ELSEVIER
Computational Geometry 23 (2002) 69–83
Computational Geometry
Theory and Applications
www.elsevier.com/locate/comgeo

Orthogonal polygon reconstruction from stabbing information [☆]

L. Jackson [a], S.K. Wismath [b,*]

[a] *Department of Computer Science, University of Calgary, Calgary, AB, Canada, T2N 1N4*
[b] *Department of Mathematics and Computer Science, University of Lethbridge, Lethbridge, AB, Canada, T1K 3M4*

Received 13 February 2001; received in revised form 29 November 2001; accepted 14 December 2001

Communicated by K. Mehlhorn

Abstract

Reconstruction of polygons from visibility information is known to be a difficult problem in general. In this paper, we consider a special case: reconstruction of orthogonal polygons from horizontal and vertical visibility information and show that this reconstruction can be performed in $O(n \log n)$ time. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Visibility graph; Reconstruction; Orthogonal polygon

1. Introduction

The *visibility graph* of a simple polygon traditionally consists of a vertex for each corner of the polygon with an edge joining a pair of vertices if the corresponding corners are internally visible. Polygon *reconstruction* results attempt to build a polygon consistent with a given visibility graph. The general polygon reconstruction problem was shown to be in PSPACE by Everett [4], who also characterized the visibility graph of spiral polygons. However, a general characterization of visibility graphs of simple polygons has proven elusive; see for example, the paper by Everett and Corneil [1].

Some authors have investigated the effect of adding extra information to make these problems more tractable. O'Rourke and Streinu [8] defined a richer combinatorial structure called the vertex-edge visibility graph which includes edge-to-edge visibility. Wismath [12] introduced an extended visibility structure called the *stab graph* and showed how parallel line segments can be efficiently reconstructed

[☆] An early version of this work was presented at the 8th Canadian Conference on Computational Geometry. Financial support from N.S.E.R.C. is gratefully acknowledged.

[*] Corresponding author.
E-mail addresses: jackson@cpsc.ucalgary.ca (L. Jackson), wismath@cs.uleth.ca (S.K. Wismath).

from it. The stab information relied on in the current paper is a weaker model of visibility than the stab graph defined there. Everett, Hurtado and Noy [2] have considered stronger variants of stabbing information and investigated the properties of the polygon that can then be derived.

Horizontal and vertical visibility information in orthogonal polygons was first studied by Booth and O'Rourke [7]. In their paper *Realization of Visibility Trees*, they characterized how the two sets of visibility information must mesh in order to form a proper orthogonal polygon. Other related results include *Cross-ratios and Angles Determine a Polygon* by Snoeyink [9] the *Bar Visibility Graphs* studied by Wismath [11] and independently by Tamassia and Tollis [10], and results pertaining to external visibility graphs by Everett, Lubiw and O'Rourke [3]. Although results in this area are primarily of theoretical interest, applications to orthogonal polygons and constrained visibility include VLSI design, robotics and GIS.

In this paper we consider the special case of reconstructing an orthogonal polygon from pure visibility information, namely the *stabs* (i.e., the horizontally or vertically visible sides) of the vertices of the polygon. Both internal and external visibility information is supplied. An algorithm that reconstructs an orthogonal polygon from such information in $\mathrm{O}(n \log n)$ time is presented.

1.1. *Definitions and preliminaries*

Initially, we assume that the orthogonal polygon to be reconstructed is simple, has more than four sides and that no pair of sides of the polygon is collinear. In Section 4, the collinearity assumption will be removed.

Associated with each corner (or equivalently for the purpose of this paper, vertex) v of a simple unknown polygon P are two *stabs*, namely the first sides of P encountered if the two sides of P at v were extended. For an orthogonal polygon (aligned with the X and Y axes) these stabs are horizontal and vertical. Let $v_1, v_2, \ldots, v_n$ denote the corners of an orthogonal polygon P with sides $\overline{v_i v_{i+1}}$. Define $\mathit{hstab}(v)$ as the first side of P encountered by extending the horizontal side of P incident at v. Similarly $\mathit{vstab}(v)$ is the first side encountered vertically from v. Both interior and exterior visibilities are provided – the symbol "∞" is used to denote a stab that encounters no side of P.

Given a known orthogonal polygon P, the *construction* of the stab information can be computed in $\mathrm{O}(n \log n)$ time via a straightforward line sweep algorithm. The problem described here is the opposite of this construction: Given only the set of stabs of a realizable unknown orthogonal polygon, reconstruct the polygon. Note that stabbing information does not *uniquely* determine an orthogonal polygon, but rather partitions polygons into equivalence classes.

Orthogonal Polygon Reconstruction (OPR). Let $V = \{v_1 \ldots v_n,\ n > 4\}$ denote a set of vertices of an unknown orthogonal polygon in clockwise order. Inputs $\mathit{hstab}(v_i)$ and $\mathit{vstab}(v_i),\ i = 1, 2, \ldots, n$, for this unknown polygon are given. Compute x and y coordinates (x_i, y_i) for each vertex v_i that produce an orthogonal polygon consistent with the stabbing information.

See Fig. 1 for an example of the input given and a resulting reconstructed polygon.

The first stage of the algorithm to solve the OPR problem identifies each vertex as either *convex* or *reflex*.[1] Only the horizontal stab information is required.

[1] A *convex* corner has an internal angle of $\Pi/2$ radians, while a *reflex* corner has an internal angle of $3\Pi/2$.

Vertex	hstab	vstab
v_1	∞	∞
v_2	∞	∞
v_3	v_5v_6	∞
v_4	v_1v_2	$v_{10}v_1$
v_5	v_9v_{10}	$v_{10}v_1$
v_6	∞	∞
v_7	∞	∞
v_8	∞	∞
v_9	v_5v_6	v_6v_7
v_{10}		

OPR Input

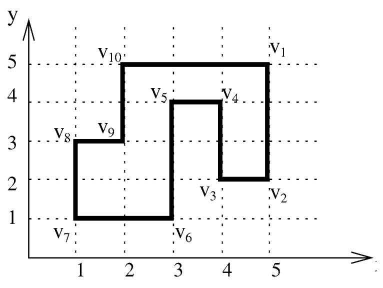

Fig. 1. An example of the OPR problem.

The second stage of the algorithm computes two partial orders representing the x and y dimensions based on the stabbing and convexity information and subsequently assigns (integer) coordinates to the vertices, yielding an orthogonal polygon consistent with the inputs. Both horizontal and vertical stab information is utilized.

2. Properties of orthogonal polygons

Before describing the reconstruction algorithm, some properties of orthogonal polygons are considered, in particular, the types of rectangles formed from the horizontal stabs.

An orthogonal polygon with only the horizontal stabbing rays drawn, partitions the plane into a collection of rectangles as in Fig. 2.

For the assumption of no collinear sides, any orthogonal polygon has a unique top-most and a unique bottom-most side. These sides bound degenerate rectangles as shown in Fig. 4; they will be labelled type 0 rectangles and can be efficiently identified.

The following lemma characterizes the horizontal rectangles created with the orthogonal polygon and its horizontal stabs.

Lemma 1. *Aside from the two type 0 rectangles, rectangles of types 1 through 12 are the only possible rectangles created from the sides of an orthogonal polygon and its horizontal stabs.*

Proof. Every rectangle has exactly four corners. It is possible that zero to three of those corners correspond to vertices of the polygon. The assumption that the polygon has more than four sides eliminates the possibility that all four corners of the rectangle correspond to vertices of the polygon. At each corresponding corner the polygon could turn toward or away from the rectangle, as shown in Fig. 5. Those rectangles with stabs to infinity are assumed to be completed by a 'pseudo' side at infinity. Depending on how the stabs hit the sides of the polygon, twelve different types of rectangles, as enumerated in Fig. 3, are possible. Horizontally and vertically symmetric situations are ignored.

- *Case 0.* When *zero* corners of a rectangle correspond to vertices of the polygon, a type 12 rectangle is created.
- *Case 1.* With *one* corner of a rectangle corresponding to a vertex of the polygon, the two possible directions (toward and away) at that corner result in the creation of rectangle types 10 and 11.

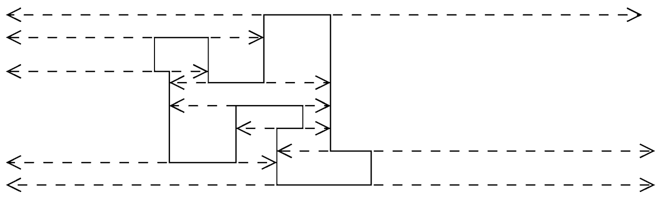

Fig. 2. An orthogonal polygon with horizontal stabs.

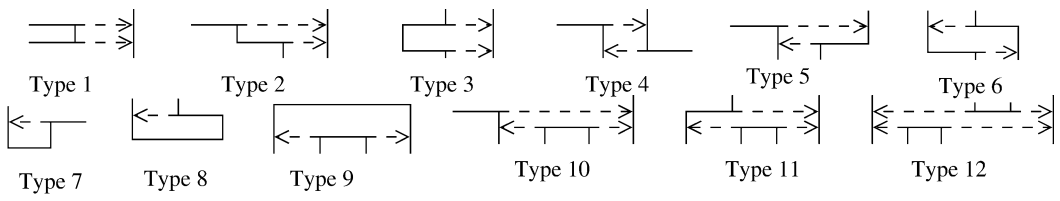

Fig. 3. The twelve possible horizontal rectangles.

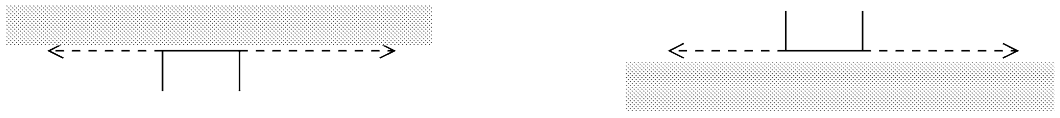

Fig. 4. The two type 0 rectangles.

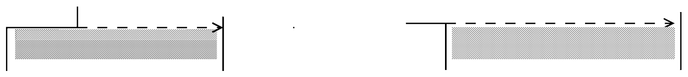

Fig. 5. Turning toward and away from rectangle.

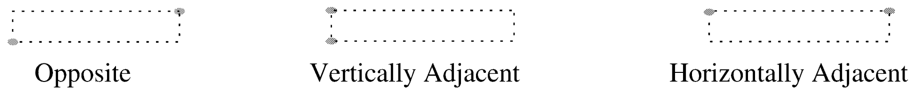

Fig. 6. Two vertices correspond to rectangle corners.

- *Case 2*. When *two* corners of a rectangle correspond to vertices of the polygon, they could be on opposite sides of the rectangle, or vertically adjacent or horizontally adjacent, as shown in Fig. 6.
  - If the corresponding corners are on opposite corners, a type 4, 5 or 6 rectangle is formed.
  - If the corresponding corners are together vertically, a type 1, 2 or 3 rectangle is formed.
  - When the corresponding corners are horizontally adjacent, the polygon sides can only turn toward the rectangle and be consecutive sides of the polygon. If either or both turned away, a stab would be created that defines part of the rectangle, and the vertex would then not correspond to a corner of the rectangle. If both turned away from the rectangle and were not consecutive sides of the polygon, collinear sides would be created. The one rectangle obtained here is a type 9 rectangle.
- *Case 3*. If *three* corners of a rectangle correspond to vertices of the polygon:

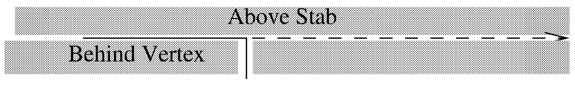

Fig. 7. A vertex is part of 3 rectangles.

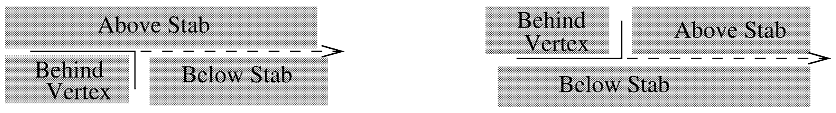

Fig. 8. Two orientations of rectangles around a vertex.

– Two of the three vertices cannot turn away from the rectangle, since this would result in an orthogonal polygon with collinear sides.
– There could be one vertex turning away from the rectangle, and two turning toward it. However, since there are no collinear sides, the three vertices must be consecutive sides of the polygon, thus forming a type 7 rectangle.
– The three vertices could all turn toward the rectangle. Again the three vertices must then be consecutive sides of the polygon, resulting in a type 8 rectangle.

Therefore, there are exactly twelve possible configurations of rectangles created from the sides and horizontal stabs of an orthogonal polygon, as enumerated in Fig. 3. □

Lemma 2. *Every vertex of the polygon is part of exactly three horizontal rectangles.*

Proof. Refer to Fig. 7. There is one horizontal rectangle above and one below every horizontal stab, and one rectangle behind the vertex. □

Corollary 3. *Around each vertex, there is either*:

- *one horizontal rectangle above and two below it, or*
- *two horizontal rectangles above and one below it.*

Proof. Since these are the horizontal rectangles created by the horizontal stabs, these are the only possible configurations. See Fig. 8. □

3. Reconstruction from stab information

In this section, we present an algorithm to reconstruct an unknown orthogonal polygon (i.e., solve the OPR problem) given only its stab information.

From the horizontal stabbing information alone, it is possible to determine the types of rectangles incident upon each vertex and hence whether the vertex must ultimately appear as convex or reflex in the polygon. Next, horizontal and vertical information is used to compute two partial orders, for the x and y coordinates of the vertices, and a final layout of the polygon is performed.

3.1. Identification of rectangles

The previous characterizations are used to form and identify the type of each horizontal rectangle. The identification of all rectangles of type 0 to type 11 will be shown to be an $\mathrm{O}(n)$ step, but identification of type 12 rectangles requires $\mathrm{O}(n \log n)$ time. This classification will subsequently be used to determine the convexity/reflexity of each vertex.

In the algorithm described below, $\mathit{hstab}(v_i)$ returns the side stabbed by the horizontal stab from vertex v_i. Define $\mathit{hstab}(v_i).\mathit{ver}$ to be the first (clockwise) vertex on the vertical side that is stabbed by $\mathit{hstab}(v_i)$. The boolean function $\mathit{IsHoriz}(\overline{v_i v_{i+1}})$ determines whether the side $\overline{v_i v_{i+1}}$ is horizontal or not. A simple modulus calculation can be used to calculate $\mathit{IsHoriz}(\overline{v_i v_{i+1}})$. If v_i is the current polygon vertex, then v_{i+1} and v_{i-1} (modulo n arithmetic) respectively refer to the next and previous vertices of the polygon. Refer to Fig. 3 while reading the descriptions.

Identifying the types 0 through 11 rectangles can be achieved by testing the following conditions for a vertex v_i:

- Type 0: $\mathit{hstab}(v_i) = \infty$ and $\mathit{hstab}(v_{i+1}) = \infty$ AND $\mathit{IsHoriz}(\overline{v_i v_{i+1}})$
- Type 1: $\mathit{hstab}(v_i) = \mathit{hstab}(v_{i+1})$ AND NOT $\mathit{IsHoriz}(\overline{v_i v_{i+1}})$
- Type 2: $\mathit{hstab}(v_i) = \mathit{hstab}(v_{i+2})$
- Type 3: $\mathit{hstab}(v_i) = \mathit{hstab}(v_{i+3})$
- Type 4: $\mathit{hstab}(v_{\mathit{hstab}(v_i).\mathit{ver}}).\mathit{ver} = v_i$ OR $\mathit{hstab}(v_{\mathit{hstab}(v_i).\mathit{ver}-1}).\mathit{ver} = v_{i+1}$
- Type 5: $\mathit{hstab}(v_{\mathit{hstab}(v_i).\mathit{ver}-1}).\mathit{ver} = v_i$ OR $\mathit{hstab}(v_{\mathit{hstab}(v_i).\mathit{ver}+2}).\mathit{ver} = v_{i-1}$
- Type 6: $\mathit{hstab}(v_{\mathit{hstab}(v_i).\mathit{ver}+2}).\mathit{ver} = v_{i-2}$ OR $\mathit{hstab}(v_{\mathit{hstab}(v_i).\mathit{ver}-1}).\mathit{ver} = v_{i+1}$
- Type 7: $\mathit{hstab}(v_i).\mathit{ver} = v_{i+2}$ OR $\mathit{hstab}(v_i).\mathit{ver} = v_{i-3}$
- Type 8: $\mathit{hstab}(v_i).\mathit{ver} = v_{i+3}$ OR $\mathit{hstab}(v_i).\mathit{ver} = v_{i-4}$
- Type 9: $\mathit{hstab}(v_i).\mathit{ver} = \mathit{hstab}(v_{i+1}).\mathit{ver} + 2$ OR $\mathit{hstab}(v_i).\mathit{ver} = \mathit{hstab}(v_{i-1}).\mathit{ver} - 2$
- Type 10: $\mathit{hstab}(v_{\mathit{hstab}(v_i).\mathit{ver}}) = \mathit{hstab}(v_{i+1})$ OR $\mathit{hstab}(v_{\mathit{hstab}(v_i.\mathit{ver}+1)}) = \mathit{hstab}(v_{i-1})$
- Type 11: $\mathit{hstab}(v_{\mathit{hstab}(v_i).\mathit{ver}-1}) = \mathit{hstab}(v_{i+1})$ OR $\mathit{hstab}(v_{\mathit{hstab}(v_i).\mathit{ver}+2}) = \mathit{hstab}(v_{i-1})$

For a vertex v_i, detecting the types 0 through 11 rectangles incident upon v_i requires $\mathrm{O}(1)$ time.

- Type 12: A type 12 rectangle is formed when two pairs of vertices have common horizontal stabs. (That is, $\mathit{hstab}(v_i) = \mathit{hstab}(v_{j+1})$ and $\mathit{hstab}(v_{i+1}) = \mathit{hstab}(v_j)$, and $\mathit{IsHoriz}(\overline{v_i v_{i+1}})$ and $\mathit{IsHoriz}(\overline{v_j v_{j+1}})$). Detecting this type of rectangle requires examining all horizontal stabs to each vertical side. Since it is possible that $\mathrm{O}(n)$ stabs could hit one side (as in Fig. 11, for example), it might appear that this operation could take $\mathrm{O}(n^2)$ time. However, in Section 3.2, a data structure is presented that reduces the overall time needed to identify all the type 12 rectangles to $\mathrm{O}(n \log n)$ time in total.

3.2. Algorithm – determine convex/reflex

The following algorithm determines the convexity of the vertices from the incident horizontal rectangles computed in the previous section. Each horizontal rectangle defined by the sides of the unknown polygon and its horizontal stabs contains two, three or four vertices of the polygon (see Fig. 2 for example). The convexity properties of the involved vertices are not independent. For each vertex, v_i, the algorithm maintains two sets, *same*$[v_i]$ and *opposite*$[v_i]$. Ultimately, all the vertices on a rectangle containing v_i will be included in either *same*$[v_i]$ or *opposite*$[v_i]$. These two sets indicate whether those vertices have the same or opposite convexity as v_i. In the final step, by carefully merging all these sets the algorithm assigns the label *convex* or *reflex* to each vertex.

This stage is described in three parts: classifying types 0 through 11 rectangles and identifying the vertices on each, identifying vertices on type 12 rectangles, and finally determining the convexity of each of the vertices.

3.2.1. *Classify and identify rectangles: types 0 to 11*

This part of the algorithm traverses through the vertices checking each horizontal stab for inclusion as part of any type 0 through 11 rectangles. For each vertex, v_i, the other vertices on the same rectangle are appended to either the *same*$[v_i]$ or *opposite*$[v_i]$ set and the number of rectangles to which it has been assigned are counted.

```
• Initialize: For each vertex $v_i$, $i = 1, 2, \ldots, n$, do:
  – $number\_of\_rectangles[v_i] := 0$.
  – initialize $same[v_i]$ to the empty set.
  – initialize $opposite[v_i]$ to the empty set.
• Classify/Identify: For each vertex $v_i$, $i = 1, 2, \ldots, n$, do:
  – if the conditions (Section 3.1) identifying types 0 to 11 rectangles are met:
        For every vertex, $v_j$, around the rectangle
            increment $number\_of\_rectangles[v_j]$.
        For every pair of vertices on the rectangle:
            update the same or opposite sets appropriately.
```

Analysis: The *initialize* and *classify/identify* loops each use $O(n)$ time. Also, the space used by the routines is bounded by $O(n)$.

Label any vertex that has been assigned to three rectangles as *classified* and the rest as *unclassified*. The next section will use this *classified/unclassified* labelling to identify the type 12 rectangles.

3.2.2. *Identify rectangles: type 12*

Any vertex that is now *unclassified* must be part of some type 12 rectangle. The difficulty is identifying which other stabs are also part of this same rectangle; refer to Fig. 9. The stab s_2 that is on the other end of the horizontal side from s_1 can be identified in constant time using the ordering of the vertices. The two stabs s_3 and s_4 of the same rectangle are more difficult to determine.

It is critical that these type 12 rectangles be computed and processed, to ensure that the *same* and *opposite* sets are complete.

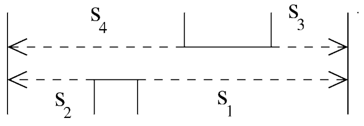

Fig. 9. A type 12 rectangle.

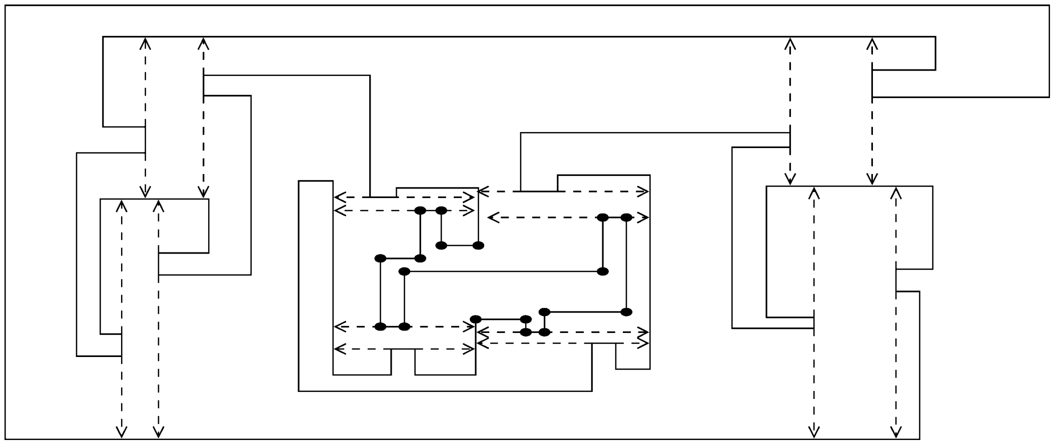

Fig. 10. A polygon with vertices isolated by type 12 rectangles.

Definition 4 (*Isolated group of vertices*). Define an *isolated group of vertices* to be a *proper* subset, K, of all the vertices of an orthogonal polygon such that these vertices appear in each other's *same* and *opposite* sets, but not in the *same* and *opposite* sets of any other vertices. Furthermore, the *same* and *opposite* sets of the vertices of K do not contain any vertices that are not in K.

The necessity of identifying type 12 rectangles is illustrated by the polygon of Fig. 10. If type 12 rectangles are not considered, the marked vertices would be isolated. In the same figure, more vertices would be isolated if we examined the vertical, instead of the horizontal rectangles of this polygon.

Even though the total number of all types of rectangles created by a polygon's horizontal stabs is $O(n)$, there could be $O(n)$ type 12 rectangles. A natural conjecture would be that each of the $O(n)$ vertical sides stabbed by type 12 rectangles, have only a constant number of such stabs. Vertical side, s, on the polygon of Fig. 11 shows that this conjecture is incorrect and a more elaborate procedure is required.

This stage of the algorithm traverses the vertices several times. The first pass initializes counters and binary search trees for each vertical side, while the second pass determines the number of type 12 vertices that stab each vertical side. The third creates a binary tree for each vertical side and matches the vertices on each type 12 rectangle. For every adjacent pair of type 12 vertices (e.g., s_1 s_2 in Fig. 9) one vertex of the pair is included in the binary tree of the vertical side stabbed by the other. When inserting into these binary trees, a vertex to be inserted that already exists in the tree was placed there by the other pair of type 12 vertices that stabbed the same vertical sides. This condition indicates that all four vertices of a type 12 rectangle have been identified.

```
• Initialize: For each vertical side, $\overline{v_i v_{i+1}}$, $i = 2, 4, \ldots, n$, do:
  – unclassified_count[$\overline{v_i v_{i+1}}$] := 0.
  – initialize binary_tree[$\overline{v_i v_{i+1}}$] to empty.
• Count stabs: For each vertex, $v_i$, $i = 1, 2, \ldots, n$, do:
  – If (number_of_rectangles[$v_i$] = 2) increment unclassified_count[stab($v_i$)].
• Create Trees: For each vertex, $v_i$, $i = 1, 2, \ldots, n$, do:
  – if (number_of_rectangles[$v_i$] = 2) AND (number_of_rectangles[$v_{i+1}$] < 3)
        if (unclassified_count[hstab($v_i$)] < unclassified_count[hstab($v_{i+1}$)])
```

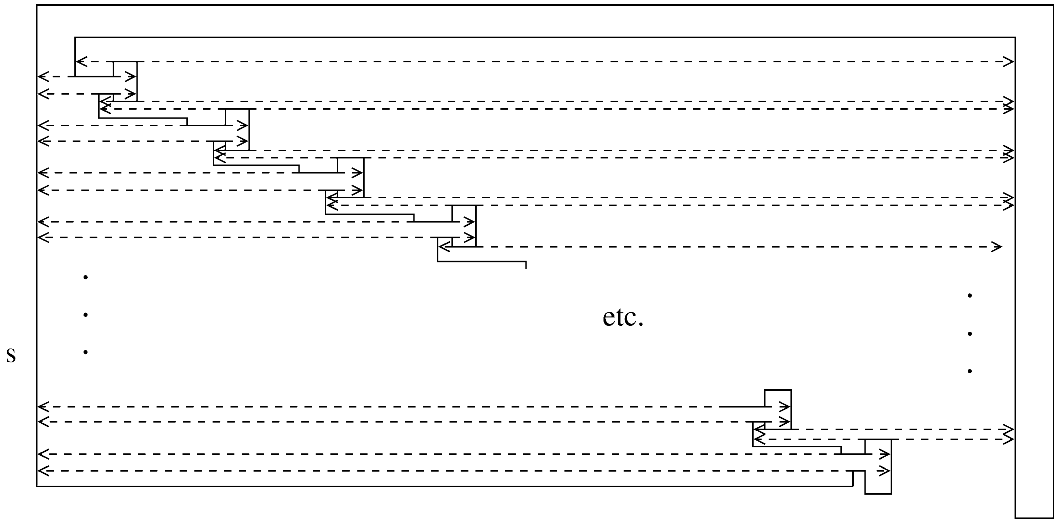

Fig. 11. A polygon with O(n) type 12 stabs to some vertical sides.

```
        /* vertical side $hstab(v_i)$ is the least stabbed of the two sides */
           if ($MEMBER(hstab(v_{i+1}), binary\_tree[hstab(v_i)])$)
    – /* a type 12 rectangle has been found. */
    – For every pair of vertices, $v_j$ and $v_k$, on the rectangle:
    – $INSERT(v_j, same[v_k])$
    – $DELETE(hstab(v_{i+1}), binary\_tree[v_i])$
    else $INSERT(hstab(v_{i+1}), binary\_tree[hstab(v_i)])$.
else /* vertical side $hstab(v_i)$ is NOT the least stabbed of the two */
    – if ($MEMBER(hstab(v_i), binary\_tree[hstab(v_{i+1})])$)
    – /* a type 12 rectangle has been found. */
    – For every pair of vertices, $v_j$ and $v_k$, on the rectangle:
    – $INSERT(v_j, same[v_k])$
    – $DELETE(hstab(v_i), binary\_tree[v_{i+1}])$
    else $INSERT(hstab(v_i), binary\_tree[hstab(v_{i+1})])$.
```

Analysis: The *initialize* and *count stabs* loops are each O(n) loops. The *create trees* loop is an O$(n \log n)$ loop, since it is executed n times, and each binary tree could have O(n) entries in it. (Searching, and inserting into a balanced binary tree of size O(n) requires O$(\log n)$ time.) Thus the overall time needed by this part of the algorithm is O$(n \log n)$. However the space required here is only O(n), since the number of entries in all the binary trees never exceeds n.

The *same* and *opposite* sets created in the algorithm of Section 3.2 are instrumental in solving the OPR problem. By definition, *same*$[v]$ is a set of vertices that share the same convexity of v on some rectangle. By applying the *same* relation on the elements of this set, we obtain a larger set of vertices that all have the same convexity as v. In an analogous fashion, the set of all vertices with the opposite convexity of v can be constructed. It is by repeated application of the *same* and *opposite* relations, that these 2 sets can be extended until all vertices have been partitioned into those with the same or opposite convexity as v. We use an inductive definition and note that no more than n applications are necessary.

Definition 5. Same and Opposite sets:
$\mathit{same}_1[v] = \mathit{same}[v]$.
$\mathit{opposite}_1[v] = \mathit{opposite}[v]$.
For $k > 1$,
$\mathit{same}_k[v] = \{w \mid \exists u \in \mathit{same}_{k-1}[v] \text{ and } w \in \mathit{same}[u]\} \cup \{w \mid \exists u \in \mathit{opposite}_{k-1}[v] \text{ and } w \in \mathit{opposite}[u]\}$.
$\mathit{opposite}_k[v] = \{w \mid \exists u \in \mathit{same}_{k-1}[v] \text{ and } w \in \mathit{opposite}[u]\} \cup \{w \mid \exists u \in \mathit{opposite}_{k-1}[v] \text{ and } w \in \mathit{same}[u]\}$.
$\mathbf{same}^+[\mathbf{v}] = \mathit{same}_n[v]$
$\mathbf{opposite}^+[\mathbf{v}] = \mathit{opposite}_n[v]$.

The following lemma establishes that for an arbitrary vertex v, the two sets $\mathit{same}^+[v]$ and $\mathit{opposite}^+[v]$ partition the vertices appropriately.

Lemma 6. *For any vertex v:*

- $v \cup \mathit{same}^+[v] \cup \mathit{opposite}^+[v] = V$, *and*
- $\mathit{same}^+[v] \cap \mathit{opposite}^+[v] = \phi$.

Proof (*By induction*). *Base case*: Number of vertices is $n = 6$. This is the smallest orthogonal polygon in our problem. Every simple orthogonal polygon with $n = 6$ vertices has exactly one reflex vertex and five convex vertices and is symmetric to the polygon in Fig. 12. Choosing the reflex vertex to be v, all of the other vertices are inserted into $\mathit{opposite}[v]$ in Section 3.2.1. Here $\mathit{opposite}^+[v] = \mathit{opposite}[v]$ and $\mathit{same}^+[v] = \phi$. Alternately, choosing any one of the convex vertices as v, after the algorithm of Section 3.2.1, $\mathit{same}[v]$ will contain at least two of the convex vertices and $\mathit{same}^+[v]$ will contain all of the convex vertices except v. $\mathit{Opposite}[v] = \mathit{opposite}^+[v] = \{\text{the reflex vertex}\}$. Thus establishing the conditions for $n = 6$.

Induction: Assume Lemma 6 is true for polygons of size k.

Let $n = k + 2$. The n vertex polygon can be reduced to a polygon of size k by removing one *orthogonal ear* as shown in Fig. 13. An orthogonal polygon always has at least two orthogonal ears as shown by O'Rourke [7, p. 47]. This follows from the fact that every orthogonal polygon is convexly quadrilateralizable [6] and that the dual of this quadrilateralization will be a tree when the polygon has no holes [7, Lemma 3.9]. Every tree has at least two nodes of degree one, or leaf nodes. These two leaf nodes correspond to quadrilaterals of the convex quadrilateralization that are one of the forms of Fig. 14 [7, Lemma 3.16]. We reduce our $n = k + 2$ polygon to a polygon of size k by removing one of these two ears using a cut along the stab from the reflex vertex of this polygon that goes through

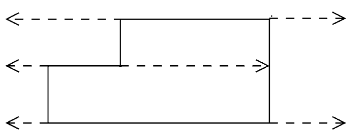

Fig. 12. An orthogonal polygon with 6 sides.

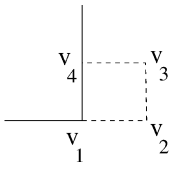

Fig. 13. Creating an $n = k + 2$ orthogonal polygon.

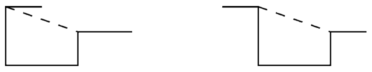

Fig. 14. Two quadrilaterals: correspond to leaf nodes of dual of quadrilaterization.

the quadrilateral, as shown in Fig. 13. Removal of one ear will result in the loss of three vertices and the inclusion of one new vertex, for an overall reduction in the polygon size of two vertices. Assume the vertex, v, of Lemma 6 on the polygon of size k was not vertex v_1 of Fig. 13. By the induction hypothesis we have $v \cup \textit{same}^+[v] \cup \textit{opposite}^+[v] = V$ and $\textit{same}^+[v] \cap \textit{opposite}^+[v] = \phi$. The algorithm determine *convex/reflex* of Section 3.2 will place vertices v_2 and v_3 of Fig. 13 in the same set ($\textit{same}^+[v]$ or $\textit{opposite}^+[v]$) of the $n = k + 2$ polygon that contained v_1 of the polygon of size k, and v_4 in the other set.

If vertex v, of Lemma 6 on the polygon of size k was vertex v_1 of Fig. 13, we must assume that vertex v of the $n = k + 2$ polygon is one of v_2, v_3 or v_4 of Fig. 13. If $v = v_1$ (respectively $v = v_2$), the determine *convex/reflex* algorithm of Section 3.2 would result in an identical $\textit{same}^+[v]$ for the $n = k + 2$ polygon as for the $n = k$ polygon, with the addition of v_3 (respectively v_2) and the loss of v_1 and an identical $\textit{opposite}^+[v]$ with the addition of v_4. A similar argument could be made if $v = v_4$ except that the $\textit{same}^+[v]$ set of the polygon of size k would be switched with the $\textit{opposite}^+[v]$ set of the $n = k + 2$ polygon and vice versa. Again, the induction hypothesis is used to ensure that $v \cup \textit{same}^+[v] \cup \textit{opposite}^+[v] = V$ and $\textit{same}^+[v] \cap \textit{opposite}^+[v] = \phi$. □

3.3. Determine convexity of vertices

The next stage of the algorithm starts with any vertex, v, that has a stab to infinity, marks it as *convex*, and initializes a queue (called *to_be_done*) with this vertex. Then a loop is created that dequeues a vertex, v, from the front of the queue, marks the vertices in $\textit{same}[v]$ with the same convexity as v, and those in $\textit{opposite}[v]$ as opposite to v. For each of these vertices, if they were not previously marked, they are enqueued to the back of the queue. The loop continues until the *to_be_done* queue is empty.

```
• initialize $to\_be\_done$ to be an $EMPTY$ queue.
• $Initialize$: For each vertex, $v_i$, $i = 1, 2, \ldots, n$, do:
  – $has\_been\_queued[v_i] := FALSE$
  – if $(hstab(v_i) = \infty)$ and $(to\_be\_done = EMPTY)$
              $ENQUEUE(v_i, to\_be\_done)$
              $has\_been\_queued[v_i] := TRUE$
              $vertex[v_i] := convex$
• $Determine\ Convexity$: While $(to\_be\_done \neq EMPTY)$ do:
  – $v_i := DEQUEUE(to\_be\_done)$
  – for every vertex, $v_j$, in $same[v_i]$ do:
          if $NOT(has\_been\_queued[v_j])$
              $ENQUEUE(v_j, to\_be\_done)$
              $has\_been\_queued[v_j] := TRUE$
```

```
            if ($vertex[v_i] = convex$) then $vertex[v_j] := convex$
            else $vertex[v_j] := reflex$
  – for every vertex, $v_j$, in $opposite[v_i]$ do:
            if $NOT(has\_been\_queued[v_j])$
                $ENQUEUE(v_j, to\_be\_done)$
                $has\_been\_queued[v_j] := TRUE$
            if ($vertex[v_i] = convex$) then $vertex[v_j] := reflex$
            else $vertex[v_j] := convex$
```

Analysis: The *initialize* loop runs in $\mathrm{O}(n)$ time. The *determine convexity* loop is executed exactly n times, as the loop does not terminate until the queue is empty, which will only happen when all are marked. Each vertex is put onto the queue once, and pulled off once, and each iteration requires constant time. Therefore, this entire stage of the algorithm uses $\mathrm{O}(n)$ time.

Lemma 7. *The algorithm of Section 3.3 correctly determines the convexity of the vertices of the polygon.*

Proof. The vertices with stabs to infinity are clearly convex vertices. The algorithm simply identifies all the vertices in the $same^+[v]$ set as a vertex, v, with a stab to infinity as convex, and all the vertices in v's $opposite^+[v]$ as reflex. Lemma 6 ensures that no vertex will be both convex and reflex, and no vertices are missed. □

Thus, the time required to determine the convexity of the n vertices is dominated by the $\mathrm{O}(n \log n)$ step needed to identify the type 12 rectangles; and the space requirement is $\mathrm{O}(n)$.

3.4. Algorithm – embed polygon

The final stage in the reconstruction process is to obtain an actual embedding of the orthogonal polygon. The algorithm creates two lists, representing the relationships between the x and y coordinates of all vertices. One list $\{x_{\min}, \ldots, x_{\max}\}$ represents the x coordinates of each of the vertical sides, the other list $\{y_{\min}, \ldots, y_{\max}\}$ represents the y coordinates of the horizontal sides. These two lists will be created in such a way that placing the sides on the x and y coordinates, the result will be an orthogonal polygon. The two lists cannot be established uniquely, since it is not possible to determine the relationships between the stabs on opposite sides of a segment. Placing all vertices on these coordinates, however, does reconstruct an orthogonal polygon that respects the input information.

(1) Find the four extreme segments, that is, segments with both horizontal and vertical stabs to infinity. The two vertical segments must be located at $x_{\min}$ and $x_{\max}$, while the two horizontal ones must be at $y_{\min}$ and $y_{\max}$. Start with a horizontal extreme segment, assign it to $y_{\min}$, then follow through the polygon order, assigning the other three extreme segments to $x_{\min}$, $y_{\max}$ and $x_{\max}$. These assignments will lay out the polygon so that its vertices are in clockwise order. $\mathrm{O}(n)$ time is needed.
(2) The segment that runs horizontally along $y_{\min}$ is laid out from right to left. Call this a *left* segment. The next segment on the polygon, a vertical segment, must be an *up* segment and the corner between the two must be a convex corner.

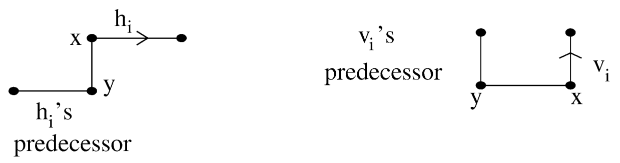

Fig. 15. Predecessor segments and preceding vertices.

Name the sides of the polygon $h_1, v_1, h_2, v_2, \ldots, h_{n/2}, v_{n/2}$ around the polygon. For a horizontal (respectively vertical) segment, define its *predecessor segment* to be the horizontal (respectively vertical) segment immediately before it. (h_i's predecessor is h_{i-1}, and v_j's predecessor is v_{j-1}.) On any segment, vertical or horizontal, define its *two preceding vertices* to be the two vertices between h_i and h_{i-1} or between v_i and v_{i-1}. Fig. 15 shows a horizontal segment and a vertical segment and their respective predecessor segments. In each case, vertices x and y are the preceding vertices to the segment.

For each of the remaining segments on the polygon, if the preceding vertices have the same convexity, the segment must be oriented opposite to its predecessor segment in the same dimension. If the preceding vertices have opposite convexity, the segment is oriented the same as its predecessor segment in the same dimension. In this way, assign *up*/*down*, *left*/*right* to each segment of the polygon. Again, this step uses a total of $O(n)$ time and space.

(3) Create two digraphs, X and Y (representing the two partial orders of the sides of P), with a node in the X graph for each vertical side, and a node in the Y graph for each horizontal side. Add arcs as follows:
- On the X graph, direct arcs from the node corresponding to the $x_{\min}$ side to every other node, and from all nodes to the node corresponding to the $x_{\max}$ side.
- For every *right* (respectively *left*) segment in the polygon, add an arc in the X digraph from the node corresponding to the side containing the first (respectively second) endpoint to the node corresponding to the side containing the second (respectively first) endpoint.
- For every stab to a *right* (respectively *left*) segment add an arc from the node corresponding to the side containing the first (respectively second) endpoint of the stabbed segment, to the node corresponding to the side containing the endpoints of the stabbing segment, and another from the node corresponding to the side containing the endpoints of the stabbing segment to the node corresponding to the side containing the second (respectively first) endpoint of the stabbed segment.

The X digraph contains less than $5n/2 - 2$ arcs. The arcs for the Y digraph are created in a similar fashion, substituting *up* for *right* and *down* for *left*. Creating the two digraphs is easily accomplished in a total of $O(n)$ time and space.

(4) Topologically sort each digraph to order the nodes from minimum to maximum in $O(n)$ time.

(5) Assign x and y integer track numbers (unique integers chosen from the range $[1, \ldots, n/2]$) to the nodes indexed according to the previous topological sorts. Follow around the polygon laying each vertex on its respective tracks, putting a segment between each pair of consecutive vertices. The resulting orthogonal polygon respects the given stabs and has no collinear sides. This step also uses $O(n)$ time.

Thus the layout algorithm uses $\mathrm{O}(n)$ time and space. This stage was inspired by the final stage of an algorithm by Booth and O'Rourke that realizes labeled edge visibility trees [7]. In that work, Booth and O'Rourke have established the correctness of the method.

Theorem 8. *The OPR problem can be solved in* $\mathrm{O}(n \log n)$ *time.*

Proof. The three main stages of the algorithm have:

- Identified the types of rectangles created by the sides and horizontal stabs of an orthogonal polygon. Lemma 6 ensures this can be done. The algorithm requires $\mathrm{O}(n \log n)$ time.
- Determined the convexity of each of the vertices. Lemma 7 shows the algorithm is correct. The algorithm requires $\mathrm{O}(n)$ time.
- Embedded the polygon in the plane. This algorithm uses the same technique as Booth and O'Rourke used in their work on Edge Visibility Trees. The algorithm used $\mathrm{O}(n)$ time. □

4. The collinear sides assumption

The previous algorithms assume that the orthogonal polygon has no collinear sides. Allowing collinear sides is a natural extension of this problem.

Lemma 1 of Section 2 showed that rectangles of types 0 through 12 (Fig. 3) were the only possible rectangles created from the sides of an orthogonal polygon and its horizontal stabs. If collinear sides were possible, the number of rectangle types would increase. Cases 0, 1 and most of case 2 of that proof, did not rely on this assumption, so those cases contain the complete set of rectangles. However, part of case 2, and cases 3 and 4 both used this assumption, so the additional rectangles of Fig. 16 would be introduced. Algorithmically, checking each of these additional types of rectangles is no more difficult than checking each of types 0 through 11.

It is possible, however, that along each *stab* of each rectangle any number of collinear sides could exist, as shown in Fig. 17. Rechecking each rectangle type, allowing for the possibilities of such sides complicates the process. An algorithm that allows collinear sides but runs in $\mathrm{O}(n^2)$ time is presented in [5].

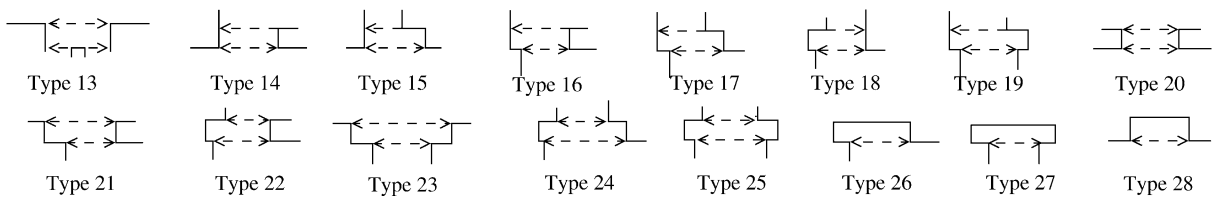

Fig. 16. Extra rectangles possible with collinear sides.

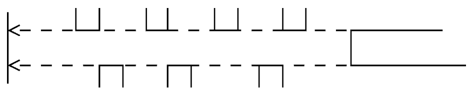

Fig. 17. Collinear sides along the *stabs* of a type 1 rectangle.

5. Conclusion

An algorithm was presented that reconstructs an orthogonal polygon when only the horizontal and vertical stabs of its vertices are known. The algorithm runs in $O(n \log n)$ time if the polygon has no collinear sides. An interesting open problem is to reduce the time to classify type 12 rectangles (and hence the overall algorithm) to $o(n \log n)$, or alternately to prove a lower bound of $\Omega(n \log n)$.

Acknowledgements

Thanks to Tom Shermer, for pointing out some errors in an earlier draft.

References

[1] H. Everett, D.G. Corneil, Negative results on characterizing visibility graphs, Computational Geometry 5 (1995) 51–63.

[2] H. Everett, F. Hurtado, M. Noy, Stabbing information of a simple polygon, Discrete Appl. Math. 91 (1–3) (1999) 67–82.

[3] H. Everett, A. Lubiw, J. O'Rourke, Recovery of convex hulls from external visibility graphs, in: Proc. Fifth Canadian Conference on Computational Geometry, 1993, pp. 309–314.

[4] H. Everett, Visibility graph recognition, PhD thesis, University of Toronto, Department of Computer Science, January 1990.

[5] L. Jackson, Polygon reconstruction from visibility information, Master's thesis, University of Lethbridge (TR-CS01-96), April 1996.

[6] J. Kahn, M. Klawe, D. Kleitman, Traditional galleries require fewer watchmen, SIAM J. Alg. Disc. Meth. 4 (1983) 194–206.

[7] J. O'Rourke, Art Gallery Theorems and Algorithms, Oxford University Press, New York, 1987.

[8] J. O'Rourke, I. Streinu, The vertex-edge visibility graph of a polygon, Computational Geometry 10 (1998) 105–124.

[9] J. Snoeyink, Cross-ratios and angles determine a polygon, in: Proc. ACM 14th Symposium on Computational Geometry, 1998, pp. 49–57.

[10] R. Tamassia, I.G. Tollis, A unified approach to visibility representations of planar graphs, Discrete Comput. Geom. 1 (4) (1986) 321–341.

[11] S.K. Wismath, Characterizing bar line-of-sight graphs, in: Proc. First Annual ACM Symposium on Computational Geometry, 1985, pp. 147–152.

[12] S.K. Wismath, Point and line segment reconstruction from visibility information, Internat. J. Comput. Geom. Appl. 10 (2) (2000) 189–200.