Orthogonal Polygon Reconstruction*

L. Jackson[†] S. K. Wismath[‡]

April 23, 1996

1 Introduction

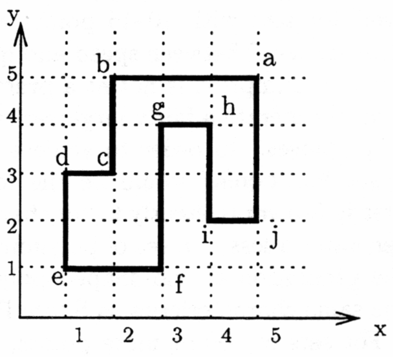
In the field of visibility research, the *Reconstruction problem* rebuilds an (unknown) object from (primarily) visibility information. In the literature, various objects have been considered (e.g., points, line segments, polygons) and a variety of visibility models suggested [Eve90] [O'R87] [Wis85]. In general, visibility information alone is not sufficient to reconstruct an object and frequently extra information is provided [CL91] [Wis94].

This paper presents an algorithm that reconstructs an (unknown) orthogonal polygon in $O(n \log n)$ time from pure visibility information - the "stab visibilities" of the vertices of the polygon. Each vertex v of a simple polygon P has two *stabs*, namely the first side of P encountered by extending the two sides of P at v. For an orthogonal polygon (aligned with the X and Y axes) the stabs are horizontal and vertical. Both interior and exterior visibilities are provided - "∞" will be used to denote a stab that encounters no side of P.

Given an orthogonal polygon P, the *construction* of this stab information can be computed in $O(n \log n)$ time via a straightforward line sweep algorithm. The *reconstruction* problem (OPR) considered here is to obtain an orthogonal polygon that is consistent with given stab information. The sides of the polygon are specified in order, and referred to as the Hamiltonian cycle. See figure 1 for an example of the given input (stabs along each *side*) and resulting reconstructed polygon. We assume that the stab information represents a simple orthogonal polygon with $n > 4$ vertices

Side	Orientation	Stab	
ab	horizontal	∞	∞
bc	vertical	∞	ef
cd	horizontal	fg	∞
de	vertical	∞	∞
ef	horizontal	∞	∞
fg	vertical	∞	ab
gh	horizontal	bc	ja
hi	vertical	ab	∞
ij	horizontal	fg	∞
ja	vertical	∞	∞

OPR Input



OPR Output

Figure 1: An Example of the OPR Problem

*This research was supported by N.S.E.R.C.

[†]Dept. of Math. and Computer Science, University of Lethbridge, and Dept. Electronics Eng. Tech., Lethbridge Community College, Alberta, Canada

[‡]Dept. of Math. and Computer Science, University of Lethbridge, Alberta, Canada

and that no pair of sides are collinear. (Collinearities can be accommodated, but at the expense of a more detailed exposition[Jac96].)

The first stage of the algorithm identifies each vertex as either CONVEX or REFLEX. Only the horizontal stab information is required, so **stab(v)** will be used to denote the *horizontal* stab at v. (This stage is equivalent to determining whether the stab is interior or exterior to the polygon.)

The second stage of the algorithm computes two orders (for 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 given input.

2 Determining Convexity

In the final reconstructed polygon P, the horizontal stabs must partition the plane into a collection of rectangles - see figure 2. Depending on how the stabs hit the sides of the polygon, twelve different

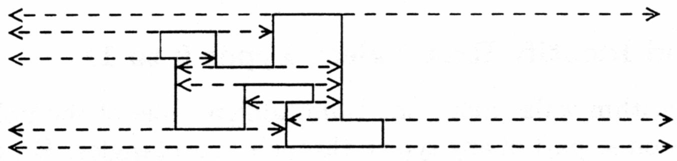


Figure 2: A Reconstructed Orthogonal Polygon with Horizontal Stabs

types of rectangles, as enumerated in figure 3, are possible, ignoring horizontal and vertically symmetric situations. It is assumed that those rectangles with stabs to infinity are completed by a pseudo side at infinity. This stage of the algorithm organizes the (horizontal) stabs into rectangles, and based on the type, identifies the convexity of the vertices.

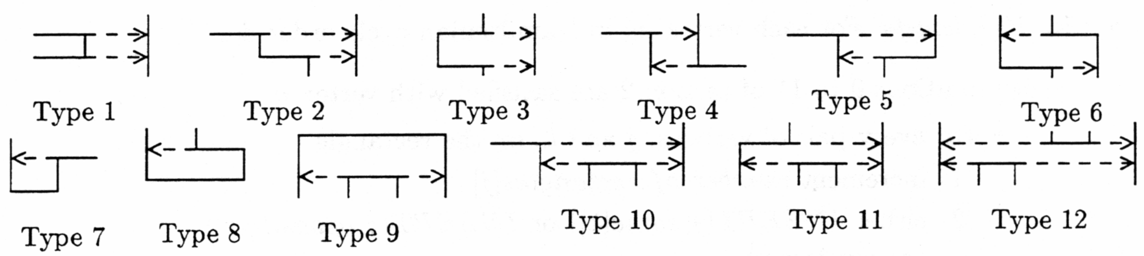


Figure 3: The Twelve Possible Horizontal Rectangles

Notice that every stab is part of exactly two rectangles, one above and one below it. For the previously stated assumption of no collinear sides, there exists a unique top most side and a unique bottom most side, each of which partially bound degenerate rectangles. These will be called type 0 rectangles and are easily identifiable.

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: The proof to this lemma is a combinatorial proof that considers the number of corners of the rectangles that correspond to the vertices of the polygon. See [Jac96] for details. □

Identification of each of the types 0 through 11 rectangles requires checking constant time conditions. The list of conditions is lengthy, but straightforward and is omitted; two examples of the conditions are given below:

- A type 1 rectangle is identified when the horizontal stab of one vertex i and its neighbour on the Hamiltonian cycle $i+1$ are both to the same side. Thus i and $i+1$ must have the same convexity.

- A type 3 rectangle is identified when the stab of one vertex i and the stab of vertex $i+3$ are to the same side, and vertices $i+1$ and $i+2$ form a vertical side. Thus, vertex $i+3$ must have the same convexity as i, but $i+1$ and $i+2$ must have the opposite convexity of i.

A type 12 rectangle is detectable when two pairs of vertices have common stabs. (That is, $stab[i] = stab[j+1]$ and $stab[i+1] = stab[j]$, assuming vertices (i and $i+1$) and (j and $j+1$) form horizontal sides). Detecting this type of rectangle will require examining all horizontal stabs to each vertical side. Since it is possible that $O(n)$ stabs could hit one side, for example, it might appear that this operation could take $O(n^2)$ time. However, in section 2.2, a data structure is presented that reduces the overall time needed to identify all type 12 rectangles to $O(n \log n)$ time in total.

2.1 Classify and Identify Rectangles: Types 0 to 11

This part of the algorithm walks along the Hamiltonian cycle of the polygon, checking each horizontal stab for inclusion as part of any type 0 through 11 rectangles. For each vertex, v, append the other vertices on the same rectangle to either its *same*$[v]$ or *opposite*$[v]$ set, and count the number of rectangles to which it has been assigned.

- *Initialize*: For each vertex v, in Hamiltonian cycle order do:

  1. $number\_of\_rectangles[v] := 0$.
  2. initialize *same*$[v]$ to the empty set.
  3. initialize *opposite*$[v]$ to the empty set.

- *Classify/Identify*: For each vertex, v, in Hamiltonian cycle order do:

  - if conditions 0 to 11 of section 2 are satisfied with vertex v:
    - For every pair of vertices, j and k, on the rectangle
      1. increment $number\_of\_rectangles[j]$
      2. either $INSERT(j, same[k])$ or $INSERT(j, opposite[k])$ appropriately[1]

Analysis: $O(n)$ time and space is used. Note that every vertex is part of exactly three horizontal rectangles, each of which contains from two to four vertices of the polygon, the *same* and *opposite* sets for each vertex will together contain no more than twelve vertices.

Now, 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.

2.2 Identify Rectangles: Type 12

A type 12 rectangle could be on either the inside or the outside of the polygon; there could be $O(n)$ type 12 rectangles. 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 figure 4. The stab s_2 that is on the other end of the horizontal side from s_1 can be identified in constant

[1]This is easily determined from figure 3

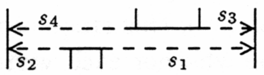


Figure 4: A Type 12 Rectangle

time, simply by looking at the stab of the next vertex on the Hamiltonian cycle. However, the two stabs s_3 and s_4 of the same rectangle are more difficult to find.

For every adjacent pair of type 12 vertices (e.g., s_1 s_2 in figure 4) one vertex of the pair is included in a binary search 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, s, in Hamiltonian cycle order do:
    – $unclassified\_count[s] := 0$.
    – initialize $binary\_tree[s]$ to empty.
• Count stabs: For each vertex, v, in Hamiltonian cycle order do:
    – If $(number\_of\_rectangles[v] = 2)$ increment $unclassified\_count[stab[v]]$.
• Create Trees: For each vertex, v, in Hamiltonian cycle order do:
    – if $(number\_of\_rectangles[v] = 2)$ AND $(number\_of\_rectangles[v+1] = 2)$
        * if $(unclassified\_count[stab[v]] < unclassified\_count[stab[v+1]])$
            · $least\_stabbed := stab[v]$ ; $most\_stabbed := stab[v+1]$
        * else
            · $least\_stabbed := stab[v+1]$ ; $most\_stabbed := stab[v]$
    – if $(MEMBER(most\_stabbed, binary\_tree[least\_stabbed]))$
        * /* a type 12 rectangle has been found. */
        * For every pair of vertices, j and k, on the rectangle:
          $INSERT(j, same[k])$
        * $DELETE(most\_stabbed, binary\_tree[v])$
    – else $INSERT(most\_stabbed, binary\_tree[least\_stabbed])$.
```

Analysis: The *initialize* and *count stabs* loops are each $O(n)$ loops. The *create trees* loop is an $O(n \log n)$ loop. So, 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.

2.3 Determine Convexity of Vertices

This 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 $same[v]$ as the same convexity as v, and those in $opposite[v]$ as opposite to v. For each of these vertices, if they were not previously marked, enqueue them to the back of the queue. The loop continues until the *to_be_done* queue is empty. The algorithm is straightforward and the pseudo-code is omitted.

Analysis: Each vertex is put onto the queue once, and pulled off once and a constant amount of work is done for each vertex. Therefore, this stage of the algorithm uses $O(n)$ time.

Thus, the time used to determine whether each vertex of an orthogonal polygon of n vertices is *CONVEX* or *REFLEX* is dominated by the $O(n \log n)$ needed to identify the type 12 rectangles. The space requirement is only $O(n)$.

3 Layout Algorithm

In this section, an efficient algorithm is presented to reconstruct an orthogonal polygon from its stabs and Hamiltonian cycle, after the convexity of the vertices is established. This algorithm creates two lists, representing the relationships between the x and y coordinates of all vertices. One list $\{x_{min}, ...x_{max}\}$ represents the x coordinates of each of the vertical sides, the other list $\{y_{min}, ...y_{max}\}$ represents the y coordinates of the horizontal sides. The lists are not unique since it is not possible to determine the relationships between the stabs on opposite sides of any boundary segment.

1. Find the four segments with both horizontal and vertical stabs to infinity. The two vertical ones must be located at $x_{min} = 1$ and $x_{max}n/2$, while the two horizontal ones must be at $y_{min} = 1$ and $y_{max} = n/2$. Completion of this step requires $O(n)$ time.

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 Hamiltonian cycle, a vertical segment, must be an *up* segment, and the corner between the two must be a convex corner.

   Name the sides of the polygon h_1, v_1, h_2, v_2, ... $h_{n/2}$, $v_{n/2}$ along the Hamiltonian cycle. For a horizontal (respectively vertical) segment, define its **predecessor segment** to be the horizontal (respectively vertical) segment immediately before it on the Hamiltonian cycle. (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}. Figure 5 shows a horizontal segment and a vertical segment and their respective

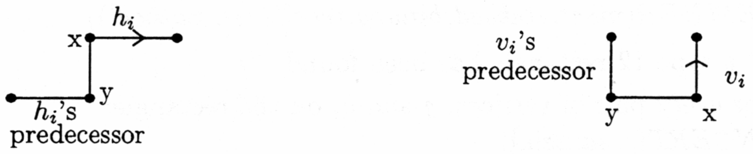


Figure 5: Predecessor Segments and Preceding Vertices

   predecessor segments. In each case, vertices x and y are the preceding vertices to the segment.

   For each of the remaining segments, in the cycle, if the preceding vertices have the same convexity, the segment must be opposite its predecessor segment, in the same dimension. If the preceding vertices have opposite convexity, the segment is 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 the x_{max} side.

- For every *right* (respectively *left*) segment in the polygon, put 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 put 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 $5n/2 - 2 = O(n)$ 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 uses 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 $[1..n/2]$) to the nodes indexed according to the previous topological sorts. Follow through the Hamiltonian cycle laying each vertex on its respective tracks, putting a segment between each pair of consecutive vertices. The resulting orthogonal polygon respects the given Hamiltonian cycle and stabs and has no collinear sides. This step, also uses $O(n)$ time.

Thus the layout algorithm uses $O(n)$ time and space.

4 Conclusion

Overall, the algorithm reconstructs an orthogonal polygon in $O(n \log n)$ time and $O(n)$ space. An interesting open problem is to reduce the time to classify type 12 rectangles (and hence the overall algorithm) to $O(n)$, or alternately to show a lower bound of $\Omega(n \log n)$.

References

[CL91] C. Coullard and A. Lubiw. Distance visibility graphs. In *Proceedings of the seventh Annual ACM Symposium on Computational Geometry*, pages 289–296, 1991.

[Eve90] H. Everett. *Visibility Graph Recognition.* PhD thesis, University of Toronto, Department of Computer Science, January 1990.

[Jac96] L. Jackson. Polygon reconstruction from visibility information. Master's thesis, University Lethbridge, April 1996.

[O'R87] J. O'Rourke. *Art Gallery Theorems and Algorithms.* Oxford University Press, New York, NY, 1987.

[Wis85] S. K. Wismath. Characterizing bar line-of-sight graphs. In *Proceedings of the first Annual ACM Symposium on Computational Geometry*, pages 147–152, 1985.

[Wis94] S. K. Wismath. Reconstruction of parallel line segments from endpoint visibility information. In *Proceedings sixth Canadian Conference on Computational Geometry*, pages 369–373, 1994.