UNIVERSITY OF WATERLOO
COMPUTER SCIENCE DEPARTMENT

Dynamical Sets of Points

Thomas Ottmann
Derick Wood

Data Structuring Group
CS-82-56

November, 1982

Dynamical Sets of Points†

Thomas Ottmann

Institut für angewandte Informatik und Formale Beschreibungsverfahren
Universität Karlsruhe
Postfach 6380
D-7500 Karlsruhe
West Germany

Derick Wood

Data Structuring Group
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1

ABSTRACT

This paper initiates the combinatorial investigation of sets of moving points, that is dynamical sets of points. In this, the first step, we assume the points are instantiated at the same instant of time each with some constant speed. We observe how dynamical point sets give rise to half-lines or rays in a onedimensional higher space time model. This in turns leads to efficient solutions for many problems concerning 1-dimensional dynamical sets, occurring naturally in our framework, using computational geometry techniques. However, this initial investigation raises more questions than it answers, for example is there an algorithm which determines for a 2-dimensional dynamical set all possible point coincidences in time better than $O(n^2)$?

1. INTRODUCTION

By watching the traces of airplanes on radar screens air controllers pursue their task of routing them safely. As a first approximation to this task we consider the following abstract problem:

Given a collection of n points in the plane, each moving at constant speed and direction, detect any, all, the first, etc. collisions of any two points.

We will speak of a *dynamical* set of points in 2-space. Here, dynamical means that the relative positions of objects changes over time. This is in contrast

† This work was carried out under NATO grant No. RG 155.81 and Natural Sciences and Engineering Research Council of Canada Grant No. A-7700.

to the traditional notion of a *dynamic* set of objects which means that objects may appear or disappear but their positions are fixed over time. There is, of course, an obvious method of detecting all collisions for a given dynamical set of n points in the plane. Check each of the $\binom{n}{2}$ pairs of points for all possible collisions. A fundamental question is whether or not this approach is optimal. We are, unfortunately, unable to answer this basic question. Instead we discuss the one-dimensional analogue of the problem and show how techniques known from the blossoming area of computational geometry may be applied to obtain solutions in the one-dimensional case which are better than the naive ones.

In the next section we pose a number of questions for dynamical sets of points in 1-space. In Section 3 related computational geometry problems are discussed which then lead in Section 4 to solutions of the problems posed in Section 2. Finally, in Section 5 we discuss our results and pose some of the many remaining open problems.

2. THE PROBLEMS

Assume that we are given a set of n objects moving either left or right at different constant speeds in 1-space. There are a number of natural questions which may be posed in this setting. For simplicity we assume that the objects are points moving on the x-axis; for every point p_i its starting x-value x_i, that is its distance from the origin at time 0, and its speed are known. We are interested in the following problems.

Collision or Coincidence Problems

Assume that the collision of two points has no effect on their speeds and directions.

(c1) Determine all pairs of points which collide at some time in the future.

(c2) For a given time t, determine all pairs of points which collide at time t.

Anihilation Problems

Assume two colliding points anihilate each other:

(a1) Determine the order of anihilation.

(a2) Determine the anihilation free objects which are left after all anihilations or after all anihilations up to a given time t.

The Problems (a1) and (a2) are only two examples of basic questions which may be posed under the anihilation assumption. Furthermore, instead of the assumption that two calling points anihilate each other, further variants of problems may be posed which are based on different assumptions about what happens with two colliding points. One might recall elastic and inelastic pulses in mechanics as examples. Thus, the next group of problems (order problems), may be split into a variety of related problems obtained by imposing different assumptions about what happens whenever two points collide. We will restrict ourselves to the case that the collision of two points has no effect on their speeds and directions.

Order Problems

(O1) For given time t, determine the sorted order of points at time t.

(O2) For given time t and object x: what is the rank of x at time t?

(O3) For given rank and time t: what is the element with that rank at time t?

(O4) For given time t: what is the leftmost and/or rightmost object at time t?

Each of the above problems has a natural translation into a problem in 2-space (space-time):

Each point p_i has a starting position x_i and a space-time orientation θ_i, $0 \leq \theta_i \leq 180^\circ$, which is uniquely determined by its speed v_i, that is θ_i is the slope of the trace of point p_i starting from x_i with speed v_i, cf. Figure 1.

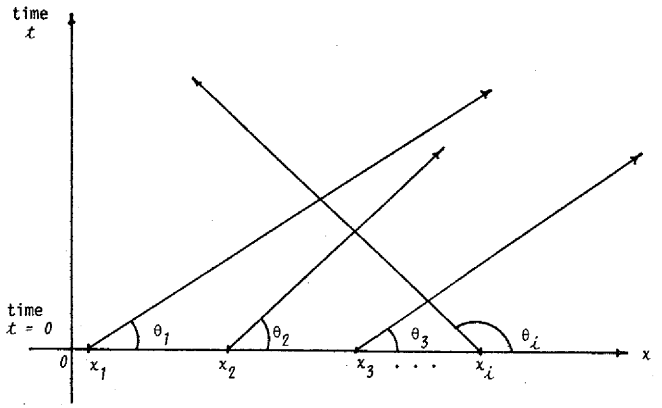


Figure 1

In the space-time model the above problems concerning moving points on a line become problems concerning half lines (or rays) in the plane:

Two half lines L_i and L_j intersect if and only if their corresponding points p_i and p_j collide. Similarly, the anihilation problems lead to the problems of determining the "earliest" intersection points of two half lines. Finally, the sorted order of the intersection points of the half lines with a horizontal line at $t = t_0$ gives the sorted order of the n points at time t_0.

In the next section we will discuss these problems concerning half lines in the plane in a slightly more general setting.

3. HALF LINES IN TWO-SPACE

Two *lines* in the plane (considered as infinite in two directions) intersect if and only if they are not parallel, i.e. if and only if their slopes are different. Thus, in order to determine all pairs of intersecting lines in a given set of n lines in 2-space it suffices to sort them by their slope. This is an $O(n \log n)$ task. It yields the equivalence classes of parallel lines in the given set and, thus, all non-intersecting pairs of n given lines in the plane in time $O(n \log n)$.

On the other hand, all k intersecting pairs of n given *line segments* in the plane can be computed in time $O((n+k)\log n)$ and space $O(n)$, cf. [BO] and [Br]. It is still an open problem whether or not the time bound is tight, or whether it may be improved to, say, $O(n \log n + k)$ as is the case when all line segments are either vertical or horizontal. (This time bound is in fact optimal in this case.)

The *Half-Line Intersection Problem* is to determine all pairwise intersections of n given half lines. Intuitively one might expect the complexity of this problem to be somewhere in between the complexity of the intersection problem for lines (which is easy) and the intersection problem for line segments (for which the available solution is not known to be optimal). We feel, however, that the Half-Line Intersection Problem is closer to the intersection problem for line segments.

Assume that n half lines in the plane are given by their respective starting points in the plane and by their slopes with respect to the positive x-axis. Let us first consider the special case resulting from the problems of Section 2 where all n starting points are an *one* line, and where all half lines are infinite in only one half plane. We may assume without loss of generality that all half lines start on the positive x-axis and are directed into the upper half plane as in Figure 1.

Let A and B be two such half lines and let the starting point of B be to the right of the starting point of A. Then, A and B intersect if and only if B is steeper than A. Hence, the problem of determining all intersecting pairs of n half lines (as in Figure 1) can be solved as follows: Represent each half line by a point (x, θ) in 2-space where x is the starting point on the x-axis and $0 \leq \theta \leq 180°$ is the slope of the half line. Reporting all intersecting pairs of half lines now reduces to the

All-Points Dominance Problem:

For each point $A = (x, \theta)$ report all points $B = (x', \theta')$ dominated by A, that is $x' < x$ and $\theta' < \theta$.

An inspection of Bentley's divide and conquer algorithm [B] for solving the All-Point ECDF Problem shows that it can easily be modified such that it yields the answer to the All-Points Dominance Problem in time $O(n \log n + k)$ where k is the size of the answer. Hence, determining all k pairs of intersecting half lines of a given set of n half lines in the plane is also an $O(n \log n + k)$ task if the given lines are as in Figure 1. This is certainly optimal. Unfortunately, we are unable to solve the (unrestricted) Half-Line Intersection Problem with the

same time bound. We are only able to reduce the Half-Line Intersection Problem to the above special case and the problem of reporting all intersecting pairs of n given line segments in the plane. Let us briefly explain the reduction: Let n half lines with arbitrary starting points in the plane and arbitrary slopes be given. Let x and $x_{\max}$ be the smallest and largest x-value, respectively, of a starting point of a half line facing to the right, that is with slope $-90^o < \theta \leq +90^o$, (respectively facing to the left, that is with slope $90^o < \theta \leq 270^o$). Figure 2 shows an example:

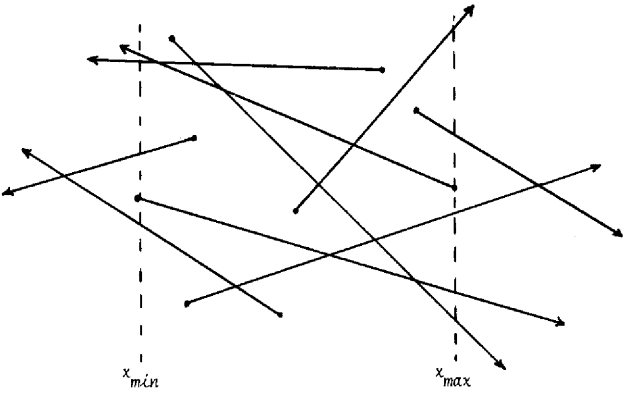


Figure 2

All intersections which occur to the left of the vertical line $x = x_{\min}$ (respectively to the right of $x = x_{\max}$) can be determined as in the special case discussed above where all starting points are on one line. Thus, it remains to report all intersections in the region $x_{\min} \leq x \leq x_{\max}$. Using the algorithm for line segments in the plane, the latter task can be solved in time $O((n+k)\log n)$ where k is the number of intersections found in the region $x_{\min} \leq x \leq x_{\max}$.

This argument shows that the Half-Line Intersection Problem is at most as difficult to solve as the line segment intersection problem but no more efficient solution is known to us.

Next we discuss the anihilation problem in terms of half lines, again in a slightly more general setting: Given a set H of n half lines with starting points on the x-axis or in the upper half plane with slopes in the range $0 \leq \theta \leq 180^o$. The sequence S of anihilation points for H and the set H_0 of half lines left over after all anihilations have occurred are defined by the following algorithm: Initially, H_0 is the given set H of n half lines and the sequence S of anihilation points is empty.

begin

Determine the intersection point p of two half lines, L and L' with minimal distance from the x-axis, let p be the next element of S;
replace H_0 by $H_0 \setminus \{L, L'\}$

end;

An example is depicted in Figure 3:

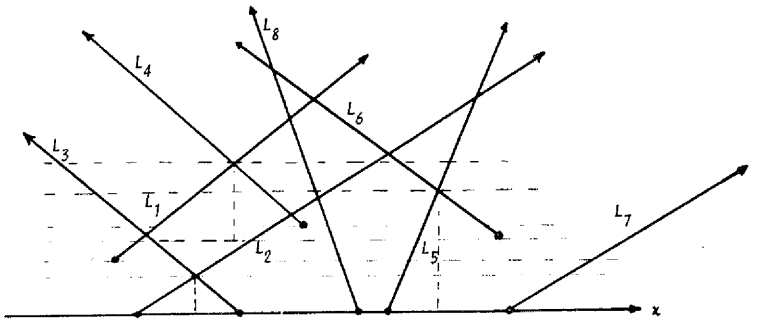


Figure 3

The sequence of anihilation points is:

$$S = L_2 \cap L_3,\ L_5 \cap L_6,\ L_1 \cap L_4$$

The set left over after all anihilations is:

$$H_0 = \{L_7, L_8\}$$

We will now show how to use the sweep line paradigm to compute the sequence S of anihilation points and the set H_0 left over after all anihilations (cf. [BO] and [SH]). A horizontal line is swept bottom to top through the set of starting points of half lines and the anihilation points in sorted order according to their distance from the x-axis. Let L be the set of all half lines which are cut by the sweep line and have not yet been anihilated. As in [SH] and [BO] the sweep line imposes a total order on L, hence we can refer to adjacent half lines with respect to this order.

Initially L contains exactly the half lines starting from the x-axis. For any two adjacent half lines L_i and L_j in L we compute their intersection

$L_i \cap L_j$. If $L_i \cap L_j \neq \phi$, that is if L_i and L_j intersect, their intersection point $L_i \cap L_j$ is entered into a priority queue Q according to its y-distance from the x-axis. Q contains at most n elements and may be organized such that the operations *extraction* and *insert* both can be carried out in time $O(\log n)$.

Starting with the above specified initial values of L and Q we repeat the following steps until all starting points of half lines have been considered and Q has become empty: The next halting position of the sweep line is

(a) either the next starting point of a half line

or

(b) the first element of Q ,

depending on which is nearer to the x-axis.

In case (a) simply insert the respective half line into the structure L, compute the intersection with the elements in L which become immediate neighbors of the newly inserted half line and insert the intersection points into Q.

In case (b) report the first element of Q, say $p = L_i \cap L_j$, as the next anihilation point (in the sequence S); remove L_i and L_j from L and delete also all those intersections from Q which involve L_i or L_j . (Assuming that each half line in L refers to the at most two intersections with immediate neighbors stored in Q these deletions can be carried out in time $O(\log n)$ also.) Finally, the half-lines which became adjacent in L after the removal of L_i and L_j are checked for intersection and any intersection points are inserted into Q.

For the *example* depicted in Figure 3 we show the values of L and Q at all halting positions $y_0, y_1, \cdots$ of the sweep line. The elements of Q are listed in sorted order according to their distance from the x-axis.

$y_0 = 0$:
$L = L_2, L_3, L_8, L_5, L_7$
$Q = L_2 \cap L_3$

$y_1 =$ y-value of $(L_2 \cap L_3)$:
report $L_2 \cap L_3$ as first anihilation point
$L = L_8, L_5, L_7$
$Q = \phi$

$y_2 =$ starting y-value of L_1:
$L = L_1, L_8, L_5, L_7$
$Q = L_1 \cap L_8$

$y_3 =$ starting y-value of L_6:
$L = L_1, L_8, L_5, L_6, L_7$
$Q = L_5 \cap L_6, L_1 \cap L_8$

y_4 = starting y–value of L_4:
$L = L_1, L_4, L_8, L_5, L_6, L_7$
$Q = L_5 \cap L_6, L_1 \cap L_4, L_1 \cap L_8$

y_5 = y–value of $(L_5 \cap L_6)$
report $L_5 \cap L_6$ as second anihilation point
$L = L_1, L_4, L_8, L_7$
$Q = L_1 \cap L_4, L_1 \cap L_8$

y_6 = y–value of $(L_1 \cap L_4)$
report $L_1 \cap L_4$ as third anihilation point
$L = L_8, L_7$
$Q = \phi$

L now contains the half lines left over after all anihilations.

The correctness of the above algorithm immediately follows from the observation: At any time, that is at any halting position of the sweep line, the intersection with minimal y-distance from the sweep line between any two half lines currently cut by the sweep line must occur between two half lines which are adjacent in L. Because all intersections between half lines which are adjacent in L have been computed and are stored in Q, the intersection with minimal y-distance from the sweep line appears as the first element in Q.

Furthermore, from the remarks for implementing the above algorithm (which we gave already) it should be clear that it can be carried out in time $O(n \log n)$ and space $O(n)$.

4. MOVING POINTS ON A LINE

In this section we discuss solutions of the problems posed in Section 2 by using the results of Section 3.

The collision problem (c1) clearly can be solved in time $O(n \log n + k)$ where k is the size of the answer because it can be reduced to the All-Points Dominance Problem.

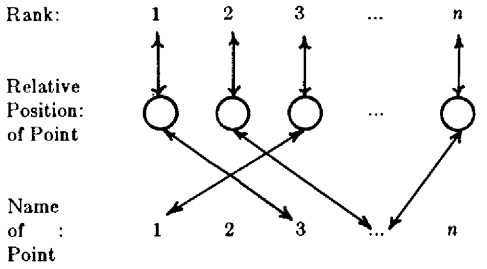
Problem (c2) has a trivial solution: In order to determine all pairs of points which collide at time t calculate the positions $p_i^{(t)}$ of points p_i at time t for $i = 1, \ldots, n$. By sorting $p_1^{(t)}, \ldots, p_n^{(t)}$ we obtain the desired answer in time $O(n \log n + k)$. However, a natural question arises for problem (c2); Assuming that there will be r queries of this kind, can the answers be found in less than $O(\, r \cdot n \log n + \sum_{i=1}^{r} k_i \,)$ time? The following improvement is obvious: First, compute *all* k collisions by the algorithm for solving problem (c1) (this takes $O(n \log n + k)$ time and $O(n)$ space). Second, sort the k collision points by time, requiring $O(k \log n)$ time and $O(k)$ space. After this preprocessing, r queries of the kind (c2) can be answered in time $O(\, r \cdot \log n + \sum_{i=1}^{r} k_i \,)$.

Next we discuss the Anihilation Problems. In the time-space model these problems reduce to the problem of computing the anihilation sequence S and the set H_0 left over after all anihilations for a given set of n half lines in the plane. As we have seen in Section 3 this takes time $O(n \log n)$ and space $O(n)$. Hence, the same time and space bounds hold for the anihilation problems (a1) and (a2). This is even true, if we assume that not all points (moving on a line) are already present at time 0 but some may arise at later times - however with a known starting position and known speed. If we want to compute the anihilation-free objects up to a given time t, it is sufficient just to stop the line sweep at that time in order to obtain the desired answer.

It is obvious that the order of anihilation cannot be computed in less than $O(n \log n)$ time and $O(n)$ space; thus, the solution for problem (a1) is optimal. The same does not hold for problem (a2). It might be possible to compute the set of anihilation-free objects in less than $O(n \log n)$ time, for we have only the trivial lower bound $O(n)$ for this problem.

In order to solve the Order Problems it is clearly sufficient to determine the relative ordering of the n given points at a given time t and then to raise the queries involved in problems (O1)-(O4) with respect to that ordering.

This leads to a variety of solutions depending on how many relative orderings have been precomputed. Let us discuss a few cases in more detail: First we may not precompute any relative ordering at all and proceed as follows: Store the initial ordering, precompute the $k = O(n^2)$ order *changes*, sort and store them according to increasing time: The first task takes time $O(n \log n + k)$, the second $O(k \log n)$, and the necessary amount of storage is $O(n + k)$. We may assume that we have direct access to every point via its name to every element in the ordering via its rank;



If we now want to solve one of the order problems for a given time t we start with the initial ordering and carry out all interchanges up to and including time t. Observe that each single interchange operation takes constant time because we assume we have direct access to each point via its name. Thus, performing the interchange operations up to and including time t takes time $O(k_t)$ where $k_t \leq k = O(n^2)$ is the number of interchange operations between time 0 and time t. It yields the relative ordering of the n points at time t in a structure as described above. It should be clear that each of the questions (O2), (O3), and (O4) can now be answered in constant time.

Let us summarize: Using preprocessing time $O((n+k)\log n)$ we have built a structure which allows us to solve all order problems in time $O(k)$ at worst. On the other hand we may precompute the relative ordering of the n given points after each interchange, thus, building $k = O(n^2)$ structures as described above, and store them according to increasing time. This will consume space $O(kn)$. However, in order to solve an Order Problem for a given time t it is sufficient to determine the structure reflecting the relative ordering valid at time t. This can always be done in time $O(\log k) = O(\log n)$.

Clearly, there is a great variety of other possibilities available for solving the Order Problems in between these two extremes of either precomputing *no* relative ordering or precomputing *all* different relative orderings between time $t = 0$ and time $t = \infty$: We may precompute each r-th different relative ordering, where r is in the range $0 < r < k$, and additionally store for each precomputed ordering the at most k/r interchanges of points occurring up to the next precomputed ordering. In order to obtain the ordering at a given time t we determine the precomputed ordering for the largest time t_1 preceeding t and update that ordering by carrying out all interchanges between time t_1 and t. This leads to a solution for the Order Problems using space $O(\frac{k}{r} \cdot n + k)$ and time $O(\log \frac{k}{r} + r)$. As an example let $r = \log_2 n$, then the Order Problems require time $O(\log n)$ and space $O(\frac{n^3}{\log n})$. We have not differentiated the treatment of the different Order Problems, since the above techniques directly carry over to the case where we assume that any two colliding points anihilate each other.

Under the non-anihilation assumption, much simpler solutions may be obtained as well: In order to solve problem (O4), for example, we can compute for any given time t the positions of all points at time t and then determine the point with minimum (respectively maximum) x-value. This yields a solution to problem (O4) in time and space $O(n)$.

5. DISCUSSION

We have raised more questions than we were able to answer. In particular we do not know whether or not the technique of treating the one-dimensional case (namely to translate problems for dynamical sets of points in 1-space into computational geometry problems in 2-space) ultimately will lead to solutions of the basic problems for dynamical sets of points in 2-space which are better than the naive ones.

As a byproduct we have obtained a natural motivation for problems concerning half lines in the plane which have not been considered before and which are of interest in their own right. Unfortunately, the most important question has been left open: Is it possible to detect all k intersections of a given set of half lines in the plane faster than in time $O((n + k)\log n)$? The reader is invited to solve this simple stated problem.

6. REFERENCES

[B] Bentley, J.L.: Multidimensional Divide-and-Conquer. *Communications of the ACM* 23 (1980), 214-229.

[BO] Bentley, J.L. and Ottmann, Th.: Algorithms for Reporting and Counting Geometric Intersections. *IEEE Transactions on Computers* C-28 (1979), 643-647.

[Br] Brown, K.Q.: Comments on "Algorithms for Reporting and Counting Geometric Intersections". *IEEE Transactions on Computers* C-30 (1981), 147-148.

[SH] Shamos, M.I., and Hoey, D.J.: Geometric Intersection Problems. *Proceedings of the 17th Annual IEEE Conference on Foundations of Computer Science* (1976), 208-215.