On the Notion of *Completeness* for Reconstruction Algorithms on Visibility Graphs

(Extended Abstract)

G. Srinivasaraghavan[*] Asish Mukhopadhyay[†]

1 Introduction

The *visibility graph* of a scene is a graph with a vertex for every object of the scene, and an edge between two objects if the two are visible to each other. There have been two main directions in the study of visibility graphs—one algorithmic in nature and the other graph theoretic. In the first, one wants to compute the visibility graph of a given scene efficiently. In the second, one studies graphs which are realizable, as combinatorial objects in their own right. One looks for properties which characterize such graphs, algorithms that can identify such a graph efficiently, algorithms that can reconstruct a scene which is a realization of a given graph etc..

This is a speculative note on what we call the *Completeness* criterion for reconstruction algorithms on visibility graphs. This notion of completeness for reconstruction algorithms we feel is a highly desirable feature and possibly necessary too, for a deep understanding of the combinatorial nature of visibility. In this note, we introduce the notion and illustrate it with an algorithm which reconstructs a *horizontally convex orthogonal polygon* (henceforth denoted as HCOP) from its orthogonal edge visibility graph consisting of a pair of trees, one of which is a caterpillar. We also sketch how the algorithm can be used as a 'skeleton' to arrive at an algorithm which reconstructs such a polygon realizing a *weighted* pair of trees. Our algorithm is a *complete* version of the algorithm for the same problem by O'Rourke [1].

The rest of this note is organized as follows. In section 2 we introduce the notion of *Completeness*. In section 3 we review briefly the algorithm of O'Rourke [1], which reconstructs a HCOP from a pair of trees, one of which is a caterpillar. We give our 'complete' version of the algorithm in section 4 and follow it with an algorithm to carry out the reconstruction from a pair of weighted trees in section 5. We end with some concluding remarks in section 6. We omit most of the details and proofs in this abstract. Full details can be found in [2].

2 The Notion of Completeness

A visibility graph is a combinatorial representation of a scene. From the way visibility graphs are defined it is evident that given a scene and a particular notion of visibility, the corresponding visibility graph is unique. On the other hand in reducing the scene to a graph, quite a bit of information (mostly metric information) is lost. Thus usually,

[*]The Institute of Mathematical Sciences, C.I.T Campus, Madras-600113, India. Email: gsr@imsc.ernet.in

[†]Department of Computer Science and Engineering, Indian Institute of Technology, Kanpur-208016, India. Email: am@iitk.ernet.in

many different scenes end up having the same visibility graph. Typically a reconstruction algorithm can potentially produce a whole class of scenes all of which have the same visibility graph (the given one). However it appears that the possibility of reconstruction algorithms which can potentially produce *every* scene with the given graph as the visibility graph has not been investigated in the literature on visibility graphs so far. An algorithm *complete* in this sense, would we feel throw much more light on the relationship between scenes and their visibility graphs in general. Also it is likely that such 'complete' algorithms, would pave the way to algorithms which handle 'weighted' visibility graphs, under some suitable interpretation of the weights. In this paper we introduce this notion of 'completeness' of a reconstruction algorithm with a complete algorithm, though in a restricted sense, for reconstructing a horizontally convex polygon from its visibility graph.

3 Reconstructing HCOPs

The visibility graph of a given orthogonal polygon P consists of a vertex e for each edge e^* of P with two vertices e and f of G being adjacent if the two corresponding edges e^* and f^* are such that there exists a rectangle of non-zero area, wholly contained in the interior of P with two opposite sides of the rectangle on e^* and f^* (e^* and f^* are then said to *see* or *be visible to* each other). O'Rourke [1, Lemma 7.3] has shown that the visibility graph of a simple orthogonal polygon consists of two disconnected trees, one each for the edges parallel to one of the axes (horizontal and vertical).

A HCOP is an orthogonal polygon which intersects any horizontal line in a single connected segment. We henceforth refer to simple HCOPs in general position (those in which no two vertices can be joined by a horizontal or a vertical segment lying wholly in the interior of the polygon) simply as 'polygons'.

A caterpillar is a tree which has a path—called the *spine*—such that every vertex of the tree not on the path is a leaf. It was also shown in [1] that the visibility graph of a HCOP consists of a pair of trees having the same number of vertices, one of which (the component corresponding to the vertical edges of P) is a *caterpillar* and that for every pair of trees one of which is a caterpillar, there exists an orthogonal polygon (not necessarily horizontally convex) whose visibility graph is the given pair. O'Rourke [1] has also given an algorithm to reconstruct a HCOP from such a pair of trees. This algorithm is clearly not complete because it can produce only HCOPs. In fact it is incomplete in a stronger sense; it cannot even produce all the possible HCOPs from the given pair of trees. Here we modify the algorithm in [1] to remove the latter incompleteness. Henceforth we use the term 'visibility graph' to mean an orthogonal edge visibility graph.

We use the labels T and C to refer to the vertical and the horizontal visibility trees of our polygon. Note that C is a caterpillar. If $T \cup C$ is the visibility graph of a polygon P, then we denote the edge of P corresponding to the vertex v of the visibility graph as, v^* and vice-versa.

3.1 Review of the known algorithm

We now give a brief description of the reconstruction algorithm in [1]. The algorithm attempts to reconstruct a polygon from a tree T and a caterpillar C. Any tree has a unique bipartition and so has a caterpillar. It would be convenient for us to refer to the partitions of C as the *left* and the *right* partitions, having l and r vertices respectively. Suppose T has an edge e which admits an embedding of T which maps e to a vertical line segment, and embeds $l-1$ vertices of T to the left of e and $r-1$

to the right. It is easily shown that then there exists a polygon with T and C as its vertical and horizontal visibility graphs respectively. A polygon obtained this way is called an *hourglass* polygon and the two trees C and T are said to *balance* each other. To obtain a joint realization of an arbitrary pair of trees C and T, a subtree C' of C is balanced with a subtree T' of T to get an hourglass polygon and the remaining vertices $C' - C$ and $T' - T$ are gathered in an isolated region by 'sliding' them to the left or right. The process is repeated with the "leftover" diminishing at each step resulting in a series of hourglass polygons, horizontally displaced and connected top to bottom, which jointly realize C and T. We describe the algorithm in a little more detail below.

Let the bipartition of C be (l_1, r_1) with $l_1 + r_1 = n$ and $l_1 - r_1 = \delta \geq 0$, n being the number of vertices in C. Choose an arbitrary vertex b_1 of T as the "base". Choose an edge e_1 of T incident on b_1 such that the remaining vertices of T may be arranged on either side of e_1, with L_1 vertices to the left and R_1 to the right (note that then $L_1 + R_1 + 2 = n$), such that the quantity $\mid (L_1 - R_1) - \delta_1 \mid$ is minimal among all edges e_1 incident on b_1 and all arrangements of vertices. One can now show that a subtree at t_1 (the other end of e_1) containing Δ_1 vertices can be 'slid' off to start another hourglass polygon and the remainder can be balanced with a part of C into an hourglass polygon. The 'slide' is illustrated in Figure 1. The reader is referred to O'Rourke[1] for the details. It suffices to say that the procedure does converge (Δ_1 strictly decreases each time) and produces the required polygon.

4 The Complete Reconstruction Algorithm

Our algorithm is a variant of the algorithm outlined above. In the description of our algorithms, we concentrate only on how a subtree descendent of t_1 is 'slid' on one side, since as it can be observed, the problem 'repeats' itself after this.

Our algorithm is based on the following observation, which is the crux of O'Rourke's algorithm: b, t and the subtree descendent g which is to be 'slid' must be such that the remaining subtrees at b and t can be arranged in such a way as to make the number of vertices embedded to the left (right) of $e = (b, t)$ in the realization, at most $l - 1$ $(r - 1)$. Of course we must be able to do this in each of the subproblems we generate during the course of the algorithm. That the above condition is sufficient for reconstructing a HCOP, is easy to see. Conversely, given a polygon P, the vertices of its visibility graph corresponding to its bottommost edge b^*, the farthest visible edge t^* of P to b^* and, the edge g^* which can see t^* and some other edge of P at a higher ordinate than t^*, clearly form such a set of vertices. Note that the horizontal convexity of P ensures uniqueness of g^*, if it exists. So any HCOP can be reconstructed from its visibility trees using a scheme which chooses b, t and g as above.

Thus the following is the generic reconstruction algorithm for HCOPs. Note from our observations above, that any algorithm which reconstructs a HCOP from a pair of trees can be rewritten to fit into this framework.

Algorithm Reconstruct

1 Choose a vertex b to correspond to the bottommost horizontal edge of the reconstructed polygon.

2 Choose a vertex t adjacent to b such that it would correspond to the vertex visible to but farthest from b in the reconstructed polygon.

3 Partition the subtrees of T at b and t, so that the partitions can be arranged on either side of the edge connecting t and b in such a way that the number of vertices embedded on the left and right of the edge are less than $(l - 1)$ and

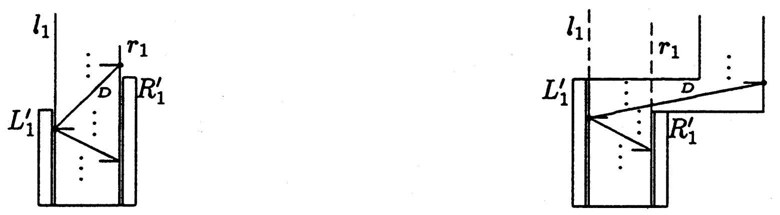

Figure 1: A 'slide'.

$(r-1)$ respectively. The subtree to be 'slid' to a side is also identified along with this.

4 Balance the caterpillar with T as far as is possible till a subtree of t is 'slid' away to a side. This starts off a new hourglass, and the above steps repeat.

End. (of Algorithm Reconstruct)

In our algorithm, we first choose an arbitrary edge e of T in place of steps 1 and 2 of the above generic algorithm. We will eventually make one of the end-vertices of e correspond to the bottommost edge b^* of P and the other to t^*. The following lemma shows that the above assignments and the partition required in step 3 can be computed efficiently for any choice of the edge e.

Lemma 4.1: *Let S be a finite set of positive integers and let l and r be any two positive integers such that $l + r = \sum S$, where $\sum S$ refers to the sum of the integers in S. Then there exist subsets L and R of S such that $\sum L < l$, $\sum R < r$ and $\mid L \mid + \mid R \mid +1 = \mid S \mid$ ($\mid A \mid$ of a set A denotes the number of elements in A).*

Proof: The following algorithm computes L and R.

Algorithm Partition

```
1 Initialize $L \leftarrow \phi$, $R \leftarrow S$ and $\delta \leftarrow \sum S - r$.
2 If $\delta < 0$ then STOP. The current $L$ and $R$ are
  the required subsets.
3 Otherwise $L \leftarrow L \cup \{i\}$ if $i \leq \delta$, $R \leftarrow R - \{i\}$
  and $\delta = \delta - i$, where $i$ is some arbitrary element
  of $R$.
  Go to Step 2.
End. (of Algorithm Partition)
```
□

Clearly every possible partition of S into L and R can be obtained by the above scheme. Moreover since the partitioning algorithm makes no assumptions about the set S to be partitioned, it is clear that the reconstruction based on the above partitioning scheme, can be carried out over all the phases of the reconstruction. We thus have a complete reconstruction algorithm for HCOPs. In the next section we use the generic algorithm to derive a reconstruction algorithm which produces such a polygon from a pair of weighted trees.

5 Weighted Reconstruction

Here we give an algorithm which reconstructs a HCOP from a pair of *weighted* trees, one of which is a caterpillar. The algorithm will report a failure if a joint-weighted-realization does not exist. The interpretation of the weights we use here is that a pair of edges see each other with weight W if the distance (vertical distance if the two edges are horizontal and the horizontal distance if they are vertical) between the two edges is W. In accordance with the above interpretation, we assume that the weights assigned to the edges of the trees are positive reals. We denote the weight of an edge (x, y)

of either tree as $W(x, y)$.

We will now see how each of the steps in the generic algorithm can be carried out for the weighted case.

Steps 1 and 2: Choices of b and t.

This choice cannot now be arbitrary for a pair of weighted trees. We will now examine the choice of b.

We first observe that since b^* is the bottommost horizontal edge of the reconstructed polygon b must satisfy the following conditions:

1. b can not be a leaf.
2. Suppose $b = v_0v_1v_2 \ldots v_k$ is a maximal path of the vertical visibility graph of P such that $v_1 \neq t$, t^* being the edge of P visible to b^* and farthest from it. Then $W(v_{i-1}, v_i) > W(v_i, v_{i+1})$, $1 \leq i < k$.
3. There must exist a unique path $b = u_0, u_1, u_2, \ldots, u_k$ from b such that $W(u_{i-1}, u_i) > W(u_i, u_{i+1})$ whenever i is odd and $W(u_{i-1}, u_i) < W(u_i, u_{i+1})$ whenever i is even, for $i \leq 1 < k$ (we assume henceforth that k is as large as possible).

Note that in condition (3) $u_1 = t$ and the subtree of T at t, to be slid away, is the one containing u_2. Also if a b satisfying the above conditions exists then there are exactly two choices for b viz., u_0 and u_k. Searching for a vertex of the kind described above can be done in time linear in the size of the graph.

Step 3: Partitioning the subtrees at b and t and the reconstruction of P.

We start the reconstruction from b and an extreme edge of the spine of C (recall the observation we made at the end of section 2). We will see below that the reconstruction proceeds uniquely at every step once the choice of the edge of C to start with, has been made. Thus we can try a reconstruction starting from one of the extreme edges, say e_c of the spine of C. If this fails then we try with the other extreme edge. Also without loss of generality let the leaf vertex of e_c be in the right partition of the vertices of C.

Suppose P is the polygon realizing T and C jointly, with b^* being the bottommost edge and t^* the farthest visible edge to b^*. For any two horizontal edges p^* and q^* of P such that p^* is visible to t^* and q^* to b^*, if both the edges are on the same side of some line segment in the interior of P joining b^* and t^* then clearly, $W(t, p) + W(b, q)$ must be strictly less than $W(b, t)$, where $W(x, y)$ denotes the vertical distance between the edges x^* and y^*. Similarly for two edges p^* and q^*, both of which are visible to b^*, it must be that thres$(p) > W(b, q)$ or vice-versa, where thres$(x) = W(b, x) - \max\{W(x, a) | a \neq b$ and $a^*$is visible to $x^*\}$. Now let thres(p) for a horizontal edge p^* which is visible to t^*, be defined as $W(b, t) - W(t, p)$. It is now easy to verify that the length of the smaller vertical edge adjacent to b^* is simply the minimum of thres(p) over all horizontal edges $\bar{p}^*$, other than t^* and b^*, which can see either t^* or b^*.

We now define thres(p) for the vertices of T which are adjacent to either b or t. The definition is exactly the same as that in the above paragraph, with 'visibility' substituted by 'adjacency'. To start the reconstruction, we take b^* as the bottommost edge. The length of b^* is clearly the weight of e_c. Since the leaf vertex of e_c is on the right, the subtree of T containing p for which thres(p) is minimum, will also be embedded on the right. The edges of C incident on the non-leaf vertex v of e_c can now be easily matched with the vertices of this subtree to get a partial realization. In this the only condition that the weights on the edges of C need to satisfy is that if two vertices x and y adjacent to v are made to correspond to counterclockwise consecutive vertical edges in that order, of the partially realized polygon, then $W(v,x) < W(v,y)$ ($W(v,x) < W(v,y)$) if the distance from b to z on T is even (odd), where z^* is the horizontal edge between x^* and y^*. Of course to make sure that the visibility between b^* and t^* is not affected, we keep track of the maximal connected portion of b^* which is currently visible from a point at $y = \infty$. The weights on the edges of C must be such that this portion is not obstructed.

To complete the algorithm, we only need to say how the construction would proceed when the vertices on the subtrees of T that have been embedded so far are not enough to match the vertices of C for a realization. The subtree of T to be 'taken in' now for carrying out the construction is the one with the vertex p for which thres(p) is the smallest among the remaining subtrees. To see why this must necessarily be so, imagine that the base b^* has been moved up to the current height (upto which the partial realization has taken place) with the edges of the polygon obtained so far deleted. Imagine removing the corresponding vertices also from the two trees. It is now as if we start the reconstruction afresh with only the current vertex of C and b active. Clearly the 'side' of (b,t) on which a new subtree of T needs to be brought in, is the one having the leaf-vertex of the extreme edge of the truncated caterpillar. Thus the subtree to be taken in, is defined uniquely at every step of the algorithm. We now know how the reconstruction can be carried out till a subtree at t is to be 'slid' to one side. This can clearly be carried on to eventually obtain a joint-weighted-realization or report a failure. Our choices at every step having been unique, a failure anywhere will imply unrealizability of the two trees for the particular starting edge C we used. We thus have an algorithm to reconstruct a HCOP from a pair of weighted trees.

6 Concluding Remarks

In this paper we have introduced and discussed the notion of *completeness* for reconstruction algorithms on visibility graphs. We believe that algorithms complete in this sense throw a lot more light on the combinatorial nature of visibility than do algorithms which are not complete. Though the example we have chosen for illustration is a rather simple one, it seems likely that the concept will prove to be a powerful one in the study of visibility graphs, besides possibly providing a framework for the study of weighted visibility graphs.

References

[1] Joseph O'Rourke. *Art Gallery Theorems and Algorithms.* Oxford University Press, New York, 1987.

[2] G. Srinivasaraghavan On Some Visibility and Intersection Problems in Computational Geometry Ph.D. thesis, Department of Computer Science and Engineering, Indian Institute of Technology, Kanpur, India, 1992.