
4. **for** $h \leftarrow 1$ **to** 4
5. **if** $Q_h(i, j) \cap S \neq \emptyset$
6. $m \leftarrow m + 1, r \leftarrow r + h$
7. **if** $r \in \{4, 6\}$ **and** $m = 2$ **then** $t \leftarrow 3$
8. **else if** $m \in \{0, 2, 4\}$ **then** $t \leftarrow 0$
9. **else** $t \leftarrow m$
10. $d \leftarrow (t + d) \bmod 4, l \leftarrow l + g$
11. **if** $t \neq 0$ **then break**
12. **return** $\langle (d, t, l), i, j \rangle$

Fig. 8. The algorithm ORTHO-HULL that uses the procedure NEXT-VERTEX and the reduction Rules **R1** and **R2** (respective procedures in Figs. 9 and 10).

pattern due to types of $L[k] \dots L[k-3]$ matches **1331**, then the reduction rule is applied by calling the procedure APPLY-R1 (Steps 7 and 8). The procedure APPLY-R1 returns the updated value of k due to reduction. Similarly, if the pattern matches **1333**, then the corresponding reduction rules (APPLY-R2) are used (Steps 9–10). APPLY-R2 returns the modified k and $\langle i, j \rangle$. If a reduction is done, the algorithm continues to evaluate the pattern of last four vertices after reduction, in the **while** loop (Steps 5–11), to check if further reductions are possible. Otherwise, when no rules are applicable (Step 11), it comes out of the **while** loop and continues to evaluate the next vertex. The variable k always points to the last vertex appended to L and is incremented with the visit of each vertex. The procedure NEXT-VERTEX computes the length corresponding to $L[k]$. It also computes the direction d and the type t of the next vertex. The value of k is incremented in Step 12, and d and t are assigned to the incremented $L[k]$. The algorithm terminates (Step 14) when the coordinates $\langle i, j \rangle$ coincides with $\langle i_s, j_s \rangle$.

In the procedure NEXT-VERTEX, in the **while** loop (Steps 2–11), the next vertex from the current vertex (i, j) is evaluated. The coordinates of the next grid point is determined in Step 3. Steps 4–10 determine whether the grid point is a vertex (Type **1** or **3**) or an edge point. The type of a grid point is determined by the object-intersecting quadrants incident at that grid point (Steps 4–6), as explained in Section 2. The algorithm continues to evaluate the subsequent grid points until a vertex ($L[k+1]$) is reached. As it proceeds to the next grid point, the length is incremented by g (Step 10). The procedure returns the type and the direction of the vertex $L[k+1]$, which it has traversed to, and the length l from the current vertex ($L[k]$) to the next vertex ($L[k+1]$). The coordinates of the next grid point are computed (Step 3) based on the direction d from (i, j) . If the direction of traversal is towards right ($d = 0$) or left ($d = 2$), then the y -coordinate remains unchanged and the x -coordinate is incremented ($d = 0$) or decremented ($d = 2$) by g ; and if the direction is upwards ($d = 1$) or downwards ($d = 3$), then the y -coordinate is decremented ($d = 1$) or incremented ($d = 3$) while the x -coordinate remains unchanged. In other words, for $d = 0, \langle i, j \rangle \leftarrow \langle i, j + g \rangle$; for $d = 2, \langle i, j \rangle \leftarrow \langle i, j - g \rangle$; for $d = 1, \langle i, j \rangle \leftarrow \langle i + g, j \rangle$; and for $d = 3, \langle i, j \rangle \leftarrow \langle i - g, j \rangle$. The above transformations are denoted by $(i, j) = d \otimes (i, j)$ in Step 3 for notational simplicity. The subsequent direction of traversal is determined in Step 10.

In procedure APPLY-R1, if the lengths of the vertices $L[k-3]$ and $L[k-1]$ are same (Step 1), then the length of $L[k-4]$ is modified as shown in Step 2. The tail of the list is reset to $k-4$ (Step 3) as the last four vertices are removed according to the Rule **R11**. Similarly, Rules **R12** and **R13**, are implemented in Steps 5–7 and Steps 8–13, respectively. While applying the Rule **R13**, as vertices $L[k-3]$ (v_1) and $L[k-2]$ (v_2) are removed, the vertices $L[k-1]$ (v_3) and $L[k]$ (v_4) are reassigned as

```

Procedure APPLY-R1 ( $L, k$ )
Steps:
1. if ( $L[k-3].l = L[k-1].l$ )
2.   then  $L[k-4].l \leftarrow L[k-4].l + L[k-2].l + L[k].l$ 
3.      $k \leftarrow k - 4$ 
4. else if ( $L[k-3].l > L[k-1].l$ )
5.   then  $L[k-3].l \leftarrow L[k-3].l - L[k-1].l$ 
6.      $L[k-2].l \leftarrow L[k-2].l + L[k].l$ 
7.      $k \leftarrow k - 2$ 
8. else
9.    $L[k-1].l \leftarrow L[k-1].l - L[k-3].l$ 
10.   $L[k-4].l \leftarrow L[k-4].l + L[k-2].l$ 
11.   $L[k-2] \leftarrow L[k]$ 
12.   $L[k-3] \leftarrow L[k-1]$ 
13.   $k \leftarrow k - 2$ 
14. return  $k$ 

```

Fig. 9. The procedure to remove the concavity formed by the vertex pattern **1331**.

```

Procedure APPLY-R2 ( $L, k, i, j, g$ )
Steps:
1. if ( $L[k-3].l < L[k-1].l$ )
2.   then  $L[k-1].l \leftarrow L[k-1].l - L[k-3].l$ 
3.      $L[k-4].l \leftarrow L[k-4].l + L[k-2].l$ 
4.      $L[k-2] \leftarrow L[k]$ 
5.      $L[k-3] \leftarrow L[k-1]$ 
6.      $k \leftarrow k - 2$ 
7. else if ( $L[k-3].l \geq L[k-1].l$ )
8.   then  $l' \leftarrow L[k-3].l - L[k-1].l, l'' \leftarrow L[k].l$ 
9.      $d \leftarrow L[k].d$ 
10.    while ( $l' \geq L[k-3].l - L[k-1].l$  or  $l'' \geq L[k-2].l$ )
11.       $\langle (d, t, l), i, j \rangle \leftarrow \text{NEXT-VERTEX}(S, i, j, d, g)$ 
12.      if ( $d = d_{k-3}$ ) then  $l' \leftarrow l' + l$ 
13.      else if ( $d = d_{k-1}$ ) then  $l' \leftarrow l' - l$ 
14.      if ( $d = d_k$ ) then  $l'' \leftarrow l'' + l$ 
15.      else if ( $d = d_{k-2}$ ) then  $l'' \leftarrow l'' - l$ 
16.    if ( $d = d_{k-2}$ )
17.      then  $L[k-2].l \leftarrow L[k-2].l - l''$ 
18.       $L[k-3].l \leftarrow l'$ 
19.       $k \leftarrow k - 2$ 
20.    else if ( $d = d_{k-1}$ )
21.      then  $L[k-3].l \leftarrow L[k-3].l - L[k-1].l$ 
22.       $L[k-2].l \leftarrow L[k-2].l - l''$ 
23.       $L[k-1].l \leftarrow L[k-3].l - L[k-1].l - l'$ 
24.       $k \leftarrow k - 1$ 
25. return  $\langle k, i, j \rangle$ 

```

Fig. 10. The procedure to remove the concavity formed by the vertex pattern **1333**.

$L[k-3]$ and $L[k-2]$ respectively (Steps 11–12). The tail of the list, k , is reset to $k-2$, as two vertices have been removed (Step 13). The value of k is reset to $k-4$ (Step 3) and $k-2$ (Step 7) respectively for Rules **R11** and **R12**. The procedure returns the updated value of k in Step 14.

The procedure APPLY-R2 implements the second set of rules corresponding to the pattern **1333**. Rule **R21**, applied when $l_1 < l_3$, is implemented in Steps 1–6. Otherwise, when $l_1 \geq l_3$ (Step 7), two variables l' and l'' are initialized in Step 8. The **while** loop (Steps 10–15) is repeated as long as the condition $l' \geq l_1 - l_3$ or the condition $l'' \geq l_2$, is satisfied. In the **while** loop, the next vertex is visited and the values of l' and l'' are modified depending upon the direction of traversal. If the direction of traversal is same as the direction of traversal from $L[k-3]$, then l' is updated to $l' + l$; if it is same as that of $L[k-1]$, then l' is updated to $l' - l$ (Steps 12–13). Similarly, l'' is also modified (Steps 14–15). Thus, each time a new vertex is visited, the parameters l' and l'' are modified. When it comes out of the **while** loop, two different rules, **R22** and **R23**, are applied depending upon the current direction of traversal. If this direction of traversal is same as that from $L[k-2]$ (Step 16), then **R22** is applied (Steps 17–19). When the direction is same as that of $L[k-1]$, Rule **R23** is applied (Steps 20–24). The value of k is readjusted to $k-2$ (Step 19) for Rule **R22**, and to $k-1$ (Step 24) for Rule **R23**. The procedure returns the updated k and $\langle i, j \rangle$ (Step 25).

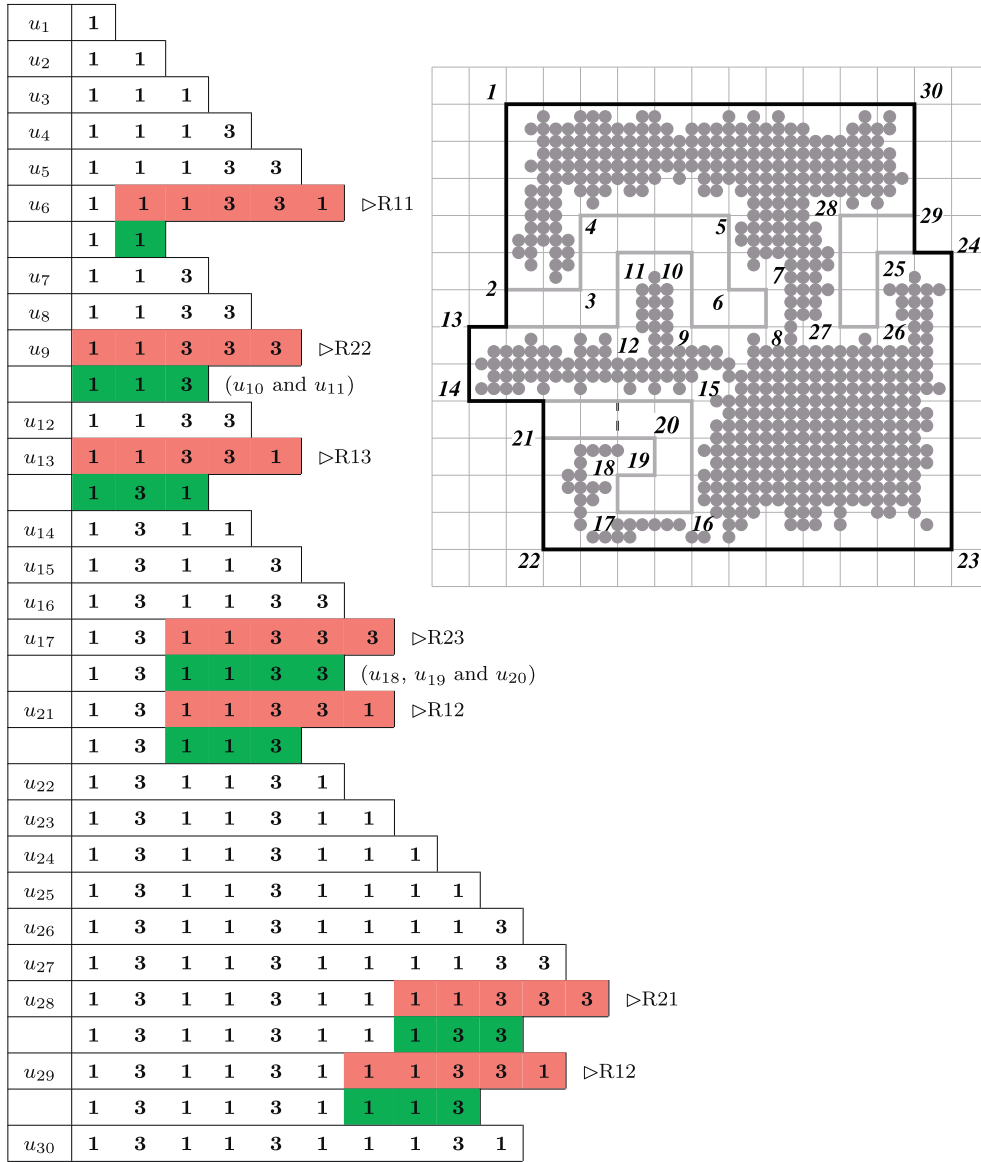


Fig. 11. Demonstration of the proposed algorithm on a digital object (shown in right). Each vertex u_i of the orthogonal polygon (right) has been labeled as 'i' for sake of simplicity. For each iteration of the algorithm, the last five vertices in the list L (left) have been highlighted in red before a rule is applied and in green after two consecutive vertices of Type 3 are removed by the rule. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

4.1. Algorithm ORTHO-HULL is correct

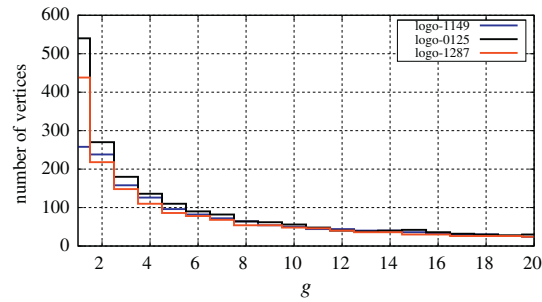
The proof of correctness of the algorithm is based on the fact that the orthogonal hull produced by the algorithm is of minimum area with each object point strictly lying inside it, and the intersection of the hull with any horizontal or vertical line is either empty or exactly one line segment (Definition 4).

To show that the orthogonal hull produced by the algorithm ORTHO-HULL is of minimum area, we first observe that if p is a point lying on the grid line and lying left while traversing around the object S , then $0 < d_{\top}(p, S) \leq g$.³ Since $h > g$, neither Q_1 nor Q_4 has any object containment. Hence, the traversal traces the grid-line segment common to Q_3 and Q_4 , and then traces

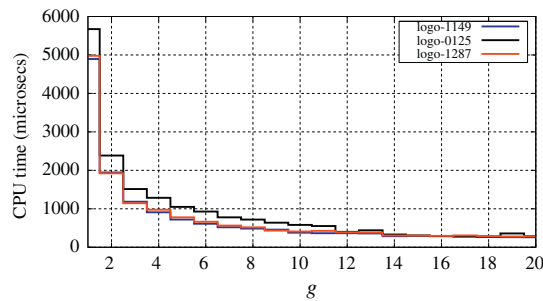
³ Here, $d_{\top}(p, q) = \max\{|i_p - i_q|, |j_p - j_q|\}$ denotes the (orthogonal) distance between two points, $p(i_p, j_p)$ and $q(i_q, j_q)$ (i.e., the Minkowski norm L_1 [23]). The distance of a point p from the object S is $d_{\top}(p, S) = \min\{d_{\top}(p, q) : q \in S\}$. That $d_{\top}(p, S) > 0$ is evident from our policy of object containment in a cell of the grid, as explained in Section 2. To show that $d_{\top}(p, S) \leq g$, let, w.l.o.g., p be any point on the horizontal grid line-segment common to Q_1 and Q_4 (Fig. 2). Let, for contradiction, $d_{\top}(p, S) = h > g$. For any point p' in Q_1 or Q_4 , $d_{\top}(p, p') \leq g$.



Fig. 12. Orthogonal hull (the edges shown in blue and the interior in green) of the binary image 'dragon' for $g = 8$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)



(a) No. of vertices vs. grid size.



(b) CPU Time vs. grid size.

Fig. 13. (a) Shows that the number of vertices of the orthogonal hull decreases with the increasing grid size. (b) The CPU time required for computation of the orthogonal hull decreases with the increase of grid size.

either the grid line-segment common to Q_2 and Q_3 (90° vertex, Fig. 2a) or the grid line-segment common to Q_1 and Q_2 (180°). In either case, the grid-line segment common to Q_1 and Q_4 (and p , there of) lies to the right during the traversal around S —a contradiction.

Thus, each cell lying left of the traversed path around the object contour has object containment, and each free cell (without object containment) lies right of the traversed path. Free cells are included in the orthogonal hull only when the reduction rules are applied. It is evident from the rules (Section 3) and their related figures (Figs. 4 and 5) that the free cells included in the orthogonal hull are minimum in number. For, if any of these free cells (making a non-convex region) is

not included, then it would create either a hole inside the hull or two consecutive vertices of Type 3. This completes the proof that the orthogonal is of minimum area.

Now, to show that the intersection of the orthogonal hull with any horizontal or vertical (grid or non-grid) line is either empty or a single line segment, we simply observe that the final polygon does not contain two consecutive vertices of Type 3. In accordance with the concavity-removal rules, it is evident that, excepting the Rules **R21** and Rule **R23**, the outcome of each rule contains no two consecutive vertices of Type 3 (Section 3.2). The outcome of **R21** or **R23** has a pattern **33** at the end, which is removed depending on the type of the next vertex. In the final iteration, the last vertex visited coincides with the start vertex, which is of Type 1, and hence, the applied rule is one among **R11**, **R12**, and **R13**. Therefore, on termination, we get no two consecutive vertices of Type 3.

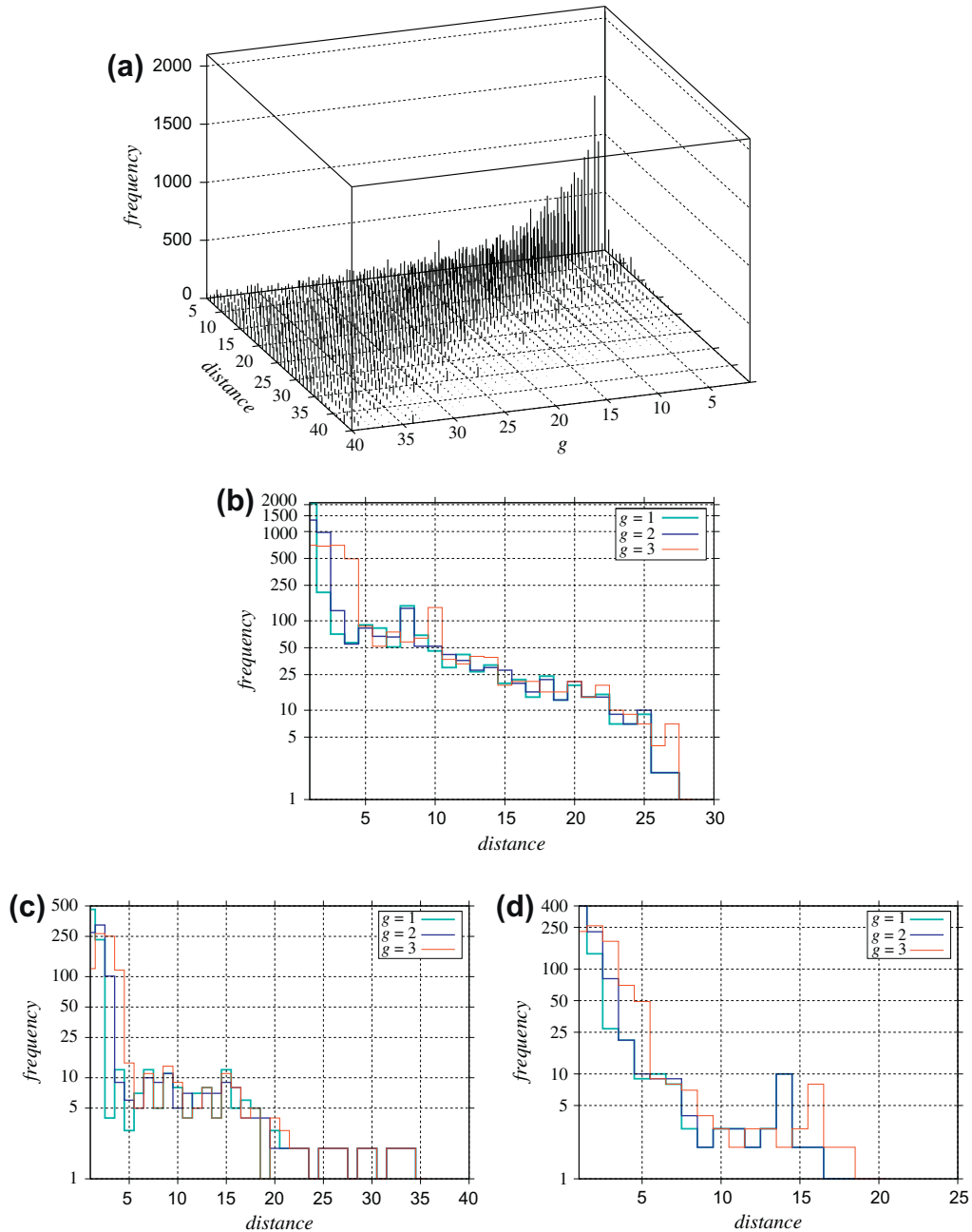


Fig. 14. (a) Frequency of errors plotted against g corresponding to the image 'dragon'. (b) Plot on the error frequency corresponding to 'dragon' for $g = 1, 2$, and 3 . (c and d) Error frequencies for two logo images, 'logo355' and 'logo353'.

Table 1

Comparison of experimental results on convex hulls and orthogonal hulls.

Image name and size	#Pixels	CH	A(CH)	T(CH)	OH			A(OH)			T(OH)		
					g = 4	8	14	g = 4	8	14	g = 4	8	14
logo245 288 × 288	9868	38	39509	682	86	48	26	35681	37929	39887	3.24	2.78	2.63
logo247 288 × 288	10096	25	32419	698	66	28	14	29929	31857	33349	2.08	1.83	1.88
logo353 288 × 288	18122	33	32241	1375	120	60	32	31117	32593	35953	3.68	2.88	2.73
logo354 288 × 288	7179	33	39237	263	88	44	42	35835	37025	41469	3.40	2.91	2.62
logo415 288 × 288	13590	21	29131	894	98	56	40	23149	24601	27735	3.33	2.81	2.544
dragon 1000 × 1000	89744	37	564925	8610	314	168	94	526961	533409	546449	39.10	34.29	33.33
'R' 288 × 288	26369	12	35141	1087	16	12	8	37237	39441	43149	3.62	2.83	2.70
'P' 288 × 288	22320	24	28576	472	40	20	18	27833	28945	33741	3.28	2.72	2.58
swastik 442 × 442	24174	11	35069	3910	204	108	60	35825	38177	41329	10.25	7.81	6.82

|CH| and |OH| denote the respective number of convex hull vertices and number of orthogonal hull vertices; A(CH) and A(OH) are their respective areas; and T(.) indicates the CPU time in milliseconds.

4.2. Time complexity

Since the object is defined as a connected component, the containment of the object in a cell incident at a grid point q is verified from the intersection of the object with the four edges of the corresponding cell. For each edge, the intersection can be checked in $O(g)$ time, where g is the grid size. Hence, checking the object containment in any cell can be done in $4 \times O(g) = O(g)$ time.

During traversal of the grid points lying immediately outside the object contour, we visit each grid point q_i from the preceding one, q_{i-1} , using the information on intersection of the object with the edges incident at q_{i-1} . For example (Fig. 2a), if q_{i-1} is of Type 1 and has been visited along the vertical edge (from its predecessor, q_{i-2}), then q_i is visited along the horizontal edge from q_{i-1} . Thus, the number of grid points visited while traversing orthogonally along the object contour is bounded by $O(n/g)$, where n is the number of points constituting the object contour. Note that, for a given object, n is fixed. The resultant time complexity for visiting all the vertices is, therefore, given by $O(n/g) \cdot O(g) = O(n)$. Note that, each grid point lying on the orthogonal path of traversal is visited either once (Fig. 2a, b, d) or twice (Fig. 2c). The time spent over detection and removal of concavities is associated with checking a pattern (of types of the last four vertices) in the list L in $O(1)$ time and applying the reduction rule, whenever necessary. If the pattern does not contain two consecutive 3s, a new vertex is visited in $O(g)$ time. If the pattern undergoes a reduction, which needs $O(1)$ time, then also the next vertex is visited. Maximum number of reductions is bounded by $O(n/g) - 4$, since at most $O(n/g)$ vertices are visited and the orthogonal hull consists of at least four vertices. Thus, the total number of list operations is bounded by $(O(n/g) - 4) \cdot O(1) = O(n/g)$. Hence, the total time complexity for finding the orthogonal hull of a digital object is given by $O(n) + O(n/g) = O(n)$. It should be noted here that the value of g can be varied to obtain fine or coarse description of the underlying shape in terms of the orthogonal hull. For the minimum value that g can assume, i.e., $g = 1$, the algorithm gives the tightest orthogonal hull, whereas g can be increased to get a slackened orthogonal hull. Typically, in our experiments, we have varied the value of g from 1 to 24.

4.3. Example

A demonstration of the algorithm is shown in Fig. 11, which illustrates how the different rules are applied for removal of some typical concave regions from a given digital object as the orthogonal traversal proceeds closely around the object contour without touching it. On the left hand side, the list L is shown as it grows with the orthogonal traversal or as it shrinks as a result of applying the reduction rules. The start vertex, u_1 , is determined by a row-wise scan of the grid points. Note that the vertices are indicated in the figure by their indices. For simplicity of the representation, the dummy vertex is not shown here in L . The leftmost column of the table indicates the vertices as they are visited and associated list is shown to its right. Only the types of the vertices are shown in the list L . The first non-convex region is detected when u_6 is visited. On visiting u_6 , the sequence of types of the last four vertices in L matches the pattern **1331** and l_3 (the length corresponding to u_3) equals l_5 (of u_5), and hence Rule **R11** (Fig. 4) is applied. As a result, all four vertices, namely u_3, u_4, u_5 , and u_6 , are removed, and the length of u_2 (equivalent to v_0 in Fig. 4) is modified to $l_2 + l_4 + l_6$. As the traversal continues, the next non-convex region is detected when u_9 is visited. On visiting u_9 , the sequence of types of the last four vertices matches the pattern **1333**, and $l_2 \geq l_8$. Note that the length l_2 has been modified to $l_2 + l_4 + l_6$ in the last reduction. So, the two variables l' and l'' are initialized as $l_2 - l_8$ and l_9 respectively. When u_{10} is visited, l' is modified to $l_2 - l_8 - l_{10}$ and l'' is not changed. As the second condition in the **while** loop in Step 9 of the procedure APPLY-R2 (Fig. 10) still holds, u_{10} is discarded (not added to the list L) and the next vertex u_{11} is visited. Now, l'' is modified to $l_9 - l_{11}$, and none of the conditions in the **while** loop holds true, and so the algorithm comes out of the **while** loop. As the direction of traversal of u_{11} is same as that of u_7 , the Rule **R22** is applied. The list L is left with a pattern **113**. Subsequently, when vertex u_{13} is visited, the Rule **R13** is applied. In the course of traversal, when u_{17} is reached, the procedure APPLY-R2 is called as the sequence of last four vertices in L matches **1333**. The vertices u_{18}

and u_{19} are visited but not added to L as one of the conditions in the **while** loop is still valid. When u_{20} is visited, none of the conditions in the **while** loop is valid, and as d_{20} equals d_{16} , reduction Rule **R23** is applied. Subsequently, Rule **R21** is applied at u_{28} and Rule **R12** at u_{29} . When the traversal finally reaches u_1 , the list L contains the vertices of the orthogonal hull of the given digital object in order.


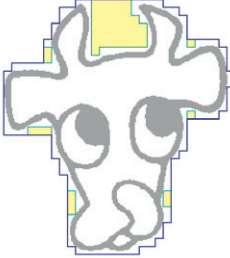
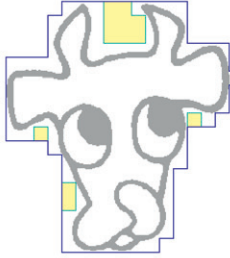
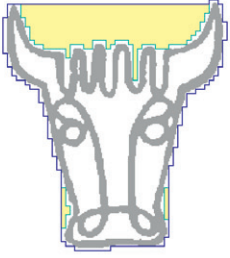
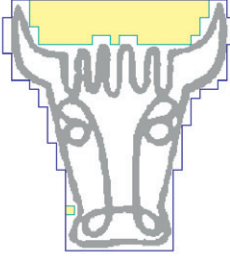
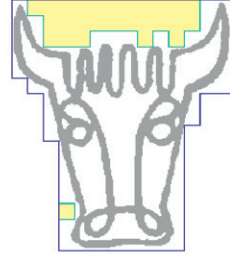
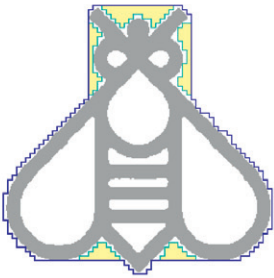
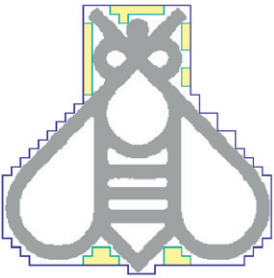
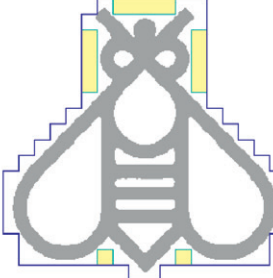

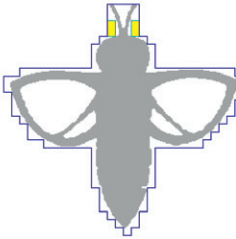
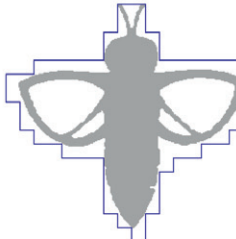
n	$g = 4$	$g = 8$	$g = 14$
logo245 9868	 86 35681	 48 37929	 26 39887
logo247 10096	 66 29921	 28 31857	 14 33349
logo416 16367	 114 36561	 52 38497	 34 42253
logo415 13590	 98 23149	 56 24601	 40 27735

Fig. 15. The orthogonal hulls (edges shown in blue) of a set of logo images for grid size $g = 4, 8$, and 14 . The non-convex regions detected by the algorithm are shown in yellow. The number of vertices of the orthogonal hull is shown in the left box and the area of the hull in the right box below the concerned image. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

5. Experimental results and discussions

The proposed algorithm is implemented in C on a Sun_Ultra 5_10, Sparc, 233 MHz, the OS being the SunOS Release 5.7 Generic, and has been tested on (i) database D1 containing 1034 logo images (received on request, from Prof. Anil K. Jain and Aditya Vailya of Michigan State Univ., USA); (ii) a collected database D2 having 520 shape images; (iii) some geometric shapes; (iv) a selected database of optical characters; (v) a set of gray-scale images.



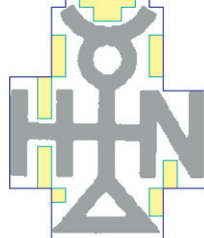



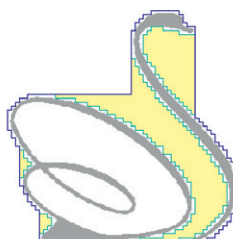
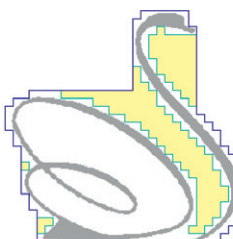
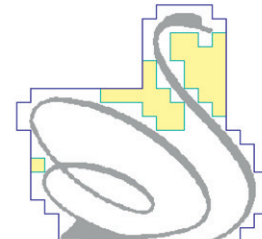
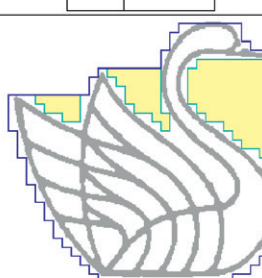
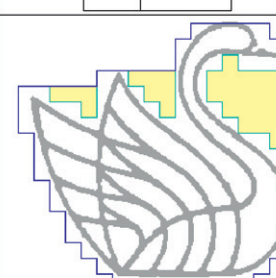
n	$g = 4$	$g = 8$	$g = 14$
logo119 16305	 <div>18 33217</div>	 <div>16 35649</div>	 <div>16 37101</div>
logo353 18122	 <div>120 31117</div>	 <div>60 32593</div>	 <div>32 35953</div>
logo354 7179	 <div>88 35385</div>	 <div>44 37025</div>	 <div>32 41469</div>
logo355 10541	 <div>90 39665</div>	 <div>48 41625</div>	 <div>28 43793</div>

Fig. 16. The orthogonal hulls for another set of logo images corresponding to $g = 4, 8$, and 14 (color codes and numerical figures as in Fig. 15). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Fig. 12 shows the result on an image of 'dragon' for $g = 8$. The orthogonal hull is shown in blue, whereas the red colored polygon indicates the smallest-area orthogonal polygon describing the object, which is traversed during the derivation of the orthogonal hull. The interior region of the orthogonal hull has been highlighted in green. The regions enclosed by the orthogonal hull (blue lines) and the isothetic polygon (red lines) show the non-convex portions detected by the algorithm. The CPU time required for computation of the convex hull for the 'dragon' image, having size 1000×1000 and consisting of 89744 object pixels, is 8.61 s, which is significantly high compared to 34.29 ms required for the computation of its orthogonal hull for $g = 8$ (shown in Table 1). The convex hull is computed using the Graham scan algorithm. The time required for computation of the convex hull is very high compared to orthogonal hulls because of the fact that the computation of an orthogonal hull is based on the orthogonal traversal around the boundary of the object, and it requires only comparison and addition/subtraction in the integer domain. The time required by Graham scan algorithm to compute the convex hull could be reduced by considering the boundary points of the object, but that requires an edge detection algorithm.

In Table 1, the area of the convex hull and the areas of the orthogonal hulls for different grid sizes (4, 8, and 14) are presented. The number of convex hull vertices, the CPU time required for computation of the convex hull, the number of vertices of an orthogonal hull, and the CPU times at different grid sizes are also presented in the table. It can be seen from this table that the number of vertices of OH(S) decreases and its area increases with the increase of grid size. Also, the computation time drops appreciably for higher grid sizes. Apart from the 'dragon' image, Table 1 also presents the results corresponding to four logo images, two OCR images ('P' and 'R'), and one geometric shape ('swastik').

Fig. 15 shows four logo images and their corresponding orthogonal hulls for grid size $g = 4, 8$, and 14. The non-convex regions have been filled in yellow to depict the result of concavity-reduction rules. First column of each row indicates the

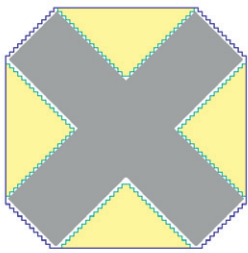
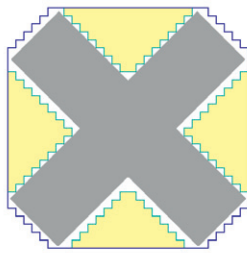
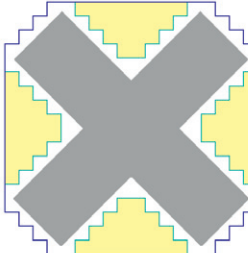
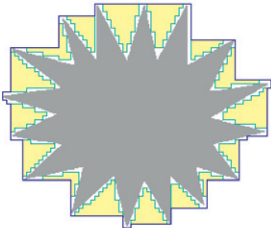
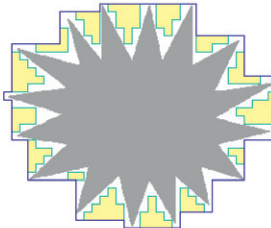
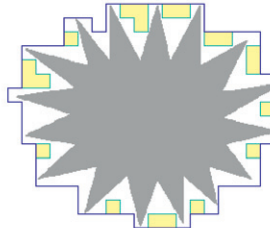
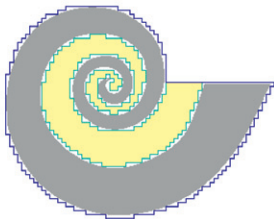

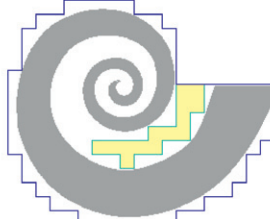
n	$g = 4$	$g = 8$	$g = 14$
32583	 96 53473	 44 54241	 28 59305
28732	 50 42445	 34 44657	 30 46173
23776	 126 38033	 64 40233	 36 43429

Fig. 17. The orthogonal hulls of a set of geometric shapes for $g = 4, 8$, and 14 (color codes and numerical figures as in Fig. 15). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

number of object pixels, and each image is accompanied with the number of vertices and the orthogonal hull area (shown in two boxes) for the corresponding grid size. The number of vertices is plotted against the grid size $g = 1-20$ in Fig. 13a for three different logo images. From these plots, it is evident that the number of vertices rapidly decreases with the increase of grid size. Fig. 13b shows the way the CPU time decreases with increasing g for the same three images. The orthogonal hulls of all four logo images in Fig. 15 reflect the symmetric nature of the objects. Fig. 16 shows another set of four logo images in which, excepting the first image ('logo119'), all images are asymmetric. The similar nature of the last three objects ('logo353',

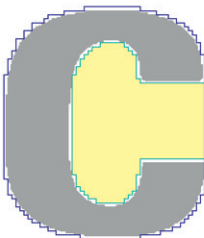
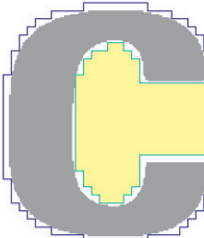
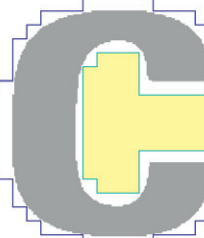
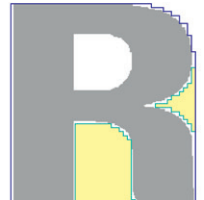




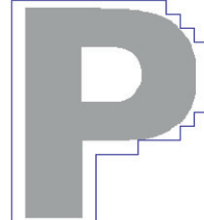
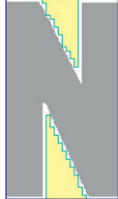


n	$g = 4$	$g = 8$	$g = 14$
24170	 80 41953	 48 44161	 32 48105
26369	 16 37237	 12 39441	 8 43149
22320	 40 27833	 20 28945	 18 33741
17592	 4 22713	 4 23617	 8 22261

Fig. 18. Results on some optical characters (color codes and numerical figures as in Fig. 15). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

'logo354', and 'logo355') are captured by their orthogonal hulls. Fig. 17 shows four geometric shapes and their orthogonal hulls for various grid sizes. The 'spiral' image has a more convoluted shape, and the algorithm successfully constructs the orthogonal hull. Similar to the images in Fig. 15, the orthogonal hulls of the first two images have also rotational symmetry, which remains almost invariant with a change in the grid size. The orthogonal hulls of four optical characters are shown in Fig. 18. The optical characters can be classified into different sets depending upon their orthogonal hulls. A clear demarcation can be made between different characters by using the non-convex regions (marked in yellow) in conjunction with orthogonal hulls. Fig. 19 shows results on binarized impressions of some real-world gray-scale images. In Fig. 14, we have furnished a 3-D plot on the frequency of errors versus g and distances of points on the orthogonal hull from the object corresponding to the image 'dragon'. The error frequency for an orthogonal distance d is given by the number of points on (the border of) $OH(S)$ for which the nearest object point is at the distance d . The other three plots (Fig. 14b–d) show the distribution of frequency versus distance for three grid sizes $g = 1, 2, 3$ corresponding to the images 'dragon', 'logo355', and 'logo353' respectively. The frequency is shown in \log_{10} – scale. It is evident that the number of object points at a smaller distance is very high, whereas, the number of object points lying at a larger distance from $OH(S)$ is low. However, this distribution is dependent on the shape of the object; if there are more concavities, then the distribution may be different, while the basic trend remains the same.



Fig. 19. The orthogonal hulls (color codes and numerical figures as in Fig. 15) of the binarized images corresponding to some real-world gray-scale images for $g = 6$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

6. Conclusion

We have presented a combinatorial algorithm to construct the orthogonal convex hulls of a digital object for various grid resolutions. The worst-case time complexity of the algorithm is linear in the size of the contour. The actual runtimes on different images demonstrate its effectiveness for speedy execution. The algorithm is a single-pass algorithm, and outputs the (types and outgoing edge-lengths of) vertices of the orthogonal hull in order. Hence, we can utilize the orthogonal hull description in some suitable application like shape analysis, shape-based image retrieval, etc. For example, the number of times the reduction rules are applied during the traversal for removal of a concavity can be used to measure the distribution of shape complexity over a large and complex object. Also, the non-convex regions along with the orthogonal hull can be used for an appropriate shape analysis. Presently, we are working on the shape analysis of objects using their orthogonal hulls, which will be reported in the future.

References

- [1] L. Atzori, G. Ginesu, A. Raccis, Jpeg2000-coded image error concealment exploiting convex sets projections, *IEEE Transactions on Image Processing* 14 (4) (2005) 487–498.
- [2] C. Audet, P. Hansen, F. Messine, Extremal problems for convex polygons, *Journal of Global Optimization* 38 (2007) 163–179.
- [3] D. Avis, D. Bremner, How good are convex hull algorithms? in: *Symposium on Computational Geometry*, 1995, pp. 20–28.
- [4] P. Balazs, On the number of hv-convex discrete sets, in: V. Brimkov, R. Barneva, H. Hauptman (Eds.), *Intl. Workshop on Combinatorial Image Analysis (IWICIA)*, LNCS, vol. 4958. Springer-Verlag, Berlin Heidelberg.
- [5] B.B. Barber, D.P. Dobkin, H. Huhdanpaa, The quickhull algorithm for convex hull, *The Geometry Center, University of Minnesota, Technical Report GCG53*, July 1993.
- [6] M.D. Berg, M.V. Kreveld, M. Overmars, O. Schwarzkopf, *Computational Geometry Algorithms and Applications*, Springer-Verlag, Berlin, 2000.
- [7] A. Biswas, P. Bhowmick, B.B. Bhattacharya, TIPS: On finding a Tight Isothetic Polygonal Shape covering a 2D object, in: *Proc. 14th Scandinavian Conf. Image Analysis (SCIA)*, LNCS, vol. 3540, Springer, Berlin, 2005, pp. 930–939.
- [8] A. Biswas, P. Bhowmick, M. Sarkar, B.B. Bhattacharya, Finding the orthogonal hull of digital object: a combinatorial approach, in: V. Brimkov, R. Barneva, H. Hauptman (Eds.), *Intl. Workshop on Combinatorial Image Analysis (IWICIA)*, LNCS, vol. 4958. Springer-Verlag, Berlin Heidelberg.
- [9] F. Bookstein, *Morphometric Tools for Landmark Data: Geometry and Biology*, Cambridge Univ. Press, 1991.
- [10] M. Bousquet-Mélou, A method for the enumeration of various classes of column-convex polygons, *Discrete Mathematics* 154 (1–3) (1996) 1–25.
- [11] L. Boxer, Computing deviations from convexity in polygons, *Pattern Recognition Letters* 14 (1993) 163–167.
- [12] B. Chazelle, An optimal convex hull algorithm in any fixed dimension, *Discrete Computational Geometry* 10 (1993) 377–409.
- [13] T.H. Cormen, C.E. Leiserson, R.L. Rivest, *Introduction to Algorithms*, Prentice Hall of India, New-Delhi, 2000.
- [14] L.d.F. Costa, J.R.M. Cesar, *Shape Analysis and Classification*, CRC Press, 2001.
- [15] E. Fink, D. Wood, Fundamentals of restricted-orientation convexity, *Information Sciences* 92 (1996) 175–196.
- [16] R. Graham, An efficient algorithm for determining the convex hull of a finite point set, *Information Processing Letters* 1 (1972) 132–133.
- [17] S.T. Hyde, S. Andersson, Z. Blum, S. Lidin, K. Larsson, T. Landh, B.W. Ninham, *The Language of Shape*, Elsevier, 1997.
- [18] R.A. Jarvis, On the identification of the convex hull of a finite set of points in the plane, *Information Processing Letters* 2 (1973) 18–21.
- [19] S.-W. Jung, T.-H. Kim, S.-J. Ko, A novel multiple image deblurring technique using fuzzy projection onto convex sets, *IEEE Signal Processing Letters* 16 (3) (2009) 192–195.
- [20] R. Karlsson, M. Overmars, *Scanline algorithms on a grid*, second ed. Dept. of Computer Science, University of Utrecht, Technical Report RUU-CS-86-18, 1986.
- [21] R. Karlsson, M. Overmars, *Scanline algorithms on a grid*, *BIT Numerical Mathematics* 28 (2) (1988) 227–241.
- [22] D.G. Kirkpatrick, R. Seidel, The ultimate planar convex hull algorithm?, *SIAM Journal on Computing* 15 (1986) 287–299.
- [23] R. Klette, A. Rosenfeld, *Digital Geometry: Geometric Methods for Digital Picture Analysis*, Morgan Kaufmann Series in Computer Graphics and Geometric Modeling, Morgan Kaufmann, San Francisco, 2004.
- [24] S.C. Nandy, K. Mukhopadhyaya, B.B. Bhattacharya, Recognition of the largest empty ortho-convex polygon in a point set, in: *Proc. Canadian Conference on Computational Geometry (CCCG)*, 2008.
- [25] T. Ottmann, E. Soisalon-Soininen, On the definition and computation of rectilinear convex hulls, *Information Sciences* 33 (3) (1984) 157–171.
- [26] A.F. Pitty, *Geomorphology*, Blackwell, 1984.
- [27] F. Preparata, M. Shamos, *Computational Geometry*, Springer, Verlag, New York, 1985.
- [28] F.P. Preparata, M.I. Shamos, *Computational Geometry—An Introduction*, Springer-Verlag, New York, 1985.
- [29] A. Rosenfeld, A.C. Kak, *Digital Picture Processing*, 2nd ed., Academic Press, New York, 1982.
- [30] M. Sonka, V. Hlavac, R. Boyle, *Image Processing Analysis and Machine Vision*, Chapman and Hall, 1993.
- [31] H.I. Stern, Polygonal entropy: a convexity measure, *Pattern Recognition Letters* 10 (1989) 229–235.
- [32] G. Swart, Finding the convex hull facet by facet, *Journal of Algorithms* (1985) 17–48.
- [33] W. Trutschnig, G. Gonzalez-Rodriguez, A. Colubi, M.A. Gil, A new family of metrics for compact convex (fuzzy) sets based on a generalized concept of mid and spread, *Information Sciences* 179 (2009) 3964–3972.
- [34] G.B. Unal, A.E. etin, Restoration of error-diffused images using projection onto convex sets, *IEEE Transactions on Image Processing* 10 (12) (2001) 1836–1841.
- [35] C. Weerasinghe, A.W.-C. Liew, H. Yan, Artifact reduction in compressed images based on region homogeneity constraints using the projection onto convex sets algorithm, *IEEE Transactions on Circuits and Systems for Video Technology* 12 (10) (2002) 891–897.
- [36] P. Widmayer, Y.F. Wu, C.K. Wong, On some distance problems in fixed orientations, *SIAM Journal on Computing* 16 (4) (1987) 728–746.
- [37] J. Wu, Z. Jiang, On constructing the minimum orthogonal convex polygon for the fault-tolerant routing in 2-d faulty meshes, *IEEE Transactions on Reliability* 54 (3) (2005) 449–458.
- [38] J. Zunic, P.L. Rosin, A new convexity measure for polygons, *IEEE Transactions on PAMI* 26 (7) (2004) 923–934.