Finding Intersections of Bichromatic Segments Defined by Points

Amr Elmasry* Kazuhisa Makino†

Abstract

Consider a set of n points in $\Re^2$, each colored either red or blue. A line segment defined by two red points is a red segment, and that defined by two blue points is a blue segment. A bichromatic intersection is an intersection between a red segment and a blue segment. We give an $O(n^2 + k)$ algorithm that reports k bichromatic intersections defined by the n points. Extending our algorithm to points on spherical curves, we can report in $O(n^2 + k)$ time the k simplices, defined by n points in $\Re^3$, containing a specified point in their interior.

1 Introduction

Geometric intersection problems are among the fundamental problems of computational geometry. One of these intersection problems is the problem of reporting all pairwise line-segment intersections. Using a plane-sweep algorithm, Bentley and Ottmann showed how to report k intersecting pairs of n line segments in $O((n + k) \log n)$ time and $O(n)$ space [4]. Later, Balaban gave an optimal algorithm for the problem running in $O(n \log n + k)$ time and $O(n)$ space [2].

A variation of the general line-segment intersection problem is the bichromatic line-segment intersection reporting; Given a set of red segments and another of blue segments, the bichromatic intersections problem is the problem of reporting the intersections between red segments and blue segments. The case where there are no possible monochromatic intersections (intersections between segments having the same color) is a special case of the general line-segments intersections problem, and hence inherits the same $O(n \log n + k)$ bound. Mairson and Stolfi [9] gave a simpler algorithm for such special case than that of Balaban. The problem becomes more difficult when monochromatic intersections exist. In such case, Agarwal [1] and Chazelle [6] showed how to report k bichromatic intersections in $O(n^{4/3} \log^{O(1)} n + k)$ time. A special case of the latter problem is when the union of each of the red and the blue segments is connected as a point set. In such case, Basch et al. gave an $O((n+k) \log^{O(1)} n)$ time algorithm for reporting k such bichromatic intersections [3].

Consider a set of n points, each colored either red or blue. Our problem is to find intersections between line segments joining two red points and those joining two blue points. Note that the number of such blue and red line segments is $\Theta(n^2)$, and the number of the bichromatic intersections k is $O(n^4)$. If we consider all the possible $\Theta(n^2)$ red and blue line segments connecting the n-point set, our intersection problem is even a special case of that of Basch et al. [3]. Applying their algorithm on these segments, we get an $O((n^2+k) \log^{O(1)} n)$ algorithm for such problem. In this paper, we introduce a simpler algorithm and get rid of the poly-logarithmic factor achieving the $O(n^2 + k)$ bound.

Consider a set of n points and a point p in $\Re^d$. A d-dimensional simplex that contains p is a minimal subset of points whose convex hull contains p. By Carathéodory theorem (see [10]), the cardinality of any such set is at most $d + 1$ (under the general position assumption, this would be exactly $d + 1$). It is a long-standing open problem to find an algorithm, for reporting all such k simplices, whose running time is polynomial in n, d, and k (see [5, 11]). For the case $d = 2$, an algorithm for reporting all such simplices in $O(n + k)$ time is introduced in [8]. In this paper, we give an $O(n^2 + k)$ algorithm for the case $d = 3$.

2 The bichromatic-intersections algorithm

A main result of the geometric duality is that the angular order of a given set of n points can be produced with respect to each of these n points, all in $O(n^2)$ time and storage [7]. We use this result, and assume that throughout the algorithm such angular orders are known and stored by a preprocessing phase.

We use a divide-and-conquer approach to solve our problem. The set of n points is divided into two sets of almost equal sizes ($\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$) via a horizontal line. This can be done by finding the median of the points with respect to their Y-coordinates. Two categories of line segments are defined: A crossing segment is a segment whose endpoints are each on a different side of the dividing horizontal line. A non-crossing segment is a segment whose two endpoints are on the same side of the dividing horizontal line. Three types of intersections are thereby possible depending on the type of the

*Department of Computer Engineering and Informatics, Beirut Arab University, Lebanon (On sabbatical from Alexandria University of Egypt); elmasry@alexeng.edu.eg

†Department of Mathematical Informatics, Graduate School of Information and Technology, University of Tokyo, Tokyo, Japan; makino@mist.i.u-tokyo.ac.jp

two intersecting segments:

1. crossing/non-crossing
2. crossing/crossing
3. non-crossing/non-crossing

We start by finding the first two types of intersections (crossing/non-crossing and crossing/crossing), and then we recursively solve the problem for each of the two subsets of points to find the third type of intersections (non-crossing/non-crossing). Being able to find the first two types of intersections for a single recursive step in $O(n^2)$ plus the time required to report such intersections, the claimed $O(n^2 + k)$ bound follows as a result of the following recursive relation: $T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + O(n^2)$ whose solution is $T(n) = O(n^2)$.

For an efficient implementation, instead of finding the median of the Y-coordinates of the points with every recursive call, we sort the n points with respect to their Y-coordinates in a preprocessing phase. An iterative bottom-up (considering the recursion tree) implementation instead of the recursive implementation is possible.

Finding crossing/non-crossing intersections

Fixing a point x, consider the line segments that cross the separating horizontal line and join x to the points on the other side with the same color as x. We call these segments the segments of the cone of x. The angular order of these segments around x is computed in a preprocessing phase. We are looking forward to find the intersections of the segments of the cone of x with the non-crossing line segments of the other color. The points that are on the same side as x and have a different color than x are identified and grouped such that the points between two consecutive segments of the cone of x are in the same group. Since the angular order of the points around x is pre-computed, these groups of points are identified in $O(n)$ time. A special group is the group of such points which are outside the segments of the cone of x (between the segment with the largest angle with the horizon and that with the smallest angle). Except for that special group, the line segments formed by the points of the same group do not have any intersections with the segments of the cone of x. On the other hand, a line segment formed by two points from two different groups intersects all the segments of the cone of x that lie between (with respect to the angular order around x) these two groups. A line segment whose both endpoints are in the special group either intersects all the segments of the cone of x or none of them. To find these intersections, we start checking the points of this special group in an increasing angular order around the point x in the clockwise direction.

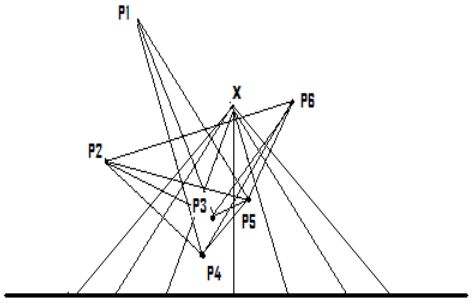

Figure 1: crossing/non-crossing intersections

For each such point y, we start another traversal for the points of the special group in increasing angular order in the anti-clockwise direction as long as the line segment formed by this point and y intersects all the segments of the cone of x. Once we reach a point in the anti-clockwise traversal that together with y forms a line segment which does not intersect the cone of x, we proceed with the clockwise direction for another point y and restart the anti-clockwise traversal.

For the example in Figure 1, p_3 and p_4 are in the same group, while p_1, p_2 and p_6 are in the special group. The line segment whose endpoints are p_2 and p_6 intersects all the line segments of the cone of x, while the line segment whose endpoints are p_1 and p_2 and that whose endpoints are p_1 and p_6 have no intersections with the segments of the cone of x.

That way, we will be able to find such bichromatic intersections for the crossing line segments of the cone of x in $O(n)$ time plus the time to report such intersections. Repeating the procedure for all such points as x, we use an extra $O(n^2)$ time for finding all the crossing/non-crossing intersections.

Finding crossing/crossing intersections

The crossing line segments joining any point to points with the same color on the other side form a cone of segments. The order of these segments around that point is determined in a preprocessing phase. Consider any two points x and y having two different colors. There are two cases depending on the position of one of these points with respect to the cone of segments of the other point. For Case 1, one of the two points lies inside the cone of segments of the other point. Assume, without

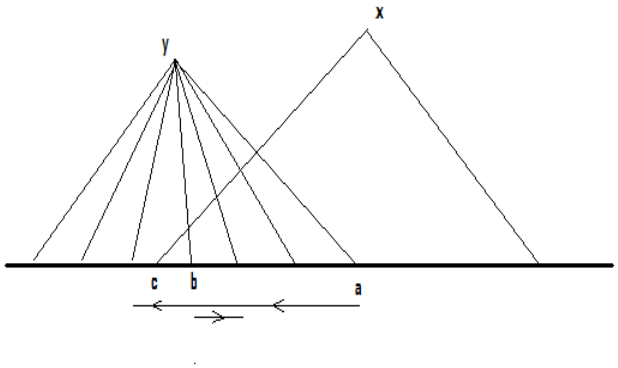

Figure 2: crossing/crossing intersections

loss of generality, that y lies in the cone of x. For all such points as y, among the segments of the cone of x, the segment that precedes y and the segment that succeeds y in the sorted angular order around x can be found and stored in $O(n)$ time. This requires a total of $O(n^2)$ time for all the possible pairs x and y. Once we know these two segments for a pair of points x and y, we can divide the segments of the cone of x to two cones of segments and treat the problem as two problems of Case 2, where none of the points is inside the cone of the other point. It follows that only Case 2 is to be treated.

As shown in Figure 2, we start from the segment with the largest angle around y among the segments of the cone of y. The intersection of this segment with the horizontal separating line is the point a. We traverse the segments of the cone of y in order until we reach the segment whose intersection with the horizontal line is the point b. This is the last line segment whose intersection with the horizontal line is to the right of the point c, where c is the intersection of the horizontal line with the segment of the cone of x having the smallest angle around x. Note that all the traversed segments of the cone of y each intersects at least one segment of the cone of x, and hence the time spent in traversing such segments can be charged to these intersections. Once we reach the point b, we start traversing the segments of the cone of y again but in the other direction this time. This can be done by maintaining a stack that holds these segments. Simultaneous to the reversed traversal of the segments of the cone of y, we traverse the segments of the cone of x in the same direction of increasing angular order. Whenever the intersection of the segment of the cone of y at the top of the stack with the horizontal line is to the right of the intersection of the next segment of the cone of x with the horizontal line, we report that this segment of the cone of x intersects all the segments of the cone of y that are still in the stack. Then we proceed traversing the next segment of x, and in accordance we perform as many pops to the stack of the segments of the cone of y as necessary.

That way, we will be able to find the intersections of the segments of the cone of x with those of the cone of y in an $O(1)$ time plus the time required for reporting these intersections. Repeating the procedure for all possible points x and y, we use an extra $O(n^2)$ time for finding all the crossing/crossing intersections.

3 Finding simplices containing a given point in $\Re^3$

Consider the case where, instead of having line segments between every pair of points, the points are defined on a spherical surface and that the curve between any pair of such points is the spherical curve thus defined. The question is whether our algorithm still applies or not. We give three basic properties for the defined curves that are enough to guarantee the validity of our algorithm with the same $O(n^2 + k)$ time bound:

1. **The uniqueness property:** For any two points x and y, there is a unique curve C_{xy} defined between these two points.

2. **The non-crossing property:** For any two points x and y, if $z \in C_{xy}$, then $C_{xz} \subset C_{xy}$.

3. **The triangular property:** For any three points x, y and z, the angle between the two tangents to C_{xy} and C_{xz} at the point x is less than π.

It is a straightforward exercise to verify that all the steps of the algorithm apply for such types of curves.

Given a set of n points and a point p in $\Re^3$, we proceed to find all the simplices of the point-set that contain p. First, the n points are projected on the surface of a unit sphere whose center is p. Second, consider an arbitrary plane passing by p, the center of the sphere. Each point on the surface of the first hemisphere is colored red. Each point on the surface of the second hemisphere is mapped to the surface of the first hemisphere as the other endpoint of the diameter that passes by this point; this mapped point is colored blue. This mapping simplifies the problem by such projection to a two-dimensional surface. It is not hard to verify that four points forming a simplex that contains p correspond to either

1. Three points of the same color, the simplex of which contains the fourth point that has a different color.

2. Two red points defining a red segment and two blue points defining a blue segment, and the two segments intersect.

To find simplices of the first type, we repeat for every point to find the two-dimensional simplices, among the points of the other color, that contain this point. This can be done by applying n calls to the two-dimensional algorithm given in [8] that runs in $O(n)$ time in addition to the time to report such simplices, for a total of an additional $O(n^2)$ time for the n calls. To find simplices of the second type, we apply our bichromatic-intersections algorithm for spherical curves. This gives a total time of $O(n^2 + k)$ for reporting these k simplices.

References

[1] P.Agarwal, Partitioning arrangements of lines: II. Applications, *Discrete Computational Geometry: Theory and Applications*, **5**:(1990), 533-573.

[2] I. Balaban, An optimal algorithm for finding segment intersections, *11th ACM Symposium on Computational Geometry*, (1995),211-219.

[3] J. Basch, L. Guibas, and G. Ramkumar, Reporting red-blue intersections between two sets of connected line segments, *4th European Symposium on Algorithms*, LNCS **1136** (1996), 302-319.

[4] J. Bentley and T. Ottmann, Algorithms for reporting and counting intersections, *IEEE Transactions on Computers C-28*, (1979), 643-647.

[5] M. Bussieck and M. Lübbecke, The vertex set of a 0/1-Polytope is Strongly P-Enumerable, *Computational Geometry: Theory and Applications*, **11**(2):(1998), 103-109.

[6] B. Chazelle, Cutting hyperplanes for divide-and-conquer, *Discrete Computational Geometry*, **9**:(1993), 145-158.

[7] B. Chazelle, L. Guibas, and D. Lee, The power of geometric duality, *BIT*, **25**:(1985), 76-90.

[8] A. Elmasry and K. Elbassioni, Output-sensitive algorithms for counting and enumerating simplices containing a given point in the plane, *17th Canadian Conference on Computational Geometry*, (2005), 248-251.

[9] H. Mairson and J. Stolfi, Reporting and counting intersections between two sets of line segments, *Theoretical Foundations of Computer Graphics and CAD*, F40 NATO ASI, (1988), 307-325.

[10] A. Schrijver, Theory of Linear and Integer Programming, *Wiley-Interscience*, (1986).

[11] G. Swart, Finding the convex hull facet by facet, *Journal of Algorithms*, **6**:(1985), 17-48.