Topology and arrangement computation of semi-algebraic planar curves

L. Alberti & B. Mourrain & J. Wintz
GALAAD, INRIA Méditerranée,
2004 route des lucioles,
06902 Sophia-Antipolis, France
{lalberti,mourrain,jwintz}@sophia.inria.fr

November 29, 2008

Abstract

We describe a new subdivision method to efficiently compute the topology and the arrangement of implicit planar curves. We emphasize that the output topology and arrangement are guaranteed to be correct. Although we focus on the implicit case, the algorithm can also treat parametric or piecewise linear curves without much additional work and no theoretical difficulties.

The method isolates singular points from regular parts and deals with them independently. The topology near singular points is guaranteed through topological degree computation. In either case the topology inside regions is recovered from information on the boundary of a cell of the subdivision.

Obtained regions are segmented to provide an efficient insertion operation while dynamically maintaining an arrangement structure.

We use enveloping techniques of the polynomial represented in the Bernstein basis to achieve both efficiency and certification. It is finally shown on examples that this algorithm is able to handle curves defined by high degree polynomials with large coefficients, to identify regions of interest and use the resulting structure for either efficient rendering of implicit curves, point localization or boolean operation computation.

Keyword: topology; arrangement; implicit; parametric; curves

1 Introduction

When manipulating a geometric object there are several operations that one wants to be able to carry out, such as visualizing it in an accurate way, have a topologically correct discretization of the object, be able to individuate the connected components of the object (or of its complement), or to test whether a point is in a specific connected component. Furthermore, one wants to be able to dynamically manipulate these geometric objects and have them interact together through operations such as intersection, complement and union.

A very flexible and theoretically robust setting for doing so is semi-algebraic geometry. Our algorithm, that handles all possible boolean operations on semi-algebraic objects in the plane, makes

it possible to consider practical examples such as those which come from CAD (Computer Aided Design) or CSG (Constructive Solid Geometry). In order to do so, we study curves and "regions" endowed with the connected component predicate in a rectangular domain $\mathcal{D}_0 = [a, b] \times [c, d] \subset \mathbb{R}^2$. Curves are simply sets defined by one polynomial equality. Regions are the connected components of the complement of a curve.

This article, which corresponds to an expanded version of [3] and [53], presents a new algorithm to answer the following question: how to interactively and accurately compute the correct topology and regions defined by one or several curves. We first describe the general subdivision framework that we propose to compute these arrangement of curves. Then, we move to the problem of representing region groups and implementing localization, intersection and update operations on them. The solution we present to this problem is a new combinatorial approach that speeds up computations. Finally, we describe the regularity criterion used in the subdivision algorithm and how to make it effective for algebraic curves.

To get a nice visual representation of the curve $\mathcal{C}$ we're interested in, we produce a piecewise linear approximation of it (within a given Hausdorff distance $\epsilon > 0$). Besides, we guarantee that this piecewise linear complex is actually topologically equivalent to $\mathcal{C}$. What we guarantee precisely is that there is an injective continuous semi-algebraic deformation of $\mathcal{C}$ into its piecewise linear approximation. This is made possible by controlling what happens close to singular points by means of the topological degree which is a quantity that can be computed on the boundary of an isolating box for the singular point. This is a new idea that allows us to control the neighborhood of a singular point only relying on 1-dimensional information around the point.

The approach we choose is subdivision. One strong point of this approach regarding rendering is that we can easily control the Hausdorff distance to the actual curve and that we can use the subdivision incrementally to create an interactive visualization environment. In other words, should the user ask to zoom in on any specific part of the curve, we can always refine the approximation by subdividing further to reduce the approximation error to a size smaller than a pixel. This interactivity feature is hardly achievable with physically based methods like the one described in [46].

On the efficiency side, this approach makes as much use of floating number computation as possible which speeds up the computation and enables us to treat curves defined by polynomials with large coefficients and large degree. Such curves appear, for instance, when applying computer algebra techniques on exact representations of geometric objects. We illustrate it on an example of a self-intersection curve of a bicubic parametrized surfaces and on a discriminant curve related to a conjecture on bivariate systems.

As for certification, the output correctness is proved under the assumption that we have a certified multivariate solver that can isolate the roots of a zero-dimensional system. Multivariate solving is a difficult problem in itself that can be solved efficiently in most cases by sleeve methods or subdivision methods for example [48, 15, 39], but no matter the approach it seems necessary, in some cases, to resort to purely algebraic techniques such as rational univariate representations of roots [6, 17]. We will not go into further details about multivariate solving and refer to previous citations and references therein instead.

Arrangements of geometric objects is a field of computational geometry which has been studied for years [1, 27]. They allow to perform boolean operations which are fundamental tools in Geometric Modeling.

The new method that we describe in this paper for computing arrangements of semi-algebraic curves, provides an efficient way to localize intersection points of region boundaries by storing geo-

metric information on the vanishing of the functions which define the algebraic curves in hierarchical structures. Its originality is to prevent useless computation by stopping the subdivision as soon as the topology of the object is known in a cell of subdivision.

Associated topology computation has lead us to a generic subdivision arrangement algorithm where input objects considered in a given domain are subdivided until being regular in a cell of subdivision, building a quadtree of cells. From this quadtree, we easily obtain regions which are organized within an augmented influence graph to describe the arrangement of objects. It is dynamic in the sense that we can maintain this structure while new objects are inserted or existing objects are removed. These insertions and deletions involve to be able to compute respectively intersections and unions of regions.

Our algorithm brings several contributions. First, the nature of a subdivision scheme allows us to focus, if desired, on regions of interest, leading to a multi-resolution arrangement algorithm, parametrized by a level of approximation. Another contribution is that thanks to its generic approach, it can be used to compute an heterogeneous arrangement, *i.e.* an arrangement of objects having different representations. The resulting regions are also equipped with data structures which lead to efficient operations and are directly usable in a CSG context.

We now give a description of the content of the subsequent subsections: Section 2 places back our method in its scientifical and historical context by explaining what is the current state of the art. Section 3 explains how the generic arrangement computation works and how it is implemented. Section 4 contains the description of the topology algorithm that is used for implicit curves. We isolate the roots of a bivariate polynomial system, using either a Bernstein subdivision solver to approximate efficiently $\mathcal{C}$ or algebraic techniques to certify the result. Section 5 shows some experimental results.

2 Previous work

There are two main types of algorithms that compares to ours concerning topology computation.

The first type is inspired by the Cylindrical Algebraic Decomposition [12] algorithm. They use projection techniques based on a conceptual sweeping line perpendicular to some axis that detects the critical topological events, such as tangents to the sweeping planes and singularities. They involve the exact computation of critical points and genericity condition tests and adjacency tests. The approach has been applied successfully to curves in 2D, and even in 3D, 4D [25, 32, 24, 23, 4] and extended to surfaces [11, 42].

However, they assume exact input equations and rely on the analysis of the curve at the critical values of its projection. From an algebraic point of view, they involve the computation of (sub)-resultants polynomial and of their roots which are algebraic numbers. This can be a bottleneck in many examples with large degree and large coefficients, for which the resultant is difficult to compute, and its real roots even harder to manipulate.
Moreover, as these algorithms work by projection, they have to compute every point in the fibers above the points in the projection. In other words, most points that they compute are actually useless for the computation of the final topological description.
The complexity of the algorithm can also vary wildly, depending on the direction of projection we choose. And non-degeneracy conditions have to be checked (which can be difficult by itself) to ensure the correctness of the algorithm. The problem is that the choice of projection is not at all related to the geometry of the curve. This is why the Cylindrical Algebraic Decomposition methods are hardly efficient in practice, and are facing complexity problems in higher dimension. They are also intrinsically delicate to apply using approximate computation.

The other type of methods relies on subdivision techniques of the original domain. This process is most commonly used to get approximations of the curve in terms of Hausdorff distance. The most famous family of algorithm using this approach is the marching cube algorithms family [36]. It does not give any guarantee on the topological correctness of its output, but it has inspired some algorithms that do certify that their output has the same topology as the curve (usually in the smooth case). They have already been used for solving several complicated equations. See [48, 15] and the recent improvements proposed in [39], exploiting preconditioning techniques. Extensions of this approach to higher dimensional objects have also been considered [47, 32, 29, 31, 34, 2, 44]. These subdivision methods usually fail when singular points exist in the domain. If a threshold on the minimal size for boxes is not set, the algorithm would run forever. Indeed at singularities, no matter the scale of approximation, the shape and topology of the algebraic objects remain similar.

Our algorithm, like the one in [45], is hybrid. It combines approximation properties with certification and adaptivity. It subdivides the domain $\mathcal{D}_0$ into regular regions in which the curve is smooth and regions that contain singular points. In the regular regions, we can approximate the curve as precisely as we want and the "singular" regions can be made as small as required. The algorithm computes the topology inside the regions by using what happens on their boundary and we use enveloping techniques to efficiently treat large input equations. This scheme is refined into two concrete algorithms, one being purely numerical and the other one using some algebraic computations. This method combines the advantages of subdivision and Cylindrical Algebraic Decomposition like methods. Its complexity is intrinsic to the geometry of the curve (like the subdivision methods) and it avoids the main drawback of projections methods because it does not need to lift points.

While current methods proposed to compute an arrangement of algebraic curves allow to handle objects of degree 1 such as line segments, objects of degree 2 such as circular arcs and curves of higher degree are still investigated [37, 54, 26, 20, 38, 28] and can be used for computing an arrangement of surfaces [41, 42]. However, their multiple representations is a major difficulty which leads to as many algorithms as representations exist.

While current methods using a sweep approach [7] focus on events, which are critical points for a projection direction, our method using a subdivision approach focuses on regularity criteria and regular regions. When using sweep methods, events are treated when the sweep line encounters points of interest where a projection on a line becomes critical, reducing the dimension of the problem but increasing its computational difficulty (for instance by computing resultants and by lifting points in the case of implicit curves).

On the contrary, a subdivision method avoids the analysis at critical values by enclosing the singular points in a domain from which the problem in hand can be solved. Such methods are less sensitive to numerical approximations of objects and of their intersection points. They have already been used in geometric modelers such as [9]. Their application to arrangement computation have recently emerged such as in [10, 30] where interval arithmetic is used to classify cells in the subdivision process. Subdivision methods are also very efficient for isolating the roots of polynomial equations, which appear in geometric problems [49, 16, 22, 39]. They have also been extended for the approximation of one or two dimensional objects [31, 2, 34].

3 Arrangement computation

In this first section we set the basic notations and definitions we use throughout the paper. Then the following sections present the specifics of the method for topology computation, and for manipulating arrangements.

3.1 Notations and definitions

The objects that we consider in this framework are semi-algebraic curves whose representation can either be discrete, parametric or implicit. We will denote by $\mathcal{O} = \{o_1, \ldots, o_n\}$ the set of input objects of the arrangement algorithm.

For a subset $S \subset \mathbb{R}^2$, we denote by S° its *interior*, by $\overline{S}$ its *closure*, and by ∂S its *boundary*. We call *domain* any closed set $\mathcal{D}$ such that $\overline{\mathcal{D}^\circ} = \mathcal{D}$ and $\mathcal{D}$ is simply connected. And we call *region* any open set R which is a connected component of the complement of an algebraic curve. The boundary of a domain $\mathcal{D}$ is denoted $\partial\mathcal{D}$.

We call *branch* of a curve $\mathcal{C}$ (relative to a domain $\mathcal{D}$), any smooth closed connected subset of $\mathcal{C} \cap \mathcal{D}$ (*i.e.* C^∞ diffeomorphic to $[0, 1]$) and maximal for the inclusion.

We call *half branch* of a curve $\mathcal{C}$ at a point $p \in \mathcal{D}^\circ$ or half branch originating from $p \in \mathcal{D}^\circ$, any smooth closed connected subset of $\mathcal{C} \cap \mathcal{D}$ which has one endpoint on $\partial\mathcal{D}$ and which is maximal for the inclusion.

A tangent to $\mathcal{C}$ is a line, which intersects $\mathcal{C}$ with multiplicity ≥ 2. We extend this definition to discrete curves as follows. For a point on one segment of the curve, the line supporting the segment is a tangent line. For a vertex belonging to two segments, a tangent line is any line through this vertex and which defines a half-plane containing the two segments. For a vertex belonging to more segments, any line through this vertex is a tangent line.

A curve $\mathcal{C}$ in a domain $\mathcal{D}$ is x-critical (resp. y-critical) if it contains a point with a vertical (resp. horizontal) tangent.

A point of $\mathcal{C}$ which has a tangent line in any direction is called *singular*.

The arrangement computation will produce *regions* which are connected components of the plane $\mathbb{R}^2$, whose interior does not intersect the objects of $\mathcal{O}$. They might also be called faces of the arrangement. These regions are constructed such that their boundary is a set of edges (a segment of a curved object and two vertices) on objects of $\mathcal{O}$.

In addition to this information, we will associate to a region a bounding box, the set of objects involved in its edges and which determine it. Sign conditions can also be associated to a region with regard of the type of objects which determine it.

The generic arrangement algorithm works incrementally by introducing objects one by one, as follows. For each new object o_k, 1. the regions defined by this new object o_k independently of the others are computed, 2. these regions are inserted in the arrangement data-structure $\mathcal{A}_{k-1}$ and yields the new data-structure $\mathcal{A}_k$.

When a region is inserted in the arrangement data-structure, conflicts are detected and "resolved". We say that two regions *conflict* together if their intersection is non empty. In what follows, to make the discussion easier, we may say that an object conflicts with a region, which means that a conflict exists between regions determined by this object and another region. We may also say that two objects conflict together, which means that a conflict exists between regions defined by these objects.

In order to be able to dynamically add or remove objects, we use an *augmented influence graph* $\mathcal{I}_a$ (see figure 11), which is an influence graph connected together with a conflict graph, that we describe now.
An *influence graph* is a directed, acyclic and connected graph. It possesses a single root, and its nodes correspond to the regions created by an algorithm during its execution. Therefore, a node corresponds to a region defined over the current set of objects at some point during the execution of the algorithm. The influence graph possesses two essential properties: at each step of the algorithm, a region defined over the current set of objects is associated with a leaf of the influence graph, and, the domain of influence of a region associated with a node of the influence graph is contained in the union of the domains of influence of the regions associated with the parents of that node.
In addition to the usual information stored in the influence graph, the augmented influence graph stores a conflict graph between the objects in the current set $\mathcal{O}$ and the regions stored in the nodes of the influence graph. This conflict graph is a system of interconnected lists: to each region stored in a node of the influence graph, corresponds a list of objects of $\mathcal{O}$ with which it conflicts, and, to each object in the current set $\mathcal{O}$ corresponds a list of regions stored in the entire influence graph that conflict with it.

The arrangement is represented by the set of leaves in the augmented influence graph, but, the latter still contains the information required to be able to remove an object from it, or to add an object to it. Indeed, leaf nodes constitute the current arrangement $\mathcal{A}_k$, where k is the number of objects in the arrangement, while other nodes (non-root and non-leaf nodes) allow to keep track of the incremental construction of the arrangement $\mathcal{A}_1 \dots \mathcal{A}_{k-1}$. We remind that a root node has no parent, an internal node has both parents and children and a leaf node has no children.

To compute an arrangement $\mathcal{A}_n$ of a collection $\mathcal{O}$ of n objects $o_1 \dots o_n$, we randomly insert the elements of the collection into the arrangement structure $\mathcal{A}$. This structure can be maintained while removing for example the object o_k, leading to the arrangement $\mathcal{A}_\Sigma$ where $\Sigma = \{o_1 \dots o_{k-1}, o_{k+1} \dots o_n\}$ is the corresponding chronological sequence.

Removing an object from an arrangement can be achieved following the general design explained in [8]: after the deletion of an object o_k, the algorithm reinserts the objects o_l of higher chronological rank $l > k$ to create new nodes and re-parent unhooked nodes in the augmented influence graph. We do not detail the removal any further but describe in great details the insertion operation in the next section.

3.2 Generic algorithm

In this section, we present the *generic* insertion operation of the algorithm, *i.e.* the parts which are not related to the type of input objects. The latter makes the assumption that type related functions will be found in a specialization of it. Inserting an object o_k into a structure built for a set $\mathcal{O}_{k-1}$ can be handled in four phases.

Computing regions. In this phase, regions are computed from the topology of o_k independently of objects of $\mathcal{O}_{k-1}$.

Segmenting the boundary. In this phase, computed regions are equipped with additional data structures to help the introduction in the current arrangement.

Locating conflicts. In this phase, newly computed regions are checked for conflict with regions defining the current arrangement.

Updating regions. In this phase, conflicts are dealt with, possibly leading to new regions which are inserted in the data structure.

3.2.1 Computing regions

Regions defined by an object within a given domain $\mathcal{D}_0$ (generally, its bounding box) cannot be computed directly from its representation. To be able to compute a set of regions, from an object in a given domain, we have to ensure that we are in a configuration where we are able to deduce its topology from information on the border of the bounding domain. For the sake of simplicity from here on, we will consider the configurations shown in figure 1, from which we can deduce the topology of regions simply by connecting points of interest of the domain, with points on its border.

When an object is not in one of these configurations, it is subdivided into smaller parts. This process is iterated until one of these configurations is detected. An object in such a configuration is said to be *regular*. The subdivision is then driven by a *regularity test*.

There is a strong link between determining if a cell has been subdivided enough and computing the regions defined by the object within the cell. These two type-related functionalities being dependent one from another, they are left for the specialization of an algorithm computing an arrangement of objects of type t. There is however a general scheme to get the regions defined by an object in a given domain.

To each object, we associate a quadtree used to keep track of the subdivision process which allows to deduce regions. The root of the quadtree stores the bounding box of o_k. A list of cells is initialized with the root node. While this list is not empty, we check its first element for regularity: if o_k is deemed regular in the cell, regions are computed from its topology within the cell and stored in the corresponding node of the quadtree. Else, the cell is subdivided into four children which are appended to the list of cells to be checked for regularity and the quadtree is updated accordingly.

Once all cells have been processed, the leaves of the quadtree contain sub-regions whose union constitutes the regions defined by o_k. To compute this union, these regions are merged traversing the tree from its leaves to its root in a process called *fusion*. The subdivision algorithm, summarized in 3.1 ends up with the root node representing the bounding box of o_k containing the regions determined

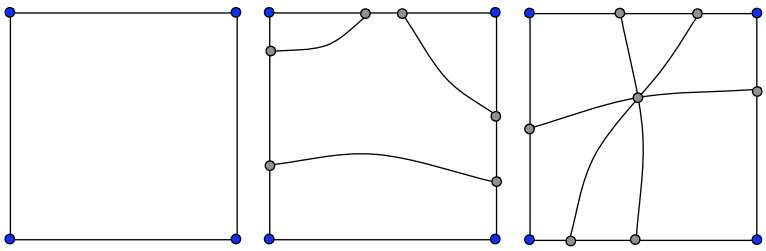


Figure 1: Allowed configurations within a cell.

by o_k.

Algorithm 3.1: A generic subdivision algorithm

Input: a list of objects $\mathcal{O}$ and a box $B_0 \subset \mathbb{R}^2$.
Output: a list of regions.
Create an quadtree $\mathcal{Q}$ and set its root to B_0;
Create a list of cells $\mathcal{C}$ and initialize it with $[B_0]$;
while $\mathcal{C} \neq \emptyset$ **do**
  $c =$ `pop`$(\mathcal{C})$;
  $o =$ `objects`$(\mathcal{O}, c)$;
  if `regular`(o, c) **then**
    $\mathcal{Q} \leftarrow$ `topology`(o, c) ;
  else
    $\mathcal{C} \leftarrow$ `subdivide`(o, c) ;
  end
end
return `fusion`$(\mathcal{Q})$;

The following operations remain to be clarified:

regularity: the specific operation which checks if regions can be computed from an object within a cell of the subdivision, *i.e.* if the object is regular in the cell.

subdivide: the generic operation which subdivides a cell into four children, saving computation effort.

topology: the specific operation which computes regions in a regular cell.

fusion: the generic operation which merges regions stored in each node of the tree.

The regularity test allows to determine if regions can be computed in a cell from information on the boundary of this cell. It is a representation dependent operation and therefore provided in one specialization of this generic algorithm (see section 3.3).

The computation of regions determined by an object in a regular subdivision cell is also provided in a specialization of the generic algorithm, since it strongly depends on the regularity test, which corresponds to different configurations of input objects within a cell (see section 3.3).

When a cell is subdivided into four child cells, we can save computation effort by inheriting the information from a parent to its children , so that we ensure this information is computed only once.

The inheritance is performed in two steps as shown in figure 2: 1. The information contained in the parent cell is inherited by all its children, from a level l to a level $l+1$ in the tree (vertical inheritance) 2. Each time the information is computed for one child, it is inherited by its neighbors (horizontal inheritance). Beyond the performance improvement, this inheritance ensures the information shared by different cells to be consistent, especially if solvers used to compute the intersections of an object with a cell are approximate, as it may be the case when using subdivision solvers (see section 3.3). Also, by using a subdivision policy that determines the location of the subdivision depending of the representation of the object, it is possible to significantly decrease the depth of the subdivision.

The fusion of a quadtree consists in merging at each level and across the levels of the tree, the regions stored in the nodes, from its leaves (only nodes where regions have been computed) to the root (where final regions determined by an object will end). The algorithm takes as input a quadtree obtained from the subdivision, in which, only leaves contain regions, and propagates these regions from the leaves to the root. At each step, the set of regions contained in the children of a non-leaf

node are merged together.

3.2.2 Segmenting the boundary of a region

Finding all the regions that conflict with a new object is a frequent operation in the arrangement computation so we need to pay special attention to its complexity. Indeed, a naive tree traversal of $\mathcal{I}_a$ where each region (stored in a node) is checked for intersection with the regions (stored in newly created nodes) to be inserted would lead to a $O(p^2)$ complexity, where p is the number of nodes in $\mathcal{I}_a$. Also, the respective quadtrees of each object are only used to determine the topology of regions and do not subdivide the same domain. Besides, this structure is related to an object whereas the concept of conflict is relative to a region. To find out the set of regions of $\mathcal{I}_a$ which conflict with regions created by o_k we build a structure called the region segmentation.
A *region segmentation* is a balanced tree data structure in which each node has two children (internal nodes), or none (leaves). Leaf nodes contain the edges defining the boundary of the region as well as their bounding boxes, and internal nodes are associated to the union of the boxes of their children. This way, the root of the tree defining the segmentation is associated to a *domain of influence*, a subset of the bounding box from which the region has been computed, containing the whole region.

For more efficiency when querying the tree, we propose a simple procedure to build a balanced tree from the list of edges defining a region. As shown in figure 3, to build the segmentation, the list of edges is "flattened", ordered and traversed to build the tree. First, a node is created for each edge and the bounding box of the edge is associated to it, yielding a list of nodes. Second, this list is traversed to build the tree. Since we want the tree to be balanced, the traversal is performed from left to right at step k, then from right to left at step $k+1$. Each time a couple of nodes is found in the list, we create a node containing the union of their bounding boxes and reparent the couple of nodes to it, until no more nodes exist in the list. This structure leads to an efficient algorithm to check whether two regions conflict together and helps dealing with them.

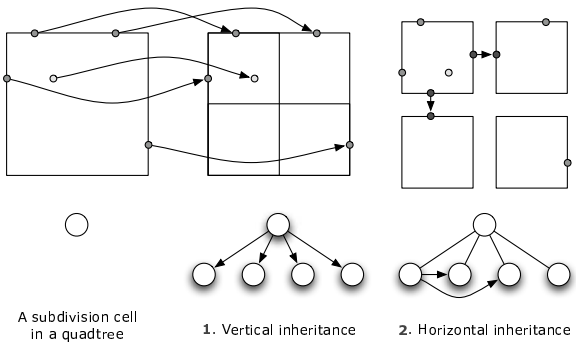


Figure 2: Inheritance.

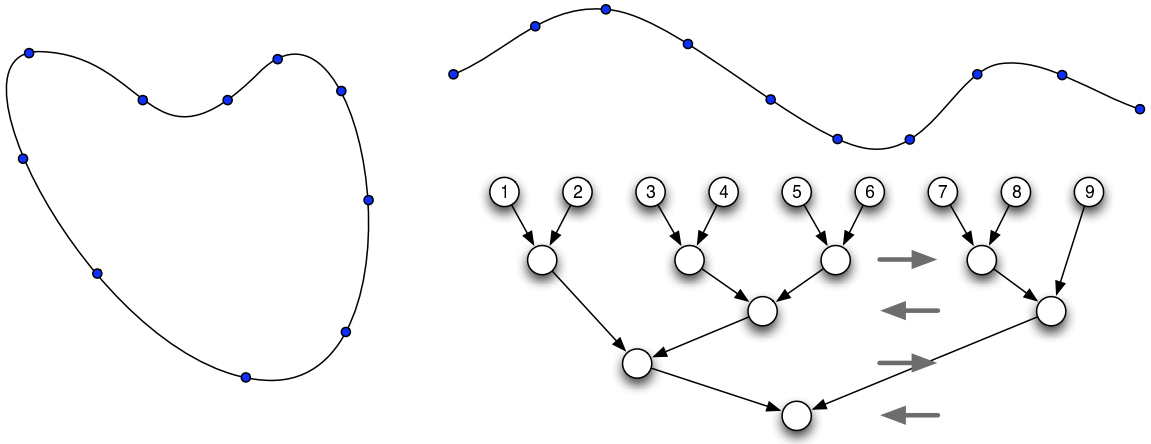


Figure 3: Building the region segmentation.

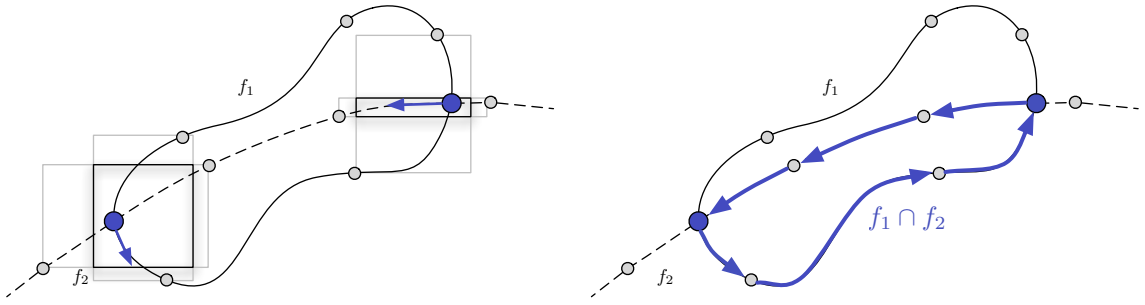
3.2.3 Locating conflicts

To get the list of conflicting zones, we simply compare the segmentations of the regions stored in the nodes of the augmented influence graph with the segmentation of the region currently inserted.

First, we compare their domain of influence, *i.e.* the roots of their respective segmentations. Second, we compare the boxes associated to internal nodes and proceed to child nodes as long as internal nodes boxes do intersect.

If the resulting list is non empty, the intersection of the curve segments are computed using type specific intersection methods according to the type of the objects defining the edges in question. If several intersection points are found, we refine the boundary segmentations in order to have at most one intersection per box.

If the resulting list of pairs of intersecting boxes is empty and if there is no inclusion of a region in the other, there is no conflict and the region can be inserted to $\mathcal{I}_a$. The inclusion test simply consists in locating one point of the region to be inserted. If the latter falls inside the other region, and since we already know there is no intersection of edges, we can conclude that the inserted region lies inside the other one. It is then inserted in the graph as a child node of the one representing the other region. To locate a point in a region, we only have to count the number of intersections of a half-line stemming out from the point in an arbitrary direction. The point is inside the region if this number of intersections is odd, the point is outside the region otherwise. The intersection test can be carried out in a reduced complexity using the region segmentation. Indeed, we only test the half-line for intersection with an edge of the region if the latter intersects its bounding box from the segmentation.



(a) Computing navigation information

(b) Walk-about border of regions

Figure 4: A generic boolean operation

3.2.4 Updating regions

When two nodes of the region segmentations intersect, a conflict is detected, yielding a conflict zone (or box). The intersection of the conflicting regions is computed in three phases. First, intersection points between regions are computed in the conflict box. Second, navigation information are computed for each intersection point, as in [55] (see figure 4a). Finally, a walkabout from an intersection point to the next along the edges of the intersecting regions is performed to construct the intersection (figure 4b).

Computing navigation information for each intersection point consists in examining intersections of the edges with the conflict box, to know which edge to follow from an intersection point to the next during the walkabout. To do so, we compute intersection points of each edge of each region (which are oriented counter-clockwise for their outside border and clockwise for their inside border if holes exist) with the bounding box.

When computing the intersection of regions, the walkabout proceeds along the edge whose intersection point with the box is to the left of the other, considering the orientation of edges. This can be easily obtained by sorting intersection points on the boundary of the conflict box, see figure 4a and 4b.

The resulting intersection regions are segmented and then inserted in the augmented influence graph as a child node of the regions from which they have been computed.

3.3 Specialization for specific curves

When computing an arrangement of curves, we only have to specialize object-type specific functions to meet the generic arrangement algorithm's requirements. We have to provide the following tools:

- Regularity criteria of a curve in a domain.
- Localisation of characteritic points: points with vertical or horizontal tangents, singular points, intersection points, extremal points.

In the following sections, we will detail how to specialize this two points for implicit, paremetric and piecewise curves.

Implicit curves. We denote by $f_k(x,y) \in \mathbb{R}[x,y]$ the polynomial defining the implicit curve corresponding to the object o_k. We assume that f_k is a squarefree polynomial in $\mathbb{Q}[x,y]$. That is, f_k has no square factors in its factorization over $\mathrm{CC}[x,y]$, or more intrinsically the ideal (f_k) generated by f_k is radical. The curve o_k is the zero set $\mathcal{Z}(f_k) = \{(x,y) \in \mathcal{D}_0 | f_k(x,y) = 0\}$ where $\mathcal{D}_0 := [a,b] \times [c,d] \subset \mathbb{R}^2$. is the rectangular domain in which we carry out all of our computations.

The assumption on the f_k being squarefree is easily enforceable as it suffices to divide f_k by its greatest common divisors with its derivatives along the coordinate axes (or with only one of its greatest common divisor with its derivative along a generic direction). For more details on computing the squarefree part of a polynomial, we refer the reader to the literature on this classical matter [52]. Of course, once having taken the squarefree part of f_k we have not modified $\mathcal{Z}(f_k)$ which is the object we are interested in.

The set of *singular points* of $\mathcal{C}$ is $\mathcal{S} := \mathcal{Z}(f, \partial_x f, \partial_y f) = \{(x,y) \in \mathbb{R}^2 | f(x,y) = \partial_x f(x,y) = \partial_y f(x,y) = 0\}$. The set of *extremal points* of f is $\mathcal{Z}_e(f) := \mathcal{Z}(\partial_x f, \partial_y f) = \{(x,y) \in \mathbb{R}^2 | \partial_x f(x,y) = \partial_y f(x,y) = 0\}$.

We recall that a tangent to the curve $\mathcal{C}$ is a line, which intersects $\mathcal{C}$ with multiplicity ≥ 2. In particular, any line through a singular point of $\mathcal{C}$ is tangent to $\mathcal{C}$.

The specific operations for the arrangement will be performed on the Bernstein representation of f_k, starting with the initial domain $\mathcal{D} = [a,b] \times [c,d]$:

$$f_k(x,y) = \sum_{i=0}^{d_{x,k}} \sum_{j=0}^{d_{y,k}} b_{i,j}^k \, B_{d_{x,k}}^i(x;a,b) \, B_{d_{y,k}}^j(y;c,d),$$

where $d_{x,k}$ is the degree in x of f_k and $d_{y,k}$ its degree in y and $B_d^i(x;u,v) = \binom{d}{i} (x-u)^i (v-x)^{d-i} (v-u)^{-d}$ (for $0 \leq i \leq d$, $u < v$).

Using the method described in section 4, we obtain the topology of the curve within a regular cell of subdivision, applying the connection algorithm 4.1. The topology of local regions within the regular cell is easily deduced from the topology of the curves lying in the cell together with its corner points. The edges of regions are oriented counterclockwise to ease up following steps of the arrangement algorithm and specify on which side of the border lies the interior of a region.

After the segmentation step, the detection of conflicts reduces either to intersecting regular segments of two different objects or to testing that an endpoint of a regular segment of an object belongs to a given region. To this end, our favorite polynomial solver (see Section 4.4) is used to solve the bivariate polynomial equations f_k, f_l associate to the objects in potential conflict in the region of interest.

Parametric curves. The computation of an arrangement of parametric curves only requires to be able to compute information on the boundary of a subdivision cell, the subdivision criteria itself, to check if a curve is regular or not within a cell, and a way to compute the intersection of two segments of parametric curves, for detecting region conflicts.

We denote by $x(t)$ and $y(t) \in \mathbb{R}(t)$ the rational functions representing the parametric curve corresponding to the object o_k. For sake of simplicity, we will assume that o_k is the image of $[0,1]$ by the map $\sigma_k : t \mapsto (x(t), y(t))$.

In order to meet the configurations presented in figure 1, we first partition the interval $[0,1]$ in intervals where $x'(t) \neq 0$ and $y'(t) \neq 0$ on the interior. On each one of these intervals, the curve is x and y monotonic. The corresponding bounding box of this curve segment is defined by image by σ_k

of the end points of the interval. This ensures that the regularity test is realized. Next, we localize which pairs of bounding boxes of two non-consecutive segments intersect, using the segment structure described in Section 3.2.2. If we find two such boxes, which are the images of the intervals I and I', we test if there exists $(t, s) \in I \times I'$ with $t \neq s$ such that $x(t) = x(s)$ and $y(t) = y(s)$. This reduces to solving a bivariate polynomial system of polynomial equations. Using our polynomial solver (Section 4.4), we isolate the solution and refine the boundary segmentation by inserting isolating boxes around the image of these points by σ_k. This allows to test simply regular cells.

To determine the topology of the regions within a cell we have to compute the intersection points of the object it contains with its border. In the case of parametric curves, this can be done easily by solving univariate equations of the form $x(t) = x_0$ or $y(t) = y_0$ using solvers as [18] and by checking that the image by σ_k of these roots is on the border of the cell.

Again, we finally turn around these points on the border of the cell in clockwise order and connect them to points of intersect inside the cell, such as self-intersection points or points where a vertical or horizontal tangent exist.

To compute the intersection points of two parametric curves σ_k, σ_l (with $k \neq l$), we solve the bivariate system $\sigma_k(t) = \sigma_l(s)$, with t, s in the intervals $\subset [0, 1]$ corresponding to the curve segments. Here, we can use again our polynomial solver (Section 4.4) for this purpose. This allows to detect conflicts between regions which edges are made up of parametric curve segments.

Piecewise-linear curves. When computing an arrangement of piecewise linear curves, the only challenge is to be able to check if the line segments defining a curve are regular in a cell of subdivision. Indeed, computing the intersection of a linear curve segment with a cell border poses no problem since they have the same representation and detecting self-intersection points is not difficult either since it consists in intersecting two line segments.

To check a piecewise linear curve for regularity in a cell, we gather the curve segments lying in the cell having the same derivative sign in the directions x and y. To each curve segment s, we associate a *monotony code* $M(s)$ given by the tests $x_{i+1} \geq x_i$ and $y_{i+1} \geq y_i$. We traverse the list of curve segments looking for transitions where the monotony code changes, *i.e.* positions i such that $M(s_i) \neq M(s_{i+1})$. The transition points are used to define bounding boxes in the the boundary segmentation and then to detect self-intersecting points, following the same approach as for parametric curves.

4 Regularity and topology of an algebraic curve

The main ingredients of our iterative approach, are to check the regularity criteria of one or several curves in a given domain and to deduce their topology inside this domain. The subdivision algorithm find a partition of $\mathcal{D}_0$ into what we call **simple domains** $\mathcal{D}_i$ for which we can compute the topology. Then we can piece together the topologies of the simple domain by gluing them on their boundaries. For each kind of simple domains, we have a *connection algorithm* that computes a piecewise linear approximation of the curve inside simple domains of that type.

In this section, we describe these operations in details for algebraic curves. The structure of this exposition is as follows: We first define the types of simple domains we will use in subsection 4.1. Then we devote subsections 4.2 and 4.3 to each type of simple domains to explain how to test whether a given domain is indeed of a given type and how to compute the topology in the domain once its type is known. Finally in subsection 4.4, we explain how to actually obtain a partition of the initial

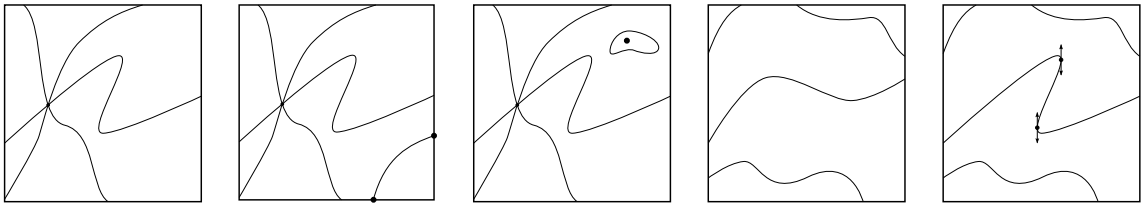


Figure 5: From left to right: A simply singular domain, a domain that fails to be simply singular because there are too many points on its boundary, another one that fails to be simply singular because there are several extremal points of f inside it, an x-regular domain, and a domain that fails to be x-regular because there are vertical tangents to $\mathcal{C}$ inside it.

bounding box into simple domains with a subdivision algorithm that makes use of the tests introduced before, together with known root isolation techniques which are shortly presented in the same section.

4.1 Regularity criteria

We distinguish three different types of simple domains: x-regular domains, y-regular domains and simple singular domains.

Definition 4.1 *A domain $\mathcal{D}$ is* x-regular *(resp.* y-regular*) for $\mathcal{C}$ if $\mathcal{C}$ is smooth in $\mathcal{D}$ and it has no vertical (resp. horizontal) tangents. This is algebraically formulated as the following condition: $\mathcal{Z}(f, \partial_y f) \cap \mathcal{D} = \emptyset$ (resp. $\mathcal{Z}(f, \partial_x f) \cap \mathcal{D} = \emptyset$).*

We might equivalently say that the curve $\mathcal{C}$ is x-regular (resp. y-regular) in $\mathcal{D}$ instead of saying that $\mathcal{D}$ is x-regular (resp. y-regular) in $\mathcal{D}$.

Remark 4.2 *Pay attention to the fact that x-regularity is a condition on the partial derivative along y. It ensures that the orthogonal projection on the x-axis is a submersion. The same remark applies to y regularity.*

Finally we say for short that a curve is *regular* in $\mathcal{D}$, or equivalently that $\mathcal{D}$ is *regular* for $\mathcal{C}$ if $\mathcal{C}$ is x-regular *or* y-regular in $\mathcal{D}$.

Definition 4.3 *A domain $\mathcal{D}$ is* simply singular *for $\mathcal{C}$ if $\mathcal{S} \cap \mathcal{D} = \{p\}$ and if the number n of half branches of $\mathcal{C}$ at the singular point p is equal to $\sharp(\partial\mathcal{D} \cap \mathcal{C})$, the number of points of $\mathcal{C}$ on the boundary of $\mathcal{D}$.*

One may at this point be slightly confused by the fact that we speak freely of *number of points on the boundary* although it could happen that the curve runs along the boundary and there are thus infinitely many points on the boundary. As a matter of fact, the domains are rectangles, and if this happened this would mean that either $(y-\alpha)$ or $(x-\alpha)$ could be factored out of the defining equation of $\mathcal{C}$ (for some $\alpha \in \mathbb{Q}$). It makes it very easy to test for this condition and it can be remediated in several ways: effectively factoring out the linear factor and considering it as a new object that can be added back to the curve afterwards by virtue of its simplicity, or avoid creating domains whose boundary are along a line of the curve by tweaking the final subdvision algorithm. For the sake of clarity we will assume the intersection of $\mathcal{C}$ with every domain considered is 0-dimensional in the rest of the discussion as we feel it would pointlessly burden the exposition.

Finally, these definitions will allow us to prove that the topology in regular domains is that of a number of lines stacked up as a function graph, and to prove that the topology inside simply singular domains is that of a cone over the trace of the curve on the domain's boundary. This will be explained in the next two subsections.

4.2 Regular domains

In this section, we consider a curve $\mathcal{C}$ in $\mathbb{R}^2$, defined by the equation $f(x, y) = 0$ with $f \in \mathbb{Q}[x, y]$ and a domain $\mathcal{D} = [a, b] \times [c, d] \subset \mathbb{R}^2$.

We are going to show that if $\mathcal{C}$ is x-regular in $\mathcal{D}$, then its topology can be deduced from its intersection with the boundary $\partial\mathcal{D}$. By symmetry the same applies when $\mathcal{C}$ is y-regular. We only require that $\partial\mathcal{D} \cap \mathcal{C}$ be 0-dimensional. This is a again a mild requirement that can be easily taken care of when choosing a partition of the initial domain or factoring out the corresponding linear component out of $\mathcal{C}$.

Remark 4.4 *This is well defined because we required that $\partial_y f$ does not vanish at any point of $\mathcal{C}$ in $\mathcal{D}$.*

Definition 4.5 *For a point $p \in \mathcal{C} \cap \partial\mathcal{D}$, and of a sufficiently small neighborhood U of p, by the implicit function theorem, $\mathcal{C}$ is a function graph over the x-axis because $\partial_y f(p) \neq 0$. We define the local right branch at p relative to U as the portion of $\mathcal{C}$ in the half plane $x > x_p$. We define the local left branch at p relative to U as the portion of $\mathcal{C}$ in the half plane $x < x_p$.*

Definition 4.6 *For a point $p \in \mathcal{C} \cap \partial\mathcal{D}$, we define its x-index.*

- `+` *if $\mathcal{C}$ enters $\mathcal{D}$ locally: there exists a local left (resp. right) tangent lying outside (resp. inside) $\mathcal{D}$.*
- `-` *if $\mathcal{C}$ exits $\mathcal{D}$ locally: there exists a local left (resp. right) tangent lying inside (resp. outside) $\mathcal{D}$.*
- `+-` *if $\mathcal{C}$ is tangent to $\mathcal{D}$ and does not enter it locally: $\mathcal{C} - \{p\}$ locally lies outside $\mathcal{C}$.*
- `-+` *if $\mathcal{C}$ is tangent to $\mathcal{D}$ and does not exit it locally: $\mathcal{C} \subset \mathcal{D}$.*

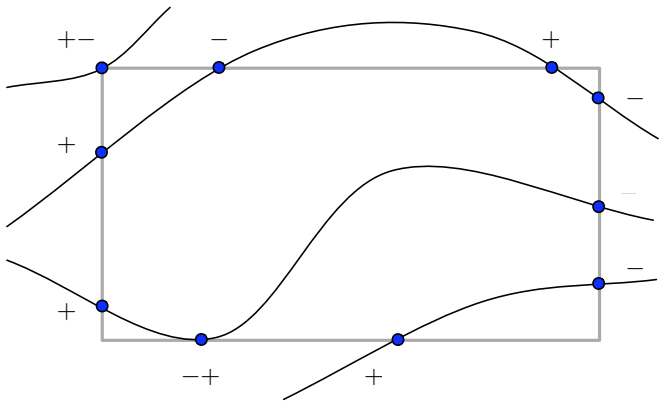


Figure 6: x-indices of an x-regular domain

Remark 4.7 *This is well defined because if there exists a local left (resp. right) tangent lying outside (resp. inside) $\mathcal{D}$, then there cannot exist a local left (resp. right) tangent lying inside (resp. outside) $\mathcal{D}$.*
And we necessarily fall into one of these cases because $\partial\mathcal{D} \cap \mathcal{C}$ is 0-dimensional.

These conditions can be effectively tested using the sign s_y of $\partial_y f$, the order k of the first x derivative of f that does not vanish, and the sign s_x of $\partial^k f$. k is well defined because if all this partial derivatives were 0, the whole horizontal line would be included in $\mathcal{C}$ which would mean $\mathcal{C} \cap \partial\mathcal{D}$ is not 0-dimensional.

Table 4.2 summarizes how to obtain the x-index I_p of a point $p \in \mathcal{D}$ from these 3 numbers for a box $\mathcal{D} = [a, b] \times [c, d]$.

In the following the points with double index (+- or -+) are considered as double points, one with "smaller x component" than the other (although they correspond to a single point that has only one x component). The one with smaller x component gets the left part of the double index, and the one to its right (bigger x component) gets the right part.

Lemma 4.8 *If $\mathcal{C}$ is x-regular in $\mathcal{D}$, then a branch of $\mathcal{C} \cap \mathcal{D}$ connects a point p of x-index $+$ to a point q of x-index $-$, such that $x_p < x_q$.*

Proof. As the curve is x-regular, it has no vertical tangent and thus no closed loop in $\mathcal{D}$. Consequently, each of the interior connected components of $\mathcal{C} \cap \mathcal{D}$ intersects $\partial\mathcal{D}$ in two distinct points $p, q \in \mathcal{C} \cap \partial\mathcal{D}$. Without loss of generality we can assume that $x_p \leq x_q$.

Let $s : t \in [0, 1] \mapsto s_t = (x(t), y(t))$ be an analytic parameterization of the branch $[p, q]$ with $s_0 = (x(0), y(0)) = p$, $s_1 = (x(1), y(1)) = q$. Assume that the x-index of q is $+$, then by definition of the x-index, there is a point $q' = s(T)$ on the branch $[p, q]$ such that $x_{q'} > x_q$. This means $x(T) > x(1)$

Location of p on $\partial\mathcal{D}$	Parity of k	Sign determinant	Resulting x-index
$p \in](a,c),(a,d)[$			$I_p = +$
$p \in](b,c),(b,d)[$			$I_p = -$
$p \in](a,c),(b,c)[$	k odd	$s_y s_x > 0$	$I_p = +$
$p \in](a,c),(b,c)[$	k odd	$s_y s_x < 0$	$I_p = -$
$p \in](a,c),(b,c)[$	k even	$s_y s_x > 0$	$I_p = +-$
$p \in](a,c),(b,c)[$	k even	$s_y s_x < 0$	$I_p = -+$
$p \in](a,d),(b,d)[$	k odd	$s_y s_x > 0$	$I_p = -$
$p \in](a,d),(b,d)[$	k odd	$s_y s_x < 0$	$I_p = +$
$p \in](a,d),(b,d)[$	k even	$s_y s_x > 0$	$I_p = -+$
$p \in](a,d),(b,d)[$	k even	$s_y s_x < 0$	$I_p = +-$
$p = (a,c)$		$s_y s_x (-1)^k < 0$	$I_p = +$
$p = (a,c)$		$s_y s_x (-1)^k > 0$	$I_p = +-$
$p = (a,d)$		$s_y s_x (-1)^k > 0$	$I_p = +$
$p = (a,d)$		$s_y s_x (-1)^k < 0$	$I_p = +-$
$p = (b,c)$		$s_y s_x (-1)^k < 0$	$I_p = -$
$p = (b,c)$		$s_y s_x (-1)^k > 0$	$I_p = +-$
$p = (b,d)$		$s_y s_x (-1)^k > 0$	$I_p = -$
$p = (b,d)$		$s_y s_x (-1)^k < 0$	$I_p = +-$

Table 4.2. Definition of the x-index in a box $\mathcal{D} = [a,b] \times [c,d]$.

and thus $\partial_t x(t)$ is negative on the whole segment $[T,1]$ as it does not vanish by x-regularity. Similarly for p, as $x_p \leq x_q$ we also have $x(T) > x(0)$ and thus $\partial_t x(t)$ is positive on the whole of $[0,T]$. Therefore $\partial_t x(T)$ is both negative and positive as pictured on figure 7a, this is a contradiction and thus the x-index of q is $-$.
The symmetrical reasoning shows that the x-index of p is $+$. Finally, since $\partial_t x(t) > 0$ for $s \in [0,1]$, we have $x_p < x_q$, which proves the lemma. □

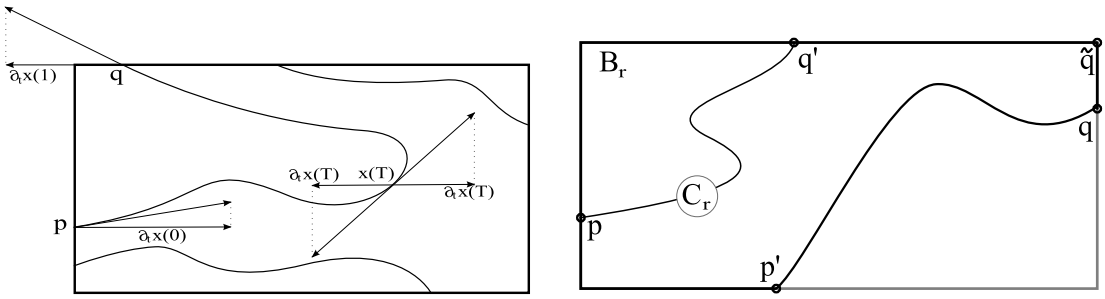
Lemma 4.9 *Suppose that $\mathcal{C}$ is x-regular in $\mathcal{D}$ and let p, q be two consecutive points of $\mathcal{C} \cap \partial\mathcal{D}$ with: q such that x_q is minimal among the points with x-index$= -$, and $x_p < x_q$, then p, q belong to the same branch of $\mathcal{C} \cap \mathcal{D}$.*

Proof. Suppose that p and q are not on the same branch. Let p' be the other endpoint of the branch going to q. Let q' be the other endpoint of the branch starting from p. By lemma 4.8, x-index$(p') = +$ and $x_{p'} < x_q$. By that same lemma, x-index$(q') = -$ and $x_p < x_{q'}$.

The branch (p', q) separates $\mathcal{D}$ in two connected components. We call C_r the one whose boundary $B_r = \partial C_r$ contains the point p (see figure 7b).

Because (p', q) and (p, q') do not intersect, p and q' are in the same connected component of $\mathcal{D} - (p', q)$ and on B_r.

Consider the sub-boundary $\{x \geq x_q\} \cap B_r$. It must be connected. Otherwise the branch (p', q) would intersect $x = x_q$ in two distinct points and the curve would have a x-critical point in between. We denote by $q, \tilde{q}$, the endpoints of $\{x \geq x_q\} \cap B_r$ (with possibily $q = \tilde{q}$). We decompose B_r as the union of arcs $B_r = (p', q) \cup (q, \tilde{q}) \cup (\tilde{q}, p')$ with $(q, \tilde{q}) \subset \partial\mathcal{D}$, $(\tilde{q}, p') \subset \partial\mathcal{D}$.



(a) p and q cannot have same x-index.

(b) Illustrative figure for proof of lemma 4.9

Figure 7: Connection algorithm.

By minimality of x_q, we have $x_{q'} \geq x_q$ so that $q' \in \{x \geq x_q\} \cap B_r = (q, \tilde{q})$. Because $x_p < x_q$ and $p \in B_r$ and $p \notin (p', q)$, we have $p \in (\tilde{q}, p') \subset \partial\mathcal{D}$.

This proves that p is in-between p' and q' and q' in-between p and q on $\partial\mathcal{D}$. Therefore, p and q cannot be consecutive points of $\mathcal{C}$ on $\partial\mathcal{D}$. By way of contradiction, we conclude that p and q must be on the same branch of $\mathcal{C}$. □

Proposition 4.10 *Let $\mathcal{C} = \mathcal{Z}(f)$. If $\mathcal{D}$ is a x-regular domain, the topology of $\mathcal{C}$ in $\mathcal{D}$ is uniquely determined by its intersection $\mathcal{C} \cap \partial\mathcal{D}$ with the boundary of $\mathcal{D}$.*

Proof. We prove the proposition by induction on the number $N(\mathcal{C})$ of points on $\mathcal{C} \cap \partial\mathcal{D}$. We denote this set of points by $\mathcal{L}$.
Since the curve has no vertical tangent in $\mathcal{D}$ and has no closed loop, each of the connected components of $\mathcal{C} \cap \mathcal{D}^\circ$ have exactly two distinct endpoints on $\partial\mathcal{D}$. Thus if $N(\mathcal{C}) = 0$, then there is no branch of $\mathcal{C}$ in $\mathcal{D}$.
Assume now that $N(\mathcal{C}) > 0$, and let us find two consecutive points p, q of $\mathcal{L}$ with x-index$(p) = +$, x-index$(q) = -$, $x_p < x_q$ and x_q minimal. By lemma 4.9, the points p, q are the endpoints of the branch of $\mathcal{C}$.
Removing this branch from $\mathcal{C}$, we obtain a new curve $\mathcal{C}'$ which is still x-regular and such that $N(\mathcal{C}') < N(\mathcal{C})$. We conclude by induction hypothesis, that the topology of $\mathcal{C}'$ and thus of $\mathcal{C}$ is uniquely determined. □

Proposition 4.11 *If $\mathcal{C}$ has at most one x-critical or y-critical point $\in \mathcal{D}$, which is also smooth, then its topology in $\mathcal{D}$ is uniquely determined by its intersection with the boundary of $\mathcal{D}$.*

Proof. Suppose $\mathcal{C}$ has at most one x-critical point in $\mathcal{D}$, which is smooth, then the curve is smooth in $\mathcal{D}$ and has no closed loop inside $\mathcal{D}$ (otherwise the number of x-critical points would be at least 2). Therefore, the branches are intersecting $\partial\mathcal{D}$ in two points. If there is no x-critical point on a

branch, by Lemma 4.8 their x-index $\in \{-,+\}$ are distinct. If the branch has a x-critical point of even multiplicity, then the x-index of the end-points of the branch in $\mathcal{C}$ are the same. If there are only two points of $\mathcal{C}$ on ∂D, then this branch is connecting the two points. As the curve is smooth, the branches are not intersecting. If there are more points, and thus more than 2 branches, the branch with the even x-critical point is separating the set of branches into two disjoints subsets of branches with no x-critical points. Changing the orientation of the x-axis if necessary, we can find consecutive points p, q on ∂D which satisfies the hypothesis of lemma 4.9. By this lemma, they are necessarily on the same branch of one of these two subsets. Removing this branch from $\mathcal{C}$ and processing recursively in this way, we end up either with no point on $\partial\mathcal{D}$ or two points on $\partial\mathcal{D}$ with the same x-index. These points are necessarily connected by the branch containing the x-critical point of $\mathcal{C}$ in $\mathcal{D}$. □

This leads to the following algorithm:

Algorithm 4.1: Connection for a x-regular domain

Input: an algebraic curve $\mathcal{C}$ and a domain $\mathcal{D} = [a,b] \times [c,d] \subset \mathbb{R}^2$ such that $\mathcal{C}$ has no vertical tangent in $\mathcal{D}$

Output: the set $\mathcal{B}$ of branches of $\mathcal{C}$ in $\mathcal{D}$

Isolate the points $\mathcal{C} \cap \partial\mathcal{D}$ and compute their x-index ;

Order the points of $\mathcal{C} \cap \partial\mathcal{D}$ with non-zero x-indices clockwise and store them in the circular list $\mathcal{L}$;

while $\mathcal{L} \neq \emptyset$ **do**

  Take a point q such that x_q is minimal among the points in $\mathcal{L}$ with x-index$= -$;

  Take the point p that follows or precedes q in $\mathcal{L}$ such that $x_p < x_q$ (thus x-index$(p) = +$) ;

  Add the arc $[p,q]$ to the set $\mathcal{B}$ of branches and remove p, q from $\mathcal{L}$;

end

Notice that a sufficient condition for the x (resp. y) regularity of f in a domain $\mathcal{D}$ is that the coefficients of ∂_y (resp. $\partial_x f$) in the Bernstein basis on $\mathcal{D}$ are all > 0 or < 0. In this case the connection algorithm can be simplified even further. See [2] for more details.

4.3 Simply singular domains

In this section we deal with simple singular domains. Recall that those domains $\mathcal{D}$ are such that they contain a unique critical point p of f where f also vanishes (*i.e.* p is a singular point of $\mathcal{C}$) and the number of half branches at p is the same as the number of points of $\mathcal{C}$ on their boundary. As mentioned earlier we will see in section 4.4, how to compute such a domain. This section will focus on explaining how to test whether a given domain is simply singular, as well as how to compute its topology.

The next two subsections explain how it is possible to effectively count the number of half branches of $\mathcal{C}$ at p. The first one introduces the essential theoretical tool to do this which is the topological degree, [35]. Then the subsection that follows deals with Khimshiashvili theorem, [33, 5, 51] which actually relates the number of half branches and the topological degree. This enables us to check effectively that the number of points in $\partial\mathcal{D} \cap \mathcal{C}$ is the same as the number of half branches at p.

The third and last subsection for simply singular domains show that the topology inside a simply singular domain is conic. This fact gives rise a very simple connection algorithm for them.

4.3.1 Topological Degree

In this section, we recall the definition of the topological degree in two dimensions and how it can be computed. See [35, 50] for more details.

Let $\mathcal{D}$ be a bounded open domain of $\mathbb{R}^2$ and $F = (f_1, f_2) : \mathcal{D} \to \mathbb{R}^2$ a bivariate function which is two times continuously differentiable in $\mathcal{D}$.
A point $p \in \mathbb{R}^2$ is said to be a *regular value* of F on $\mathcal{D}$ if the roots of the equation $F(x, y) = p$ in $\mathcal{D}$ are simple roots, *i.e.* the determinant of the Jacobian J_F of F at these roots is nonzero).

Definition 4.12 *Let $p \in \mathbb{R}^2$ and suppose further that the roots of the equation $F(x, y) = p$, are not located on the boundary $\partial\mathcal{D}$.*

Then the topological degree of F at p relative to $\mathcal{D}$, *denoted by* $\deg[F, \mathcal{D}, p]$, *is defined by*

$$\deg[F, \mathcal{D}, p] = \sum_{\mathbf{x}\in\mathcal{D}:F(\mathbf{x})=q} \text{sign det } J_F(\mathbf{x}),$$

for q a regular value of F on $\mathcal{D}$ in the connected component of $\mathbb{R}^2 - F(\partial\mathcal{D})$ containing p.

It can be proved that this construction does not depend on the regular value q in the same connected component of $\mathbb{R}^2 - F(\partial\mathcal{D})$ as p [35]. If p is a regular value of F on $\mathcal{D}$, we can take $q = p$.

Remark 4.13 *The topological degree has a geometric interpretation known as the degree of the "Gauss map". It is the number of times $F(p)$ goes around $F(\mathcal{D})$ when p goes around $\mathcal{D}$ one time. And it is negative when F reverses the orientation of $\mathcal{D}$.*
The red arrows in fig. 8 picture the $F(p)$ on the boundary. This viewpoint allows to use the strong geometric intuition behind the gradient field when F is the gradient map of f.

Let us now give a more explicit formula for computing this topological degree, which involves only information on the boundary of $\mathcal{D}$.

Proposition 4.14 *[50] Assume here that the boundary $\mathcal{D}$ is a polygon and that it is decomposed in reverse clock-wise order into the union of segments*

$$\partial\mathcal{D} = \cup_{i=1}^{g}[p_i, p_{i+1}], \quad p_{g+1} = p_1,$$

in such a way that one of the component f_{σ_i} ($\sigma_i \in \{1, 2\}$) of $F = (f_1, f_2)$ has a constant sign ($\neq 0$) on $[p_i, p_{i+1}]$. Then

$$\deg[F, \mathcal{D}, (0, 0)] = \frac{1}{8}\sum_{i=1}^{g}(-1)^{\sigma_i - 1} \begin{vmatrix} \text{sg}(f_{\sigma_i}(p_i)) & \text{sg}(f_{\sigma_i}(p_{i+1})) \\ \text{sg}(f_{\sigma_i+1}(p_i)) & \text{sg}(f_{\sigma_i+1}(p_{i+1})) \end{vmatrix} \tag{1}$$

where $f_1 = f_3$ and $\text{sg}(x)$ denotes the sign of x.

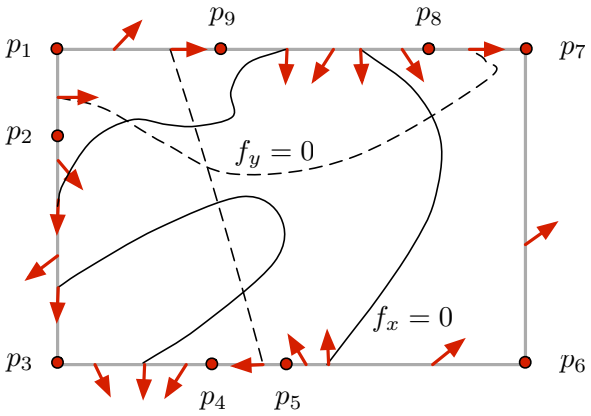


Figure 8: Computing the topological degree

Thus in order to compute the topological degree of F on a domain $\mathcal{D}$ bounded by a polygon, we need to separate the roots of f_1 from the roots of f_2 on $\partial\mathcal{D}$ by points $p_1, \ldots, p_{g+1}$ at which we compute the sign of f_1 and f_2. This will be performed on each segment of the boundary of $\mathcal{D}$, by a univariate root isolation method working simultaneously on f_1 and f_2, that we will described in the next section.

Figure 8 shows a sequence of points $p_1, \ldots, p_9$, which decomposes ∂D into segments on which one of the two functions ($f_1 = 0$ and $f_2 = 0$ are represented by the plain and dash curves) has a constant sign. Computing the sign of these functions and applying formula (1) yields the topological degree of $F = (f_1, f_2)$ on $\mathcal{D}$ at $(0, 0)$.

4.3.2 Counting the number of branches

Let us consider a curve $\mathcal{C}$ in a domain $\mathcal{D} \subset \mathbb{R}^2$, defined by the equation $f(x, y) = 0$ with $f(x, y) \in \mathbb{R}[x, y]$. Let $\nabla f = (\partial_x f, \partial_y f)$ be the gradient of f. A point $p \in \mathcal{C}$ is singular if $\nabla f(p) = 0$. We defined a real half branches of $\mathcal{C}$ at p, as a connected component of $\mathcal{C} - \{p\} \cap \mathcal{D}(p, \epsilon)$ for $\epsilon > 0$ small enough.

The topological degree of ∇f can be used to count the number of half branches at a singular point, based on the following theorem:

Theorem 4.15 (Khimshiashvili [33, 5, 51]) *Suppose that p is the only root of $\nabla f = 0$ in $\mathcal{D}$. Then the number N of real half branches at p of the curve defined by $f(x, y) = f(p)$ is*

$$N = 2\,(1 - \deg[\nabla f, \mathcal{D}, (0, 0)]). \tag{2}$$

We will denote by $N(f, \mathcal{D})$ the number given by Formula (2).

In order to count the number of branches of $\mathcal{C}$ at a singular point $p \in \mathcal{C}$, first we isolate the singular point p in a domain $\mathcal{D}$, so that ∇f does not vanishes elsewhere in $\mathcal{D}$. Then we compute the topological degree $\deg[\nabla f, \mathcal{D}, (0, 0)]$, as described previously, by isolating the roots of $\partial_x f$ and $\partial_y f$ on $\partial\mathcal{D}$.

Let us describe now the algorithm used to compute the topological degree of ∇f in a domain $\mathcal{D} = [a,b] \times [c,d]$. According to formula (1), this reduces to separating the roots of $\partial_x f$ $\partial_y f$ on the boundary of $\mathcal{D}$, which consists in 2 horizontal and vertical segments. The problem can thus be transformed into isolating the roots of univariate polynomials on a given interval. Hereafter, these polynomials will be called $g_1(t), g_2(t)$ and the interval $[u,v] \subset \mathbb{R}$. For instance, one of the 4 cases to consider will be $g_1(t) = \partial_x f(t,c), g_2(t) = \partial_y f(t,c)$, $u = a, v = b$. We recall briefly the subdivision method described in [43, 40, 18], which can be used for this purpose. First we express our polynomials $g_1(t)$, $g_2(t)$ of degree d_1, d_2 in the Bernstein bases $(B^i_{d_k}(t;u,v))_{i=0,\ldots,d_k}$ $(k = 1,2)$, on the interval $[u,v]$:

$$g_k = \sum_{i=0}^{d_k} \lambda_{k,i}\, B^i_{d_k}(t;u,v), k = 1,2,$$

where $B^i_d(t;u,v) = \binom{d}{i}(t-u)^i(v-t)^d(v-u)^{-d}$. The number of sign variations of the sequence $\lambda_k = [\lambda_{k,0}, \ldots, \lambda_{k,d_k}]$ $(k = 1,2)$ is denoted $V(g_k;[u,v])$. By a variant of Descartes rule [6], it bounds the number of roots of g_k on the interval $[u,v]$ and is equal modulo 2 to it. Thus if $V(g_k;[u,v]) = 0$, g_k has no root in the interval $[u,v]$, if $V(g_k;[u,v]) = 1$, g_k has exactly one root in the interval $[u,v]$. This is the main ingredient of the subdivision algorithm [18], which splits the interval using de Casteljau algorithm [19] if $V(g_k;[u,v]) > 1$; store the interval if $V(g_k;[u,v]) = 1$ and remove it otherwise. It iterates the process on each subintervals until the number of sign variation is 0 or 1. The complexity analysis of the algorithm is described in [18]. See also [14].

In our case, we need to compute intervals on which one of the polynomial g_1 or g_2 has a constant sign. Thus we replace the subdivision test by the following:

- if $V(g_1;[u,v]) = 0$ or $V(g_2;[u,v]) = 0$, we store the interval $[u,v]$;
- otherwise we split it and compute the Bernstein representation of g_k $(k = 1,2)$ on the two subintervals using de Casteljau algorithm and repeat the process.

This yields the following algorithm for computing the topological degree of $\nabla f = (f_1(x,y), f_2(x,y))$ on $\mathcal{D}$:

If we assume that $\partial_x f$ and $\partial_y f$ have no common root on the boundary of $\mathcal{D}$, it can be proved (by the same arguments as those used in [6, 40, 18]) that this algorithm terminate and output a sequence of intervals on which one of the functions g_1, g_2 has no sign variation. The complexity analysis of this method is described in [43]. This analysis can be improved by exploiting the recent results in [18].

4.3.3 Conic structure and connection algorithm

Finally we prove that the topology in a simple singular domain $\mathcal{D}$ is conic and write a connection algorithm for theses domains.

Let $A \subset \mathbb{R}^n$ and $p \in \mathbb{R}^n$. We call cone over A with center p the set $p \star A := \bigcup_{q \in A}[p,q]$.

Proposition 4.16 *Let $\mathcal{D}$ be a convex simple singular domain, i.e. $\mathcal{D}$ is convex such that there is a unique singular point s and no other critical point of f in $\mathcal{D}$, and such that the number of half branches of $\mathcal{C}$ at s is $\sharp(\partial\mathcal{D} \cap \mathcal{C})$. Then the topology of $\mathcal{D}$ is conic, i.e. for any point p in the inside $\mathcal{D}$, $Z(f) \cap \mathcal{D}$ can be deformed into $p \star (\partial\mathcal{D} \cap \mathcal{C})$.*

Proof. s is the unique critical point of f in $\mathcal{D}$. If the endpoint of a half branch at s is not on $\partial\mathcal{D}$, the half branch has to be a closed loop inside $\mathcal{D}$. In that case, f would be extremal at some point p $(\neq s)$

Algorithm 4.2: Topological degree of (f_1, f_2)

Input: a polynomial $f(x, y) \in \mathbb{Q}[x, y]$ and a domain $\mathcal{D} = [a, b] \times [c, d]$
Output: N the topological degree of ∇f on $\mathcal{D}$ at $(0, 0)$
$\mathcal{B} := \{\}$ (a circular list representing the boundary $\partial\mathcal{D}$) ;
foreach *side segment* I *of the box* $\mathcal{D}$ **do**
  Compute the restriction $g_1(t)$ (resp. $g_2(t)$) of f_1 (resp. f_2) on this side segment I and its representation in the corresponding Bernstein basis ;
  $\mathcal{L} := \{I\}$;
  while $\mathcal{L} \neq \emptyset$ **do**
    pop up an interval $[p, q]$ from $\mathcal{L}$;
    if $V(g_1; p, q) = 0$ *or* $V(g_2; p, q) = 0$ **then**
      insert p, q clockwise in the circular list $\mathcal{B}$;
    else
      split $[p, q]$ in half an insert the two subintervals in $\mathcal{L}$;
    end
  end
end
Compute N given by formula (1) for the points in the circular list $\mathcal{B}$;

inside the loop, and p would be another critical point of f inside $\mathcal{D}$. Thus, by way of contradiction, the endpoints of half branches at s have to be on $\partial\mathcal{D}$.

The number of half branches at s is exactly $\sharp(\partial\mathcal{D} \cap \mathcal{C})$. As no two half branches can have the same endpoint on $\partial\mathcal{D}$ (that would be another singular point in $\mathcal{D}$), all points on $\partial\mathcal{D}$ are endpoints of half branches at s. Thus, at this point, we know that the connected component of s inside $\mathcal{D}$ is conic.

But in fact, there is no other connected component: Suppose we have another connected component α of $\mathcal{C}$ intersecting $\mathcal{D}$. As all points of $\partial\mathcal{D} \cap \mathcal{C}$ are connected to s, we have $\alpha \subset \mathcal{D}$. α is a smooth 1-dimensional manifold because s is the only singular point. Therefore α is a closed loop inside $\mathcal{D}$ (s might be inside it). We look at the complement of $\mathcal{C}$ in $\mathbb{R}^2$, it has a bounded connected component because one of them is inside the loop α. As f vanishes on the boundary of this component, f has an extremum inside it. This extremum cannot be s as it is in the complement of f, which is impossible. Thus, $\mathcal{C} \cap \mathcal{D}$ is connected.

This concludes our argument as we have proved that $\mathcal{C} \cap \mathcal{D}$ is equal to the connected component of s inside $\mathcal{D}$ and that it has the topology of a cone over $\partial\mathcal{D} \cap \mathcal{C}$ which is what we claimed. □

Remark 4.17 *We do not have to suppose that $\mathcal{D}$ is convex, simply connected would suffice. But we only work with convex sets (boxes) and the denomination "conic topology" originates from the convex case.*

In the end the connection algorithm is extremely simple. We just proved that the topology inside these domains is conic, that is $\mathcal{C} \cap \mathcal{D}$ can be deformed into a cone over $\mathcal{C} \cap \partial\mathcal{D}$. Therefore the connection algorithm for (convex) simply singular domains is to first compute the points q_i of $\mathcal{C} \cap \partial\mathcal{D}$, then choose an arbitrary point p inside $\mathcal{D}$ and finally for every q_i, connect q_i and p by a half branch segment $\mathfrak{b}_i = [p, q_i]$.

4.4 The subdivision procedure

Let $\mathcal{D}_0 = [a, b] \times [c, d]$ be a domain of $\mathbb{R}^2$. The goal of this section is to describe effective methods to partition $\mathcal{D}_0$ into simple domains. The difficult step of this approach is to isolate the roots of

$$\mathcal{Z}_e(f) = \{(x, y) \in \mathcal{D}_0, \partial_x f(x, y) = 0, \partial_y(f)(x, y) = 0\}.$$

which are on $\mathcal{C}$, with the following property:

- There is only one point p of $\mathcal{Z}_e(f)$ in each isolating domain $\mathcal{D}$ (and it is on $\mathcal{C}$, that is singular)
- The number of points in $\mathcal{C} \cap \partial\mathcal{D}_0$ is the number of half-branches at the singular point p (that is $N(f, \mathcal{D}) = 2\,(1 - \deg[\nabla f, \mathcal{D}, 0])$).

We present two approaches. The first one exploits the Bernstein representation of f and subdivision techniques to isolate the roots of $\mathcal{Z}_e(f)$, while identifying domains where the curve is regular. It outputs an approximation of $\mathcal{C}$ to a precision that is given as input to the algorithm. We prove that, for a sufficiently high precision, the algorithm output has the same topology as $\mathcal{C}$. The second algorithm is based on algebraic techniques (namely Rational Univariate Representation) and is guaranteed to output the correct topology.

The two following methods do the isolation work in a different way but they share the test described in section 4.3 to count the number of half branches at a singular point.

4.4.1 Recursive Bernstein subdivision method

We describe here the subdivision method used to obtain such isolating domains, with a specialization of the approach used in [39]. See also [48, 15]. This method which we recall for polynomials in $\mathbb{Q}[x, y]$ applies for general multivariate polynomials. We are going to consider the system $f(x, y) = 0$, $\partial_x f(x, y) = 0$, $\partial_y f(x, y) = 0$ in the domain $\mathcal{D}_0 = [a, b] \times [c, d]$.

Each of these polynomials is expressed in the Bernstein basis on $\mathcal{D}_0$:

$$h(x, y) = \sum_{i=0}^{d_x} \sum_{j=0}^{d_y} \gamma_{i,j}\, B^i_{d_x}(x; a, b)\, B^j_{d_y}(y; c, d),$$

where $h \in \{f, \partial_x f, \partial_y f\}$ and d_x is the degree of h in x, d_y the degree of h in y. By using a method described in [39] we can quickly generate a set of boxes where the curve is x or y-regular and small set of boxes of size smaller than a given precision $\epsilon > 0$ that isolates the part of the curve where we don't yet know what is happening.

The principle of this method is to either reduce a box by using convexity inequalities on Bernstein bases or to split the boxes if the inequalities do not apply. This is the main loop of the subdivision algorithm, which is combined with preconditioning techniques to improve the performance of the solver. The computation is iterated until the size of the box is smaller than ϵ.

It is not the purpose of this article to go into the details of the method expounded in [39]. In this method the domain are boxes $[a, b] \times [c, d]$ and when they are reduced, they are reduced in only one. That is, either the $[a, b]$ or the $[c, d]$ part of the box is shrinked. When that happens, the convexity inequalities imply that one of the functions f, $\partial_x f$, $\partial_y f$ does not vanish in the regions which are removed from the box. Thus the curve $\mathcal{C}$ in these regions is regular and according to section 4.2, its topology can be deduced from the intersection of the curve with the boundary of the region.

This method can be adapted to our implicit curve problem, and yields the following algorithm:

Algorithm 4.3: Subdivision algorithm for the topology of $\mathcal{C}$

Input: a curve $\mathcal{C}$ defined by $f(x, y) = 0$, $\mathcal{D}_0 = [a, b] \times [c, d]$, a rendering precision $\epsilon > 0$ and a computation precision ν with $\epsilon \geq \nu > 0$

Output: A graph of points $\in \mathcal{D}$ connected by segments

$\mathcal{L} = \{\mathcal{D}_0\}$; $\mathcal{S} = \{\}$;

while $\mathcal{L} \neq \emptyset$ **do**

  Pop up a domain $\mathcal{D}$ from $\mathcal{L}$;

  if $\mathcal{D} > \nu$ **then**

    reduce or split the domain $\mathcal{D}$ according to the Bernstein coefficients of $f, \partial_x f, \partial_y f$ and insert the resulting domains in $\mathcal{L}$;

    apply the connection algorithm of regular domain 4.1 on the removed regions ;

  else

    add $\mathcal{D}$ to the set of singular domains $\mathcal{S}$ and update its connected components;

  end

end

foreach *minimal box* $\mathcal{D}$ *containing a connected component of* $\mathcal{S}$ **do**

  if $|\mathcal{D}| < \epsilon$ *and* $\mathcal{D}$ *does not intersect such another minimal box and if* $\sharp(\mathcal{C} \cap \partial\mathcal{D}) = 2\,(1 - \deg[\nabla f, \mathcal{D}, (0, 0)])$ **then**

    apply the algorithm of connection 4.3.3 in $\mathcal{D}$;

  else

    replace ν by $\frac{\nu}{2}$ and apply the same algorithm on $\mathcal{D}$.

  end

end

This algorithm decomposes the initial domain into regions where the topology is known and a set of non-intersecting boxes of size $\leq \epsilon$ where $\sharp(\partial\mathcal{D} \cap \mathcal{C}) = 2\,(1 - \deg[\nabla f, \mathcal{D}, 0])$ (this is (2)). If ϵ corresponds to the size of a pixel, the visualization of the curve will be correct, except in these pixel boxes, which we call singular regions. Inside them equation (2) holds, and if in addition there is a unique critical point of f, which is also on $\mathcal{C}$, then the computed topology is correct.

During the subdivision process we have to zoom on domains or equivalently to scale the variables ($x := \lambda\, x,\ y := \lambda\, y$). In order to handle the numerical instability problems, which may happen in this scaling step or when we have to deal polynomials with large coefficients and degrees, we use the following enveloping techniques, which allows us to compute with fixed precision numbers: To analyze the curve $\mathcal{C}$ defined by the polynomial $f \in \mathbb{Q}[x, y]$ on a domain $\mathcal{D} = I \times J$,

- we convert f to the Bernstein basis on the domain $\mathcal{D}$ using exact arithmetic: $f(x, y) = \sum_{i,j}\ \gamma_{i,j}\, B^i_{d_x}(x; I)\, B^j_{d_y}(y; J)$
- we round up and down to the nearest machine precision number $\underline{\gamma_{i,j}} \leq \gamma_{i,j} \leq \overline{\gamma_{i,j}}$, so that we have $\underline{f}(x, y) \leq f(x, y) \leq \overline{f}(x, y)$ on $\mathcal{D}$.
- We use the interval coefficients $[\underline{\gamma_{i,j}}, \overline{\gamma_{i,j}}]$ to test the sign conditions and to remove the regular regions.

It can be proved that if ϵ is small enough, then this algorithm compute the topology of $\mathcal{C}$ (but for space limitation reasons, we do not include the proof here).

Remark that if $Z(f)$ is smooth in a domain $\mathcal{D}$, this algorithm can be run with $\epsilon = 0$ and will terminate (and output the correct topology) as every subdomain will ultimately be x-regular or y-regular.

One could think that "scaling" f to have its coefficient lie inside a "good" range could be useful to enhance rounding. This is to a very large extent a fallacy as IEEE standard machine floating numbers are encoded in the format $0.m10^e$ where m (the significand) and e (the exponent) are integers with a fixed number of digits in base 2 (23 for m and 8 for e). The operation of scaling would therefore affect primarily e without improving in any way the rounding of the rational number which primarily takes place in m. Scaling would prove useful in situations where the amplitude of the exponents of the coefficients of the polynomial is less than $2^8 (= 256)$ and the lower exponents are below -128. This situation hardly ever arises in practice where the exponents typically lay between -30 and 30 even in the most degenerate cases encountered after a great deal of subdivisions. And this is by no means close to 10^{-128} .

4.4.2 Subdivision based on rational univariate representation

Choosing the precision parameter ϵ smaller than some bound was enough to certify the output of the previous algorithm. The drawback is that the bounds are difficult to compute and are bad because uniform. The algebraic technique we present hereafter, namely RUR (rational univariate representation), is guaranteed to yield the correct topology. It allows the algorithm to use coarser approximations of roots (when the critical points of f are far away from each others).

We explain in short what RUR are in the bivariate case (see [6] for more details). When given a system of equations $E = \{f_1 = 0, f_2 = 0\}$ in $\mathbb{R}^2$ with 0-dimensional solution space, it is possible to find polynomials $P, P_1, P_2 \in \mathbb{R}[u]$ so that we have $\mathcal{Z}(E) = \left\{ \left(\frac{P_1}{P'}(\alpha), \frac{P_2}{P'}(\alpha) \right) \mid \alpha \in \mathbb{R}, \quad P(\alpha) = 0 \right\}$ where P is squarefree and P' is its derivative. In other words, the roots of E are the image of the

roots of P by a rational map. A RUR of the roots of E can be computed by finding a separating linear function and using resultant or Groebner basis techniques.

In our case the following problem arises: $\mathcal{Z}_e(f)$ can have 1-dimensional components. Because we are dealing with curves in $\mathbb{R}^2$, we can easily separate the 1-dimensional part from the 0-dimensional part by computing $g := \gcd(\partial_x f, \partial_y f)$. We define

$$\mathcal{Z}_e^1(f) = \mathcal{Z}(g), \quad \mathcal{Z}_e^0(f) = \mathcal{Z}\left(\frac{\partial_x f}{g}, \frac{\partial_y f}{g}\right).$$

Among the points in $\mathcal{Z}_e^0(f)$ we want to be able to tell those that are in $\mathcal{C}$, that is those which are singular points of $\mathcal{Z}(f)$. This way we can isolate the singular points of $\mathcal{C}$ from the rest of $\mathbb{N}_e(f)$. Since f is square-free, the singular locus $\mathcal{S}$ of f is 0-dimensional and $\mathcal{Z}_e^0(f) \cap \mathcal{C} = \mathcal{Z}_e^1(f) \cap \mathcal{C}$.

Therefore we compute (P, F_1, F_2) a RUR for $\mathcal{Z}_e^0(f)$ instead of $\mathcal{Z}_e(f)$ to isolate the critical points of f. And to tell which points are on $\mathcal{C}$ we compute $Q = \gcd(P, \operatorname{num} f(F_1, F_2))$ where num takes the numerator an irreducible rational fraction. It can be check easily that (Q, F_1, F_2) is a RUR for $\mathcal{Z}_e^0(f) \cap \mathcal{C}$ by using the fact that P' and P have no common roots.

Now, we use this RUR to isolate the roots of squarefree polynomial P using a univariate solver (see e.g.. [18]). By using interval arithmetic one can find isolating intervals for the roots of $\mathcal{Z}_e^0(f)$ by computing the images of the isolating intervals of the roots of P by $F_1 := \frac{P_1}{P'}$ and $F_2 := \frac{P_2}{P'}$. This generates boxes containing these roots. If the boxes intersect we refine the isolating intervals of the roots of P until the boxes do not intersect anymore. Finally, using again interval arithmetic, we check that g does not vanish in these isolating boxes. Otherwise we refine them until it doesn't.

Keeping the boxes which correspond to roots of Q, we obtain isolating boxes which contain a single singular point. For each isolating box $\mathcal{D}$, we compute the topological degree. If $N(f, \mathcal{D})$ is not the number of points of $\mathcal{C} \cap \partial\mathcal{D}$, we refine the isolating box.

This yields isolating boxes for the singular points of $\mathcal{C}$, which are simply singular. The complement of the isolating boxes is divided into boxes on which we apply the previous subdivision algorithm for smooth curves.

5 Examples

Algorithms presented in this paper have been completely implemented with the algebraic geometric modeling software Axel[1]. The efficiency of the topology algorithm presented here allows a real time manipulation of algebraic objects within the software, whereas current solutions usually only propose ray tracing algorithm for visualization. Here are some significant illustrations. More examples [2] and videos [3] can be found on the software's website.

The curve in figure 9 is the preimage in the parameter space of a self-intersection point of a bicubic surface. Its equation has been obtained by resultant computation. It is of total degree 76 and of degree 44 in each parameter. Its coefficients are of maximal bit size 590. It takes 7s to visualize this curve.

Figure 10 shows the discriminant curve of a bivariate system with few monomials that gives a counter-example to Kushnirenko's conjecture [13]. It is of degree 47 in x and y, and the maximal bit size of its coefficient is of order 300. It takes less that 10 seconds to visualize it within the modeler.

[1] `http://axel.inria.fr`
[2] `http://axel.inria.fr/user/screenshots`
[3] `http://axel.inria.fr/user/screencasts`

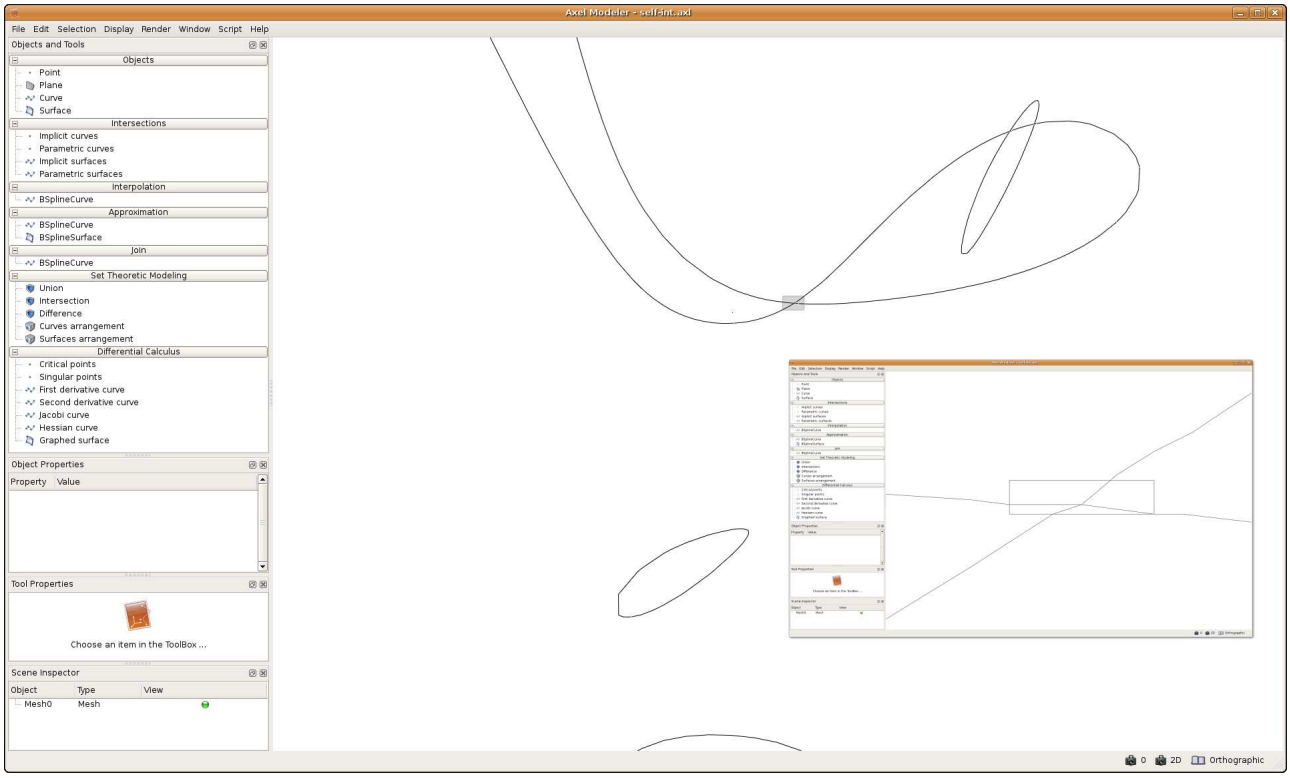


Figure 9: Topology of a high degree curve in Axel.

The area that is emphasized looks like a cusp point, but when we blow it up in the thumbnail shown in bottom right corner of figure 10, we see that it is actually made of 3 cusp points and 3 crossings. The counter-example comes from this area.

The dynamic part of the arrangement algorithm has lead to a daemon in the modeler which looks for insertions and removal of objects to maintain its data structure. The generic part has lead to an abstract algorithm implemented following the template method and visitor design patterns [21]. We have put a special emphasis on keeping the structure accessible to the user. Using the model-view-controller pattern, data structures such as the augmented influence graph and the various quadtrees can be interactively displayed and queried for point location (see figure 11).

Figure 11 illustrates how the algorithm behaves with a large amount of implicit curves of degree up to 9 in degenerate cases, that is to say with tangential intersection points and coincident singular points. In this example, the subdivision process takes no longer than 7.224 seconds, the corresponding quadtree depth is 10.

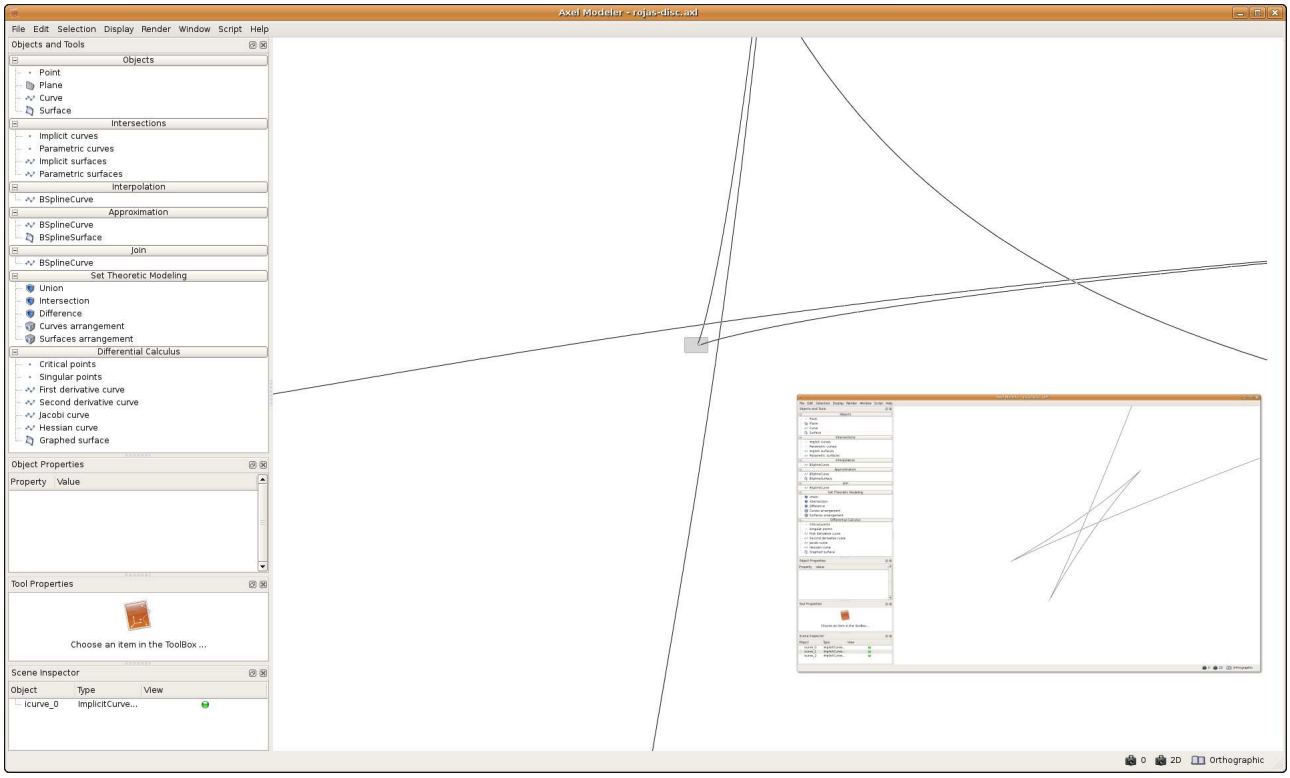


Figure 10: Topology of a curve with hidden cusp points.

6 Summary and outlook

Topology computation methods are usually compared in terms of exactness and efficiency. The method we propose certifies the topology of singular curves even for very high degrees. Moreover, using the subdivision scheme together with the Bernstein representation of input object coupled with enveloping techniques, we achieve very fast computations. To the best of our knowledge, our method is more than 20 times faster than best existing other implementations.

Arrangement computation is a new research field in computational geometry in which sweep methods are systematically employed, as a natural evolution of vertical decomposition algorithms. Such methods are very sensitive to numerical errors whereas ours benefits from the topology algorithm efficiency and correctness. Keeping track of the subdivision process also permits faster queries on the arrangement result.

Both the topology and the arrangement algorithm remain to be extended to treat surfaces. The technique has already proved reliable for computing the topology of implicit space curves [34] and the subdivision scheme naturally extends to higher dimensions.

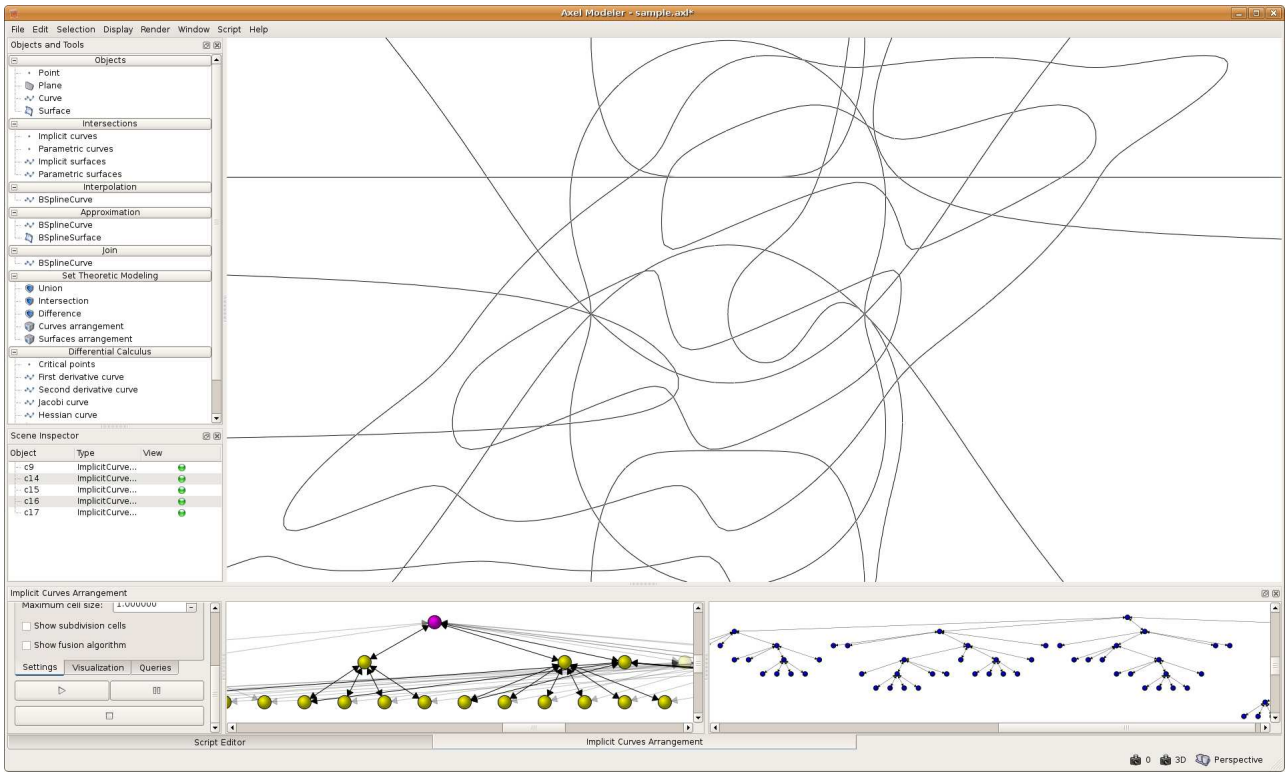


Figure 11: Computing an arrangement in Axel.

References

[1] P. Agarwal and M. Sharir. Arrangements and their applications. *Handbook of Computational Geometry (J. Sack, ed.)*, pages 49–119, 2000.

[2] L. Alberti, G. Comte, and B. Mourrain. Meshing implicit algebraic surfaces: the smooth case. In L.L. Schumaker M. Maehlen, K. Morken, editor, *Mathematical Methods for Curves and Surfaces: Tromso'04*, pages 11–26. Nashboro, 2005.

[3] L. Alberti and B. Mourrain. Visualisation of implicit algebraic curves. In Alexa Marc, Gortler Steven, and Ju Tao, editors, *Pacific Graphics*, pages 303–312, Lahaina, Maui, Hawaii États-Unis d'Amérique, 2007. IEEE Computer Society.

[4] J. Gerardo Alcázar and J. Rafael Sendra. Computing the topology of real algebraic space curves. *J. Symbolic Comput.*, 39:719–744, 2005.

[5] K. Aoki, T. Fukuda, and W. Z. Sun. On the number of branches of a plane curve germ. *Kodai Math. Journal*, 9:179–187, 1986.

[6] S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry*. Springer-Verlag, Berlin, 2003. ISBN 3-540-00973-6.

[7] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, C-28(9):643–647, September 1979.

[8] J.-D. Boissonnat and M. Yvinec. *Algorithmic geometry*. Cambridge University Press, New York, NY, USA, 1998.

[9] A. Bowyer. Svlis: Introduction and user manual, 1994.

[10] A. Bowyer, J. Berchtold, D. Eisenthal, I. Voiculescu, and K. Wise. Interval methods in geometric modeling. *GMP*, 00:321, 2000.

[11] J.-S. Cheng, X.-S. Gao, and M. Li. Determining the topology of real algebraic surfaces. In *Mathematics of Surfaces*, number 3604 in LNCS, pages 121–146. Springer-Verlag, 2005.

[12] G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Proc. 2nd GI Conference on Automata Theory and Formal Languages*, volume 33 of *Lecture Notes Comput. Sci.*, pages 134–183. Springer-Verlag, 1975.

[13] A. Dickenstein, M.J. Rojas, Rusekz K., Shihx, and J. Extremal real algebraic geometry and a-discriminants. Preprint, 2007.

[14] A. Eigenwillig, V. Sharma, and C. K. Yap. Almost tight recursion tree bounds for the descartes method. In *ISSAC '06: Proceedings of the 2006 International Symposium on Symbolic and Algebraic Computation*, pages 71–78. ACM Press. New York, NY, USA, 2006.

[15] G. Elber and M.-S Kim. Geometric constraint solver using multivariate rational spline functions. In *Proc. of 6th ACM Symposium on Solid Modelling and Applications*, pages 1–10. ACM Press, 2001.

[16] Gershon Elber and Myung-Soo Kim. Geometric constraint solver using multivariate rational spline functions. In *Proceedings of the sixth ACM Symposium on Solid Modelling and Applications*, pages 1–10. ACM Press, 2001.

[17] M. Elkadi and B. Mourrain. *Introduction à la résolution des systèmes polynomiaux*, volume 59 of *Mathématiques et Applications*. Springer, 2007.

[18] Ioannis Z. Emiris, Bernard Mourrain, and Elias P. Tsigaridas. Real algebraic numbers: Complexity analysis and experimentations. In *Reliable Implementation of Real Number Algorithms: Theory and Practice*, LNCS, page (to appear). Springer-Verlag, 2008.

[19] G. Farin. *Curves and surfaces for computer aided geometric design : a practical guide*. Comp. science and sci. computing. Acad. Press, 1990.

[20] E. Fogel, D. Halperin, L. Kettner, M. Teillaud, R. Wein, and N. Wolpert. Arrangements. In Jean-Daniel Boissonnat and Monique Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*, pages 1–66. Springer-Verlag, Mathematics and Visualization, 200.

[21] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[22] J. Garloff and A. P. Smith. Investigation of a subdivision based algorithm for solving systems of polynomial equations. In *Proceedings of the Third World Congress of Nonlinear Analysts, Part 1 (Catania, 2000)*, volume 47, pages 167–178, 2001.

[23] G. Gatellier, A. Labrouzy, B. Mourrain, and J.-P. Técourt. Computing the topology of 3-dimensional algebraic curves. In *Computational Methods for Algebraic Spline Surfaces*, pages 27–44. Springer-Verlag, 2005.

[24] L. González-Vega and I. Necula. Efficient topology determination of implicitly defined algebraic plane curves. *Comput. Aided Geom. Design*, 19(9):719–743, 2002.

[25] T. A. Grandine and F. W. Klein. A new approach to the surface intersection problem. *Computer Aided Geometric Design*, 14:111–134, 1997.

[26] D. Halperin. Arrangements. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 24, pages 529–562. CRC Press LLC, Boca Raton, FL, 2004.

[27] D. Halperin and M. Sharir. Arrangements and their applications in robotics: recent developments. In *WAFR: Proceedings of the workshop on Algorithmic foundations of robotics*, pages 495–511, Natick, MA, USA, 1995. A. K. Peters, Ltd.

[28] I. Hanniel and R. Wein. An exact, complete and efficient computation of arrangements of bézier curves. In *SPM '07: Proceedings of the 2007 ACM symposium on Solid and physical modeling*, pages 253–263, New York, NY, USA, 2007. ACM.

[29] J. Hass, R. T. Farouki, C. Y. Han, X. Song, and T. W. Sederberg. Guaranteed Consistency of Surface Intersections and Trimmed Surfaces Using a Coupled Topology Resolution and Domain Decomposition Scheme. *Advances in Computational Mathematics*, 2007.

[30] Younis Hijazi and Thomas Breuel. Computing arrangements using subdivision and interval arithmetic. In *Proceedings of the Sixth International Conference on Curves and Surfaces Avignon*, 2006.

[31] Seong Joon-Kyung, G. Elber, and Kim Myung-Soo. Contouring 1- and 2-Manifolds in Arbitrary Dimensions. In *SMI'05*, pages 218–227, 2005.

[32] J. Keyser, T. Culver, D. Manocha, and S. Krishnan. Efficient and exact manipulation of algebraic points and curves. *Computer-Aided Design*, 32(11):649–662, 2000.

[33] G. N. Khimšiašvili. The local degree of a smooth mapping. *Sakharth. SSR Mecn. Akad. Moambe*, 85(2):309–312, 1977.

[34] Chen Liang, B. Mourrain, and J.P. Pavone. Subdivision methods for 2d and 3d implicit curves. In *Computational Methods for Algebraic Spline Surfaces*. Springer-Verlag, 2006.

[35] N. G. Lloyd. *Degree theory*. Cambridge University Press, Cambridge, 1978. Cambridge Tracts in Mathematics, No. 73.

[36] W. Lorensen and H. Cline. Marching cubes: a high resolution 3d surface construction algor ithm. *Comput. Graph.*, 21(4):163–170, 1987.

[37] V. Milenkovic. Calculating approximate curve arrangements using rounded arithmetic. In *SCG '89: Proceedings of the fifth annual symposium on Computational geometry*, pages 197–207, New York, NY, USA, 1989. ACM Press.

[38] V. Milenkovic and E. Sacks. An approximate arrangement algorithm for semi-algebraic curves. In *SCG '06: Proceedings of the twenty-second annual symposium on Computational geometry*, pages 237–246, New York, NY, USA, 2006. ACM.

[39] B. Mourrain and J.-P. Pavone. Subdivision methods for solving polynomial equations. Technical Report 5658, INRIA Sophia-Antipolis, 2005.

[40] B. Mourrain, F. Rouillier, and M.-F. Roy. *Bernstein's basis and real root isolation*, pages 459–478. Mathematical Sciences Research Institute Publications. Cambridge University Press, 2005.

[41] B. Mourrain, J.-P. Técourt, and M. Teillaud. On the computation of an arrangement of quadrics in 3d. *Comput. Geom. Theory Appl.*, (30):145–164, 2005. Special issue, 19th European Workshop on Computational Geometry, Bonn.

[42] B. Mourrain and J.P.. Técourt. Isotopic meshing of a real algebraic surface. Technical Report 5508, INRIA Sophia-Antipolis, 2005.

[43] B. Mourrain, M. Vrahatis, and J. Yakoubsohn. On the complexity of isolating real roots and computing with certainty the topological degree. *Journal of Complexity*, 18:612–640, 2002.

[44] S. Plantinga and G. Vegter. Isotopic approximation of implicit curves and surfaces. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 245–254, New York, NY, USA, 2004. ACM.

[45] Helmut Ratschek and Jon G. Rokne. Scci-hybrid methods for 2d curve tracing. *Int. J. Image Graphics*, 5(3):447–480, 2005.

[46] A. Rösch, M. Ruhl, and D. Saupe. Interactive visualization of implicit surfaces with singularities. In *Implicit Surfaces*, pages 73–87, 1996.

[47] T. Sederberg. Algorithm for algebraic curve intersection. *Computer Aided Design*, 21:547–554, 1989.

[48] E. C. Sherbrooke and N. M. Patrikalakis. Computation of the solutions of nonlinear polynomial systems. *Comput. Aided Geom. Design*, 10(5):379–405, 1993.

[49] E.C. Sherbrooke and N.M. Patrikalakis. Computation of the solutions of nonlinear polynomials systems. *Computer Aided Geometric Design*, 10(5):379–405, October 1993.

[50] F. Stenger. Computing the topological degree of a mapping in $\mathbf{R}^n$. *Numer. Math.*, 25(1):23–38, 1975.

[51] Z. Szafraniec. On the number of branches of a 1-dimensional semianalytic set. *Kodai Math. J.*, 11(1):78–85, 1988.

[52] J. von zur Gathen and J. Gerhard. *Modern computer algebra*. Cambridge University Press, New York, 1999.

[53] J. Wintz and B. Mourrain. A subdivision arrangement algorithm for semi-algebraic curves: an overview. In Alexa Marc, Gortler Steven, and Ju Tao, editors, *Pacific Conference on Computer Graphics and Applications 2007*, pages 449–452, Lahaina, Maui, Hawaii États-Unis d'Amérique, 2007. IEEE Computer Society.

[54] N. Wolpert. Jacobi curves: Computing the exact topology of arrangements of non-singular algebraic curves. *In ESA 2003, LNCS 2832, pages 532–543, 12*, 2003.

[55] B. Zalik, M. Gombosi, and D. Podgorelec. A quick intersection algorithm for arbitrary polygons. In L. Szirmay-Kalos, editor, *Spring Conference on Computer Graphics - SCCG98*, pages 195–204, 1998.