Computing Arrangements using Subdivision and Interval Arithmetic

Younis O. Hijazi and Thomas M. Breuel

Abstract. Computing arrangements of curves is a fundamental and challenging problem in computational geometry, with many practical applications in a wide range of fields, including robot motion planning and computer vision. This paper describes a method for computing arrangements of implicitly defined curves. Our method for computing arrangements is an adaptation of methods successfully used for the exploration of large, higher dimensional, non-algebraic arrangements in computer vision. While broadly similar to subdivision methods in computational geometry, its design and philosophy are different; for example, it replaces exact computations by subdivision and interval arithmetic computations and prefers data-independent subdivisions. It can be used (and is usually used in practice) to compute well-defined approximations to arrangements, but can also yield exact answers for specific problem classes.

§1. Introduction

Arrangements are subdivisions of Euclidean space created by multiple lower-dimensional surfaces, and are widely used in robotics, computer graphics, molecular modeling, and computer vision. For survey papers on arrangements, see [13, 1, 14]. The output of an arrangement algorithm is often the *arrangement graph*, the planar graph in which nodes correspond to intersections of curves, and edges correspond to curve segments joining the nodes [4].

The first algorithms for computing planar arrangements of lines were based on *sweeps*, an enumeration of the geometric structures encountered when a line–often parallel to the y-axis–is moved (swept) across the plane [3, 10]. Recent results tend to generalize such methods to more general classes of curves such as conics/cubics [11], and algebraic curves [16]. Subdivision methods have also recently found increasing interest for the

Conference Title
Editors pp. 1–6.

ISBN 0-0-9728482-x-x

computation of arrangements [15, 22, 8]. Closely related to subdivision methods for the computation of arrangements are subdivision methods for the computation of implicit curves and/or surfaces [20, 19], intersections of Bézier curves [23], intersections of surfaces [2], and ray tracing [17, 9].

In computer vision, exploration of arrangements by subdivision methods has been used for computing globally optimal solutions to geometric matching problems under bounded error [5, 6].

This paper describes an algorithm for the computation of exact or approximate arrangement graphs of a collection of implicitly defined curves in the plane using a subdivision method and interval arithmetic; only the static case is considered.

§2. The CAPS algorithm

In this section, we present the CAPS (Curves Arrangements by Planar Subdivisions) algorithm which computes the arrangement of curves, i.e. a subdivision of the plane that consists of vertices (0-cells), edges (1-cells), and faces (2-cells). We assume that curves are defined implicitly using equations $f_i(x, y) = 0$ and that their inclusion functions are convergent [18].

An *implicit curve* is the set of zeroes of a function $f : \Omega \subseteq \mathbb{R}^2 \to \mathbb{R}$ (where Ω is an open subset of $\mathbb{R}^2$). Evaluations of f will be carried out in terms of interval arithmetic, so $z = f_{[\,]}(x, y)$ (sometimes written as just $f(x, y)$) stands for $[\underline{z}, \overline{z}] = f_{[\,]}([\underline{x}, \overline{x}], [\underline{y}, \overline{y}])$. Here, $f_{[\,]}(x, y)$ is a natural inclusion function [18] corresponding to a real-valued function $f(x, y)$; it satisfies $f_{[\,]}(x, y) \supseteq \{z : z = f(\xi, \eta), \xi \in x, \eta \in y\}$, and any sufficiently well-behaved f converges towards the real value as its argument intervals shrink. Note that f has a zero within a box $[\underline{x}, \overline{x}] \times [\underline{y}, \overline{y}]$ only if $0 \in [\underline{z}, \overline{z}] = f_{[\,]}(x, y)$. Our general approach towards exploring the arrangement will be to evaluate f on nested families of intervals, excluding boxes from further consideration if we can prove that they do not contain zero.

The input of the CAPS algorithm consists of a collection of implicit curves and an initial bounding box (optionally, additional curves may be added for handling intersections at infinity), and recursively invokes the `classify` function. This function then computes the set of curves that might cross the bounding box.

If provably none of the curves do, the function does not recurse further. If there is exactly one candidate that may intersect the box, the algorithm attempts to prove that this candidate actually intersects the box (below), and, depending on the result, marks the box as either empty, containing one curve, or indeterminate. If there are more than one candidate curve intersecting the box, and the algorithm has not reached the maximal recursion depth, the algorithm subdivides the box and recurses; subdivision is by simple bisection of each dimension of the bounding box

(more complex or data-dependent schemes are possible but do not seem to improve performance in preliminary experiments). If the search has reached the maximum recursion depth, it does not subdivide further and instead terminates and marks the box as indeterminate.

Proving that a box contains zero curves can be done simply by demonstrating that $f_{[\,]}(x, y)$ does not contain zero for any of the curves. In order to determine that a curve actually intersects a box, it is not sufficient to show that $f_{[\,]}(x, y)$ contains zero for the box; rather, we need to show that the real-valued function $f_{[\,]}(x, y)$ actually changes sign somewhere within the box. We can do this by identifying a subregion where the function is provably positive ($\underline{z} > 0$) and another subregion where the function is provably negative ($\overline{z} < 0$). Therefore, the algorithm recursively subdivides the box until it either finds that all subregions are strictly positive or strictly negative (non-intersecting), until it obtains an example of a strictly positive and a strictly negative region (intersecting), or until the box size is below a threshold (indeterminate).

As part of its search, the box classification algorithm constructs a quad-tree decomposition of the original box, with each leaf in the quad-tree labeled as containing zero curves, one curve, or indeterminate. For each leaf, the algorithm also records the curves that intersect or potentially intersect the corresponding box. In order to compute the actual arrangement graph, we view this labeled quad-tree as a subdivision of the plane and apply a standard region labeling and region adjacency graph algorithm for quad-trees. The label for each quad-tree node consists of the set of curves for which the CAPS algorithm could not prove that the curve fails to fall outside that node. In this process, all adjacent quad-tree nodes with the same label (i.e., the same set of curves) are grouped together into regions (for the purposes of this exposition, assume that no curve passes exactly through the corner of a quad-tree node and define connectedness analogous to four-connectedness in image processing). The labeled regions are transformed into a graph using the following general approach (some special cases omitted):

- each region containing zero curves is omitted (it is part of the dual graph),
- each region labeled with exactly one curve is transformed into an edge,
- each region labeled with more than one curve is transformed into a vertex and associated with all its adjacent regions representing curves.

Optionally, we can additionally compute, for each region containing exactly one curve, a polygonal or spline approximation of the curve passing through that region, as shown in [21].

In general, the graph computed by this method will not be the arrangement graph as commonly defined in computational geometry. Let us consider some of the differences.

Most importantly, if the input curves permit self-intersections, these self-intersections may or may not not be present in the computed graph. In practice, self-intersections are often *a priori* impossible because of the nature of the input curves. Furthermore, in many applications, computation of self-intersections are not required, and we can view the graph computed by the CAPS algorithm as a well-defined transformation of the arrangement graph. When self-intersections are both possible and desired, the CAPS algorithm can be modified to compute them by introducing an additional subdivision scheme (not described here).

For regions containing exactly two curves, there are many different ways in which these curves could meet inside the region. Contacts and even numbers of intersections are distinguished from odd numbers of intersections based on the topology of the neighboring regions. The considerations for distinguishing contacts and single intersections on the one hand, and multiple intersections on the other, are analogous to self-intersections: they can usually be excluded based on the class of curves under consideration, they may not be of interest, and/or they could be calculated explicitly using additional subdivision techniques.

Subject to these considerations, we can claim a number of different, exact results like the following: *If the CAPS algorithm is applied to curves that do not self-intersect and have no multiple intersections, and if it yields a graph in which all nodes correspond to regions containing no more than two curves and in which no contacts occur, then the graph is an arrangement graph for the curves.* Note that we can control whether the CAPS algorithm yields such graphs through when we terminate the search, and we can find analogous statements about CAPS-like algorithms modified to cope with self- and multiple intersection.

For many practical applications, however, approximations of the following form are of greater interest: *If the CAPS algorithm expands nodes to a terminal size of $\frac{\delta}{2}$, it yields an arrangement graph for an arrangement of curves obtained by continuously distorting the original arrangement by no more than δ at each point.* (This claim, of course, would require a proof; not shown). We can view this as a *δ-weak* solution [12] for the arrangement problem, meaning that it represents a solution that is obtainable from the true solution through a small geometric perturbation (note that this is not necessarily the same as a small perturbation of the curves themselves). In practical applications (like geometric matching) weak solutions for well-defined accuracies have turned out to be sufficient, and this is how these methods have been used in practice [7].

§3. Complexity

In this section, we consider some informal analyses of the average-case and worst-case combinatorial complexity of the CAPS algorithm in terms of

the output complexity of different classes. As a measure of complexity, we use the tree size. The threshold where we stop splitting is called ϵ, and we denote by L the size of the initial box's side, and d, the depth of the quadtree, given by $d = \log_2(L\epsilon^{-1})$.

3.1. The linear case

Let us say that lines l_1 and l_2 are ϵ-close if one of the two l_1 and l_2 are parallel and $distance(l_1, l_2) \leq \epsilon$ or if l_1 and l_2 intersect and $angle(l_1, l_2) \leq \epsilon$.

Close lines trigger the worst-case behavior of the CAPS algorithm, since it requires exploration of large number of boxes. If two lines are ϵ-close, we need $O(\epsilon^{-1})$ boxes to cover the gap, and $O(n\epsilon^{-1})$ for n lines, and this dominates the tree.

Of course, in the linear case, we can easily use exact methods for determining whether the lines actually intersect. Alternatively, we can use recursive interval arithmetic methods, but searching, say, for an intersection of the two lines directly rather than by covering the gap. Nevertheless, the analysis of the linear case provides a good introduction for the analysis of the more general case.

In order to get some idea of the average case, let us classify the relationships in which two lines l_1 and l_2 can appear: there are no intersection inside the bounding box explored by the search (C_1), there is one intersection with $angle(l_1, l_2) > \epsilon$ (C_2), and there exist ϵ-close lines (C_3). We assume that the expected value of the lines' configuration C gives us the average-case complexity. If we denote by c_i the previously listed configurations' complexity and by p_i their associated probabilities, we have $E(C) = \sum_i p_i c_i$. If we are in case C_1 (meaning no intersection) the tree will be very small and the algorithm will terminate very quickly, in constant time; so, c_1 is $O(1)$. For C_2, the complexity is the size of the tree, given by $d = \log_2(L\epsilon^{-1})$, i.e. $O(\log(\epsilon^{-1}))$. Finally, c_3 is in $O(\epsilon^{-1})$ as previously shown in the worst-case scenario.

Now that we were able to express the complexity of each situation, we would like to know how likely it is to happen and therefore compute its probability. We first examine the one-dimensional case: if parameters are within the interval $[-K, K]$, equally likely, we get, for the two random variables k_1 and k_2 : $p(k_1) = p(k_2) = \frac{1}{2K} = O(1)$ and we can then prove that $p(|k_1 - k_2| < \epsilon) = \frac{\epsilon}{2K}$. Considering the case of lines in $2D$, we need to define a distribution. Within the bounding box, we represent a line by two parameters: a point $M = (x, y)$ and an angle α between 0 and 2π. The same way as in the one-dimensional case, if we have random variables k_1 and k_2 in the box $[-K, K] \times [-K, K]$ (equally likely) we obtain $p(k_1) = p(k_2) = \frac{1}{4K^2} = O(1)$. This leads us to $p_1 = p_2 = O(1)$.

We are now interested in the probability that the ϵ-close case occurs. For two lines (M, α_1) and (N, α_2) we have $p(\mid \alpha_1 - \alpha_2 \mid \leq \frac{\epsilon}{2L}) \frac{\epsilon}{2L} = \frac{\epsilon}{4\pi L} \frac{\epsilon}{2L} = O(\epsilon^2)$. We finally get that probabilities p_1 and p_2 are $O(1)$ and p_3 is $O(\epsilon^2)$

and thus $E(C) = \sum_i p_i c_i = O(1)O(\log(\frac{1}{\epsilon})) + O(\epsilon^2)O(\frac{1}{\epsilon}) = O(\log(\frac{1}{\epsilon}))$. This analysis has been done for only two lines. For n lines we have an upper bound of $\frac{n(n-1)}{2} = O(n^2)$ intersections and thus conclude that the average-case combinatorial complexity for n lines is $O(n^2 \log(\frac{1}{\epsilon}))$. Notice that despite the term n^2 in the average-case complexity, the worst-case complexity is much worse than the average one because of the term $\frac{1}{\epsilon}$.

3.2. Algebraic curves and other families

A planar algebraic curve can be represented as $P(x, y) := \sum_{i,j} p_{ij} x^i x^j = 0$, where $p_{ij} \in \mathbb{R}$. The degree d of an algebraic curve is $d = max_{i,j}(i+j)$. For instance, the algebraic curve defined by the equation $2x^3y + 5xy^2 - y - 1 = 0$ is of degree 4. Bézout's theorem tells us that for two algebraic curves of degrees p and q respectively, the number of intersection points of those two curves is bounded by the product of the degrees, i.e. $N = pq$.

If there are no ϵ-close curves and suppose the problem locally linearized, the combinatorial complexity of two algebraic curves of degrees respectively p and q would be in $O(pq \log(\frac{1}{\epsilon}))$. Thus, for n algebraic curves the complexity is $O(K_n \log(\frac{1}{\epsilon}))$ where K_n is the result of summing up two by two all possible Bézout bounds of the input algebraic curves, which is both quadratic in n and the degrees of the curves: $K_n = \sum_{i \neq j} d_i d_j$ where d_i is the degree of the ith input algebraic curve and (i, j) are all the possible pairs within $1, ..., n$.

We can now ask what the probability of the ϵ-close configuration is on average. Therefore, consider a mapping from the product of the parameter spaces of all input curves to a given bounding box, sub-space of $\mathbb{R}^2$. We wish to get some idea of how frequent close position and general position cases are, since the non-intersecting close position cases are particularly hard for our approach. There are three cases we need to distinguish: an intersection with an angle larger than a fixed epsilon, contacts, or close, non-intersecting curves.

Contacts exist only for a set of parameter values with measure zero. Intersections at large angles, and ϵ-close approaches, on the other hand, exist for parameter regions with finite measure (i.e., greater than zero). However, if the mappings from parameter space to curves is sufficiently smooth, these parameter regions will still have a small measure for small ϵ, so that for sufficiently smooth distributions of input problems, the probability of computationally costly cases of arrangements of curves is low.

The above arguments are merely a sketch of a possible average case analysis of the complexity of CAPS-style algorithms for classes of curves or surfaces; a detailed analysis remains to be carried out.

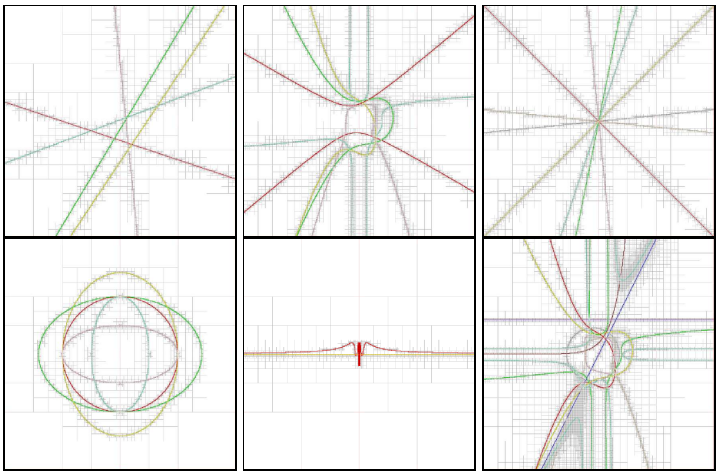

Fig. 1. From top left to bottom right: Arrangement of lines, algebraic curves, degenerate lines, degenerate polynomials, inverse sine, and arbitrary curves.

§4. Experimental Results

Running times for the algorithm are shown in Table 1, corresponding to curves of Figure 1. We can see that the algorithm performs well for straight lines and polynomials, and can handle difficult curves such as inverse sine.

Class of curves	Quadtree size	Runtime (s)
Lines (5)	12097	0.8
Algebraic curves (5)	32875	1.3
Degenerate (7 lines)	19546	1.9
Degenerate (5 polynomials)	251440	2.8
Trigonometric functions (3)	12271	0.2
Inverse Sine (2)	20800	12.6
Arbitrary curves (10)	78736	7.5

Tab. 1. Computation time for the quad-tree structure (C++, Xeon 3.6Ghz) and different families of curves. Parameters used were $\epsilon = 10^{-6}$ and an initial bounding box of $[-10, 10] \times [-10, 10]$.

§5. Discussion

The original motivation for developing CAPS-like methods was the solution of problems in computer vision for which sweep methods were not practical: subdivisions of $\mathbb{R}^4$ or $\mathbb{R}^6$ based on implicitly defined curves involving polynomials and trigonometric functions, and involving often thou-

sands of surfaces. This paper has described an application of those ideas to the problem of computing arrangements of implicitly defined curves, permitting us to compare CAPS-style approaches with commonly-used approaches from computational geometry (including subdivision and approximate methods developed in the context of computational geometry).

First, while CAPS can be used for exact computations for some problem types and instances, CAPS can also be used to obtain well-defined approximate (δ-weak) solutions through early termination. Such solutions are often sufficient for practical applications and yield well-characterized approximations for curves for which no exact methods are known or feasible.

CAPS is designed to use floating point arithmetic instead of exact computations, even when exact computations are possible (say, for algebraic curves), but can still return well-defined solutions through the use of interval arithmetic. Either approach can be used for computing exact or approximate arrangement graphs; which approach is preferable in practice depends on the domain and remains to be determined.

For practical applications, the combination of the CAPS-approach of a strict hierarchical exploration with faster intersection tests for pairs of curves may be desirable. Initial experiments (not shown) suggest that the use of such exact tests results in speedups for simple cases (e.g., linear), but may be more costly (and is often simply impossible) for more difficult cases (e.g., algebraic or trigonometric).

Our experience with CAPS-like algorithms and design choices suggests that they are more efficient and practical than traditional algorithms for computations involving arrangements in important cases. We'd like to remind the reader that the original motivation for CAPS-style algorithms was that algorithms from computational geometry for computations involving arrangements were either too slow, or simply not applicable to the kinds of arrangements encountered in computer vision. We hope that this paper will be only the first step in a more careful exploration of the similarity and differences between CAPS-like algorithms and algorithms developed in the traditional framework of computational geometry, and that it may lead to reconsideration of some design choices and assumptions often made in geometric algorithms research.

Acknowledgments. This work was supported by the German Science Foundation (DFG IRTG 1131) as part of the International Research Training Group on "Visualization of Large and Unstructured Data Sets - Applications in Geospatial Planning, Modeling, and Engineering". Many thanks to Christoph Garth, Christoph Lampert, and Burkhard Lehner for the fruitful discussions and their help for this paper and to Hans Hagen, Bernd Hamann, Ken Joy, and Georg Umlauf for their support.

References

1. Agarwal, P. K., and Sharir, M., *Arrangements and their applications*, Handbook of Computational Geometry (J. Sack, ed.), 49-119, 2000.
2. Barnhill, R. E., and Kersey, S. N., A marching method for parametric surface/surface intersection, Computer Aided Geometric Design **7**, 257–280, 1990.
3. Bentley, J. L., and Ottmann, T. A., Algorithms for reporting and counting geometric intersections, IEEE Trans. Comput. **C-28**(9), 643–647, 1979.
4. Bose, P., Everett, H., and Wismath, S., Properties of arrangements graphs, International Journal of Computational Geometry and Applications **13**(6), 447–462, 2003.
5. Breuel, T. M., Fast recognition using adaptive subdivisions of transformation space, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 445–451, 1992.
6. Breuel, T. M., On the use of interval arithmetic in geometric branch and bound algorithms, Pattern Recogn. Lett. **24**(9-10), 1375–1384, Elsevier Science Inc., 2003.
7. Breuel, T. M., Implementation techniques for geometric branch-and-bound matching methods, Computer Vision and Image Understanding **90**(3), 258–294, 2003.
8. Carvalho, P. C., de Figueiredo, L. H., and Cavalcanti, P. R., Computing arrangements of implicit curves, Extended abstract in Anais do VERMAC, 19–22, 1998.
9. Duff, T., Interval arithmetic and recursive subdivision for implicit functions and constructive solid geometry, SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques, 131–138, 1992.
10. Edelsbrunner, H., and Guibas, L. J., Topologically sweeping an arrangement, J. Comput. Syst. Sci. **38**, 165–194, 1989.
11. Eigenwillig, A., Kettner, L., Schoemer, E., and Wolpert, N., Complete, exact, and efficient computations with cubic curves, 20th Annual ACM Symposium on Computational Geometry, 409–418, 2004.
12. Groetschel, M., Lovasz, L., Schrijver, A., *Geometric algorithms and combinatorial optimization*, Springer-Verlag, 1988.
13. Halperin, D., *Arrangements*, Jacob E. Goodman and Joseph O'Rourke, editors, Handbook of Discrete and Computational Geometry, chapter 21, 389–412, 1997.
14. Hijazi, Y. O., Arrangements of planar curves, Hans Hagen, Andreas Kerren, and Peter Dannenmann (Eds.), Visualization of Large and Un-

structured Data Sets, GI-Edition Lecture Notes in Informatics (LNI), Vol. S-4, 59–68, 2006.

15. Liang, C., Mourrain, B., and Pavone, J. P., Subdivision methods for 2d and 3d implicit curves, to appear in Computational Methods for Algebraic Spline Surfaces, Springer-Verlag, 2006.
16. Milenkovic, V., and Sacks, E., An approximate arrangement algorithm for semi-algebraic curves, SCG '06: Proceedings of the twenty-second annual symposium on Computational geometry, 237–246, 2006.
17. Mitchell, D., Robust ray intersection with interval arithmetic, Proceedings on Graphics Interface 1990, 68–74, 1990.
18. Moore, R. E., *Interval Analysis*, Prentice Hall, Englewood Cliffs, NJ, 1966.
19. Paiva, A., Lopes, H., Lewiner, T., and de Figueiredo, L. H., Robust adaptive meshes for implicit surfaces, 19th Brazilian Symposium on Computer Graphics and Image Processing, 205–212, 2006.
20. Plantinga, S., and Vegter, G., Isotopic approximation of implicit curves and surfaces, SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing, 245–254, 2004.
21. Sederberg, T. W., and Farouki, R. T., Approximation by interval Bézier curves, Computer Graphics and Applications **12**, 87–95, 1992.
22. Wintz, J., Mourrain, B., Subdivision method for computing an arrangement of implicit planar curves, In proceedings of the Algebraic Geometry and Geometric Modeling 2006 conference, 2006.
23. Yap, C. K., Complete subdivision algorithms, I: intersection of Bézier curves, SCG '06: Proceedings of the twenty-second annual symposium on Computational geometry, 217–226, 2006.

Younis O. Hijazi and Thomas M. Breuel
International Research Training Group 1131
DFKI and University of Kaiserslautern
Erwin Schroedinger Strasse
67608 Kaiserslautern, Germany
hijazi@iupr.org tmb@iupr.net