Dealing with degeneracies and numerical imprecisions when computing visibility graphs

RIVIÈRE Stéphane *

March 18, 1996

1 Introduction

Most algorithms in computational geometry assume that the input is in "general position", that is that no degenerate cases occur (such as three aligned points , or three convergent lines ...). But in practice, when the size of the input becomes significant, degenerate configurations occur and programs fail. Things get worse when the program must face imprecise computations.

Some general tools have been developed to cope with these problems. For solving degeneracies symbolic perturbation schemes are employed (e.g. [EM90], [Yap90]) : coordinates of input are symbolically perturbed so that degeneracies are removed. Other techniques such as the epsilon geometry ([GSS89]) do the actual geometric computations but consider that the results have an imprecision up to ε and decisions made on these results take this imprecision into account. Finally techniques are developed to do exact computations (e.g. [FV93]) and avoid imprecision problems.

These general techniques can be used as "black boxes" for geometric computation in algorithms. This has the advantage that it can be applied to a large range of algorithms, but these black boxes have in general an extra cost which can reduce the performance of the program.

We consider here two algorithms for computing visibility graphs and study how to apply the idea of symbolic perturbation, not for black boxes creation, but for direct modifications of the algorithm, making it robust.

2 The simplicity paradigm applied to visibility

General techniques cope with degenerate cases (e.g. aligned points) by simulating a non degenerate configuration by symbolically perturbing the coordinates of the input by a quantity $f(\varepsilon)$. Symbolically means that, when computing the sign of a determinant e.g. for these points, calculation is not done with the perturbed coordinates, but symbolic calculus shows that the sign of the perturbed determinant is the one of a cofactor of the actual (null) determinant.

When dealing with visibility, perturbations must be done wisely. A degenerate situation, where three objects are aligned, is the limit between two situations : the one where all three objects can see each others, and the one where extremal objects are hidden from each other (see figure 1).

When encountering a degenerate configuration, we want to simulate the simplest situation, that is the situation where extremal objects do not see each other, so that only visibility edges $(1, 2)$ and $(2, 3)$ are computed.

We show how these ideas are applied to two sweep algorithms for computing visibility graphs.

*iMAGIS-IMAG - BP 53
38041 GRENOBLE CEDEX 09 - FRANCE
e-mail : Stephane.Riviere@imag.fr
iMAGIS is a joint project of CNRS/INRIA/INPG/UJF

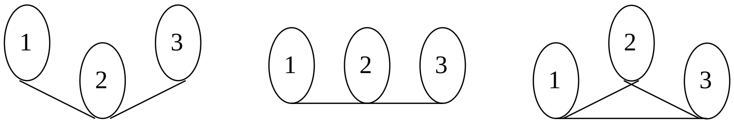

Figure 1: Visibility configurations around a degeneracy

3 Visibility graph of polygonal scenes

We study here the optimal algorithm of [Riv95] for computing visibility graphs of polygonal scenes.

We recall here the important points of the algorithm. The sweep of the visibility graph is done using two graphs : the upper and lower rotation trees. Given a direction θ, for each vertex p of the scene we associate the first vertex seen when looking forward (resp. backward) along θ and turning counterclockwise (see figure 2a). Rotation trees are subgraphs of the visibility graph and are used to sweep it by making θ vary from θ_0 to $\theta_0 + \pi$. Modifications are discrete : at each step, an edge (p, q) is passed (and reported by the program) and the new edge (p, q') (resp. (p', q)) of the upper (resp. lower) rotation tree is computed.

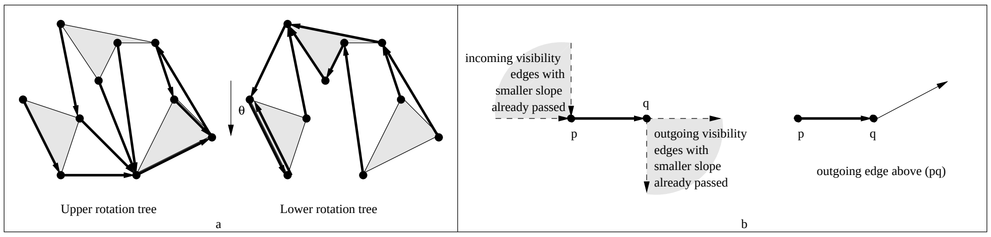

Figure 2: Sweeping the visibility graph with rotation trees

Considerations about visibility coherence show that an edge (p, q) can be passed and updated efficiently if all the visibility edges incoming at p (resp. outgoing from q) with smaller slope have already been passed.

This condition is met if the edge is in both rotation trees and if the outgoing edge of q (resp. of p) in the upper (resp. lower) rotation tree is strictly above (p, q).

Degeneracies may stop the algorithm, since an edge aligned with another can become definitively non-passable. If the "strictly above" relation is loosened to "above" then visibility coherence is lost, wrong decisions are made and generally the program aborts.

The key to cope with this problem is to consider a degenerate configuration as the corresponding non degenerate one where the extremal points are hidden from each other, as shown in figure 3.

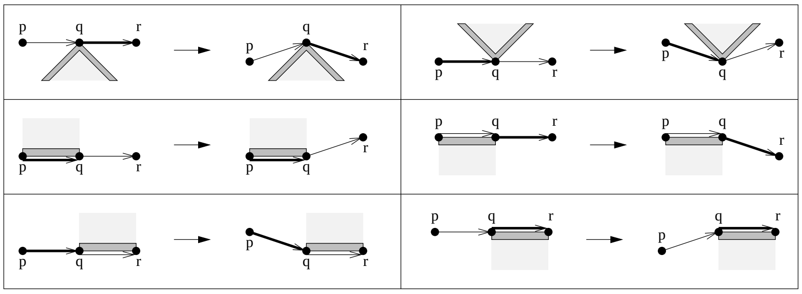

Figure 3: Simulating non degenerate situations

.

The test "strictly above" is then completed to handle degenerate cases. We just have to add two tests to verify that $[pq]$ is a side of a polygon and one test two check if an edge is strictly above another one. The rest of the program is unchanged.

4 Visibility graph of convex curved objects

We study here the algorithm of [PV93a] for computing visibility graphs of curved objects, applied to scenes of convex objects (the recent algorithm of [PV95] is optimal, but more involved and not implemented). We recall the important points of the algorithm (for more details on implementation see [Riv93]). Given a direction θ, the tangents to the objects of direction θ divide the space in regions where visibility along θ is constant. An auxiliary graph is constructed : its vertices are the tangents and two tangents are linked by an edge if they bound a same region (see figure 4 a); The edge is then associated to this region. The graph is modified when θ becomes the slope of a free bitangent of the scene (see figure 4 b) : a region shrinks to the bitangent and disappears, and a new region is created.

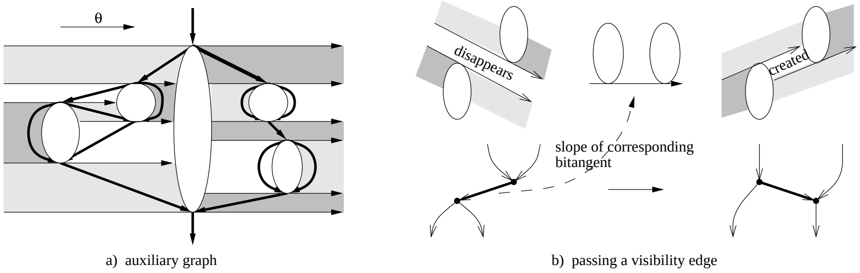

Figure 4: Auxiliary graph for computing visibility graph

The algorithm maintains the auxiliary graph from θ_0 to $\theta_0 + \pi$ reporting all the free bitangents met. Each edge has an associated "date of death" which is the slope of the bitangent causing the associated region to disappear. After initialization, edges are put in a priority queue according to their date of death. At each step, the minimum edge is taken from the queue, the graph is (locally) updated, new dates of death are computed and the priority queue is updated.

The calculation of bitangents is done numerically, and the computed slopes are accurate up to an imprecision of ε (see figure 5a).

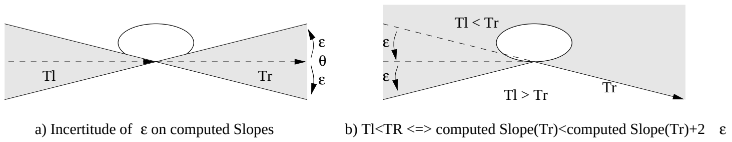

Figure 5: Numerical imprecision

This has important consequences when the program encounters a (nearly) degenerate configuration : it cannot order the bitangents of the three objects correctly according to one of the two possible schemes (see figure 6). The two schemes are mixed and the program must abort.

To cope with this problem, local comparisons are made between adjacent bitangents before putting them in the priority queue. First, the imprecision ε is taken into account when comparing the slope of two consecutive bitangents (see figure 5 b). Then a bitangent is put into the priority queue only if its slope is smaller than the one of its left and right adjacent bitangents. The modifications made when passing a bitangent involve more elements of the auxiliary graph, but these modifications remain local, and still concern only a constant number of elements.

These modifications allow us to handle degenerate cases and to cope with numerical imprecisions.

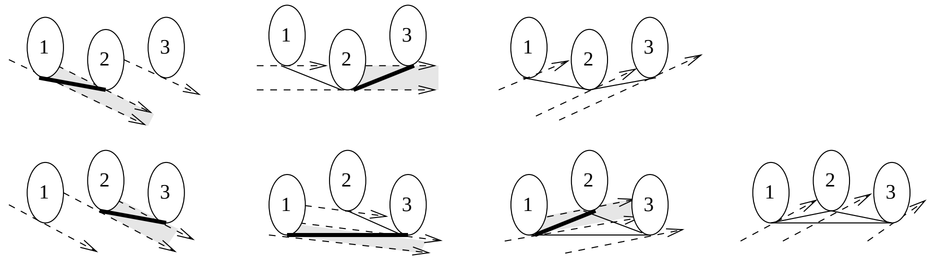

Figure 6: Sweeping schemes concerning a triplet of objects

5 Conclusion

We have shown how the idea of simulating simplicity led us to modify two algorithms for computing visibility graph so that they can handle degenerates scenes.

The advantage of studying actual modifications on applying black boxes is that in general performance is less affected. Modifications may appear to be minor ones with very little extra cost as seen with the algorithm for polygonal scenes. This study seems also to show that algorithms with topological sweep seem to be more easy to modify than algorithms with straight sweep, for which global order is important.

These algorithms are used to construct the visibility complex, a data structure encoding visibility relations between objects in the plane (see [PV93b] for more details). This data structure is then used for answering visibility queries, such as computing a view around a point in the scene. The corresponding programs face the same sort of problems when encountering degeneracies or numerical imprecisions. The principles we followed for computing visibility graphs also apply in the more general frame of visibility computation for allowing easy modifications of the programs, and making them robust and efficient.

References

[EM90] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: a technique to cope with degenerate c ases in geometric algorithms. *ACM Trans. Graph.*, 9:66–104, 1990.

[FV93] S. Fortune and C. J. Van Wyk. Efficient exact arithmetic for computational geometry. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 163–172, 1993.

[GSS89] L. J. Guibas, D. Salesin, and J. Stolfi. Epsilon geometry: building robust algorithms from imprecise com putations. In *Proc. 5th Annu. ACM Sympos. Comput. Geom.*, pages 208–217, 1989.

[PV93a] M. Pocchiola and G. Vegter. Sweep algorithm for visibility graphs of curved obstacles. manuscrit, 1993.

[PV93b] M. Pocchiola and G. Vegter. The visibility complex. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 328–337, 1993.

[PV95] M. Pocchiola and G. Vegter. Computing the visibility graph via pseudo-triangulation. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, 1995.

[Riv93] S. Rivière. Experimental comparison of two algorithms for computing visibility graphs. manuscrit, 1993.

[Riv95] S. Rivière. Topologically sweeping the visibility complex of polygonal scenes. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages C36–C37, 1995.

[Yap90] C. K. Yap. A geometric consistency theorem for a symbolic perturbation scheme. *J. Comput. Syst. Sci.*, 40:2–18, 1990.