Point-Location in The Arrangement of Curves

Sepideh Aghamolaei

Sharif University of Technology, Tehran, Iran
aghamolaei@ce.sharif.edu

Abstract. An arrangement of \( n \) curves in the plane is given. The query is a point \( q \) and the goal is to find the face of the arrangement that contains \( q \). A data-structure for point-location, preprocesses the curves into a data structure of polynomial size in \( n \), such that the queries can be answered in time polylogarithmic in \( n \).

We design a data structure for solving the point location problem queries in \( O(\log C(n) + \log S(n)) \) time using \( O(T(n) + S(n)\log(S(n))) \) preprocessing time, if a polygonal subdivision of total size \( S(n) \), with cell complexity at most \( C(n) \) can be computed in time \( T(n) \), such that the order of the parts of the curves inside each cell has a monotone order with respect to at least one segment of the boundary of the cell. We call such a partitioning a curve-monotone polygonal subdivision.

Keywords: Point-location · Pseudo-line Arrangement · Data Structures · Computational Geometry.

1 Introduction

An arrangement of curves is a subdivision of the plane induced by those curves. The arrangement of pseudo-lines has complexity \( O(n^2) \), where a pseudo-line is a curve that goes to infinity in two directions, and two pseudo-lines can intersect at most once. An output-sensitive algorithm for the arrangement of Jordan arcs with running time \( O((n + k) \log n) \) exists, where \( k \) is the number of intersections in the arrangement [4].

Bzher curves based on Bernstein basis polynomials from computer graphics for non-rasterized curves are examples of real-world examples of polynomial curves.

Testing whether a given pseudoline arrangement can be converted into an straight-line arrangement is known as the stretchability problem is NP-hard and is equivalent to existential theory of the reals [3]. A generalized version of this problem is the d-stretchability discusses the possibility of replacing each pseudoline with a polynomial of degree \( d \) which requires \( \Omega(\sqrt{n}) \) for all simple arrangements [4].

Finding a point in a planar subdivision and returning the face that contain that point is called point location. Point location in a polygonal subdivision with \( n \) segments can be solved in \( O(n \log n) \) preprocessing to build the data-structure and \( O(\log n) \) query time for \( n \) segments [2]. Without using a point-location datastructure, the problem can be solved in \( O(nQ(n)) \) time, for arbitrary shapes.
with an inclusion-exclusion oracle $Q(n)$, by testing whether the point is inside or outside each shape.

Since the problem of converting the arrangement of pseudo-lines into a polygonal arrangement is NP-hard [3], the point-location on polygonal subdivisions is not helpful. The problem of point-location in a set of congruent disks has already been discussed, and an algorithm with $O(\log n + k)$ query time and $O(n^3 \log n)$ preprocessing exists for it [1].

Contributions

- We improve the previous work on the point-location of congruent disks to use $O(\log n)$ query time, instead of $O(\log n + k)$, at the cost of increasing the preprocessing time to $O(n^4 \log n)$.
- We give a data structure for the point location problem in the arrangement of a set of curves with $O(T(n) + S(n) \log S(n))$ preprocessing and $O(\log n + \log C(n))$ query time, if an algorithm for constructing a planar subdivision of complexity $S(n)$ and cell size at most $C(n)$ in time $T(n)$ is given.

2 The Point-Location Data-Structure for Curves

In Algorithm [3] we use a polygonal partitioning such that the part of the curve inside each partition is monotone with respect to at least one boundary of that cell, which we call a curve-monotone partitioning. Then, we use a combination of point-location in polygonal subdivisions and binary search on the ordered set of polynomial arcs bounded by a segment of the subdivision (Algorithm [4]).

2.1 A Curve-Monotone Polygonal Subdivision for Disjoint Arcs of Polynomials of Maximum Degree 2

After breaking the input into a set of disjoint $xy$-monotone convex/concave curves, Algorithm [1] computes a subdivision of the plane, such that the order of the curves in each cell is monotone with respect to each cell boundary.

To do this, first compute the bounding box of each curve. Some of the bounding rectangles might be intersecting, or inside one another:

(a) For a sequence of boxes inside one another (nested rectangles), merge their boxes by considering the outermost bounding box only, if there are no other rectangles outside this sequence inside them.

(b) For intersecting boxes, add the tangents to the curves at the points of intersections with bounding rectangles of other curves and their own. Then, starting from a point on the boundary of the intersected area, move towards one of the curves, until hitting the other curve’s bounding box or tangent, then repeat this on the other curve. If it is the intersection of two curves, draw the common tangent of the curves to separate them.

(c) For disjoint rectangles inside another rectangle, we divide them by extending the edges of the rectangles inside the bigger rectangle, until intersecting the boundary of the extension of an edge of another rectangle.
(d) For disjoint rectangles, extend their edges until intersecting the boundary or the extension of an edge to create a set of disjoint rectangles.
(e) For intersecting nested bounding boxes, first handle the intersecting segments, then handle the disjoint and then the nested rectangles.

An example of these cases is shown in Figure 1.

Fig. 1. Four possible arrangements of bounding boxes and their subdivisions.

Algorithm 1 Curve-Monotone Subdivision

Input: A set of curves \(P_1, \cdots, P_n\)

Output: A subdivision of the plane such that the distances of \(\{P_i\}_{i=1}^n\) in each cell can be uniquely defined.

1: for \(i = 1, \cdots, n\) do
2: \(Q \leftarrow Q \cup \{\text{Break each curve } P_i \text{ into } xy\text{-monotone totally concave or totally convex parts}\}\).
3: end for
4: \(A = \text{Build the arrangement of curves } Q\).
5: \(B = \text{Connect the intersections and vertices using straight line segments}\).
6: \(C = \text{Partition } B \text{ by drawing the bounding boxes of the curves in } Q \text{ or the mutual tangents of the curves in } Q \text{ when their bounding boxes intersect, except for the curves that are inside another curve}\).
7: Triangulate the cells of \(C\) with complexity more than 3.
8: Store for each segment \(s\) the list of curves inside cell \(c\) intersecting it as \(L_{s,c}\) along with the index of their adjacent cells.
9: return \(C\) and \(L_{s,c}, s \in c, c \in C\)

2.2 A Subdivision For Disks

Algorithm 2 computes a planar subdivision for a set of arbitrary disks. An example of the possible cases of Algorithm 2 is shown in Figure 2. The size of the arrangement is \(O(n^2)\), since at most \(\binom{n}{2}\) lines connect the centers of the disks, there are \(4\binom{n}{2}\) mutual tangents, and the bounding box adds at most 4 vertices and therefore a multiple of at most 4. Because of the triangulation, the cell complexity is at most 3.
Algorithm 2 Subdivision for Disks

Input: A set of disks $P_1, \ldots, P_n$
Output: A subdivision of the plane $D$

1: $A =$ Build the arrangement of the disks $P_1, \ldots, P_n$.
2: $B =$ Build the bounding box of the input, connect the intersections of the disks, their pairwise tangents and their centers to each other and to the boundary with straight lines.
3: $C =$ Triangulate the cells of $B$ with complexity more than 3.
4: Store for each segment $s$ the list of curves inside cell $c$ intersecting it as $L_{s,c}$ along with the index of their adjacent cells.
5: return $C$ and $L_{s,c}, s \in c, c \in C$

Fig. 2. Four examples of the subdivisions of Algorithm 2 for a set of disks.

2.3 The Point-Location Algorithm

Assuming a curve-monotone planar subdivision is given, Algorithm 3 builds a data-structure for the point-location problem. Assuming the algorithm for constructing the curve-monotone planar subdivision has time complexity $T(n)$, space complexity $S(n)$ and maximum cell complexity $C(n)$, Algorithm 3 takes $T(n) + O(S(n) \log S(n))$ for building the subdivision, sorting and constructing the point-location data-structure.

An example of a valid partitioning for the point-location using Algorithm 3 is shown in Figure 3.

Theorem 1. Algorithm 3 using Algorithm 7 for computing the subdivision has time complexity $O(n^4 \log(n))$ and space complexity $O(n^4)$. 
Point-Location in The Arrangement of Curves

Algorithm 4 Query

Input: A point \( q \), a point location data structure \( D \) augmented by a set of lists \( L_{s,c} \)

Output: The cell of \( D \) that contains \( q \)

1: \( c = \) The output of point location of \( q \) in \( D \).
2: \( X = \emptyset \)
3: for \( s \in \) the boundary of \( c \) do
4: \( X \leftarrow X \cup \) Binary search on the lists \( L_{s,c} \), using the polynomial’s equation to detect above/below to find the cell of the arrangement.
5: end for
6: return the only non-empty result in \( X \).

Proof. The sizes of intermediate structures is as follows:

\[ |Q| = O(n), \ |A| = O((n^2)^2), \ |B| = |A|, \ |C| \leq |B|^2 = O(n^4), \sum_{s \in C, c \in C} |L_{s,c}| = O(|C|) \]

Triangulating the regions does not increase the overall complexity, since this adds a linear number of edges.

Breaking each polynomial into monotone parts takes \( O(1) \) time. Computing \( D \) takes \( O(|C| \log |C|) = O(n^4 \log(n)) \) time. Triangulating the regions takes linear time, and sorting the set of segments in each cell takes \( O(|C| \log |C|) \). So, the time complexity is dominated by the time complexity of computing \( D \).

Theorem 2. Algorithm 4 has time complexity \( O(\log n) \).

Proof. The point-location in \( D \) takes \( O(\log(n)) \) time, since it is a polygonal point location for an input of size \( O(n) \). Based on Algorithm 3, the complexity of each cell is at most 4, and each list has size at most \( n \), so the binary searches take \( O(\log(n)) \) time. So, the total time complexity of a query is \( O(\log n) \).

3 Open Problems

Approximate Point-Location. While building an exact point-location data-structure might be expensive in terms of time complexity, it is possible to build approximate point-location data structures by enclosing the original curve between two
simpler curves. Then, the exact point-location data-structure needs to be used for the area between the enclosing simplified curves. For the one-sided error, such as never reporting a point inside (outside) the shape as outside (inside), the data-structure built on the bounding (enclosed) curve is enough to get an approximate solution.

**Application: Point-Location among Contour Lines.** For a given objective function and its discretization, the contour lines separate the input into regions such that the value of the objective function is the same between two contour curves. By building a point-location data structure on such an arrangement, it is possible to get the values with error dependent on the discretization used when building the contour lines. An example of this method exists for the length query problem [1].

**References**

1. Aghamolaei, S., Keikha, V., Ghodsi, M., Mohades, A.: Windowing queries using Minkowski sum and their extension to MapReduce. Journal of Supercomputing (2020)
2. De Berg, M., Van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational geometry. In: Computational geometry, pp. 1–17. Springer (1997)
3. Shor, P.: Stretchability of pseudolines is NP-hard. Applied Geometry and Discrete Mathematics–The Victor Klee Festschrift (1991)
4. Toth, C.D., O’Rourke, J., Goodman, J.E.: Handbook of discrete and computational geometry. Chapman and Hall/CRC (2017)