Information Processing Letters 21 (1985) 55–61
North-Holland

16 August 1985

# ON FAST COMPUTATION OF DISTANCE BETWEEN LINE SEGMENTS

Vladimir J. LUMELSKY

*Information Systems Laboratory, Corporate Research and Development, General Electric Company, P.O. Box 8, Schenectady, NY 12301, U.S.A.*

Communicated by G.R. Andrews
Received 24 August 1984
Revised 7 December 1984

*Keywords*: Algorithm, minimum distance, straight lines, straight line segments, robotics, image processing, CAD systems, VLSI design

## 1. Introduction

Determining the minimum Euclidean distance (MinD) between two segments of straight lines is a typical problem in robotics (e.g., for collision avoidance), image processing, CAD systems, VLSI design, and other areas of information processing dealing with geometrical data. [1] For straight lines of infinite length, there are simple formulae based on the computation of a common perpendicular between two lines [1].

However, in the case of segments of finite length, the classical formulae for distance cannot be applied, and special algorithms are needed; moreover, since MinD computation is often a basic operation frequently used (e.g., in the robot collision avoidance problem it may be repeated hundreds of times, in real time, for each point of the robot path, with few dozens of points per second), the algorithms have to be efficient.

In the sequel, a *line* AB means a straight line of infinite length passing through points A and B; a *segment* AB means a segment of the line AB connecting (and including) points A and B; and *distance* refers to a Euclidean distance. *Global* MinD is the minimum distance between two lines, and *actual* MinD is the minimum distance between two segments defined as the minimum of distances between any of the points of one segment and any of the points of the other segment. These are distinguished from other minimum distances (e.g., between a segment and a line). In these terms, the problem in question is that of finding the actual MinD. Points on lines or segments corresponding to a minimum distance are called *points of minimum distance*; the line passing through these points is called a *line of minimum distance*. A range interval [0, 1] used in the parametric description of lines is assumed to be a closed interval (that is, the endpoints 0 and 1 are included). Instead of shorter vector notations, individual coordinates are used throughout the text to make the computational aspects clearer.

Examples in Fig. 1 show why the final length of straight line segments may complicate MinD computation. (Although points of the global MinD always coincide in the two-dimensional space, in the figure they are shown as two points, M and N, to consider the general case.) Note that, even in the case of two dimensions, the actual MinD is not necessarily zero. This depends on how the points of minimum distance are positioned relative to the

[1] For two-dimensional space, the problem of minimum distance between two segments may be considered as a generalization of the problem of detection of pairs of intersecting segments extensively developed in the area of VLSI design algorithms (see, for example, [2]).

segments. They may lie inside the segments (points M and N, Fig. 1(a))—in which case classical formulae for the distance between two lines can be used. Or, they may correspond to an endpoint of one segment and some inside point of the other segment (distance AP and DQ, Fig. 1(b), (c)). Or, they may correspond to the endpoints of both segments (distance BC, Fig. 1(d)). If endpoints are involved, each of the four endpoints has to be tested. This involves a significant amount of computations. Interestingly, although all of the possible cases are of practical interest, their variety is such that sometimes not all of them are considered. For example, in [3], where methods of descriptive geometry are used to detect interference between two segments, the suggested methods only apply to those cases for which the line of minimum distance is perpendicular to at least one of the segments (as in Fig. 1(a), (b), (c)); in other words, using these methods, one would obtain a wrong answer for the example of Fig. 1(d).

Assuming that input consists of coordinates of four points representing the endpoints of both segments, a more or less straightforward algorithm would involve the following operations:

Compute the global MinD and coordinates of both points (bases) of the line of minimum distance; if both bases lie inside the segments, the actual MinD is equal to the global MinD; otherwise, continue. Compute distances between the endpoints of both segments (a total of four distances). Compute coordinates of the base points of perpendiculars from the endpoints of one segment onto the other segment; compute the lengths of those perpendiculars whose base points lie inside the corresponding segments (up to four base points and four distances). Out of the remaining distances, the smallest is the sought actual MinD. Altogether, this represents the computation of six points and of nine distances.

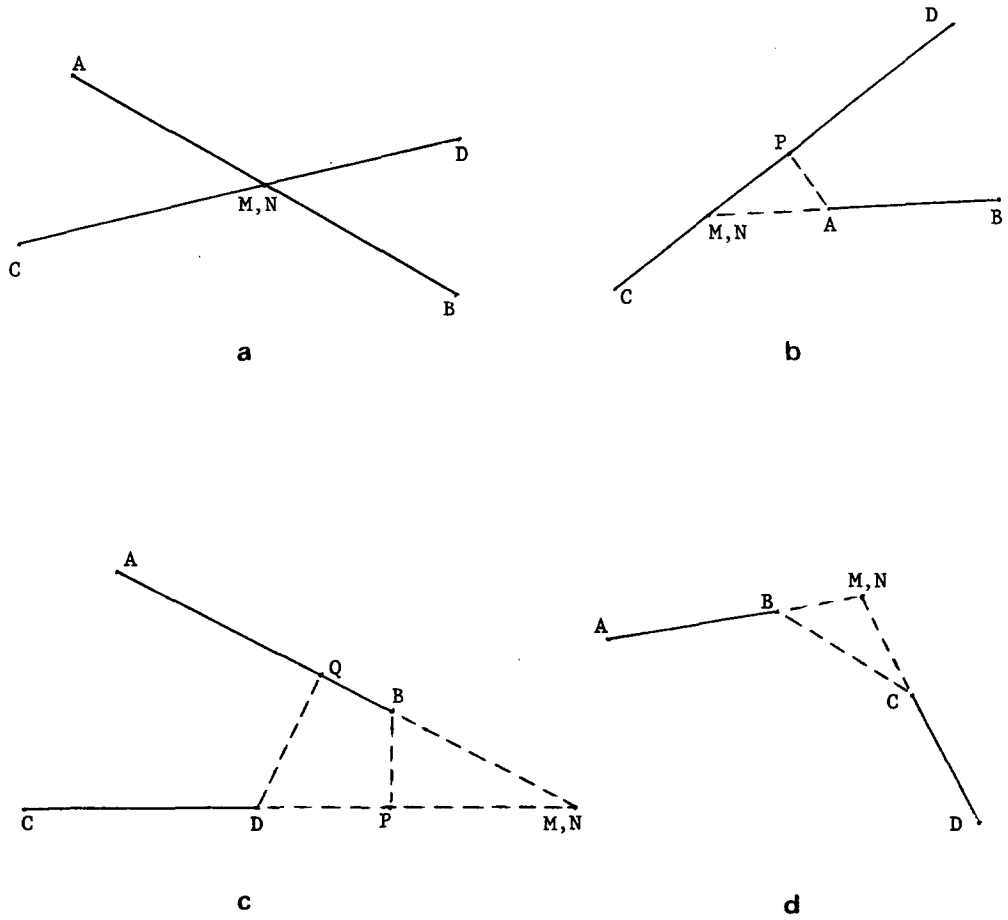


Fig. 1.

The objective of the algorithm presented here is to minimize the amount of computations needed to find the actual MinD. In the suggested scheme, a parametric description of segments is used. Values of the parameters automatically provide information on whether or not points of MinD are located inside the segments and, if not, where they are positioned relative to the segments.

This information allows the algorithm to skip the computation of a number of distances and provides for a rapid convergence of the procedure to the set of points corresponding to the actual MinD. The description and the algorithm can be applied to any number of dimensions.

## 2. Analysis

Consider two segments, AB and CD. For a straight line passing through two points, A and B, with coordinates $A(x_{1A}, x_{2A}, \ldots, x_{nA})$ and $B(x_{1B}, x_{2B}, \ldots, x_{nB})$ in an n-dimensional space of variables $x_1, x_2, \ldots, x_n$, the equation of the line may be written in a symmetric form [1]:

$$\frac{x_1 - x_{1A}}{x_{1B} - x_{1A}} = \frac{x_2 - x_{2A}}{x_{2B} - x_{2A}} = \cdots = \frac{x_n - x_{nA}}{x_{nB} - x_{nA}}. \quad (1)$$

By setting

$$\frac{x_i - x_{iA}}{x_{iB} - x_{iA}} = t, \quad i = 1, \ldots, n \quad (2)$$

a parameter t is introduced, and (1) is rewritten in a parametric form as

$$x_i = x_{iA}(1 - t) + x_{iB}t, \quad i = 1, \ldots, n. \quad (3)$$

Values $0 \leqslant t \leqslant 1$ correspond to points inside segment AB, values $t < 0$ correspond to points on line AB located 'to the left' of the segment (that is, to points that are closer to A than to B), and $t > 1$ corresponds to points 'to the right' of the segment (see Fig. 2). The value of t provides, therefore, a sense of direction for a point on line AB relative to fixed points A and B. This feature will be used in the algorithm to quickly find points of the actual MinD. Similarly, parametric form equations for segment CD are introduced using parameter u:

$$x_i = x_{iC}(1 - u) + x_{iD}u, \quad i = 1, \ldots, n. \quad (4)$$

Instead of the distance d(X, Y) between two points X and Y, its square $DD = d^2(X, Y)$ ('distance') will be used throughout. For a point $X(x_{1X}, x_{2X}, \ldots, x_{nX})$ of line AB and a point $Y(x_{1Y}, x_{2Y}, \ldots, x_{nY})$ of line CD, DD is computed as

$$DD = \sum_{i=1}^{n} (x_{iX} - x_{iY})^2 \quad (5)$$

or, by substituting (3) and (4) into (5),

$$DD = \sum_{i=1}^{n} [(x_{iB} - x_{iA})t - (x_{iD} - x_{iC})u - (x_{iC} - x_{iA})]^2. \quad (6)$$

Using notation,

$$x_{iB} - x_{iA} = d_{i1}, \qquad x_{iD} - x_{iC} = d_{i2},$$
$$x_{iC} - x_{iA} = d_{i12},$$

rewrite (6) as

$$DD = \sum_{i=1}^{n} (d_{i1}t - d_{i2}u - d_{i12})^2. \quad (7)$$

Below, all the sums are assumed to be over $i = 1, \ldots, n$ (as in (7)). Values of t and u corresponding to the points of the global MinD between lines AB and CD may be computed by minimizing (7) over t and u. To do that, equate to zero the partial derivatives, $\partial DD/\partial t = 0$, $\partial DD/\partial u = 0$. If expanded, these become

$$\frac{\partial DD}{\partial t} = 2\sum (d_{i1}t - d_{i2}u - d_{i12})d_{i1} = 0,$$
$$\frac{\partial DD}{\partial u} = -2\sum (d_{i1}t - d_{i2}u - d_{i12})d_{i2} = 0, \quad (8)$$

or

$$t\sum d_{i1}^2 - u\sum d_{i1}d_{i2} = \sum d_{i1}d_{i12},$$
$$t\sum d_{i1}d_{i2} - u\sum d_{i2}^2 = \sum d_{i2}d_{i12}. \quad (9)$$

From these, expressing one parameter in terms of

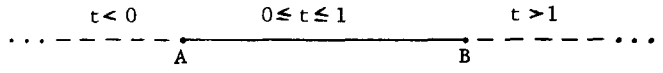


Fig. 2.

the other, we obtain

$$t = \frac{uR + S_1}{D_1}, \qquad u = \frac{tR - S_2}{D_2} \tag{10}$$

or, finding t and u from (9) explicitly:

$$t = \frac{S_1D_2 - S_2R}{D_1D_2 - R^2}, \qquad u = \frac{-(S_2D_1 - S_1R)}{D_1D_2 - R^2}, \tag{11}$$

where

$$R = \sum d_{i1}d_{i2},$$

$$S_1 = \sum d_{i1}d_{i12}, \qquad S_2 = \sum d_{i2}d_{i12},$$

$$D_1 = \sum d_{i1}^2, \qquad D_2 = \sum d_{i2}^2;$$

note that $D_1$ and $D_2$ are squares of the lengths of segments AB and CD. Using the same five terms R, $S_1$, $S_2$, $D_1$, $D_2$ in the steps of the algorithm below helps to shorten the computations.

These t and u correspond to the global MinD and to some points on lines AB and CD which may or may not lie within segments AB and CD. Now we will analyze three major cases which include all possible situations that may take place depending on the values of t and u.

*Case* 1. The values of both parameters are in the range $0 \leqslant t \leqslant 1$, $0 \leqslant u \leqslant 1$. This means that the points of MinD are located within the segments; for two-dimensional space, such an example is shown in Fig. 1(a). Here, the sought MinD corresponds to the global MinD and may be computed directly using (7).

*Case* 2. The value of at least one of the parameters t and u is outside the range [0, 1] (as in examples in Fig. 1(b), (c), (d)). A number of different situations falls into this case. Consider two lines AB and CD, with points M and N being their points of the global MinD (see Fig. 1(d)).

**Lemma 1.** *For a line* AB, *if the value of its parameter* t *is not in the range* [0, 1], *then the minimum distance between segment* AB *and some line* CD *corresponds to one of the segment* AB *endpoints—in particular, to the endpoint closest to the point of the global* MinD *of line* AB (*to line* CD).

**Proof.** Consider a specific case—say, $t > 1$. According to (3), this means that point M (on line AB) of the global MinD between two lines is closer to point B than to A (Fig. 1(d)). Take a variable point X on line AB and move it continuously, starting at point M, in the direction of segment AB. Because AB and CD are straight lines, the distance between X and line CD will increase monotonically. The first point on segment AB that X will meet is B; if X keeps moving along AB, its distance to CD will increase. Therefore, point B is the point of MinD between segment AB and line CD. Similarly, if $t < 0$, point A of segment AB is a point of MinD between segment AB and line CD. □

Lemma 1 provides justification and a method for choosing an endpoint of a segment as a point of MinD between a segment and a line, if the lemma conditions are satisfied. For the distance between two segments, first consider the case when one of the parameters t, u is in the range [0, 1] and the other is outside this range.

**Lemma 2.** *If the parameter of a line* (*say*, t *of* AB) *is outside the range* [0, 1] *while the parameter of the other line*, CD, *is inside this range, then one of the endpoints of segment* AB *is the point of the actual* MinD—*in particular, this is the endpoint of segment* AB *closest to the point of the global* MinD *between lines* AB *and* CD.

**Proof.** Consider a specific case—say, $t < 0$, $0 \leqslant u \leqslant 1$ (Fig. 1(b)). According to (3), for line AB, its point M of the global MinD (to line CD) is closer to point A than to B. Take a variable point X on line AB and move it continuously, starting at point M, in the direction of segment AB. For any point X, there is a point Y on line CD corresponding to the minimum distance from X to line CD; this is the base of the perpendicular from X onto line CD. While point X is moving along line AB toward point B, point Y moves continuously, starting at point N inside segment CD, toward one end of segment CD (say, D, as in Fig. 1(b)). Because AB and CD are straight lines, the distance between X and line CD will increase monotonically. Now, one of two cases takes place.

(1) X meets A before Y meets D (Fig. 1(b)); then point X = A and the corresponding point Y

inside segment CD are points of the actual MinD.

(2) Y meets D before X meets A; in this case, if X keeps moving toward A, the point (on segment CD) of minimum distance from X to segment CD must be D (this is because, according to Lemma 1, D is the closest point of segment CD to the perpendicular from X onto line CD).

In both cases, the first point of segment AB that X meets is A; if X kept moving along AB, its distance to segment CD would increase. Therefore, point A of segment AB is the point of the actual MinD between segments AB and CD. Similarly, if $t > 1$, then point B of segment AB is a point of the actual MinD. □

Lemma 2 provides justification for choosing an endpoint of a segment (actually, the value 0 or 1 of its parameter) every time its parameter value falls outside (and the value of the other segment parameter falls inside) the range [0, 1], as a point of the actual MinD.

In general, if t is outside the range [0, 1], then one of the endpoints of segment AB does not necessarily correspond to the actual MinD between two segments. This is demonstrated in Fig. 1(c); although, for segment AB, point B is closest to line CD, not B but point Q (which is the base point of the perpendicular from an endpoint of segment CD onto segment AB) is the point of the actual MinD of segment AB. What disqualifies point B is the fact that the perpendicular BP from B onto line CD, although it is shorter than DQ, has its base outside segment CD. This implies that, in order to check which of the two candidate perpendiculars (if any) corresponds to the actual MinD, the algorithm has to use Lemma 2 twice. If the base points of both perpendiculars fall outside the opposite segments, then it follows from Lemma 2 that the endpoints themselves are the points of the actual MinD (Fig. 1(d)).

*Case* 3. This case includes special situations when one or both parameters t and u are undefined because the denominator in (11) is equal to zero; note that when this occurs, the numerators in both equations (11) are also zero. This occurs when either one or both segments degenerate into a point, or when the segments are parallel. These situations are handled at the early stage of the algorithm, so that no computational difficulties arise because of division by zero in (10) or (11). Parallelism of the segments means either that points of the actual MinD are necessarily segment endpoints, or that many segment points, including at least some segment endpoints, may be chosen as the points of the actual MinD (see Fig. 3(a), (b)). Therefore, in case of parallel segments, segment endpoints are to be analyzed; this brings us to the situations considered in Case 2. Another special situation relates to collinear segments (with or without overlap). Again, any segment endpoint may be chosen as the first candidate for the actual MinD point, with subsequent analysis as in Case 2.

As the above analysis shows, parameters t and u have to be modified frequently to make them correspond to the segment endpoints. Such modification is equivalent to moving a point lying in line AB (or CD) outside segment AB (CD), to the closest endpoint of segment AB (CD), which is represented by the rules

$$\begin{aligned} &\text{if } t < 0 \text{ then } t = 0,\\ &\text{if } t > 1 \text{ then } t = 1,\\ &\text{if } u < 0 \text{ then } u = 0,\\ &\text{if } u > 1 \text{ then } u = 1. \end{aligned} \tag{12}$$

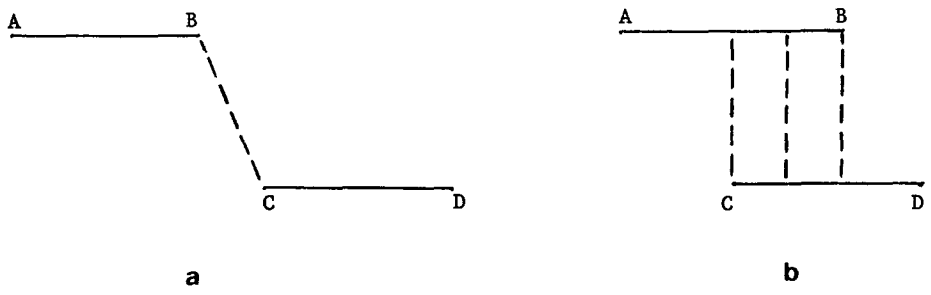


Fig. 3.

## 3. Algorithm

The input consists of coordinates of the endpoints of the two segments. The algorithm includes the following steps.

*Step* 1. Check for special cases; compute $D_1$, $D_2$, and the denominator in (11):
- (a) If one of the two segments degenerates into a point, assume that this segment corresponds to the parameter u, take $u = 0$, and go to Step 4.
- (b) If both segments degenerate into points, take $t = u = 0$, and go to Step 5.
- (c) If neither of two segments degenerates into a point and the denominator in (11) is zero (the segments are parallel), take $t = 0$ and go to Step 3.
- (d) If none of (a), (b), (c) takes place, go to Step 2.

*Step* 2. Using (11) compute t. If t is not in the range [0, 1], modify t using (12).

*Step* 3. Using (10) compute u. If u is not in the range [0, 1], modify u using (12); otherwise, go to Step 5.

*Step* 4. Using (10) compute t. If t is not in the range [0, 1], modify t using (12).

*Step* 5. With current values of t and u, compute the actual MinD using (7).

The rationale behind the algorithm is as follows. Distances are not computed directly until the very end; instead, the corresponding values of parameters t and u are found and used to choose the direction for the following steps. The algorithm manipulates the values of the parameters until those values that correspond to the actual MinD are found; then, the distance computation is done only once. In each of the intermediate steps, only one endpoint of a candidate line of minimum distance is implicitly tested; if this endpoint does not satisfy the conditions for being the actual MinD, its partner on the line of minimum distance is never tested.

First, special cases are tested (see Step 1). Specifically, the segments are checked for parallelism. If they are parallel, then the points of the global MinD are undefined, and one of the segment endpoints is taken (e.g., by setting $t = 0$) arbitrarily, to define the distance from the other line (segment). If the segments are not parallel (follow, for example, Fig. 1(c)), then in Step 2 the point of the global MinD (actually, its t value) for one of the segments (here, AB) is found. If it is in the range [0, 1] and if (in Step 3) the value of u also falls within [0, 1], then the global MinD coincides with the actual MinD, and that is computed by immediately going to Step 5. If the point of the global MinD found in Step 2 is outside the range [0, 1] (as in Fig. 1(c), $t > 1$, point M), then, according to Lemma 1, an endpoint of segment AB closest to line CD (here, point B) is found by setting the value of t according to the rules (12) (here, $t = 1$). Now, if the base P of a perpendicular from B onto line CD falls within segment CD, then, according to Lemma 2, BP represents the sought actual MinD. This is checked in Step 3. If u is outside the range [0, 1] (as in Fig. 1(c), $u > 1$), then, according to Lemma 1, point D ($u = 1$) is another candidate for being the point of the actual MinD.

At this stage, point D must be one of the points of the actual MinD, and the only remaining thing is to find a partner for point D. This is done in Step 4: if the value of t corresponding to the current value of u falls within the range [0, 1], then Q is the second point of the actual MinD (as in Fig. 1(c), $0 < t < 1$); conversely, if t falls outside the range [0, 1], then an endpoint of segment AB found from (12) is the second point of the actual MinD. At this stage, the points of the actual MinD are defined for any possible case, and the value of the actual MinD is computed (Step 5).

To compute the distance between two segments in n-dimensional space, the algorithm requires $5n + 12$ multiplications/divisions and $8n + 1$ additions (for details, see [4]). For example, in case of three-dimensional space, this amounts to 27 multiplications and 25 additions. The straightforward algorithm sketched in Section 1 requires about five times as many operations.

As to its computational characteristics, the algorithm is robust and rather insensitive to round-off errors. Those values of the parameters t and u that fall outside the range [0, 1] do not have

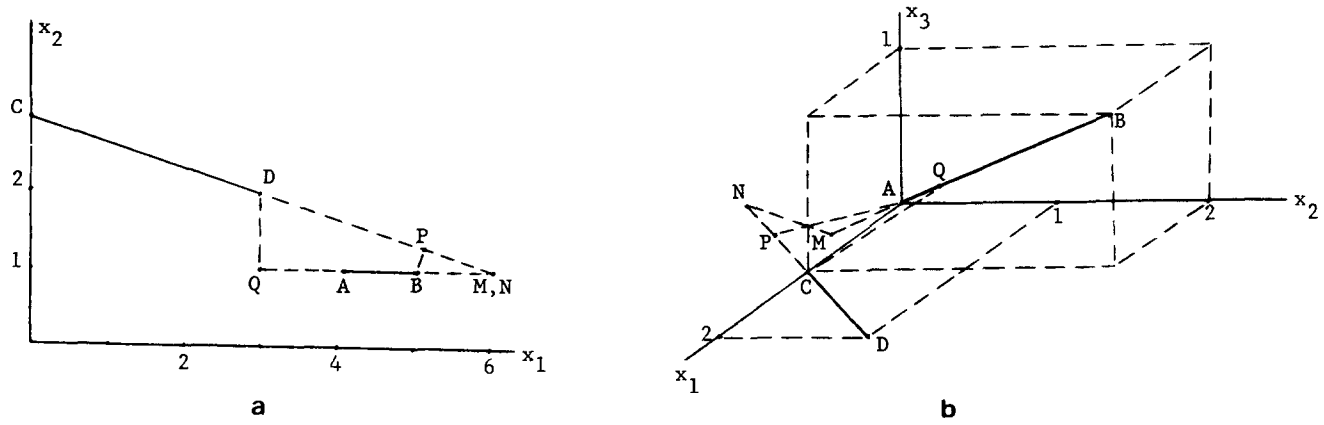


Fig. 4.

to be very accurate because immediately after computation they are changed to values 0 or 1 according to rule (12). Division by zero is avoided by separate handling of special cases in Step 1 of the algorithm. Possible division by small numbers in (10) and (11) is not a real problem because, first, not more than three divisions per problem are executed and not more than two of them require normal accuracy, and, second, small denominators in (10) and (11) always correspond to small numerators. Simple thresholds on the minimum values of those denominators (to define numerically the notion of a special case) assure numerical stability.

## 4. Example

A three-dimensional case is considered (see Fig. 4); coordinates of the segment endpoints are: A(0, 0, 0), B(1, 2, 1), C(1, 0, 0), D(2, 1, 0). The intermediate results of the algorithm steps are as follows:

*Step* 1: None of the special cases takes place.

*Step* 2: $t = -0.333$ (this corresponds to the first point of the global MinD—point M on line AB); modified to $t = 0$ (point A). (Note that the second point of the global MinD, point N on line CD, which, according to (10), corresponds to $u = -1$, is never considered.)

*Step* 3: $u = -0.5$ (point P); modified to $u = 0$ (point C).

*Step* 4: $t = 0.167$ (point Q).

*Step* 5: Using current $t = 0.167$ and $u = 0$ (points Q and C), compute $DD = 0.8333$.

## Acknowledgment

The author wishes to thank Charles Fiduccia, Boris Yamrom, and Hai-Ping Ko for valuable discussions.

## References

[1] G.A. Korn and T.M. Korn, Mathematical Handbook (McGraw-Hill, New York, 1968).

[2] M.I. Shamos and D.J. Hoey, Geometric intersection problems, in: Proc. 17th Ann. Conf. on Foundations of Computer Science, 1976.

[3] S.E. Rusinoff, Practical Descriptive Geometry (American Technical Society, 1947).

[4] V.J. Lumelsky, Fast algorithm for computation of the minimum distance between segments of straight lines, Tech. Rept. No. 84CRD206, General Electric Corporate Research Center, Computer Science Branch, 1984.