Finding Intersections of Algebraic Curves in a Convex Region using Encasement

Joseph Masterjohn* Victor Milenkovic† Elisha Sacks‡

Abstract

We present a subdivision based technique for finding the intersections of two algebraic curves inside a convex region. Even though it avoids computing resultants, the technique is guaranteed to find all intersections with bounded backwards error. The subdivision, called an *encasement*, also encodes the arrangement structure of the curves. We implement the encasement algorithm using adaptive precision interval arithmetic. We compare its performance to the CGAL library implementation of resultant based curve intersection techniques. We provide CPU and CPU/GPU versions of the algorithm and implementation. On the CPU, encasement generates all curve intersections, to accuracy 10^{-8}, 10 to 30 times faster than CGAL for degrees 8 to 18, and it handles degrees up to 20 that CGAL cannot handle. The GPU speeds up the calculation by a factor of 3 to 4.

1 Introduction

An algebraic curve f is the zero set of a bivariate polynomial $f(x, y)$. Given a convex polygon B, we find all intersections of curves f and g inside B. Curve intersection is a core geometric calculation. It is a key step in calculating the *arrangement* of n curves $f_1, \ldots, f_n$: a partition of B into intersection vertices, open curve segments, and open regions. We have in mind scientific or industrial applications, which provide the polynomial coefficients as floating-point numbers. Broadly speaking, numerical programs use double-float for calculations and aim for single-float accuracy in the output. An arrangement with accuracy $\delta = 10^{-8}$ would more than satisfy the latter. Nevertheless, exact arrangement computation is required to support CG algorithms that manipulate arrangements. These algorithms require the signs of predicates evaluated on the vertices of the arrangement. An incorrect sign can lead to program failure or to nonsensical output. This is the robustness problem of Computational Geometry.

Exact Computational Geometry (ECG) uses extended precision and algebraic algorithms to determine the signs of primitives. This approach can be numerically expensive even when heuristics are used, such as floating point filtering. In the case of curve arrangements, an exact algorithm requires construction of resultant polynomials. These have high degree and bignum coefficients.

Numerical methods, such as subdivision and curve tracing, are often stymied by ill-conditioned inputs. Usually, the subdivision is by axis-parallel lines, which can require a large number of cuts to separate features. Curve tracing is faster but is even less reliable.

1.1 Prior Work

Algebraic methods compute the turning points and the intersection points of bivariate polynomial curves via resultants and other algebraic computation. For example, the CGAL arrangement package [5, 15] implements a sweep algorithm for plane algebraic curves using Exacus [4].

Subdivision methods [7, 1] provide a faster means to isolate the intersection points of algebraic curves and to trace algebraic curves, but they cannot guarantee correctness and are prone to failure on ill-conditioned inputs. They use convex bounding polyhedra during intersection isolation, but the outputs are axis-parallel enclosures. They typically operate on polynomials given in the Bernstein-Bézier basis and involve a non-robust numerical subdivision phase followed by a robust (no false positive) certification phase on candidate intersections [9]. They can isolate the vertices and edges of an arrangement, but an axis-parallel enclosure requires $\Omega(1/\epsilon)$ cells for ϵ-separated curves (ϵ distant under the Hausdorff metric). Other work focuses on improving the efficiency rather than reliability of the subdivision phase through the use of low degree approximations [3], blending schemes for quick elimination of regions containing no roots [2], and deflation techniques [13].

Wang, Chiang, and Yap [14] formalize resolution-exact subdivision methods for motion planning, but this work is also limited to axis-parallel enclosures.

*Department of Computer Science, University of Miami, joe@cs.miami.edu

†Department of Computer Science, University of Miami, vjm@cs.miami.edu

‡Computer Science Department, Purdue University, eps@cs.purdue.edu

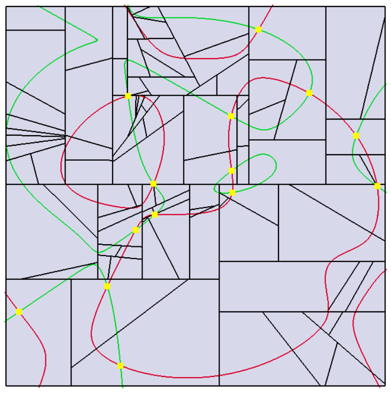


Figure 1: Encasement of f (red) with respect to g (green) and their intersections (yellow) in B (bounding square).

1.2 Encasement-Based Intersection Construction

We present an algebraic curve intersection algorithm based on *convex encasement*. The algorithm combines subdivision methods with exact computational geometry to achieve both efficiency and guaranteed accuracy.

Algebraic curves f and g are generic (in general position) if they are nonsingular (i.e. no solution to $f(x, y) = f_x(x, y) = f_y(x, y) = 0$) and have no tangent intersections. A *convex encasement* of f with respect to g in a convex polygonal region B is a partition of B into convex polygonal cells such that 1) no cell contains a loop of f or g or more than one segment of f, 2) if a cell intersects both f and g, it contains a single intersection point of f and g (Fig. 1).

An encasement isolates the components of f and the intersections of f and g. The arrangement in B of a set of curves F can be reduced to the encasement in B of each pair f, g from F (Sec. 7.1).

1.2.1 Intersection Algorithm Summary

The curve intersection algorithm (Sec. 7) takes two bivariate polynomials as inputs, perturbs the coefficients by $\delta = 2^{-26} \approx 10^{-8}$, and constructs an encasement of the corresponding generic algebraic curves f and g by recursive subdivision of B by straight lines. If a cell C violates the definition of encasement, for example by containing a loop of f, the algorithm splits C by a line L. The following summarizes the selection of L with details in the indicated sections.

Loop splitting (Sec. 2) Construct a *critical set* S for f in B: S does not intersect f and contains all local extrema of $f(x, y)$. Since a loop of f must surround an extremum, it must surround a connected component of S. If a cell C contains a connected component of S, L is selected to intersect it and hence splits any loop surrounding it (Fig. 2). Likewise g.

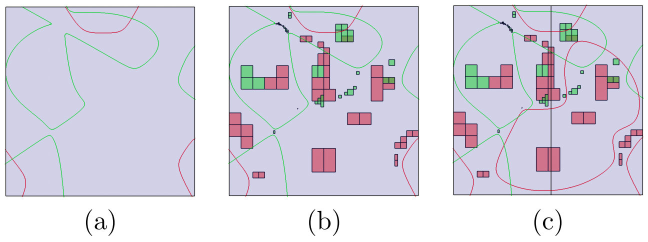


Figure 2: Curves f (red) and g (green) with undetected loop not shown (a). Critical regions (b) (colors match curves). Splitting lower middle f-region with vertical line reveals and splits missing loop (c).

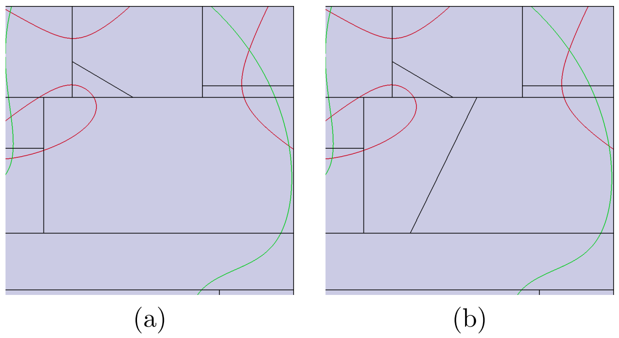


Figure 3: Cell with two segments (red) of f and one segment (green) of g (a). Segments of f separated by splitting line (b).

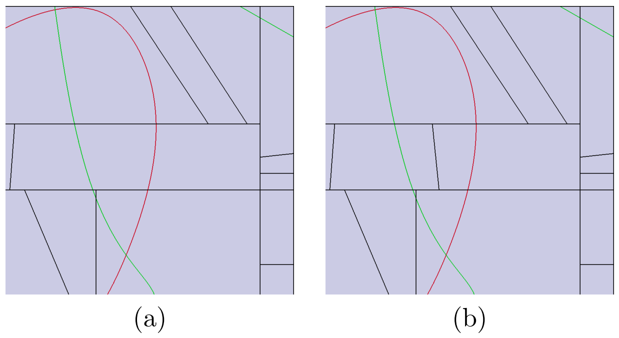


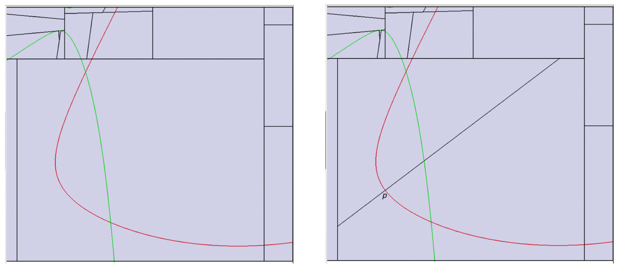
Figure 4: Cell (center) with non-intersecting segments of f and g (a). Separated by splitting line (b).

Self-separation (Sec. 3) If C contains more than one segment of f, L is selected to separate one segment from another (Fig. 3).

Curve separation (Sec. 4) If f has a single segment ab in C and no intersections with g, L separates f from g (Fig. 4).

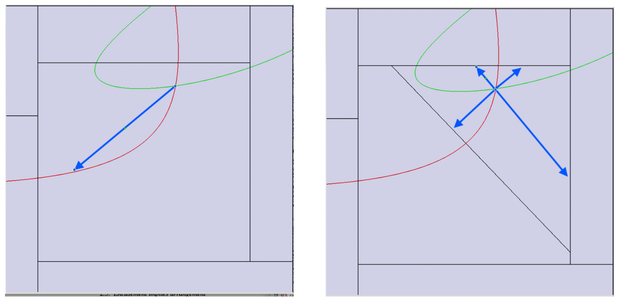
Intersection separation (Sec. 4) If ab intersects g an even number of times inside C, L splits C between two of the intersections (Fig. 5).

Intersection isolation and encasement (Secs. 5 and 6) If ab intersects g an odd number of times in C, we construct an axis-parallel rectangle $R \subset C$ containing a



(a) (b)

Figure 5: Cell with two intersections (a). Separated by splitting line through $p \in f$ in direction of $\nabla f(p)$ at a local minimum of $g(p)$ on f (red) (b).



(a) (b)

Figure 6: Segment g (red) blocks the intersection's "view" of boundary along angle bisector (a). Angle bisectors reach boundary of smaller cell proving intersection is unique (b).

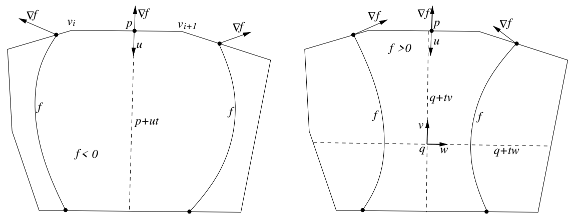
single intersection. This is standard zero isolation using a 2D interval: R is not a cell. Split C by up to four lines to encase that intersection in a cell that excludes all other intersections (Fig. 6).

Self-separation and curve separation might not be possible using a single split if the two segments are close and curved. In that case, multiple splits are required.

1.2.2 Contribution

Encasement based curve intersection improves on prior work in several ways. We ensure correctness, with accuracy δ, for all inputs by perturbing polynomials to remove singular points and tangent intersections then employing adaptive precision interval arithmetic. Replacing boxes with convex polygons reduces the space complexity for ϵ-separated curves to $\Omega(1/\sqrt{\epsilon})$. The number of splits, other than for self-separation or curve separation, is in $O(d^2)$, for d the maximum degree of f and g. We introduce a stronger criterion for showing that a cell contains a single intersection.

On the CPU, encasement generates all curve intersections, to accuracy δ, 10 to 30 times faster than CGAL for degrees 8 to 18, and it handles degrees up to 20 that CGAL cannot handle. The GPU speeds up the calculation by a factor of 3 to 4.



(a) (b)

Figure 7: p minimizes $f(p)$ on v_iv_{i+1}, $f(p) < 0$, and direction u decreases $f(x, y)$. Split by $p + ut$ (a). p maximizes $f(x, y)$ and $f(p) > 0$ but direction u decreases $f(x, y)$. If this happens for every edge, C must contain a saddle point q. Split by $q + tv$ and $q + tw$, where v and w are the eigenvectors of the Hessian of $f(x, y)$ at q (b).

2 Critical sets

A region S is a *critical set* of a curve f with respect to B if it does not intersect f and it contains all local extrema of $f(x, y)$ in B. To construct a critical set, let R be the bounding rectangle of B. If we can show that $f(x, y)$ is nonzero in R, return R. If we can show that one of the partial derivatives $f_x(x, y)$ or $f_y(x, y)$ is nonzero in R (hence R does not contain an extremum), return $\emptyset$. Otherwise, bisect R across its longer dimension, recurse on the two halves, and return the the union of the results. Since $f(x, y)$ is nonsingular, the algorithm terminates.

We test if a polynomial is nonzero on a rectangle with a generalization of Descartes' rule of signs. If $R = [a, b] \times [c, d]$, the rational function $g(x, y) = f(1/(x + 1/(b - a)) + a, 1/(y + 1/(d - c)) + c)$, takes on the same set of values on $[0, \infty] \times [0, \infty]$ as $f(x, y)$ on R, and $g(x, y)$ is nonzero on $[0, \infty] \times [0, \infty]$ (hence $f(x, y)$ on R) if all the coefficients of $x^m y^n g$ have the same sign, where m and n are the degrees of $f(x, y)$ in x and y.

3 Self-separation

We can find the intersections of a curve f with the boundary of a cell C by substituting the parametric form $v_i + t(v_{i+1} - v_i)$ of each edge v_iv_{i+1} into $f(x, y)$ and solving for the zeros of the univariate in $t \in [0, 1]$ [11]. Since f has no loops in C after loop splitting, the number of segments of f inside C is half the number of intersections with the boundary. If there are more than two intersections, we split C in a manner that partially or completely separates at least one pair of segments.

For each clockwise oriented edge v_iv_{i+1} of the boundary, solve for all p such that $(v_i - v_{i+1}) \cdot \nabla f(p) = 0$

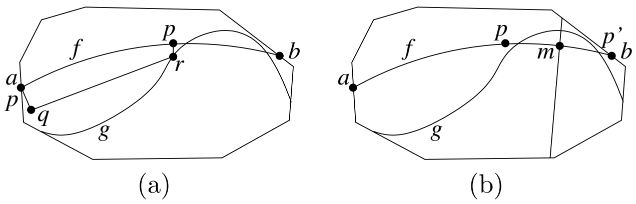


Figure 8: Chain from a (initial p) to q to r to p separates ap from g (a). $g(x,y)$ is decreasing at p and p' ($= b$) towards b and a, so we split at minimum m of $g(x,y)$ on f, separating two intersections with g (b).

(Fig. 7(a)). The vector $u = s\nabla f(p)$ with $s = \text{sign}((v_i - v_{i+1}) \times \nabla f(p))$ points inward. If $\text{sign}(f(p)) = s$, $|f(p+tu)|$, $t > 0$, increases at $t = 0$. We split by the line $p + tu$. If there is more than one such p, we choose the one between the closest pair of boundary intersections. If no such p exists on any edge, we claim that $\nabla f(p)$ makes at least one full counterclockwise turn as p traverses the boundary clockwise. This claim is a specialization of the generalized Poincaré-Hopf index theorem [8]. We isolate an intersection q of f_x ($f_x(x,y) = 0$) and f_y in a rectangle with the same property (Sec. 5), which implies that q is a saddle point (Fig. 7(b)). Let v and w be principle directions of f at q. We split by the lines $q + tv$ and $q + tw$.

4 Curve or intersection separation

If a cell C contains a single segment ab of f and $\text{sign}(g(a)) = \text{sign}(g(b))$, f crosses g an even number of times inside C. If there is a local minimum m of $g(x,y)$ on f, we expect that it separates two intersections, so we split at m. Some minima may not separate pairs of intersections, but there are at most $O(d^2)$ minima for d the maximum total degree of $f(x,y)$ and $g(x,y)$. Otherwise, we try to certify zero intersections by constructing a splitting line that separates f from g. The details of curve/intersection separation are complicated. We provide a summary here. Details are in a forthcoming full paper. We discuss separating f from g in terms of constructing a polygonal chain, but actually we split along the lines of the segments in the chain.

Suppose we are at a point $p \in f$, initially $p = a$. We have separated ap from g. Specifically, ap does not intersect a segment of g with both endpoints to the right of ab. (The left has to be handled similarly.) The sign of $\nabla f(p) \times \nabla g(p)$ tells us that $g(x,y)$ is increasing at p in the direction of b. We move away from f in the direction of $\nabla f(p)$ to q halfway to g, meaning $g(q) = g(p)/2$. Next we move in a direction perpendicular to $\nabla f(q)$, "parallel" to f. If we hit f first, that is the new position of p. If we hit g or the boundary of C at r, we drop back to f in the direction opposite of its gradient to the new p on f, with ap separated from g (Fig. 8(a)). Since the parallel move is off f and parallel to it, it goes far before hitting f. Since $g(x,y)$ is increasing in the direction of b, g is getting farther from f in that direction so the parallel move goes far before hitting g.

If $g(x,y)$ is decreasing on f at p in the direction of b, we try to work from the opposite direction, starting with $p' = b$. If $g(x,y)$ is decreasing at both p and p' in the direction of b and a, $g(x,y)$ has a local minimum on f between p and p'. We isolate the minimum m, which is an intersection between f and $\nabla f(x,y) \times \nabla g(x,y) = 0$ (Sec. 5). Then we split f by a line through m in the direction $\nabla f(m)$ (Fig. 8(b)). If there is a pair of intersections with g between p and p' and only one minimum, this will put the two intersections in different cells. If not, m becomes a new starting point for separations because $g(x,y)$ is increasing on f in both directions away from m.

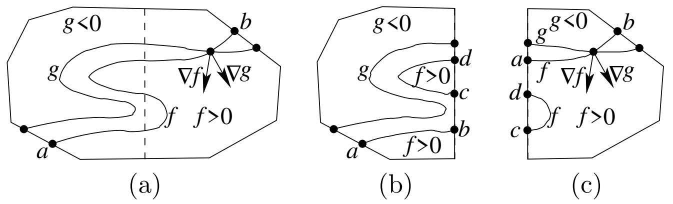


Figure 9: a is a positive tail of f but b is a negative head (a), so $i = -1 = \text{sign}(\nabla f(p) \times \nabla g(p))$ (a). a and d are positive tails and b and d are positive heads $i = 0$ (b). a and c are positive tails, b is negative head, and d is positive head so $i = -1$ (c).

5 Intersection isolation

If $\text{sign}(g(a)) = -\text{sign}(g(b))$, f intersects g an odd number of times inside C, and we isolate one of these intersections to a rectangle $R \subset C$. Let P be the precision of the arithmetic: initially double-float ($P = 53$). Isolation uses two operations: subdivide(D) subdivides a convex region D containing an intersection by the bisector of its longer dimension and returns the half D' containing an intersection (Sec. 5); and Newton(R) iterates 2D Interval Newton's method [12] on a rectangle R until it stops shrinking.

While $0 \in \nabla f(\text{bbox}(D)) \times \nabla g(\text{bbox}(D))$ or $\text{Newton}(\text{bbox}(D)) = \text{bbox}(D)$, $D \leftarrow \text{subdivide}(D)$. Return Newton(bbox($D$)). Isolation does not alter C: the subdivisions are temporary. The output R can be made smaller by doubling the P used to create it and returning Newton(R). We speed up the method by running ordinary Newton's method on each cell centroid. If it converges to a point inside the cell, we expand it to a rectangle based on its condition.

Subdivision might result in one or both halves containing more than one segment of f. We can tell which half contains the intersection by examining the inter-

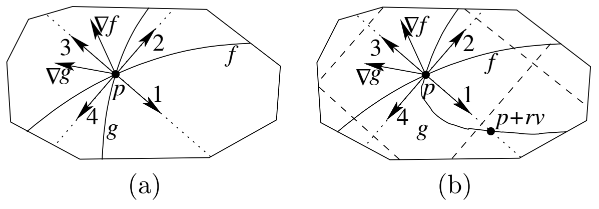


Figure 10: Each angle bisector of the gradient vectors can "see" the boundary (a). Split the cell 90% of the way to the nearest intersection (b) or boundary. Boundary splits are dropped if not needed.

sections of f. A *tail* is a point $a \in f \cap \partial C$ such that $f > 0$ in a neighborhood counterclockwise from a. If a is on edge $v_i v_{i+1}$ of the boundary, the condition is equivalent to $(v_{i+1} - v_i) \cdot \nabla f(a) < 0$. A *head* has the opposite sign. We say $a \in f \cap \partial C$ is *positive* if $g(a) > 0$, otherwise *negative*. A cell contains an intersection if the *intersection number* i of positive heads minus positive tails is nonzero (Fig. 9(a)). This number also equals the winding number of $(f(p), g(p))$ around the origin as p travels around the boundary: $(f(p), g(p))$ sweeps counterclockwise through the first quadrant for each positive head and clockwise for each positive tail. For at least one $p \in f \cap g \cap C$, $\text{sign}(\nabla f(p) \times \nabla g(p)) = \text{sign}(i)$. Subdivision calculates the intersection number for each half and selects the one whose sign is the same as the original cell (Fig. 9(c)).

6 Intersection encasement

The output R of intersection isolation is an adaptive-precision 2D interval representation of an intersection point p of f and g inside a cell C. However, C can contain an even number of additional intersections. Intersection encasement uses up to four splits to isolate p within a cell that excludes all other intersections.

Let v_1, v_2, v_3, v_4 be vectors that bisect the angle between $\pm\nabla f(p)$ and $\pm\nabla g(p)$. If g also intersects C in a single segment and if the four rays $p + tv_i$, $t > 0$, reach the boundary of C without intersecting f or g, the four curve segments connecting p to the boundary of C via f or g are isolated, and f and g have no other intersection in C (Fig. 10(a)).

Otherwise, for each $1 \leq i \leq 4$, compute $t = r_i > 0$ the minimum value at which $p + tv_i$ intersects f, g, or the boundary of C, and split C by the line perpendicular to v_i through the point $p + 0.9 r_i v_i$ (Fig. 10(b)). While f or g intersects the boundary of the cell containing p more than twice, halve each r_i and split again.

7 Encasement based intersection algorithm

The encasement based algebraic curve intersection algorithm takes two bivariate polynomials, $f(x, y)$ and $g(x, y)$, and a desired accuracy δ as input and perturbs their coefficients by η uniform in $[-\delta, \delta]$. When generating a splitting line, it rounds its coefficients to double-float and perturbs them. However, if perturbation puts the line on the wrong side of a vertex, it expresses each coefficient as the sum of two double-floats and perturbs the smaller one, and so forth as necessary. It uses interval arithmetic, increasing precision [6] as necessary to correctly determine signs of predicate expressions.

Separation of curves might require multiple splits, but the separation algorithms are correct for linear curves, and each split shrinks the. Since the curves are generic, their deviation from linear also shrinks, ensuring termination [10].

7.1 Encasement implies arrangement

Using encasement of pairs of curves, we can construct an arrangement of n curves inside B. Given curves $F = \{f_1, f_2, \ldots, f_n\}$, calculate intersections of all pairs. For each $f \in F$, calculate its intersections with the partial derivatives f_x and f_y. Starting with B, add intersections of f with other curves sequentially. First add intersections with f_x and f_y. If a cell contains two intersections, split it with a horizontal or vertical line. After adding the intersections with f_x and f_y, apply self-separation of f. Each cell now contains an x or y-monotonic segment of f and hence the cell can be split with a vertical or horizontal line without creating a cell with more than one segment. Add the remaining intersections of f with other curves, splitting vertically/horizontally as appropriate. The result is the *intersection encasement* $\mathcal{I}(f)$ of f in B with respect to F.

An arrangement segment is a segment of f connecting two cell boundary intersections, in a cell $C \in \mathcal{I}(f)$ not containing an intersection, or a segment ap connecting a boundary intersection a to a curve intersection $p \in C$ with g. To trace the boundary of an arrangement cell, we need to take a "left turn" at p to the segment pc or pd of g in its encasement. The choice is determined by the the sign $\text{sign}(\nabla f(p) \times \nabla g(p))$, a byproduct of isolating the intersection (Sec. 5). Hence the arrangement cells can be traced using only information stored in the intersection encasements.

7.2 GPU speedup

The GPU version subdivides B (or its bounding box if it is not a rectangle), into rectangular cells and assigns the task of showing f or g has no zeros on a cell C to a thread. Cells which fail this test are subdivided. This process stops when the number of failing cells stabilizes. CPU based encasement is run on each resulting cell. Details in full paper.

8 Results

The first set of experiments uses random curves of degree d from 3 to 20. We use $B = [-1, 1] \times [-1, 1]$. To generate a curve, we select $d(d + 1)/2 - 1$ points at random in B and interpolate through them. For each degree d, we generate a set F_d of 16 curves. The test is to generate all the intersections of every pair of curves in F. We compare the CGAL curve arrangement library with the CPU and CPU/GPU versions of encasement. For CGAL, we monotonize each curve once, compute the arrangement of every pair, and then calculate each vertex in double-float. For CPU encasement, we generate the critical regions for each curve once, calculate the encasement for each pair of curves, and increase the precision P until the interval contains at most one double-float point. For the GPU algorithm, we use an initial subdivision small enough to ensure that 90% of subcells are eliminated. The CPU results use an Intel Core i5-3570K over-clocked at 4.2GHz and 8GB RAM. The GPU results in addition use an Nvidia GTX 780 with 4GB DRAM. Results are in Table 8.

For $d > 10$, the CPU version of encasement is about 30 times faster than CGAL. CGAL times out for degrees greater than 18. For degrees up to 13, using the GPU speeds up encasement by a factor of 3. At degree 20, there is no benefit.

The three right columns of Table 8 help to analyze the number of splits required for encasement. Isolating i intersections requires at least i splits. The number of splits is almost proportional, rising slowly from $5i$ up to $7.74i$ for $d = 3$ to $d = 22$.

For the second experiment, we tested the robustness of encasement and the cost of encasing near degenerate cases. We generated pairs of curves with a tangent intersection, which is perturbed to a near-tangency. Table 8 shows the effect of the tangent intersection. Since i ordinary intersections require about $5i$ to $7i$ faces to encase, it appears that a tangency requires about 50 to 60 faces to encase. Since the perturbation is 2^{-26}, this is proportional to the number of bits of accuracy, which is still a very reasonable number.

d	CGAL	CPU	GPU	I	S	S/I
3	0.05	0.01	0.14	5	25	5.0
4	0.16	0.03	0.15	11	49	4.4
5	0.33	0.07	0.17	15	65	4.3
6	0.67	0.16	0.18	16	99	6.1
7	1.56	0.38	0.23	23	179	7.7
8	8.29	0.74	0.30	30	200	6.6
9	17.06	1.56	0.41	35	267	7.6
10	32.65	1.99	0.48	47	291	6.1
11	54.62	2.68	0.89	57	432	7.5
12	119.53	3.28	1.09	59	440	7.4
13	161.72	5.01	1.66	74	542	7.3
14	178.71	8.76	2.40	76	509	6.6
15	367.97	9.35	3.72	95	727	7.6
16	418.51	13.22	5.87	100	743	7.4
17	597.84	19.76	9.00	114	951	8.3
18	881.81	28.89	15.72	135	1062	7.8
19	∞	33.09	17.81	130	1010	7.7
20	∞	43.28	38.86	151	1168	7.7

Table 1: Degree d, CGAL, CPU encasement, and GPU/CPU encasement running times in seconds, number of intersections I, number of cell/line splits in the resulting encasement S, and ratio of S/I.

9 Conclusion

Although it uses perturbation, encasement is an exact algorithm, hence correct. The perturbation adds a controllable backwards error. The choice $\delta = 2^{-26} \approx 10^{-8}$ randomizes half the bits of the input, which makes it generic with high probability. For most applications, a 10^{-8} error is an acceptable price for a 10 to 30 times improvement in running time. The GPU is consumer grade, and so it has an acceptable price for an additional factor of 3 in running time.

We were hoping for more speed up from using a GPU, but the current version uses a quadratic approximation to $f(x, y)$, at a cost of d^2, instead of expanding $f(1/(x+1/(b-a)) + a, 1/(y + 1/(d-c)) + c)$ (Sec. 2), which has d^3 complexity. Also, it is limited to axis-parallel subdivision.

Another goal of this research is 3D surface intersections and arrangement. We believe subdivision by non-axis-parallel planes will be similarly beneficial.

Acknowledgments

Masterjohn and Milenkovic are supported by NSF grant CCF-1526335. Sacks is supported by NSF grant CCF-1524455.

d	time	I	S	S/I
3	0.02	3	67	22.3
4	0.03	4	70	17.5
5	0.05	4	61	15.3
6	0.09	1	51	51.0
7	0.16	2	54	27.0
8	0.31	3	84	28.0
9	0.48	2	73	36.5
10	0.86	6	82	13.7
11	1.47	7	118	16.9
12	1.75	8	152	19.0
13	2.23	2	81	40.5
14	2.84	8	161	20.1
15	4.64	6	127	21.2
16	5.26	6	133	22.2
17	6.85	4	143	35.8
18	9.36	6	127	21.2
19	11.7	10	93	9.3
20	16.2	9	128	14.2

Table 2: Degree d, encasement running time for tangentially intersecting curves.

References

[1] M. Barton, G. Elber, and I. Hanniel. Topologically guaranteed univariate solutions of underconstrained polynomial systems via no-loop and single component tests. *Computer-Aided Design*, 43(8):10351044, 2011.

[2] Michael Bartoň. Solving polynomial systems using no-root elimination blending schemes. *Computer-Aided Design*, 43(12):1870–1878, 2011.

[3] Michael Bartoň and Bert Jüttler. Computing roots of polynomials by quadratic clipping. *Computer Aided Geometric Design*, 24(3):125–141, 2007.

[4] Eric Berberich, Arno Eigenwillig, Michael Hemmer, Susan Hert, Lutz Kettner, Kurt Mehlhorn, Joachim Reichel, Susanne Schmitt, Elmar Schömer, and Nicola Wolpert. Exacus: Efficient and exact algorithms for curves and surfaces. In *European Symposium on Algorithms*, pages 155–166. Springer, 2005.

[5] Efi Fogel, Dan Halperin, and Ron Wein. *CGAL Arrangements and Their Applications: A Step-by-Step Guide*. Springer, 2012.

[6] Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. MPFR: A multiple precision binary floating point library with correct rounding. *ACM Transactions on Mathematical Software*, 33:13, 2007.

[7] I. Hanniel and G. Elber. Subdivision termination criteria in subdivision multivariate solvers using dual hyperplanes representations. *Computer Aided Design*, 39:36978, 2007.

[8] Benoit Jubin. A generalized Poincare-Hopf index theorem. `http://arxiv.org/abs/0903.0697`, 2009.

[9] Bert Jüttler and Brian Moore. A quadratic clipping step with superquadratic convergence for bivariate polynomial systems. *Mathematics in Computer Science*, 5(2):223–235, 2011.

[10] Joseph Masterjohn. Encasement: A robust method for finding intersections of semialgebraic curves. *Open Access Theses*, 699, 2017. `https://scholarlyrepository.miami.edu/oa_theses/699`.

[11] Kurt Mehlhorn and Michael Sagraloff. A deterministic algorithm for isolating the real roots of a real polynomial. *Journal of Symbolic Computation*, 46:70–90, 2011.

[12] Ramon E. Moore. *Methods and Applications of Interval Analysis.* SIAM Studies in Applied Mathematics. SIAM, Philadelphia, 1979.

[13] Bernard Mourrain and Jean Pascal Pavone. Subdivision methods for solving polynomial equations. *Journal of Symbolic Computation*, 44(3):292–306, 2009.

[14] Cong Wang, Yi-Jen Chiang, and Chee Yap. On soft predicates in subdivision motion planning. *Computational Geometry: Theory and Applications*, 48(8):589–605, 2015.

[15] Ron Wein, Eric Berberich, Efi Fogel, Dan Halperin, Michael Hemmer, Oren Salzman, and Baruch Zukerman. 2D arrangements. In *CGAL User and Reference Manual.* CGAL Editorial Board, 4.11 edition, 2017.