Reporting Intersections among Thick Objects[*]

Antoine Vigneron [†]

Abstract

Let E be a set of n objects in fixed dimension d. We assume that each element of E has diameter smaller than D and has volume larger than V. We give a new divide and conquer algorithm that reports all the intersecting pairs in $O(n\log n + \frac{D^d}{V}(n+k))$ time and using $O(n)$ space, where k is the number of intersecting pairs. It makes use of simple data structures and primitive operations, which explains why it performs very well in practice. Its restriction to unit balls in low dimensions is optimal in terms of time complexity, space complexity and algebraic degree.

1 Introduction

We present a new algorithm for the following air or sea traffic control problem: given a set E of n point vehicles in 2 or 3-space, find all the pairs that are closer than a given safety distance. The number of these pairs will be denoted by k throughout the paper. Our algorithm runs in $O(n\log n+k)$ time and uses $O(n)$ space. Its implementation is remarkably fast. For example, if the vehicles are uniformly distributed in a rectangle and we chose a small safety distance (such that $k<n$), our algorithm is only three times slower than quicksort [10]. The two reasons why it is so fast are the following. First, we make use of very simple data structures, in fact our algorithm was implemented with three static arrays of linear size. Second, our algorithm has low *algebraic degree*. This complexity measure has been recently developed by Liotta et al. [11] and is defined as the maximum degree of the polynomials evaluated by the algorithm. It is closely related to robustness in the sense that it gives the precision required to perform exact computation. Our algorithm has degree two, thus it essentially requires double precision. Therefore it allows to use build-in floating point arithmetics which is far more efficient than software multi-precision. For small values of k, it explains why our algorithm proves much faster than a Voronoi diagram computation whose degree is no less than four. Therefore one of the contributions of this work is to be the first experimental evidence of the validity of algebraic degree as a complexity measure.

This problem of finding intersecting pairs in a set of unit balls is known as the Fixed–Radius Near–Neighbors Search, Dickerson and Eppstein solved it in $O(n\log n+k)$ time in fixed arbitrary dimension [7]. Similarly, our algorithm is optimal only when the dimension d is constant, but our experimental results are very good and it extends to the following more general setting. Let E be a set of n objects in fixed dimension d. They are supposed to be thick in the sense that each one of them has diameter smaller than D and has volume larger than V. We can find the k intersecting pairs of objects of E in $O(n\log n + \frac{D^d}{V}(n+k))$ time and in $O(n)$ space. Our algorithm has optimal algebraic degree when a point of each object is given, which is a reasonable assumption in practice. In the real–RAM model of computation [12], its time complexity is optimal when $\frac{D^d}{V}=O(1)$, which means that we consider thick objects of roughly the same size and in low dimension.

[*]This work has been partially supported by the ESPRIT IV LTR Project No 28155 (GALIA), by the Action de Recherche Coopérative Géométrica, and by the Hong Kong Research Grant Council.

[†]Dept. of Computer Science, Hong Kong University of Science & Technology, Clear Water Bay, Kowloon, Hong Kong. Email: {antoine}@cs.ust.hk.

Pairwise geometric intersections reporting has been extensively studied. On a theoretical standpoint, it is well understood for line segments in the plane [1, 6] and boxes in any dimension [8]. These algorithms, however, are not the best choice in practice, which motivated Zomorodian and Edelsbrunner to design a practical intersection algorithm [13] for boxes that does not match the best theoretical bound [8]. Here we give a practical algorithm too, but for different types of objects, in particular we do not require them to be axis–parallel boxes, but we put restrictions on their volume and diameter. For curve segments intersection, robustness problems led to the design of low degree algorithms [3, 4, 5], however an $\Omega(n\sqrt{k})$ lower bound was proven [3] in this context. Our result shows that it is possible to get around this lower bound, and obtain an efficient low degree algorithm for curved objects, if we make assumptions on their volume and diameter.

2 The algorithm

We consider a set E of n objects in $\mathbb{R}^d$, where d is constant. Each object $A \in E$ has diameter at most D and volume at least V. We also assume that we are given a point of each object $A \in E$, we call it the *reference point* of A. We will present an efficient algorithm to report all intersecting pairs in E^2, assuming that we can decide whether a pair $(A, B) \in E^2$ intersects in constant time. The only other geometric predicates that it uses are comparisons between the coordinates of the reference points. Its running time is $O(n \log n + \frac{D^d}{V}(n + k))$, where k is the number of intersecting pairs. It uses $O(n)$ space, which means that the intersecting pairs are output without necessarily being stored, and each pair is output exactly once.

Note that the intersection predicate is necessary to solve our problem. Thus our algorithm has optimal algebraic degree when one point of each object is given. For instance, the objects are balls in our traffic control application, so detecting intersection means comparing distances, which is a degree two predicate. In this case, our algorithm has degree two.

Our predicates have to be implemented for the particular type of objects that is considered. If these objects do not have constant complexity, for instance if they are polyhedral figures, the intersection predicate may take more than constant time, so one would need to multiply our time bounds by this quantity in order to obtain the actual complexity of our algorithm in terms of standard arithmetic operations.

2.1 Computation model

The coordinates of the reference point of an object object $A \in E$ are denoted by $(x_1(A), x_2(A), \ldots x_d(A))$. The first type of geometric predicates that we will use are of the form $x_i(A) \leq x_i(B)$ and $x_i(A) \leq x_i(B) - 2D$. We assume that any such comparison can be made in unit time.

We will also need an intersection predicate. In order to unify the presentation of our algorithm, we will relax its definition. We will keep two properties of usual geometric intersection predicates (for objects with diameter at most D and volume at least V) that are crucial for our algorithm. First, if two object A and B intersect, then for all i we have $|x_i(A) - x_i(B)| \leq 2D$. Second, if $F \subset E$ is a set of disjoint objects whose reference points lie in an hypercube with edge length $4D$, then the cardinality of F is at most $\frac{(6D)^d}{V}$ (this comes from the fact that these objects all lie within an hypercube of volume $(6D)^d$ and are disjoint).

So, a boolean function $\mathrm{inter}(\cdot, \cdot)$ over E^2 is called a *generalized intersection predicate* over E with parameters C and D if and only if it has the following properties:

1. if $\mathrm{inter}(A, B)$ is true, then for all i we have $|x_i(A) - x_i(B)| \leq 2D$.
2. for all hypercube H with edge length $4D$ and for all $F \subset E$, if $\mathrm{inter}(A, B)$ is false for all distinct A,B in F, and if all the reference points of F are in H, then the cardinality of F is at most C.

In particular, if E is a set of geometric objects with diameters at most D and volumes at least V, then the geometric intersection predicate $A \cap B \neq \emptyset$ is a generalized intersection predicate with parameters $\frac{(6D)^d}{V}$ and D. So our algorithm will work with a usual geometric intersection predicate as well as a generalized one. Thus, in the remainder of this note, when we consider a generalized intersection predicate $\mathrm{inter}(\cdot, \cdot)$, we will simply say that A and B intersect (resp. are disjoint) if and only if $\mathrm{inter}(A, B)$ is true (resp. false).

The reason why we need this relaxed definition is the following. Our algorithm will make recursive calls to itself in dimension $d-1$, by ignoring the last coordinate of the reference points, but still using the intersection predicate between the original d–dimensional objects. If we run the $d-1$ dimension algorithm in this way on a subset $E \cap (\mathbb{R}^{d-1} \times [x, y])$ where $y - x \leq 4D$, then the induced intersection predicate in dimension $d-1$ is a generalized intersection predicate with same parameters C and D.

In short, our computation model means that we can make simple comparisons between the coordinates of reference points in unit time, and that we have a generalized intersection predicate with parameters C and D that we can evaluate in unit time as well. With these assumptions, we will show that we can report all the intersecting pairs in $O(n \log n + C(n + k))$ time.

2.2 Algorithm in one dimension

In this section, we assume that $d = 1$. The idea of this algorithm is first to compute a maximal set of disjoint objects E_{max} by walking from left to right along E, and report the intersections with objects of $E \setminus E_{max}$. Then we apply the same procedure to $E \setminus E_{max}$. The following lemma shows how to perform a step of the recursion.

Lemma 1 *Suppose $d = 1$ and E is sorted according to the coordinates of the reference points of its elements. Then it can be partitioned into (E_{max}, E') in $O(C(n' + k'))$ time where $|E_{max}| = n' > 0$ and k' is the number of intersecting pairs in $E_{max} \times E$. Within the same time bounds, and using $O(|E|)$ space, the intersecting pairs of $E_{max} \times E$ can be reported and E' can be sorted in the same order as E.*

Proof: Let $E = \{A_1, A_2, ... A_n\}$ where $x_1(A_i) \leq x_1(A_j)$ for all $i < j$. We process the objects A_i in increasing order of i. Both E_{max} and E' are maintained in a list sorted according to the x_1-coordinate. No two objects of E_{max} intersect. First we insert A_1 in E_{max}.

When we reach A_i, we check its intersection with the objects previously inserted in E_{max} whose x_1–coordinates are at least $x_1(A_i) - 2D$. By definition of generalized intersection predicates, we do not need to check the other elements of E_{max}, which are too far from A_i to intersect it. Moreover, since these objects are disjoint, and within distance $2D$ from A_i, by definition of generalized intersection predicates, there are at most C of them. Thus this step can be performed in $O(C)$ time by visiting the last elements inserted in E_{max} from right to left. If no new intersection is reported, we insert A_i in E_{max}. Otherwise, A_i is inserted in E' and the intersections are reported.

The algorithm we just described effectively partitions E as we intended to do, but it only reports the intersecting pairs $(A, B) \in E \times E_{max}$ such that $x_1(A) \geq x_1(B)$. The other intersections are found by running the same algorithm with a slight modification. Namely, we have constructed E_{max} already so we can check the intersections of A_i with the objects $B \in E_{max}$ such that $x_1(B) \in [x_1(A_i), x_1(A_i) + 2D]$.

Each object of E is processed in $O(C)$ time so this algorithm runs in $O(Cn)$ time. Besides, each element of E' intersects at least one element of E_{max} so $n \leq n' + k'$. Therefore the running time of our algorithm is $O(C(n' + k'))$. $\square$

Lemma 2 *If $d = 1$ and E is sorted according to the coordinates of the reference points, then the k intersecting pairs of E can be reported in $O(C(n + k))$ time and using $O(n)$ space.*

Proof: Partition E as in Lemma 1 and report the intersections in $E_{max} \times E$. If $E' \neq \emptyset$, set $E = E'$ and apply the same procedure recursively. To analyze this algorithm, we note that at each step we pay $O(C)$ time either to put an object in E_{max} or to report an intersection involving an element of E_{max}. These objects and intersections will not be considered at deeper levels of recursion since E_{max} is discarded. Therefore the overall time complexity is $O(C(n+k))$. $\square$

Corollary 1 *If $d = 1$, then the intersecting pairs in E^2 can be reported in $O(n \log n + C(n+k))$ time and $O(n)$ space, where k is the number of these pairs.*

2.3 Generalization to any dimension

Our algorithm can be generalized to any fixed dimension d. The idea is essentially that we can handle a set of objects that are close to an hyperplane $\mathcal{P}$ by projecting the reference points into $\mathcal{P}$ and running the $d-1$ dimensional algorithm. So we can use the following divide and conquer approach: chose a horizontal hyperplane $\mathcal{P}$ that splits the reference points evenly, find the intersecting pairs among objects whose distance to $\mathcal{P}$ is less than $2D$, and recurse on both sides of $\mathcal{P}$. In the remainder of this section, we will prove the following result:

Theorem 1 *Let E be a set of n objects in fixed dimension d. We assume that E is associated with a generalized intersection predicate with parameters C and D. We can report all the intersecting pairs in $O(n \log n + C(n+k))$ time and using $O(n)$ space, where k is the number of reported pairs.*

2.3.1 Description of the algorithm

By Corollary 1 we have an algorithm for $d = 1$. So we will proceed by induction on d. We assume that $d > 1$ and that in dimension $d-1$, there is an $O(n \log n + C(n+k))$ time and $O(n)$ space algorithm to report generalized intersections, and we will show how to use it to obtain an algorithm with the same time bound in dimension d.

We follow a divide and conquer approach. We distinguish between four cases. If the height of E is at most $4D$ (which we call Case 1), we can ignore the last coordinates of the reference points and thus reduce the problem to dimension $d-1$. If it does not occur, then we distinguish between three exclusive cases. In the main case (that we call Case 2) we split E evenly by a horizontal hyperplane, recurse on both sides and handle the neighborhood of the splitting hyperplane with the $d-1$ dimensional algorithm. The remaining two cases (cases 3 and 4) are boundary cases that we introduce for technical reasons.

For any $F \subset E$, we denote by $\max(F)$ (resp. $\min(F)$, resp. $\mathrm{med}(F)$) the maximum (resp. minimum, resp. median) value of $x_d(A)$ over all $A \in F$.

Case 1: when $\max(E) - \min(E) \leq 4D$.
We can use the $d-1$ dimensional algorithm in the following way. For all $A \in E$, we construct a new $d-1$ dimensional object A' with reference point $(x_1(A), x_2(A) \dots x_{d-1}(A))$. Let E' be the set of all such objects. For all $(A, B) \in E^2$, we denote $\mathrm{inter}(A', B') = \mathrm{inter}(A, B)$. The reference point of $A' \in E'$ is in a $d-1$ dimension hypercube H if and only if the reference point of A is in $H \times [\min(E), \max(E)]$, therefore the extension of the predicate $\mathrm{inter}(\cdot, \cdot)$ to E' is a generalized intersection predicate with parameters C and D in dimension $d-1$. So we can report in $O(|E'| \log |E'| + C(|E'| + k'))$ the k' (generalized) intersecting pairs in ${E'}^2$. By construction of E', they coincide with the intersecting pairs in E^2. Since $|E'| = n$ and $k' = k$, the time bound for this case can be rewritten $O(n \log n + C(n+k))$.

Case 2: when $\min(E) + 2D < \mathrm{med}(E) < \max(E) - 2D$.
Let E_1 (resp. E_2) be the set of all $A \in E$ such that $x_d(A) < \mathrm{med}(E)$ (resp. $x_d(A) > \mathrm{med}(E)$). Let E_m be the set of all $A \in E$ such that $x_d(A) \in [\mathrm{med}(E) - 2D, \mathrm{med}(E) + 2D]$. We compute E_1, E_2

and E_m in $O(|E|)$ time by brute force. Note that $\max(E_m) - \min(E_m) \leq 4D$, so, as in Case 1, we can report the k_m intersecting pairs in ${E_m}^2$ in $O(|E_m|\log|E_m| + C(|E_m| + k_m))$ time.

Now, we handle recursively E_1 and E_2, that is, we call recursively on E_1 and E_2 the d dimensional algorithm to report intersecting pairs. A problem here is that we may report intersection pairs that have been already reported because they belong to ${E_m}^2$. This can be fixed by passing a flag and the value of $\mathrm{med}(E)$ at the recursive calls to report intersections in E_1 and E_2. The flag will tell the program not to report pairs (A, B) such that $\{x_d(A), x_d(B)\} \subset [\mathrm{med}(E) - 2D, \mathrm{med}(E) + 2D]$.

Case 3: when $\mathrm{med}(E) \leq \min(E) + 2D < \max(E) - 2D$.
Let E_2 (resp. E_m) be the set of all $A \in E$ such that $x_d(A) > \min(E) + 2D$ (resp. $x_d(A) \leq \min(E) + 4D$). As in Case 1 we can report all the intersecting pairs in E_m by using the $d-1$ dimensional algorithm. Then we handle E_2 by calling recursively the d dimensional algorithm. We can avoid the output of duplicates using the same approach as in Case 2.

Case 4: when $\min(E) + 2D < \max(E) - 2D \leq \mathrm{med}(E)$.
Let E_1 (resp. E_m) be the set of all $A \in E$ such that $x_d(A) < \max(E) - 2D$ (resp. $x_d(A) \geq \max(E) - 4D$). As in Case 1 we can report all the intersecting pairs in E_m by using the $d-1$ dimensional algorithm. Then we handle E_2 by calling recursively the d dimensional algorithm. We can avoid the output of duplicates using the same approach as in Case 2.

2.3.2 Analysis

Here we consider the $d-1$ dimensional algorithm as a black box, that is, its internal data (for instance, the sets E_1, E_2, E_m of objects in $d-1$ dimension) are not considered, we only use its overall running time.

In cases 2,3 and 4, we have $|E_1| \leq |E|/2$ and $|E_2| \leq |E|/2$, therefore the time spent for splitting E into E_1, E_2 and E_m over all recursive calls is $O(n\log n)$. Any object can be handled at most once by Case 1, as it is a terminal case, so the overall contribution of Case 1 is $O(n\log n + C(n+k))$ time. The only remaining contribution we need to study is the contribution of the calls to the $d-1$ dimensional algorithm in cases 2,3 and 4. It is $O(\log n + C)$ time for each occurrence of an object in a set E_m, and $O(C)$ time for each occurrence of an intersecting pair.

So, in order to prove our time bound, it suffices to prove that a given object can appear in two sets E_m at most during the course of the algorithm. Assume that A appears in a set E_m for the first time during the course of the algorithm. If it is not in E_1 or E_2, then it will not be handled recursively, so it will not appear again in E_m. So assume, without loss of generality, that $A \in E_m \cap E_2$. Then $x_d(A) \leq \min(E_2) + 2D$, so afterwards, in any recursive call, we will have $x_d(A) \leq \min(E) + 2D$. Assume that A appears again in E_m in a recursive call. It cannot appear again in E_2 since $x_d(A) \leq \min(E) + 2D$. If it is not in E_1 either, then it will not be handled recursively and so we are done. So we assume that $A \in E_1 \cap E_m$. Then $x_d(A) \geq \max(E_1) - 2D$, and remember that $x_d(A) \leq \min(E_1) + 2D$, so it follows that $\max(E_1) - \min(E_1) \leq 4D$, therefore the next recursive call will be terminal and A will not appear in a set E_m again.

This completes the proof that our algorithm runs in $O(n\log n + C(n+k))$ time. It is easy to see that it uses $O(n)$ space.

Corollary 2 *Let E be a set of n objects in fixed dimension d. We assume that each element of E has diameter at most D and has volume at least V. We also assume that we can detect pairwise intersection in unit time and that we know one point of each object. Then we can report all the intersecting pairs in $O(n\log n + \frac{D^d}{V}(n+k))$ time and using $O(n)$ space, where k is the number of reported pairs.*

We note that we could have achieved the same time bound without using our one dimension algorithm, by using brute force in dimension zero, and use it as a base for induction. This would shorten our proof, however, we still described the one dimensional algorithm. The main reason is that our implementation uses it, and it performs very well. This way, our experimental results can be replicated. We suspect that, using the reduction to dimension zero, it would be slower in practice. Besides, our one dimension algorithm is new (to our knowledge) and may be interesting in its own rights.

3 Experimental results

I implemented this algorithm in two dimensions. The program reports each intersecting pair exactly once, in other words, it does not produce duplicates. For sake of simplicity, and because I expect it to be far more efficient, I implemented a simpler randomized version. The only modification I made was to choose an object at random instead of choosing the object with median y-coordinate to split E into E_1 and E_2. The average time analysis of this randomized algorithm is essentially the same as quicksort [10] and yields the same $O(n \log n + \frac{D^2}{V}(n+k))$ time bound in expectation.

I tried it with a personal computer based on an AMD K6-333Mhz processor. The following results concern same radius disks whose centers are uniformly distributed within a rectangle. I assume the coordinates to be given in single precision, then I can simply use built-in hardware double precision for the intersection predicate. When $n = 10^6$ and I chose (by trying a few values) a radius such that k is smaller than n, the running time is about 15 seconds. The initial sorting takes 5 seconds, which means that for small values of k, our algorithm is only about three times slower than quicksort.

First I compared it with the naive algorithm that checks all the $\frac{n(n-1)}{2}$ pairs of disks. When the radius is large, and therefore $k = \frac{n(n-1)}{2}$, our algorithm is four times slower. Of course, when E is sparse our algorithm is far more efficient. Then I tried it against a Voronoi diagram computation. I believe it to be an interesting comparison for two reasons. First, no experimental result has been published for the previous fixed–radius near neighbors algorithms [2, 7]. Second, Dickerson and Eppstein algorithm [7] starts by computing a Delaunay triangulation, which is equivalent to computing a Voronoi diagram. When $k < n$, my program is about 100 times faster than an exact Voronoi diagram computation with CGAL [9].

4 Conclusion

The main weakness of our approach is that it only works well for objects that have roughly the same size, namely when $\frac{D^d}{V}$ is small. So it would be interesting to try to reduce this factor in our time bound. A more specific question is the following: we gave an optimal degree-two algorithm for reporting intersections among unit disks or 3-balls. Can we find find an efficient degree-two algorithm for disks or 3-balls of different radius?

References

[1] I. J. Balaban. An optimal algorithm for finding segment intersections. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 211–219, 1995.

[2] J. L. Bentley, D. F. Stanat, and E. H. William, Jr. The complexity of finding fixed-radius near neighbors. *Inform. Process. Lett.*, 6:209–212, 1977.

[3] J.-D. Boissonnat and J. Snoeyink. Efficient algorithms for line and curve segment intersection using restricted predicates. *Comput. Geom. Theory Appl.*, 16(1):35–52, 2000.

[4] J.-D. Boissonnat and A. Vigneron. Elementary algorithms for reporting intersections of curve segments. *Comput. Geom. Theory Appl.*, 21:167–175, 2002.

[5] T. M. Chan. Reporting curve segment intersections using restricted predicates. *Comput. Geom. Theory Appl.*, 16(4):245–256, 2000.

[6] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *J. ACM*, 39(1):1–54, 1992.

[7] M. T. Dickerson and D. Eppstein. Algorithms for proximity problems in higher dimensions. *Comput. Geom. Theory Appl.*, 5:277–291, 1996.

[8] H. Edelsbrunner. A new approach to rectangle intersections, Part II. *Internat. J. Comput. Math.*, 13:221–229, 1983.

[9] A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schönherr. On the design of CGAL a computational geometry algorithms library. *Softw. – Pract. Exp.*, 30(11):1167–1202, 2000.

[10] C. A. Hoare. Quicksort. *Computer Journal*, 5(1):10–15, 1962.

[11] G. Liotta, F. P. Preparata, and R. Tamassia. Robust proximity queries: An illustration of degree-driven algorithm design. *SIAM J. Comput.*, 28(3):864–889, 1998.

[12] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 3rd edition, October 1990.

[13] A. Zomorodian and H. Edelsbrunner. Fast software for box intersections. In *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, pages 129–138, 2000.