An Adapted Version of the Bentley-Ottmann Algorithm for Invariants of Plane Curves Singularities

Madalina Hodorog, Bernard Mourrain, Joseph Schicho

An Adapted Version of the Bentley-Ottmann Algorithm for Invariants of Plane Curves Singularities

Mădălina Hodorog[1], Bernard Mourrain[2], and Josef Schicho[1]

[1] Johann Radon Institute for Computational and Applied Mathematics, Austrian Academy of Sciences, Altenbergerstrasse 69, Linz, Austria
[2] INRIA Sophia-Antipolis, 2004 route des Lucioles, B.P. 93, 06902 Sophia-Antipolis, France
{madalina.hodorog,josef.schicho}@oeaw.ac.at, Bernard.Mourrain@inria.fr

Abstract. We report on an adapted version of the Bentley-Ottmann algorithm for computing all the intersection points among the edges of the projection of a three-dimensional graph. This graph is given as a set of vertices together with their space Euclidean coordinates, and a set of edges connecting them. More precisely, the three-dimensional graph represents the approximation of a closed and smooth implicitly defined space algebraic curve, that allows us a simplified treatment of the events encountered in the Bentley-Ottmann algorithm. As applications, we use the adapted algorithm to compute invariants for each singularity of a plane complex algebraic curve, i.e. the Alexander polynomial, the Milnor number, the delta-invariant, etc.

Keywords: adapted Bentley-Ottmann algorithm, sweep technique, graph data structure, implicitly defined space algebraic curve, topological invariants, plane curves singularities

1 Introduction

Computational geometry algorithms are used in many applications domains, such as robotics, computer vision, computer aided design, geographic information systems, etc. One of these algorithms, i.e. the Bentley-Ottmann algorithm for reporting the pairwise intersections among a set of objects in the plane, proved itself useful in many applications from combinatorial geometry and computer graphics. A generalized version of the Bentley-Ottmann algorithm [4] computes the pairwise intersections among geometric objects in $\mathbb{R}^d$. The Bentley-Ottmann algorithm uses a *sweep* technique, i.e. a sweep plane (or a sweep line in $\mathbb{R}^2$) sweeps the space $\mathbb{R}^d$ (or $\mathbb{R}^2$) that contains a set of geometric objects. At certain positions called event points, the sweep is interrupted and the problem is locally solved. The sweep is greedy, without any backtracking.

In this paper, we propose an adapted version of the Bentley-Ottmann algorithm [3] for computing all the intersection points among the edges of the

projection of a 3-dimensional graph. In addition, the adapted algorithm computes some extra information on each intersection point and on the pair of edges that contains it. For our purpose, the adapted Bentley-Ottmann algorithm operates on a 3-dimensional graph data structure, which represents the piecewise linear approximation of a closed and smooth space algebraic curve, implicitly defined as the intersection of two algebraic surfaces. We compute this space algebraic curve as the link of the singularity of a plane complex algebraic curve, as described in [10].

We manage the adapted version of the Bentley-Ottmann algorithm in a simpler way than in the original version because the 3-dimensional graph has some special properties [6]: (i) it consists of several cycles; (ii) it is a regular graph, i.e. it contains no loops or multiple edges; (iii) and its projection contains at most one crossing point. The first two properties are always guaranteed since the 3-dimensional graph represents the piecewise linear approximation of an implicitly defined space algebraic curve, which is closed and smooth (i.e. it does not intersects itself). We perform a test to check whether the third property holds for the given 3-dimensional graph and if the test fails, then we report a failure message. Using the free algebraic geometric modeler Axel [13] we compute efficient and robust results.

For our purpose, the adapted Bentley-Ottmann algorithm offers essential benefits: it allows us to compute the Alexander polynomial of the singularity of a plane complex algebraic curve as reported in [8]. From the Alexander polynomial we compute other invariants of the singularity, e.g. the Milnor number and the delta-invariant. In this way, we recover topological local information on each singularity of a plane complex algebraic curve. Thus we can use the adapted algorithm to solve a specific problem from algebraic geometry, i.e. the problem of computing several topological invariants for each singularity of a plane complex algebraic curve. These topological invariants play an important role in the classification and the analysis of the singularities of a plane complex algebraic curve as discussed in [2].

2 Description of the Algorithm

2.1 Data Structures

For our study, we define a 3-dimensional graph data structure as follows:

Definition 1 *A (3-dimensional) graph is defined as a pair* $\mathcal{G} = \langle V, E \rangle$*, where* V *is a list of points (vertices) in the 3-dimensional space together with their Euclidean coordinates, and* E *is a list of edges connecting them, i.e.* $V = \{p(x, y, z) \in \mathbb{R}^3\}$ *and* $E = \{e(i, j) | i, j \in V\}$.

We are interested in the following elements of a 3-dimensional graph:

Definition 2 *A point (or vertex) in the 3-dimensional graph is a 4-tuple of the form* $p(index, x, y, z)$*, where* $index \in \mathbb{Z}$ *uniquely identifies each point in the*

graph, and $(x, y, z) \in \mathbb{R}^3$ *are the Euclidean coordinates of the point. An edge in the* 3*-dimensional graph is defined as a 2-tuple* $e(s, d)$*, where* s *is the index of the source point of* e *and* d *is the index of the destination point of* e*, see Figure 1.*

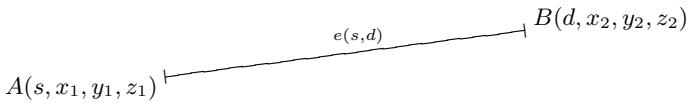


Fig. 1. An edge $e(s, d)$ in a 3-dimensional graph. The edge e is determined by its source point $A(s, x_1, y_1, z_1)$ and by its destination point $B(d, x_2, y_2, z_2)$, where $s, d \in \mathbb{Z}$ uniquely identify the points A, B and $(x_1, y_1, z_1), (x_2, y_2, z_2) \in \mathbb{R}^3$ are the Euclidean coordinates of A, B.

We introduce the following notations: (i) we use $xycoord(index)$ for denoting the x, y coordinates of $index$ and $ycoord(index)$ for denoting the y coordinate of $index$; (ii) we access the i-th component of a list sw with the underscore notation for the index i, i.e sw_i. We consider that the indexes of a list start from 0.

We recall that a *path* in the 3-dimensional graph is a sequence of consecutive edges in a graph, and a *cycle* (*circuit*) is a path which ends at the vertex it begins. In addition, a *loop* is an edge that connects a vertex to itself, and *multiple edges* are two or more edges connecting the same two vertices, see [6] for details.

We assume that the 3-dimensional graph is simple (regular), i.e. it has no multiple edges or loops as in Figure 2. For our purpose, we are interested in the projection of a 3-dimensional graph which always consists of several cycles, see Figure 3(b) for an example. We consider the edges of a 3-dimensional graph $\mathcal{G}$ to be "small" edges, i.e. the projection of any edge of $\mathcal{G}$ has at most one crossing point. If this property is not true for a certain pair of edges from a 3-dimensional graph, then we report a failure message during runtime.

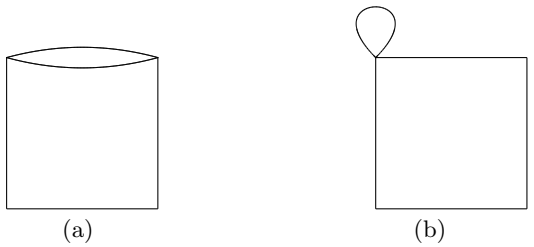
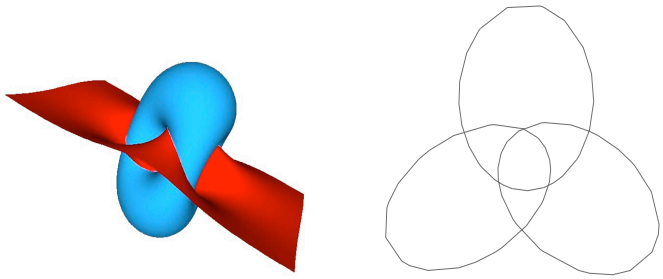


Fig. 2. (a) A graph with multiple edges. (b) A graph with a loop.



(a) (b)

Fig. 3. (a) Two algebraic surfaces that implicitly define as their intersection a closed and smooth space algebraic curve computed as a 3-dimensional graph $\mathcal{G}$ with 3 cycles. (b) The projection of the 3-dimensional graph $\mathcal{G}$ with 3 cycles from (a). Pictures produced with GENOM3CK in Axel, see Section 4 for details

Remark 1 *The 3-dimensional graph that we study in this paper represents the piecewise linear approximation of a closed and smooth implicitly defined space algebraic curve. We define this curve as the link of the singularity* $(0,0)$ *of a plane complex algebraic curve* $\mathcal{C}$*, which characterizes completely the topology of the curve* $\mathcal{C}$ *around its singularity* $(0,0)$*. For instance, in Figure 3(b) we visualize the link of the singularity* $(0,0)$ *of the plane complex algebraic curve defined by the squarefree polynomial* $x^3 - y^3 = 0$*. In the literature [1], [12], the 3-dimensional graph computed as the piecewise linear approximation of an implicitly defined space algebraic curve is called the topology of the curve. We use the Axel [13] free algebraic geometric modeler to compute the 3-dimensional graph as presented in [8], [10]. For the special case of smooth implicitly defined space algebraic curves, Axel uses certified algorithms to compute their topology.*

We state the problem that we want to solve:

Problem 1 *Given a 3-dimensional graph* $\mathcal{G} = \langle V, E \rangle$ *as in Definitions 1 and 2, which has only "small" edges, which is regular and which consists of several cycles, our goal is to compute the intersection points among all the edges of the projection of* $\mathcal{G}$*. In addition, we compute some extra information: (i) for each intersection point* P *find the pair of edges* (e_m, e_n) *that contains it. (ii) the pair of edges* (e_m, e_n) *is ordered, i.e.* e_m *is under* e_n *in* $\mathbb{R}^3$*.*

2.2 Methods

To solve Problem 1, we first compute the intersection points of all the edges of the projection of a 3-dimensional graph, and for each intersection point, we compute the pair of edges that contains it. For this purpose, we design a sweep line based algorithm as the Bentley-Ottmann algorithm from [3]. We distinguish several steps for our algorithm, that we describe in comparison with the original

Bentley-Ottmann algorithm:
Step 1 (Ordering criteria). The edges of the projection of the 3-dimensional graph $\mathcal{G}$ are oriented from left to right and they are ordered in the list of edges $E = \{e_0, ..., e_N\}$ as in Figure 4: (1) by the x-coordinates of their source points; (2) if the x-coordinates of the source points of two edges coincide, then the two edges are ordered by the two slopes of their supporting lines; (3) if the x-coordinates of the source points and the slopes of two edges coincide, then the two edges are ordered by the y-coordinates of their destination points. The ordering criteria is necessary for the correctness of the algorithm.

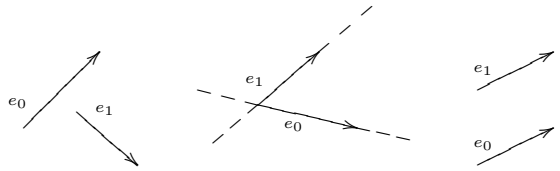


Fig. 4. Ordering criteria for the edges

Step 2 (Sweep line paradigm). As in the Bentley-Ottmann algorithm, we consider a vertical sweep line l that sweeps the plane from left to right. While l moves, it intersects several edges from E, which are stored in a list denoted SW and that we call the sweep list. SW changes while l sweeps the plane and it is updated only at certain points of the edges from E called event points. In this algorithm, the sweep list SW is ordered by the y-coordinates of the intersections of the edges of E with the sweep line l. As in the Bentley-Ottmann algorithm, SW represents the status of the algorithm.
Step 3 (Sweep line management). We observe that in E each *index* appears two times since E always contains several cycles. This allows us to manage SW in a simpler way in our adapted Bentley-Ottmann algorithm than in the original version. While we traverse E, we insert the current edge $e_m(s_m, d_m)$ from E in SW in the right position and that is: (1) we search for an edge $e_n(s_n, d_n)$ in SW such that its destination coincide with the source of $e_m \in E$, i.e. $d_n = s_m$; if we find such an $e_n \in SW$ we replace it with $e_m \in E$; (2) if such an edge $e_n \in SW$ does not exist, we insert e_m in SW depending on its position against the current edges from SW. We assume $SW = \{e_0^i, e_1^i, e_2^i, ..., e_k^i\}$, with $e_q^i \in E$ for all $q \in \{1, ..., k\}$. There exists a unique index j with $0 \leq j \leq k$ such that $ycoord(s_m)$ is larger than the y-coordinates of all the intersections of $e_0^i, ..., e_j^i$ with l, and smaller than the y-coordinates of all the intersections of $e_{j+1}^i, ..., e_k^i$ with l. This index j can be found by checking all the signs of the determinants constructed with $(xycoord(s_m), 1), (xycoord(s_j^i), 1)$ and $(xycoord(d_j^i), 1)$. Then we insert e_m in SW between the two edges e_j^i and e_{j+1}^i and we obtain $SW = \{e_0^i, e_1^i, ..., e_j^i, e_m, e_{j+1}^i, ..., e_k^i\}$. When we insert an edge from E into SW on the

right position, we have to additionally update SW depending on the encountered event points:

- we test each inserted edge in SW against its two neighbors for intersection. If an intersection point P is found we report it together with the pair of edges that contains it. In addition, we swap the edges that intersect in SW. As opposed to the original Bentley-Ottmann algorithm after swapping the edges in SW, we do not test the edges against their new neighbors for intersections because we consider only "small" edges.
- we test each inserted edge in SW against its two neighbors for common destination. In addition, when two edges are swapped in SW after reporting their intersection point, we test them against their new neighbors for common destination. Whenever we find two consecutive edges with common destinations we erase them from SW. As opposed to the original Bentley-Ottmann algorithm after deleting edges from SW, we do not test the new neighbors for intersection because we consider only "small" edges.

We notice that in the adapted Bentley-Ottmann algorithm we basically process the pre-ordered list of edges E in a for-loop in a way which makes the explicit use of a sweep list redundant.

Remark 2 *We mention briefly a way to modify the adapted Bentley-Ottmann algorithm such that in the case of a* 3*-dimensional graph* $\mathcal{G}$ *with "long" edges (i.e. the projection of any edge of* $\mathcal{G}$ *has at least one crossing point), the algorithm would detect all the intersection points and would not only report a failure message at runtime. The main idea is to update the ordered list of edges* E *and the sweep list* SW *each time the algorithm reports an intersection point as follows: if the algorithm reports the intersection point* $P(x, y) \in \mathbb{R}^2$ *together with the pair of edges* (e_1, e_2) *that contains* $P(x, y)$*, then for* $i = 1, 2$ *we split each edge of intersection* e_i *in two new edges* e_i^l, e_i^r*. The new vertices* e_i^l *are determined by the source point of* e_i *and by the coordinates of* $P(x, y)$*, while the new vertices* e_i^r *are determined by the coordinates of* $P(x, y)$ *and by the destination point of* e_i*, as described in Figure 5. Then we update the lists* SW *and* E *as follows: we replace the edges* e_i *by* e_i^l *in* SW*, and we insert the* e_i^r *edges in* E *following the ordering criteria from* ***Step 1****, Figure 4.*

In the following we assume that we have computed: (1) a list $I = \{(x_i, y_i) \in \mathbb{R}^2\}$ of the intersection points of all the edges of the projection of a 3-dimensional graph; (2) and a list EI of pairs of edges for I such that the i-th element of EI represents the pair of edges that contains the i-th intersection point from I. In the example from Figure 3(b), our adapted Bentley-Ottmann algorithm computes all the 6 intersection points together with the list of pairs of edges that contain these intersection points.

To solve Problem 1, we now have to order each pair of edges from EI depending on the Euclidean space coordinates of the intersection points from I. For instance, in Figure 6 we consider $P(x, y) \in I$ the intersection point of the pair of edges $(e_1, e_2) \in EI$. We order this pair such that the first component

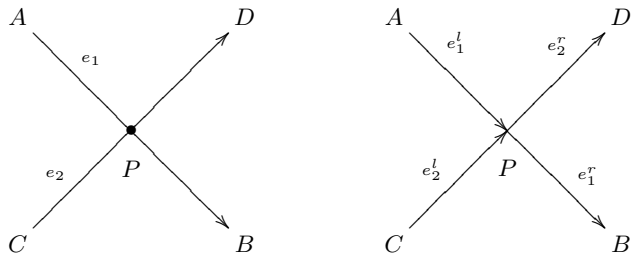


Fig. 5. Refinements of the algorithm

always lies under the second component in $\mathbb{R}^3$. We assume that for $i = \{1, 2\}$ the source and the destination points of e_i are $A_i(a_i, b_i, 0)$, $B_i(d_i, e_i, 0)$, which are the projections of $A_i^{'}(a_i, b_i, c_i)$, $B_i^{'}(d_i, e_i, f_i)$ from $\mathbb{R}^3$. To order the pair of edges we proceed as follows:

1. For $i = \{1, 2\}$ we compute the equations of the support lines L_i for the edges e_i in $\mathbb{R}^2$. We use the determinant formula for the equations of the lines L_i and we obtain:
$$L_i(x, y) : \det \begin{pmatrix} a_i & b_i & 1 \\ d_i & e_i & 1 \\ x & y & 1 \end{pmatrix} = 0\,, \tag{1}$$
and thus $L_i(x, y) : (b_i - e_i)x + (d_i - a_i)y + a_i e_i - b_i d_i = 0\,.$
2. We compute the coordinates z_1, z_2 of $P_1(x, y, z_1)$ and $P_2(x, y, z_2)$ in $\mathbb{R}^3$ as in Figure 6. As an example we compute z_1 (we proceed in the same way for z_2). Firstly we compute α_1 from
$$\alpha_1 L_2(A_1) + (1 - \alpha_1)L_2(B_1) = 0\,. \tag{2}$$
Then we compute z_1 as $z_1 = \alpha_1 c_1 + (1 - \alpha_1)f_1\,.$
3. If $z_1 < z_2$ then e_1 is under e_2 in $\mathbb{R}^3$ and we return the pair (e_1, e_2) for $P(x, y)$ (i.e. e_1 is the undergoing edge and e_2 is the overgoing edge for (e_1, e_2)); otherwise e_2 is under e_1 in $\mathbb{R}^3$ and we thus return the pair (e_2, e_1) for $P(x, y)$ (i.e. e_2 is the undergoing edge and e_1 is the overgoing edge for (e_2, e_1)), as in the example from Figure 6.

3 Applications of the Algorithm

Our main goal is to compute the topological invariants for each singularity of a plane complex algebraic curve. For this purpose, it is essential for the adapted Bentley-Ottmann algorithm to compute the extra information on each detected pair of edges that contains an intersection point, i.e. to order the edges as described in Subsection 2.2, Figure 6 such that we distinguish the undergoing

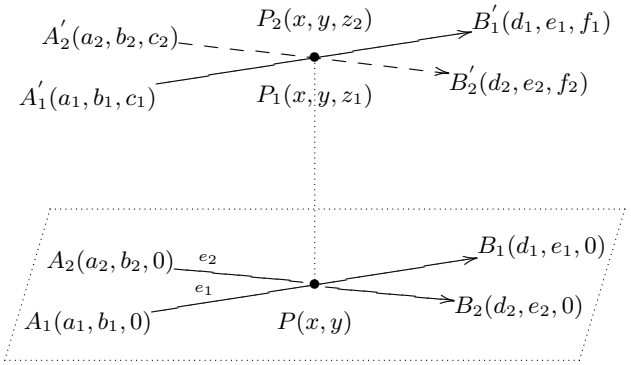


Fig. 6. Ordering the pair of edges that contains an intersection point

edge from the overgoing edge in the pair. This extra information allows us: to compute for each 3-dimensional graph a special type of projection in the 2-dimensional space called diagram. A diagram is a special type of projection in the 2-dimensional space together with the information on each crossing telling which branch goes under and which goes over. This information is captured by creating a break in the branch going underneath. In our case, this information is provided by the undergoing edge. For instance, in Figure 7 we visualize the diagram of the graph data structure $\mathcal{G}$ from Figure 3.

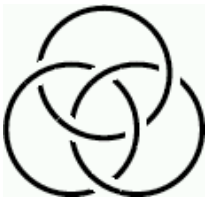


Fig. 7. Diagram of the graph $\mathcal{G}$ from Figure 3

Additionally, we compute all the cycles of a 3-dimensional graph [7]. For example, in Figure 8 we notice the 3 cycles of the graph data structure from Figure 3.

From this diagram and the cycles of the graph, we compute the Alexander polynomial of each singularity of the plane complex algebraic curve as presented in [8]. Furthermore, from the Alexander polynomial we compute other topological invariants for the plane complex algebraic curve, i.e. the Milnor number, the delta-invariant of each singularity and the genus, as described in [10].

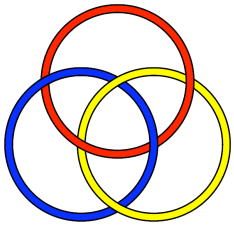


Fig. 8. The 3 cycles of the graph $\mathcal{G}$ from Figure 3

4 Implementation of the Algorithm

We implemented the adapted Bentley-Ottmann algorithm in GENOM3CK [9], a library which was originally developed for GENus cOMputation of plane Complex algebraiC Curves using Knot theory. The library is written in the free algebraic geometric modeler Axel [13] and in the free computer algebra system Mathemagix [11], i.e in C++ using Qt Script for Applications and Open Graphics Library. At present, the library is available for both Macintosh and Linux. More information about GENOM3CK (including documentation, download and installation instructions) can be found at `http://people.ricam.oeaw.ac.at/m.hodorog/software.html`. For an example, see Figure 9, where we visualize the 6 intersection points among all the edges of the projection of the 3-dimensional graph from Figure 3(b).

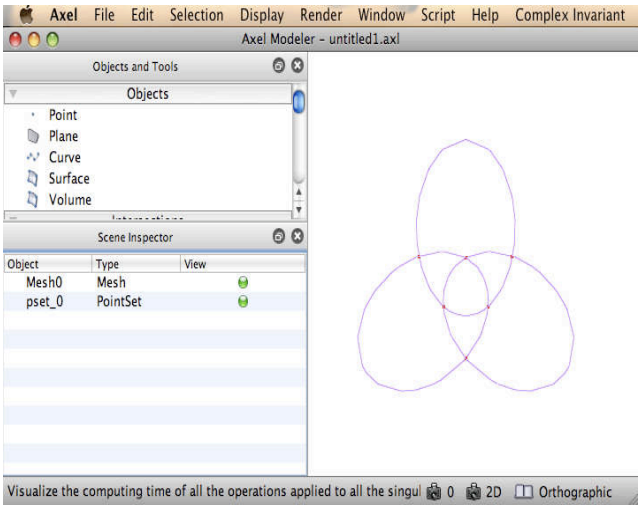


Fig. 9. Implementation of the adapted Bentley-Ottmann algorithm in GENOM3CK

We give some reasons for motivating our choice to use Axel [13] for the implementation of the algorithms. The Computational Geometry Algorithms Library, CGAL [5], implemented in C++ is the standard library in the computational geometry community. The library provides data structures and algorithms that operate on geometric objects and thus it represents a good candidate to implement algorithms in computational geometry. Still for our purpose, the Bentley-Ottmann algorithm implemented in CGAL must provide for each detected intersection point the extra information on the reported pair of edges of intersection as discussed in Subsection 2.2, Figure 6. In our study, we compute the input data for the adapted Bentley-Ottmann algorithm (i.e. the 3-dimensional graph) using Axel as explained in Remark 1. Axel offers algebraic and geometric tools for computing the topology of smooth space algebraic curves as a 3-dimensional graph data structure in a certified way. To use directly this computed 3-dimensional graph and to obtain the essential extra information on each intersection point, we use Axel for the implementation of the adapted Bentley-Ottmann algorithm in a simplified form. Consequently, the adapted Bentley-Ottmann algorithm allows us to design more algorithms for computing the topological properties of each singularity of a plane complex algebraic curve as described in [10]. Another reason for the choice of the implementation system was that we wanted to write our package in one language, and we also needed algebraic functions for surface-surface intersection, which are provided by Axel's libraries. To our knowledge, at present this is not implemented (yet) in CGAL.

5 Conclusion

We presented an adapted version of the Bentley-Ottmann algorithm for computing all the intersection points among the edges of the projection of a regular 3-dimensional graph, which consists of several cycles. We computed efficient and robust results using for the implementation free systems as Axel and Mathemagix. As applications, we use the adapted algorithm to solve a particular problem from algebraic geometry, i.e. the problem of computing the topological invariants for each singularity of a plane complex algebraic curve.

Acknowledgments. Many thanks to Julien Wintz, for the contribution to the implementation of the library in its starting phase and for creating the Axel software. This work is supported by the Austrian Science Funds (FWF) under the grant W1214/DK9. Bernard Mourrain is partially supported by the Marie Curie ITN SAGA [PITN-GA-2008-214584] of the European Community's Seventh Framework Programme [FP7/2007-2013].

References

1. Alcazar, J.G., Sendra, R.: Computation of the Topology of Real Algebraic Space Curves. Journal of Symbolic Computation 39 (6), 719–744 (2005)

2. Arnold, V.I., Varchenko, A.N., Gusein-Zade, S.M.: Singularities of Differentiable Maps: Volume 1. Birkhäuser, Boston (1985)
3. Berg, M., Krefeld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry: Algorithms and Applications. Second Edition. Springer, Berlin (2008)
4. Bieri, H., Schmidt, P.M.: An On-Line Algorithm for Constructing Sweep Planes in Regular Position. In: Bieri, H., Noltemeier, H. (eds.), International Workshop on Computational Geometry 1991. LNCS, vol. 553, pp. 27–35. Springer Berlin Heidelberg (1991)
5. CGAL - Computational Geometry Algorithms Library, `http://www.cgal.org/`
6. Diestel, R.: Graph Theory. Graduate Texts in Mathematics, Springer-Verlag Heidelberg (2005)
7. Hodorog, M., Schicho, J.: Computational Geometry and Combinatorial Algorithms for the Genus Computation Problem. DK Report 2010-07, Johannes Kepler University, Linz (2010)
8. Hodorog, M., Mourrain, B., Schicho, J.: A Symbolic-Numeric Algorithm for Computing the Alexander Polynomial of a Plane Curve Singularity. In: Ida, T., Negru, V., Jebelean, T., Petcu, D., Watt, S., Zaharie, D. (eds.), 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, pp. 21-28. IEEE Computer Society, Los Alamitos (2010)
9. Hodorog, M., Mourrain, B., Schicho, J.: GENOM3CK - A Library for Genus Computation of Plane Complex Algebraic Curves using Knot Theory. In: ACM SIGSAM Communications in Computer Algebra, vol. 44, no. 4, issue 174, pp. 198-200. Association for Computing Machinery, Special Interest Group on Symbolic and Algebraic Manipulation (2010)
10. Hodorog, M., Schicho, J.: A Symbolic-Numeric Algorithm for Genus Computation. In: Langer, U., Paule, P. (eds.), Numerical and Symbolic Scientific Computing: Progress and Prospects, to appear. Springer Wien (2011)
11. Hoeven, V.D.J., Lecerf, G., Mourrain, B.: Mathemagix Computer Algebra System, `http://www.mathemagix.org/`
12. Liang, C., Mourrain, B., Pavone, J.P.: Subdivision Methods for 2d and 3d Implicit Curves. In: Jütller, B., Piene, R. (eds.) Geometric Modeling and Algebraic Geometry, pp. 199–214. Springer-Verlag Berlin Heidelberg (2008)
13. Wintz, J., Chau, S., Alberti, L., Mourrain, B.: Axel Algebraic Geometric Modeler, `http://axel.inria.fr/`