Extensible Point Location Algorithm

Rashmi Sundareswara and Paul Schrater
Department of Computer Science and Engineering
University of Minnesota, Twin Cities, USA
sundares@cs.umn.edu, schrater@umn.edu

Abstract

We present a general walk-through point location algorithm for use with general polyhedron lattices and polygonal meshes assuming the usage of nothing more than a simple linked list as a data structure to store the polyhedra. The generality of the approach stems from using barycentric coordinates to extract local information about the location of the query point that allows a 'gradient descent'-like walk toward the goal.

1 Introduction

Many areas of computer graphics and geometric modelling benefit from the the ability to rapidly vary the level-of-detail of a mesh [1, 11], combine meshes, or automatically reconstruct meshes from point data [1, 12]. For example, real-time transfer and display of 3D models between networked computers can benefit from progressive incremental reconstruction from the vertices, using methods like Delaunay Triangulation [1]. If such a method is employed, then one need not send connectivity information along with the point positions. Any culling of the shape is also done in real-time. Such an online computation of the shape, also known as progressive coding [1], shows the user increments of the original intended shape, so that the user may interact with the whatever shape that is presented. In most of these applications, specialized data structures are sent across along with the new points to be inserted, but the need arises for combining meshes that have no data structures which relate them to each other, in which case we have to locate the points dynamically, with no initial information given about a point. This need arises especially when one wants to combine different meshes generated by software packages.

This paper presents a novel method to locate a point in a triangulation, or in any subdivision or combination of many such subdivisions (see figure 1), using barycentric coordinates. The paper assumes a very simple data structure - a simple linked list - to store the polyhedra. Each polyhedron in the list has pointers to its neighbors that are constructed when the polyhedron is split. Having this simple data structure circumvents the problem of having a search tree for point search, and consequently, allows for easier transfer of polygon-data between sites. This algorithm can be generalized to any subdivision or combination of subdivisions. This is desired especially when combining meshes of different software rendering packages, many of which are optimized such that they don't always output triangles if the region is planar. The disadvantage of having just this simple data structure is a slower point search. A review of existing methods is done in the following section.

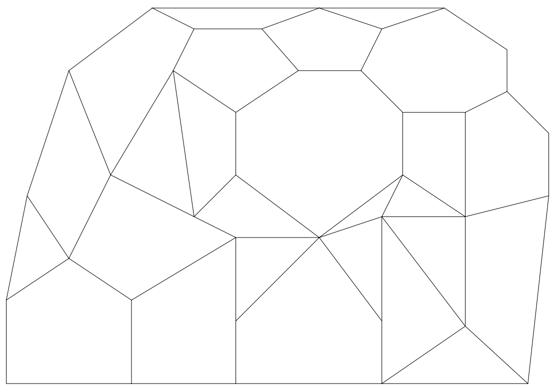

Figure 1. A mesh containing polygons of different sizes

2 Previous Work

Point location methods can be classified into three main categories:

1. Divide and Conquer, Plane Sweep and Trapezoidal Maps [7, 4]: The complexity of these algorithms is $O(log(n))$ per point, but some of its limitations include knowing all the points before hand and the need

of a specialized data structure, for point location, in addition to one needed to store the triangulation. Also, these algorithms would be good choices if the programmer intends for them to be used locally without the intention of combining them with other triangulations.

2. Directed Acyclic Graphs (DAG): The complexity of the point-location step per point is $O(log(n))$. This would be a good algorithm to choose if one does not know the positions of all points beforehand. However the $O(log(n))$ complexity is achieved at the cost of introducing a specialized data structure, in addition to one storing the current valid triangulation. Among DAG algorithms, [4] constructs a DAG, where each node corresponds to a triangle; when the triangle is subdivided or flipped, the node gets children corresponding to newly created triangles. The current valid triangulation is therefore in the leaves and the starting large triangle in the root. Location of the inserted point in this data structure can be done in $(O(log(n))$ expected and $O(n)$ worst time. If the order of insertion of the points is randomized, the tree is nearly balanced so the possibility of the worst case is low.

3. Walk-through algorithms: The complexity of these point-location algorithms is $O(\sqrt(n))$. In these methods, there is no need of a special extra data structure such as this DAG. All that is needed is the existing data structure that contains the triangles (or any subdivision). The only requirement that the algorithms adhere to is the support of $O(1)$ time access between any two neighboring triangles. This is done by having pointers from each triangle to its neighbors. This is easily constructed when the triangle is formed from its parent triangle. Each time, a new point needs to be located, the algorithm walks through the triangles (starting from a random triangle) until it finds the triangle containing the point. The decision for crossing over to a neighboring triangle is determined by:

   (a) Segment intersection test [13, 5]: Here the line segment is drawn from the query point to a good starting point, then the walk starts off from the starting triangle to query point by only crossing over the triangles intersected by the segment.

   (b) Counter Clockwise Wise Search (CCW) [Guibas and Stolfi] [8, 9]. This is essentially the same as the Segment Intersection test, but with less computation, which shows, because the path taken ends up being circuitous (see figure 2). Here we compute the determinant of an edge with respect to to the query point and compare that with the determinant with respect to the opposite vertex for that edge. If the two determinants are opposite is sign with respect to each other, then we cross over the edge. The disadvantage with this algorithm is the ambiguity that results when the query point is in a region of two intersecting half-planes. The reason for the ambiguity is because there are two candidate edges which equally good for crossing over (see figure 3). For both such edges, the query point lies on the opposite side of the edge as the query point. This accounts for the meandering in the overall path.

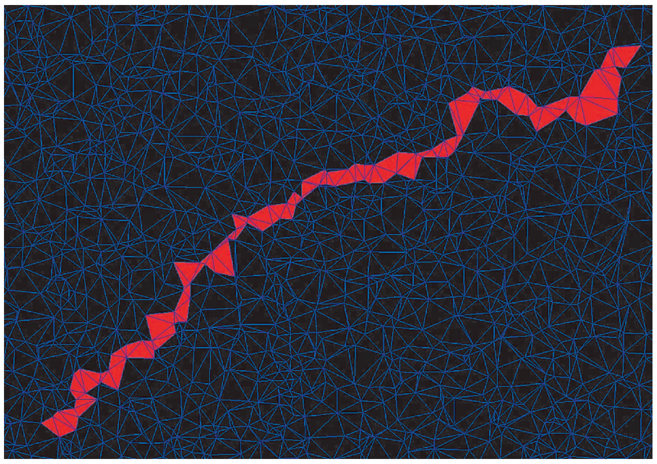

Figure 2. Path taken by CCW. Point is located in the last triangle, in the upper right corner.

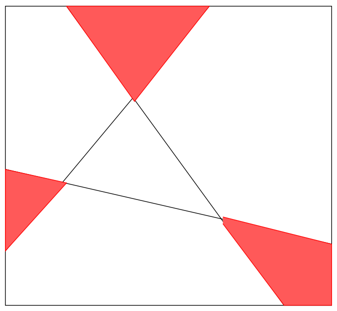

Figure 3. The gray regions are regions of ambiguity for the CCW test.

   (c) [14] provides a variant of the walk through algorithm, where query sites are bucketed, similar to bucketing algorithms by [2]. But this uses a special data structure of maintenance of the buckets.

What our algorithm provides is the advantage of the walk-through algorithms, but with the benefit that it is easily extensible to heterogenous polygonizations, i.e. this algorithm will easily adapt to a mesh containing different n-sided polygons (see figure1). The complexity of our algorithms is the same ($O(\sqrt{(n)})$ per point)the walking algorithms described above. This is because we are inherently bound by the connectivity the the triangle mesh allows us. For proof of this, refer to [5, 6, 13]. However, our algorithm does provide a slight reduction in the constant of complexity bound in case of triangular meshes.

In this paper, we compare our algorithms's performance with the Guibas and Stolfi's [9] and Mucke's [13] algorithms. We ran our tests on triangle meshes from 10k to 100k triangles. At the end of the paper, we include a discussion on the extensibility of the algorithm.

3 Theory

Walk-through methods solve point localization by making a series of decisions to move into adjacent triangles judged closer to the query point. The key element in walk-through algorithms is the decision step. Previous decision step have been quite closely tied to triangulations in a plane. We propose a very general decision step that allows extensions of walk-through point location to arbitrary polygonizations.

The decision problem involves determining which of the adjacent polygons is "closest" to the query point, terminating when the current polygon contains the query point. The difficulty is in rapidly establishing the distance from polygons to the query point based solely on the local information: the current polygons coordinates and the query point coordinates.

One solution is to treat the points in the current polygon as an affine basis for the mesh, and compute the location of the query point with respect to this basis. By computing the barycentric (affine) coordinates [3, 10, 15] of the vertices of adjacent polygons it is possible to quickly find the polygon closer to the query point. The idea relies on the following results:

Given at least $m + 1$ distinct points $A_0, A_1, \dots, A_{m+1}$ in $\Re^m$, a query point p can be represented as a weighted combination of these points,

$$p = \sum_{i=0}^{m} \alpha_i A_i = A_j + \sum_{i=0, i\neq j}^{m} \alpha_i (A_i - A_j)$$

where the vector of weights α are the *barycentric* coordinates of the point p, and $\sum_{i=0}^{m} \alpha_i = 1$ (see figure 4). If and only if all the coordinates $0 < \alpha_i < 1$ then the point lies within the polygon [3, 10].

Barycentric coordinates can be computed as a matrix division problem by setting up the matrix equation:

$$\begin{bmatrix} A_0 & A_1 & \dots & A_{m+n} \\ 1 & 1 & \dots & 1 \end{bmatrix} \vec{\alpha} = \begin{bmatrix} p \\ 1 \end{bmatrix}$$

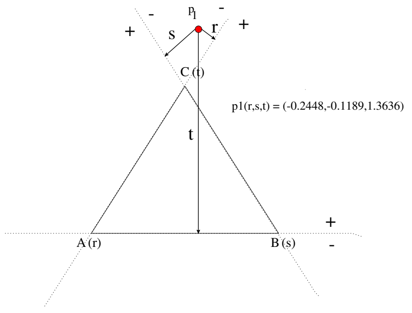

Figure 4. Barycentric division: regions of intersections of halfplanes in the triangle, shown with directionality

which have the form:

$$Q\vec{\alpha} = P$$

and solution:

$$\vec{\alpha} = Q^{-p} P$$

where $Q^{-p} = Q^{-1}$ if the number of vertices is $m + 1$, and $Q^{-p} = (Q^T Q)^{-1} Q^T$ if the number of vertices is greater than $m + 1$.

Note that this representation is origin-free (any polygon vertex can be thought of as the origin, yet the barycentric coordinates will be the same). In addition, the relative magnitude of the barycentric coordinates gives important information about the location of the current polygon's vertices with respect to the query point: larger coordinates have vertices closer to the query point.

This leads to a very general idea for walk-through point-localization in n-dimensional meshes:

- Given the current polyhedron, choose at least $n + 1$ points in general array from that polyhedron and/or its neighbors.

- Compute the barycentric coordinates of the query point.

- Pick the vertices corresponding to the n highest coordinates, and move into the adjacent polyhedron that shares those vertices.

- Terminate when all the barycentric coordinates are positive.

For example, for a planar mesh, walk across the edge defined by the vertices corresponding to the highest two barycentric coordinates. For 3-D meshes (e.g. tetrahedronal lattices), walk across the triangle formed from the largest 3 barycentric coordinates.

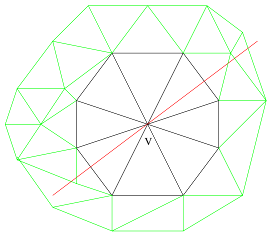

Figure 5. Difficulty with intersection test: All triangles sharing v are considered as being intersected by the line segment.

# Triangles	Seg. Inter.(secs)	CCW (secs)	BC(secs)
10000	2.765	2.515	2.391
20000	6.46	5.687	5.391
40000	15.484	13.313	12.594
60000	26.328	22.329	20.953
80000	38.719	32.656	30.453
100000	52.25	43.688	40.844

Table 1. Comparison of three Walking algorithms:Segment Intersection Test, Counter-Clockwise Test and the Barycentric Test.

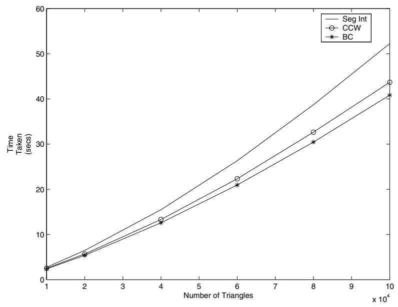

Figure 6. Plots of the time taken for the three Walking Algorithms.

4 Implementation

We implemented this general idea for the case of planar triangulations. Our algorithm, starts [1]off at a random triangle and walks toward the triangle(s) containing the query point, using the following strategy:

1. Compute the Barycentric Coordinates (r, s, t) of query point P with respect to the current triangle (A, B, C).

2. Cross over the edge defined by the two largest barycentric coordinates.

3. Stop once all the barycentric coordinates are positive

For the triangle case, this is equivalent to crossing over the edge *opposite* the smallest barycentric coordinate, *iff* the coordinate is negative (otherwise terminate).

When we first begin step1, we compute the barycentric coordinates of the first triangle with respect to the the query point. We then cross over to the neighboring triangle bordering the edge that represents the lowest of the barycentric coordinates.

In doing this sort of cross-over, we are inherently walking in the direction of the steepest descent from the current triangle to the point: We choose the edge whose opposite vertex's representative barycentric coordinate is the lowest, because it tells us that the query point is on the other side of this edge. If there are two negative barycentric coordinates, we choose the one that is smaller. By the sum-to-1 property of barycentric coordinates, all coordinates can never be negative.

5 Results

We compare our work with two techniques, one by Guibas and Stolfi [8],(CCW Test) and the other by Mucke [13] (Segment Intersection Test). The advantage that our algorithm provides over the Guibas and Stolfi method is the unambiguity regarding which edge to traverse when the query point is located in an ambiguous region with respect to an edge of the current triangle.

This problem is eliminated when using the Mucke algorithm. But in our implementation of the Mucke algorithm, we had to add an extra determinant test to make sure that we weren't looping around a local triangulation around a vertex v, when the segment from the starting point to the query point intersected v (see figure 5).

This extra test is not needed when all the triangles in the mesh are consistently oriented. The advantage of our method over Mucke, is that our algorithm can implemented without taking care of how the mesh is oriented. This is especially important when combining many meshes of different orientations (and of different sizes). We implemented the Mucke algorithm by assuming no specific orientation. All tests were run on a Pentium 433 dual Hertz processor. As table 1 and figure 6 show, our algorithm performed comparably with the other algorithms. In fact, ours performed a constant factor better than both algorithms. As mentioned earlier in our Theory section, all three algorithms' complexity is $O(\sqrt(n))$. Please note that this constant factor would be different if the mesh were of a differ-

[1]Instead of starting at a random triangle, one can compute a good starting triangle by following step1 of Mucke [13]. Since doesn't affect the core of this algorithm, we have proceeded without it

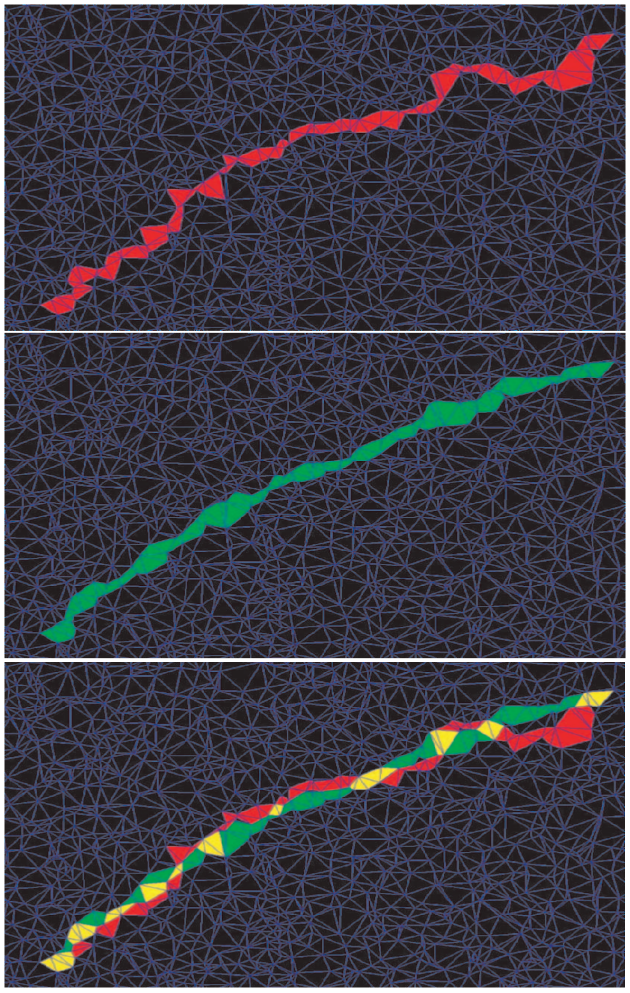

Figure 7. The first image shown is of the path taken by CCW, Second is BC, and the third, represents both, where lighter regions are the intersections of both paths.

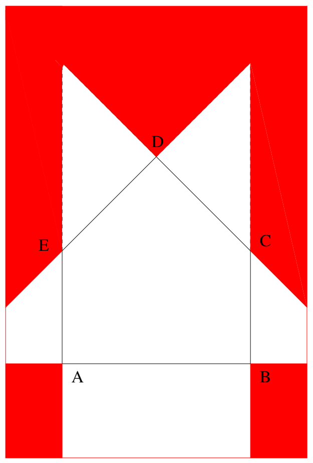

Figure 8. Regions of ambiguity for a pentagon.

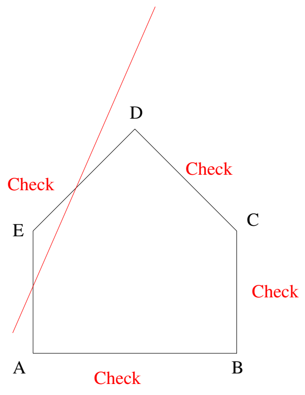

Figure 9. Checks that the segment intersection will perform.

ent subdivision (discussed in our next Section). Figure 7 illustrates paths taken by CCW, BC the intersection of both.

6 Extension and Future Work

The real merit of the algorithm, however, lies in its extensibility. This algorithm is easily adaptable to meshes which contain polygons of any size. The other algorithms discussed will inevitably end up doing computations that are redundant. For example: The Guibas and Stolfi algorithm will suffer from heavy ambiguity when faced with many regions of intersecting half planes resulting in a meandering path (see figure 8).

The Mucke [13] algorithm will compute up to (n-1) segment intersection tests in addition to orientation tests, which altogether will be redundant since an edge gleans no information from a previous edge's test (see figure 9).

The barycentric test provides an elegant method to decide which edge to traverse: find the edge corresponding to the two largest barycentric coordinates (see figure 10).

This method would be especially valuable when a reductions in the triangle mesh is desired. Reducing a mesh would involve polygons of all sizes embedded in the same mesh. If the goal is to keep the connectivity of the hetergenous mesh, but still allow point location in this mesh, our algorithm would be a good choice for such a goal.

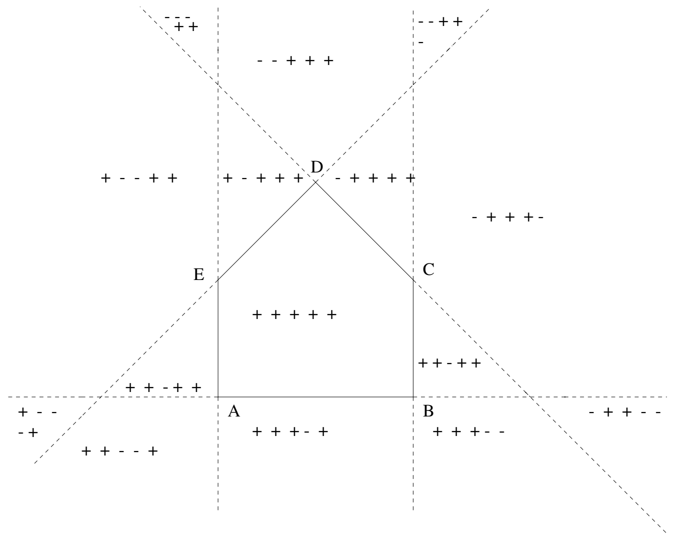

Figure 10. Regions of the intersection of half-planes and corresponding directionality with respect to the vertices A,B,C,D and E.

7 Conclusion

We have presented a simple yet general walk-through point location algorithm useful for non-homogenous meshes and higher dimensional polyhedral lattices with theoretical complexity $O((n)^{\frac{1}{m}})$ [13, 5], where n is the number of points and m is the dimensionality of the points.

References

[1] P. Alliez, and M. Desbrun, "Progressive Compression for Lossless Transmission of Triangle Meshes". *Proceedings from SIGGRAPH 2001*, pages 195-202.

[2] J.L. Bentley, B.W. Weide, and A.C Yao, "Optimal Expected Algorithms for Closest point problems", *ACM Transactions on Mathematical Software*,6(4):563-580, 1980.

[3] H. S. M. Coxeter "Barycentric Coordinates". *Introduction to Geometry*, 2nd ed. New York: Wiley, pp. 216-221, 1969.

[4] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, *Computational Geometry. Algorithms and Applications*, Springer-Verlag Berlin Heidelberg, 1997

[5] L. Devroye, E. Mucke, B. Zhu, "A note on Point Location of Delaunay Triangulation of Random Points". *Algorithmica*, 22(4): 477-482, Dec 1998.

[6] L. Devroye, P. Bose, "Intersections with Random Geometric Objects". *Manuscript, School Of Computer Science, McGill University, 1995*

[7] R.A Dwyer, "A fast divide and conquer algorithm for constructing Delauany Triangulations". *Algorithmica* 2:137-151, 1987

[8] L. Guibas, J. Stolfi, "Randomized incremental construction of Delaunay and Voronoi Diagrams". *Algorithmica* 7:381-413, 1992.

[9] L. Guibas, J. Stolfi, "Primitives for manipulation of general subdivisions and computation of Voronoi diagrams". *ACM Transaction on Graphics*, 4(2):75-123, 1985.

[10] M. Hausner, *A Vector Space Approach to Geometry*, 1965

[11] H. Hoppe, "Progressive Meshes", *Proceedings ACM SIGGRAPH 1996*, pp. 99-108, August 1996.

[12] H. Hoppe, T. DeRose, T. Duchamp, J. MacDonald, W. Stuetzle, "Surface Reconstruction from Unorganized Points", *Computer Graphics (SIGGRAPH 1992 Proceedings)* 26(2): 71-78,July 1992.

[13] E. Mucke, I. Saias and B. Zhu (1996), "Fast Randomized Point Location Without Preprocessing in Two and Three-dimensional Delaunay Triangulations". *Proceedings of the 12th Annual Symposium on Computational Geometry*, pages 274-283, 1996

[14] P. Su, R.L.S Drysdale, "A Comparision of Sequential Delaunay Triangulation Algorithms". *Symposium of Computatinal Geometry*, pages 61-70, 1995

[15] P. Yiu "The Uses of Homogeneous Barycentric Coordinates in Plane Euclidean Geometry". *International Journal Math. Ed. Sci. Tech.* 31, 569-578, 2000