Point Location in Fat Subdivisions

Mark H. Overmars

RUU-CS-91-40
November 1991

Utrecht University

Department of Computer Science

Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

Point Location in Fat Subdivisions

Mark H. Overmars

Technical Report RUU-CS-91-40
November 1991

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands

ISSN: 0924-3275

Point Location in Fat Subdivisions*

Mark H. Overmars†

Abstract

In this paper the point location problem is studied in fat subdivisions in d-dimensional space. Subdivisions are called fat if no cell has long and skinny parts. Is is shown that point location queries in such subdivisions can be performed in a very simple way in time $O(\log^{d-1} n)$ with a data structure using $O(n \log^{d-1} n)$ storage. The problem is also studied in the situation where the largest cell is only a factor R larger than the smallest cell. In this case we obtain a solution with only $O(\log R)$ query time using linear storage.

Keywords

Computational Geometry, Point location, Subdivisions.

1 Introduction

A fundamental problem in computational geometry is the point location problem: Given a subdivision of d-dimensional space in a number of cells, store it such that for a given query point p the cell containing p can be determined efficiently. Efficient data structures for point location play a crucial role in many applications.

In the plane, i.e. $d = 2$, the problem has been solved in an optimal way. Data structures have been given that use $O(n)$ storage and take $O(\log n)$ time for a point location query, where n is the complexity of the subdivision (see Preparata [8] for a survey). In the sequel we assume each cell has constant complexity and, hence, n is the number of cells in the subdivision.

In three dimensions, the best known solution, given by Goodrich and Tamassia[5], achieves a query time of $O(\log^2 n)$ using $O(n \log n)$ storage. For higher dimensions no good bounds are available.

*This work was partially supported by the ESPRIT II Basic Research Action No. 3075 (project ALCOM) and by the Netherlands Organization for Scientific Research (N.W.O.).

†Department of Computer Science, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, the Netherlands.

In many practical cases subdivisions have special shapes that might facilitate searches. For example, when the subdivision is rectangular, i.e., consists of axis-parallel hyperrectangles, Edelsbrunner, Haring and Hilbert [3] describe a method that has a query time of $O(\log^{d-1} n)$ using $O(n)$ storage. The bound was recently improved by de Berg, van Kreveld and Snoeyink [2].E.g., for $d = 3$ they obtain a query time of $O(\log n)$.

In this paper we consider point location in fat subdivisions, i.e., subdivisions in which each cell is fat. Cells are called fat if they don't contain long skinny parts. For a precise definition of fatness, see section 2. Typical fat objects are: hypercubes, hyperspheres, simplices in which each internal angle is at least θ degrees for some constant angle θ, etc. We prove that such a subdivision can be preprocessed into a data structure of size $O(n \log^{d-1} n)$ such that point location queries can be performed in time $O(\log^{d-1} n)$. The method is based on some interesting properties of fat objects that might be useful in other cases as well.

When the ratio between the sizes of the cells is also bounded by some value R we give a second approach, based on perfect hashing, that yields a structure of size $O(n)$ with a query time of $O(\log R)$.

The paper is organized as follows. In section 2 we give a precise definition of fatness and proof some useful properties. In section 3 we describe the data structure used and proof bounds on the time and storage required. In section 4 we describe our second solution to the problem. Finally, in section 5 we give some conclusions and directions for further research.

2 Fat cells

In [10] the following definition of fatness is given:

Definition 2.1 *An object obj in d-dimensional space is called* k-fat *if for any point* $p \in obj$ *and any hypersphere* B *with* p *as a center that does not fully contain obj in its interior, the portion of* B *covered by obj is at least* $1/k$.

In other words, no piece of *obj* is long and skinny. Many classes of object are k-fat for a small k, for example hyperspheres, hypercubes, simplices where each internal angle is bounded below by some constant (see [6]), etc.

In this paper we will consider fat subdivisions:

Definition 2.2 *A subdivision of d-dimensional space is called* k-fat *iff each cell of the subdivision is a k-fat object.*

In many practical situations subdivisions will be k-fat for some relatively small k. Actually we will not use the fact that we have a subdivision. We will simply deal with sets of non-intersecting k-fat objects. The following property of such sets will be crucial for our results:

Theorem 2.1 *Let V be a set of non-intersecting k-fat objects in d-dimensional space for some constant k. For each object $obj \in V$ let C_{obj} be the smallest axis-parallel hypercube containing obj. Let C be the smallest of these hypercubes. Then the number of objects in V intersecting C is bounded by a constant.*

Proof. Fatness is defined with respect to hyperspheres. We could also define fatness in terms of axis-parallel hypercubes surrounding the point p. It is easy to see that any object that is k-fat w.r.t. spheres for some constant k is k'-fat w.r.t. hypercubes for a slightly larger constant k'. So we can assume that for any point p inside an object and any hypercube C_p with p as a center that does not fully contain obj in its interior, the portion of C_p covered by obj is at least $1/k'$.

Let us assume the measure of C is 1. Now take a cube C' around C with each edge twice as long, i.e., with measure 2^d. Each object obj intersecting C must have a point p inside C. Now take a hypercube C_p around p with measure 1 (i.e., the same size as C). This cube will be completely contained in C'. C_p cannot fully contain obj because it is no larger than C_{obj} (which was at least as large as C). Hence, at least $1/k'$ of C_p is covered by obj, i.e., obj covers an area of at least $1/k'$ inside C'. Because the objects do not intersect each other, this means that there can be no more than $k'2^d$ objects intersecting C. □

Let the objects $obj_1, obj_2, \ldots$ in V be ordered by size of their smallest enclosing hypercubes. The smallest enclosing hypercube of obj_i we denote as C_i. Now we define

$$V_i = \{obj_j | obj_j \cap C_i \neq \emptyset \text{ and } j \geq i\}$$

In words, V_i is the set of objects with index larger or equal to i that intersect C_i. From the above theorem it immediately follows that:

Corollary 2.2 *The size of each set V_i is bounded by a constant.*

Proof. Let V' be the set of all objects with index greater or equal to i. Then clearly obj_i is the object in this set whose enclosing hypercube has minimal size. Hence, the above theorem states that the number of objects intersecting C_i is constant. These are exactly the objects in V_i. □

The global strategy for solving the point location problem now is as follows: For each object we compute C_i and V_i. For a query point p we compute the lowest indexed cube C_i containing p. Next we check each of the objects in V_i to see whether p is contained in it and report the answer (if any). Because V_i has constant size this will take constant time. Hence, the query time will be dominated by the cost of finding C_i. The correctness of the method follows from the following lemma:

Lemma 2.3 *Let p be a point lying in some object obj_j. Let C_i be the lowest indexed cube containing p. Then $obj_j \in V_i$.*

Proof. As p is contained in obj_j and, hence, in C_j, j must be larger or equal to i. Because p lies inside C_i obj_j must intersect C_i. Hence, it is stored in V_i. □

3 The data structure

We have now reduced the point location problem among non-intersecting fat objects to the following problem:

> Given a set of n hypercubes in d-dimensional space, report the smallest cube containing a given query point.

Note that the cubes can intersect one another, even though the original objects did not. Let the cubes, ordered by increasing size, be $C_1, C_2, \ldots$ as above.

If the dimension d is one, i.e., the cubes are segments of the real line, we proceed as follows. The endpoints of the segments split the real line into at most $2n + 1$ elementary intervals. For every point in such an interval the set of segments covering the point is the same. With the interval we store the lowest indexed segment that covers it. This will be the answer for all points in the interval. We store the intervals in a balanced binary tree to find the interval in which a given point p lies in time $O(\log n)$. The segment stored there will be the required answer. The structure clearly uses $O(n)$ storage.

When $d > 1$ we construct a segment tree (see e.g. [9]) on the projections of the cubes on the d-th coordinate axes. For each internal node δ we create a $(d-1)$-dimensional structure on the first $d-1$ coordinates of the cubes whose segments should be stored at δ. Querying the structure now goes as follows. We search with the d-projection of the point p in the segment tree. For each node on the search path we perform a query with the first $(d-1)$ coordinates of p on the associate tree. In this way we find at most $O(\log n)$ answers, one for each node on the search path. Of these answers we take the smallest. It immediately follows that the query time is bounded by $O(\log^d n)$ and that the structure uses $O(n \log^{d-1} n)$ storage. To improve the query time with a factor of $O(\log n)$ note that the one-dimensional structures are ordered lists. Hence, we can use fractional cascading (see [1]). This leads to the following result:

Theorem 3.1 *Given a set of n non-intersecting fat objects in d-dimensional space, they can be stored in a data structure using $O(n \log^{d-1} n)$ storage such that the object containing a query point p can be determined in time $O(\log^{d-1} n)$.*

Proof. As shown in the previous section, it suffices to show that the lowest indexed cube containing p can be found within the stated bounds. The structure described achieves these bounds. □

4 A hashing approach

In many applications cells of the subdivision are not only fat, but also the ratio between the size of the smallest and the size of the largest cell is bounded. (Of course this only applies if the subdivision is restricted to some bounded region of the space. In most applications this is indeed the case or can easily be achieved.) In this section we will describe a different approach, based on hashing, that leads to much better time bounds in such cases.

Let C_1 again be the smallest surrounding hypercube of an object. C_n will be the largest surrounding hypercube. Let the ratio of C_n and C_1 be R. We will give a structure with $O(\log R)$ query time using linear storage.

Let us first consider the case in which $R \leq 2$. We tile the space with hypercubes of the size of C_1. Because the bounding hypercube of each object is at most twice the size of the tile, each object can intersect at most 3^d tiles, i.e., only a constant number. Also the reverse is true: each tile is intersected by at most a constant number of objects. This follows immediately from theorem 2.1. Let S be the set of non-empty tiles and let us store with each tile in S the set of objects intersecting it. This will take $O(n)$ storage. We can now perform a query with a point p by first determining the tile containing p and next checking all objects stored at this tile to find the one containing p. To this end we represent each non-empty tile by its left lower vertex and we divide all coordinates by the size of C_1. In this way, each tile is represented by d integers. We store these integers in a perfect hash table (using either a worst-case or a randomized construction [4, 7]). Now we divide the coordinates of p by the size of C_1, round them down to integers and perform a query in the hash table. This will give us the cell containing p in constant time. Checking the objects stored there also takes constant time.

Lemma 4.1 *If $R \leq 2$, there exists a data structure, using $O(n)$ storage, such that point location queries can be performed in $O(1)$ time.*

When $R > 2$ we proceed as follows. We subdivide the set of objects into $O(\log R)$ groups such that for each group the ratio between the smallest and largest surrounding hypercube is at most 2. Next we apply the above method to each group separately. This will lead to the following result.

Theorem 4.2 *Let V be a set of n non-intersecting fat objects in d-dimensional space. Let R be the ratio between the measure of the smallest and the largest cell. V can be stored in a data structure using $O(n)$ storage such that the object containing a particular query point p can be determined in time $O(\log R)$.*

Proof. When the ratio between the measure of the smallest and largest object is R then the ratio between the smallest and the largest surrounding hypercube is $c_k.R$ where c_k is a constant, depending on the fatness constant k only. The bounds now follow from the above discussion. □

5 Conclusions

In this paper we have given two approaches for solving point location queries in subdivisions of fat cells that might be useful in practice. A number of interesting questions remain. In the solution in section 3 no use was made of the fact that the objects stored are cubes. The method works equally well for sets of axis-parallel hyperrectangles. An interesting question is whether we can use this fact to improve the bounds.

A second question involves dynamic subdivisions. The data structure described allows for insertions and deletions of hypercubes. This though is not enough. Even though each hypercube stores only a constant number of cells, a cell might be stored in all hypercubes. Deleting such a cell would be very costly. To solve this problem one probably needs a quite different approach.

A final question deals with point location among intersecting fat objects. The methods described are heavily based on the fact that the objects do not intersect. It would be interesting to see whether they can be extended to sets of intersecting objects. The property that a cube does only store a constant number of objects is no longer valid. So we probably need a different mechanism for assigning objects to cubes.

References

[1] Chazelle, B., and L. Guibas, Fractional cascading I: A data structuring technique, *Algorithmica* **1** (1986), pp. 133-162.

[2] de Berg, M., M. van Kreveld, and J. Snoeyink, Two- and three-dimensional point location in rectangular subdivisions, Techn. Rep. RUU-CS-91-29, Department of Computer Science, Utrecht University, 1991.

[3] Edelsbrunner, H., G. Haring, and D. Hilbert, Rectangular point location in d dimensions with applications, *Computer Journal* **29** (1986), pp. 76–82.

[4] Fredman, M.L., J. Komlós, and E. Szemerédi, Storing a sparse table with $O(1)$ worst case access time, *J. ACM* **31** (1984), pp. 538–544.

[5] Goodrich, M.T., and R. Tamassia, Dynamic trees and dynamic point location, *Proc. 23rd ACM Symp on Theory of Computing*, 1991.

[6] Matoušek, J., J. Pach, M. Sharir, S. Sifrony, and E.Welzl, Fat triangles determine linearly many holes, Techn. Rep. 174/90, Computer Science Department, Tel Aviv university, 1990.

[7] Mehlhorn, K., and S. Näher, Bounded ordered dictionaries in $O(\log \log n)$ time and $O(n)$ space, *Inform, Proc. Lett.* **35** (1990), pp. 183–189.

[8] Preparata, F.P., Planar point location revisited, *Intern. J. of Found. Comp. Science* **1** (1990), pp. 71– 86.

[9] Preparata, F.P., and M.I. Shamos, *Computational geometry: An introduction,* Springer-Verlag, New York, 1985.

[10] van der Stappen, F., D. Halperin, and M.H. Overmars, The complexity of the free space for a robot moving amidst fat obstacle, 1991, to appear.