On reliable visualization algorithms for real algebraic curves and surfaces

Dissertation
zur Erlangung des
Doktorgrades der Naturwissenschaften (Dr. rer. nat.)

der

Naturwissenschaftlichen Fakultät III
Agrar- und Ernährungswissenschaften,
Geowissenschaften und Informatik

der Martin-Luther-Universität Halle-Wittenberg

vorgelegt von

Christian Stussak
Geb. am 17. April 1982 in Torgau

Diese Arbeit wurde am Institut für Informatik der Martin-Luther-Universität Halle-Wittenberg in der Arbeitsgruppe von Doz. Dr. Peter Schenzel angefertigt und folgenden Gutachtern zur Bewertung vorgelegt:

1. Doz. Dr. Peter Schenzel (Martin-Luther-Universität Halle-Wittenberg)
2. doc. RNDr. Pavel Chalmoviansky, PhD. (Comenius-Universität Bratislava)

Datum der Verteidigung:

24. Oktober 2013

Abstract

This work presents algorithms for visualizing real algebraic plane and space curves as well as a certain type of algebraic surface given by embedding blowups of the plane into affine space.

The curve tracing based rasterization of real algebraic plane curves is exact in the sense that a pixel is painted if and only if it contains a point of the curve. The exactness is achieved by symbolic and certified numerical methods, where the latter also help to reduce the computation time. The efficiency of the new method is illustrated by means of complexity analysis and practical experiments with curves that are especially difficult to process.

Real algebraic space curves are visualized by determining a line strip approximation to their segments using a projection and lifting approach. Intentionally ignoring a finite set of points of the curve allows to introduce a new notion of genericity. In combination with a novel approach for lifting multiple components of the projection, this results in a robust and efficient visualization algorithm.

A blowup of the plane in finitely many points yields a surface that can be embedded into affine space using a rational parametrization. Since this parametrization has base points, its direct visualization is time consuming and prone to numerical errors. An implicitization technique is proposed and it is shown that the implicit definition removes the limitations related to the parametrization. The result is an interactive visualization in real time.

In all three visualization algorithms, resultants serve as building blocks for achieving results of high quality. In order to alleviate the cost of this symbolic operation, a parallel implementation of bivariate polynomial resultants on graphics processing units is presented. It is based on the parallel processing of homomorphic images of the input and shows high speedups compared to the sequential approach.

Zusammenfassung

In der vorliegenden Arbeit werden Algorithmen zur Visualisierung von reellen algebraischen Kurven in der Ebene und im Raum, sowie von Flächen, welche bei der Einbettung von Aufblasungen der Ebene in endlichen vielen Punkten in den affinen Raum entstehen, entwickelt.

Die Erzeugung von Rasterbildern ebener algebraischer Kurven durch einen Kurvenverfolgungsalgorithmus ist exakt in dem Sinne, dass ein Pixel genau dann gezeichnet wird, wenn es einen Punkt der Kurve enthält. Die Exaktheit wird durch eine Kombination symbolischer und selbstvalidierender numerischer Verfahren gewährleistet. Die Effizienz des neuen Verfahrens wird anhand einer Komplexitätsanalyse sowie durch Experimente mit schwer visualisierbaren Kurven demonstriert.

Reelle algebraische Raumkurven werden mit Hilfe einer Linienzugapproximation dargestellt. Diese werden durch ein Verfahren erzeugt, welches die Kurve in die Ebene projiziert, dort vorverarbeitet und anschließend in den Raum anhebt. Indem bewusst auf endlich viele Punkte der Kurve verzichtet wird, kann ein neuer Generizitätsbegriff eingeführt werden. Zusammen mit einem neuen Ansatz zur Anhebung mehrfacher Komponenten der Projektion führt dies zu einem robusten und effizienten Visualisierungsalgorithmus.

Eine Aufblasung der Ebene in endlich vielen Punkten ergibt eine Fläche, welche durch eine rationale Parametrisierung in den affinen Raum eingebettet werden kann. Da diese Parametrisierung Definitionslücken enthält, ist ihre direkte Visualisierung rechenintensiv und anfällig für numerische Fehler. In dieser Arbeit wird eine Implizitisierungstechnik vorgeschlagen und es wird gezeigt, dass die implizite Definition der Fläche die Einschränkungen der Parametrisierung aufhebt. Im Ergebnis erhält man eine interaktive Visualisierung der Fläche in Echtzeit.

Alle drei Visualisierungsverfahren nutzen Resultanten als wichtiges Hilfsmittel zur Erzeugung qualitativ hochwertiger Ergebnisse. Um die Kosten dieser symbolischen Operation abzumildern, wird die Implementierung eines parallelen Algorithmus zur Berechnung von Resultanten bivariater Polynome auf Grafikhardware vorgestellt. Diese basiert auf der gleichzeitigen Verarbeitung mehrerer homomorpher Bilder der Eingabe. Dies führt zu hohen Beschleunigungsraten gegenüber einer sequentiellen Implementierung.

Contents

1. Introduction **1**
1.1. Main contributions 2
1.2. Outline 5

2. Foundations **7**
2.1. Complexity of algorithms 7
2.2. Basic geometric properties of algebraic curves and surfaces 8
2.3. Resultants, subresultants and the GCD of polynomials 10
  2.3.1. Size of remainders and subresultants 17
  2.3.2. Specialization properties 20
  2.3.3. Polynomial content 21
  2.3.4. Chain rule 22
  2.3.5. Resultants and the extension theorem 22
2.4. Polynomial real root counting 23
  2.4.1. Sturm chains and signed subresultant sequences 24
  2.4.2. Descartes' rule of signs 28
2.5. Numerical filtering 30
  2.5.1. Interval arithmetic 31
  2.5.2. Choice of precision 31
2.6. Summary of algorithm complexities 32

3. Exact rasterization of real algebraic plane curves **35**
3.1. Motivation 35
3.2. Related work 37
3.3. Algorithm outline 39
3.4. Precomputations 40
  3.4.1. Isolation of critical points 40
  3.4.2. Viewport specific precomputations 41
  3.4.3. Intersections with the viewport boundary 42
3.5. Rasterization 42
  3.5.1. Non-critical slices and boxes 43
  3.5.2. Rows with critical points 48
3.6. Numerical filtering 49
  3.6.1. Numerical computation of signed subresultant quotient sequences 50
  3.6.2. Numerical real root counting 50
3.7. Analysis of the asymptotic complexity 53
  3.7.1. Analysis of the preprocessing stage 53
  3.7.2. Adjustment of the viewport 54
  3.7.3. Analysis of the rasterization stage 56

    3.7.4. The improved trivial algorithm 57
    3.7.5. Summary and comparison 57
  3.8. Implementation 58
  3.9. Benchmarks of challenging curves 59
    3.9.1. Preprocessing 59
    3.9.2. Adjustment of the viewport 64
    3.9.3. Rasterization 65
  3.10. Conclusions and future work 69

4. Robust graphical display of real algebraic space curves **71**
  4.1. Motivation 71
  4.2. Related work 71
  4.3. Algorithm outline 72
  4.4. Squarefree factorization of the projected curve 74
  4.5. Non-constant leading coefficients 76
    4.5.1. The content 76
    4.5.2. Asymptotes 77
    4.5.3. Curves at infinity of different multiplicity 79
  4.6. Simplification of the lifting surfaces 81
    4.6.1. Eliminating the content 81
    4.6.2. Rational parametrization 82
    4.6.3. Computing the squarefree part 84
    4.6.4. Syzygies 85
  4.7. Numerical approximation 87
    4.7.1. Lines in the direction of projection 87
    4.7.2. Approximating the projections and lifting 88
  4.8. A note on asymptotic complexity 89
  4.9. Implementation 91
  4.10. Results and discussion 91
  4.11. Conclusion and future work 92

5. Interactive visualization of blowups of the plane **97**
  5.1. Motivation 97
    5.1.1. The definition of blowups 98
    5.1.2. The embedding into a torus 99
    5.1.3. On the drawbacks of the parametrization 100
    5.1.4. Implicitization based real-time visualization 101
  5.2. Derivation of the implicit form 102
    5.2.1. Algebraic preparations 102
    5.2.2. Elimination 103
    5.2.3. Resultants 104
    5.2.4. Degree bounds 106
    5.2.5. A simplified geometric construction 106
    5.2.6. Deformations 108
    5.2.7. The degeneration 109
  5.3. Visualization of the implicit form 109
    5.3.1. Clipping $V(\mathfrak{F})$ to $\mathbb{T}_{f,g}$ 110

5.3.2. Texturing the surface 113
5.3.3. Highlighting the exceptional fibers 114
5.4. Implementation and results 114
5.4.1. RealSurf as a tool for visualizing toroidal blowups 114
5.4.2. Discussion of examples 115
5.5. Conclusion and perspectives 117

6. Parallel computation of resultants on graphics processing units 119
6.1. Motivation 119
6.2. Related work 119
6.3. Algorithm outline 120
6.4. Divide phase: applying homomorphisms 120
6.4.1. Reduction modulo a prime number 121
6.4.2. Evaluating polynomials 122
6.5. Conquer Phase: Calculating resultants of univariate polynomials 123
6.5.1. Parallelization on global memory 123
6.5.2. Parallelization on shared memory 124
6.6. Combine phase: reconstructing the integer resultant 124
6.6.1. Polynomial interpolation 124
6.6.2. Lifting from prime fields to integers 126
6.7. Experimental results 127
6.8. Conclusion and future work 128

A. Description and rasterizations of challenging real algebraic plane curves 131
A.1. Solitary points 131
A.2. Smooth curves 132
A.3. High tangencies of halfbranches 134
A.4. Many isolated singularities 135
A.5. Singularities with more than four halfbranches 137
A.6. Discriminants 138
A.7. Curves with several difficulties 141
A.8. Random polynomials 142

B. Absolute running times of plane curve renderings 145

C. Defining polynomials of example curves 151
C.1. The real algebraic plane curve “Bundle” 151
C.2. Real algebraic space curves 151

D. Notations and abbreviations 153

Bibliography 155

List of figures 165

List of tables 167

List of algorithms 169

1. Introduction

Visualizations are ubiquitous in science today and they always were. In past centuries, hand drawings and plaster models have been used to teach students and to guide research. The development of the field of computer graphics opened completely new possibilities, especially in the presentation of abstract mathematical models. Usually, no direct counterpart in the physical world exists. Even if one is able to build a physical model, it often provides only a rather restricted insight.

Computer-based visualizations solve many problems of classic approaches, but, as it is often the case, new challenges arise at the same time. While physical models are build on the basis of known facts, users of computer-based visualization programs often have only little knowledge of the properties of the object to visualize. This raises the question of exactness of a visualization. Clearly, the notion of exactness highly depends on the purpose of the visualization. Consider for example space curves: A mathematician might be interested in the topology of the curve. Displaying a graph which is topologically equivalent to the curve suffices in this case. In contrast, an engineer would probably require an accurate geometric image of the curve if it represents the boundary of a workpiece which is the difference of two surfaces. Such constructions frequently arise in computer aided geometric design (CAGD), where detailed geometric information is crucial for further processing.

Designing robust visualization algorithms is often a challenge. From a theoretical point of view, the basic operations are usually considered to be exact, but this is not the case on most processor architectures in practice. Even the most simple geometric primitives can introduce severe errors in subsequent computations. Consider for example the orientation of three points in the plane. If these points are close to being collinear, numerical computations become unstable. A solution is to switch to arbitrary precision arithmetic for integral and rational numbers. However, this is not sufficient if one moves over from linear to non-linear geometry. Handling points on circular arcs already requires square roots, and computations with general algebraic objects introduce algebraic numbers of higher degree.

Dealing with non-linear objects is the concern of computer algebra. Instead of storing explicit sets of points, geometric objects are represented and manipulated by algebraic expressions. This usually ensures correctness at the expense of complicated implementations, high algorithmic complexity and time-consuming computations. But correct results do not always require the exact representation of all involved objects. In geometry, many computations can be reduced to the determination of the signs of algebraic expressions evaluated at certain points. The exact values of these expressions, like in the case of the orientation of three points, is of minor interest. This is called the exact geometric computation (EGC) paradigm [YD95]: Use whatever method is appropriate to speed up the computation in simple cases but never output a wrong sign, even in degenerate situations.

Simple cases are often called generic and many algorithms try to transform non-generic problems into generic ones in order to simplify subsequent computations. Unfortunately, such transformations can increase the computation time significantly. Although generic cases are the most common ones, degenerate cases prevent the correctness of the result. Therefore, it is inevitable in many applications to detect and to solve non-generic cases using more sophisticated methods from computational geometry and computer algebra.

The geometric objects considered in this thesis are real algebraic curves and certain types of real algebraic surfaces defined implicitly as the real zero set of polynomials in two respectively three variables. The restriction to the *real* zero set makes it especially hard to work with them since many statements from algebraic geometry about the connection between a zero set (variety) and its algebraic representation (ideal) are only valid over an algebraically closed field but not over real closed fields. A famous example is Hilbert's Nullstellensatz. One of its consequences is that every proper ideal over the complex numbers has a nonempty zero set. The polynomial $x^2 + 1$ has a complex root $\sqrt{-1}$ but no real root although it defines a proper ideal. In the multivariate case, various additional problems related to the projection of varieties arise and it is often hard to cope with them efficiently.

When it comes to the visualization of algebraic varieties, one has to decide what exactness means and how much exactness is really necessary. First of all, the discrete raster images displayed on a computer screen only provide a finite resolution. In contrast, algebraic varieties are defined by the zero sets of continuous functions. They can have very small geometric features which are likely to be overlooked during the rasterization process. Determining their existence and approximating their position with the help of computer algebra and numerical methods is often very time consuming. Therefore, one has to find a compromise between the speed of the visualization and the correctness of the result. This often relates to the complexity of the input. It is easier to deal with real algebraic plane curves than with algebraic space curves and surfaces.

In this work, three approaches with different trade-offs between exactness and speed are presented. First, real algebraic plane curves will be rasterized correctly up to pixel level at the expense of performing a relatively costly precomputation in some situations. Secondly, the visualization algorithm for real algebraic space curves approximates all one-dimensional segments of the curve but intentionally ignores isolated points in order to speed up the computation. Finally, the rendering of a real algebraic surface may show numerical errors in exchange for an interactive visualization in real time. The latter is achieved by tailoring the implementation directly to an architecture which is specifically designed for the data-parallel processing of numerical algorithms: the graphics processing unit (GPU). Since the basic method has been presented in previous works of the author, the focus in this work is on the rendering of a specific type of algebraic surface which is particularly difficult to visualize. Recent developments also opened the possibilities for performing exact symbolic computations on the GPU. This allows to provide a prototypical parallel implementation of a specific symbolic method at the end of this thesis, which may be of great use for the analysis and rendering of algebraic curves in the future.

1.1. Main contributions

This section will be used to summarize the main contributions of this work. See also Figure 1.1 for some of the images that the newly developed algorithms are able to create.

Exact rasterization of real algebraic plane curves

Most previous approaches for visualizing algebraic plane curves rely entirely on numerical methods. This often leads to wrong results. Computing certified solutions of bivariate

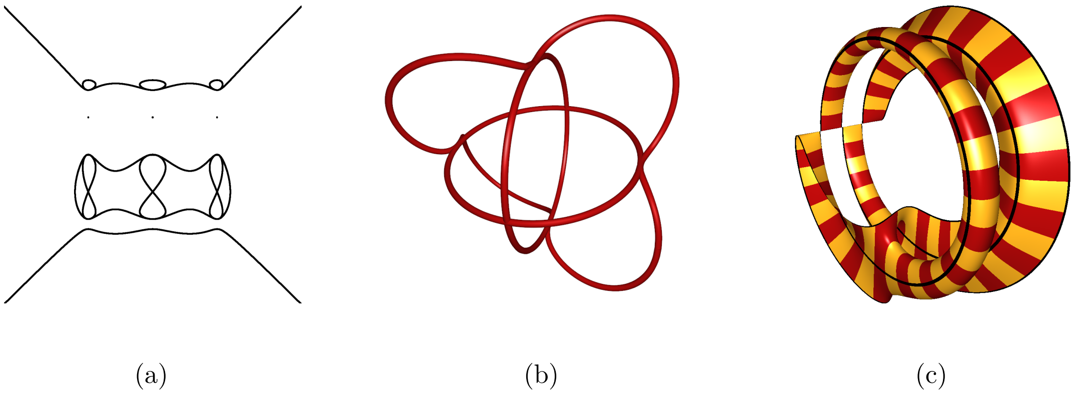

(a) (b) (c)

Figure 1.1.: Result of the visualization algorithms presented in this work: (a) An exact rasterization of a real algebraic plane curve. Note the three isolated points. (b) A rendering of a real algebraic space curve. (c) A visualization of a blowup of the plane in 3 double points embedded into a torus.

polynomial systems has become feasible for moderate problem instances in recent years due to various advances in the field. This applies to theoretical questions as well as to efficient and widely available implementations.

For this reason, a new rasterization algorithm for real algebraic plane curves is proposed which exploits global information of the curve in order to produce exact results in all cases. In the new method, the so-called critical points of the curve are computed and used to decompose the image plane into rectangular regions which contain only monotone curve segments. The monotony and the knowledge of start and end points of the curve segments are exploited in the rasterization in order to develop an efficient curve tracing method which never subdivides the image plane below pixel level. It relies on a very simple test for a sign change of the defining polynomial in simple cases and on a more sophisticated counting scheme for real roots in case of a bad separation of the segments. The subresultant based real root counting benefits from the symbolic precomputation used to determine the critical points. This allows to apply numerical filtering techniques for increased efficiency, which would not have been possible without having the global information at hand. For the first time, the asymptotic binary complexity of curve rasterization algorithms is analyzed. The comparison of the new method with a previously known one that also yields exact results confirms the efficiency of the new one from a theoretical point of view. The new algorithm is implemented and compared with two other exact methods by means of a large number of curves which are considered to be challenging instances. Although the new algorithm requires additional precomputation in order to gather more global information than others, the actual curve tracing is shown to be faster in general. This often allows to interactively explore the curve geometry as soon as the precomputation finishes.

A preliminary version of the new approach appeared in [Stu11]. It still lacks parts of the implementation as well as the complexity analysis presented here.

Robust graphical display of real algebraic space curves

The visualization of real algebraic space curves given as the intersection of two algebraic surfaces has not attracted much attention so far. Known sampling techniques that are applicable for rendering algebraic plane curves and algebraic surfaces are not easily extensible for visualizing space curves since their codimension is two instead of one. In addition, many tangent based curve tracing methods only work in special cases because the tangent planes of the surfaces defining the curve might be coplanar.

Therefore, most algorithms for analyzing and visualizing algebraic space curves are based on symbolic computations. In this work, a projection and lifting based method for computing the topology of implicit algebraic space curves is adapted and turned into a rendering algorithm. The space curve is projected onto the plane using resultants and decomposed into several simpler curves with the help of subresultants. The previous method only works for curves in a rather limiting generic position, i.e. the coordinate system is sheared if the defining surfaces have asymptotes in the direction of projection or if two one-dimensional segments of the curve have the same projection.

By dropping the requirement of correct topology, a new notion of genericity is introduced. The new condition is easy to check and to ensure. The problem of asymptotes is solved by a simple geometric transformation along the direction of projection. The geometric transformation to ensure genericity is only needed during the analysis of the curve while the lifting procedure is carried out in the original coordinate system. In addition, a new algorithm is proposed for lifting curve segments that coincide in the projection. It is based on a symbolic computation combined with a certified real root counting method for polynomials with approximate coefficients. Provided that the resolution for rendering the projected curve has been chosen appropriately, the result of the computation is an approximation of all one-dimensional components of the space curve by line segments. A (partial) analysis of the new approach suggests that its complexity is comparable to its predecessor. However, the experimental results confirm a significant gain of performance when the new method is used.

Interactive visualization of blowups of the plane

A rather important geometric transformation in algebraic geometry is the so-called blowup. Its main application is the resolution of singularities of algebraic varieties. The blowup of the plane in a finite set of points X yields a surface with the so-called exceptional fiber over X. A plane curve that is singular in X can become a smooth curve on the blowup surface. Since these surfaces can not be visualized directly, one has to find an appropriate embedding into affine space. Such an embedding is given by what we will call a toroidal blowup: The real projective lines over a disc of the plane are mapped to circles within a torus. The embedding is given by a parametric description of the surface, which is usually considered to be easy to visualize. Unfortunately, the parametrization of the toroidal blowup is undefined over the points in X, which leads to severe numerical instabilities in its neighborhood. Using triangulation methods, such issues are difficult to resolve and known algorithms are quite slow. In addition, the generated mesh is static, which leaves no room for interactive deformations of the blowup.

In order to solve this problem, a resultant based implicitization technique is proposed. In general, the implicit equation has a much higher degree then the parametric one making its visualization computationally intense and even more prone to errors. It is shown that this is not the case for toroidal blowups. The degree increases only by a small constant factor, and the implicit surface is well defined even at the points which correspond to the exceptional fiber. Furthermore, a linear time algorithm is described to compute the implicit equation. The final rendering mechanism is implemented via raytracing on graphics hardware so that many examples of blowups and their deformations can be visualized interactively and in real time.

The implementation has been carried out on the basis of prior work of the author published in [Stu07; Stu09; SS10; SS11] and the overall result will appear in [SS13].

Parallel computation of resultants on graphics processing units

Resultants are a major tool when working with algebraic varieties as indicated above but their computation can be costly. Porting algorithms to new platforms like graphics hardware has become a new trend recently. Their enormous capacity for processing parallel tasks also attracts researchers from the computer algebra community.

In this work, one of the first implementations of a symbolic operation on graphics processing units is presented: the parallel computation of bivariate polynomial resultants over the integers. The algorithm is based on a well-known divide-conquer-combine strategy, that is adapted to harness much of the parallelism available on graphics hardware. First, the bivariate integer polynomials are mapped homomorphically to univariate polynomials over finite fields. Then, univariate resultants of these polynomials are computed and afterwards combined into the bivariate resultant by interpolation techniques. All steps of the algorithm are parallelized. It is also shown that the notion of unlucky homomorphisms is unnecessary in the context of resultants and how this greatly simplifies the algorithm. A complete parallel implementation is provided, which shows huge speedups over its sequential counterpart.

Most of the above results have been published in [SS12].

1.2. Outline

This thesis is structured as follows. Chapter 2 is dedicated to the algebraic, geometric and algorithmic foundations that serve as a basis for the upcoming investigations. Most concepts will also be examined with respect to their computational complexity. The notation introduced there can also be found in summarized form in Appendix D.

In Chapter 3, we will look into the exact rasterization of real algebraic plane curves. First, the notion of exactness of a rasterization is introduced and illustrated with some examples. After providing an overview of the algorithm, it is explained which global information about the curve is useful in a rasterization context and how it can be computed. Next, the actual rasterization process is described. Since the algorithm is designed to yield exact results, it is shown how numerical filtering techniques can be used to speed up the rendering in many cases without violating the exactness constraint. Then, an analysis of the asymptotic complexity is provided and backed up with benchmark results. The results are discussed and the chapter concludes with a perspective on future work.

In Chapter 4, we will consider real algebraic space curves instead of plane curves. The chapter is arranged similarly to Chapter 3 but this time we focus on the symbolic part of the algorithm since the numeric part contains only few new results.

In Chapter 5, we introduce the notion of blowups of the plane and then investigate how they can be visualized interactively with the help of an appropriate embedding into affine space. The derivation of the implicit form of this embedding and of various of its properties is the aim of Section 5.2. Next, important visualization issues like clipping and texturing are investigated.

In the subsequent part of this chapter, an implementation of the visualization algorithm is presented. Several visualizations of blowups are provided and discussed. In the last section, some approaches for future research are given.

In Chapter 6, we explore how to harness the parallel processing power of current graphics hardware for algorithms from computer algebra by means of bivariate resultants. The chapter is structured according to the different phases of the algorithm: First it is explained how the input can be divided so that several parallel threads can work on the problem. The parallel computation of univariate resultants is described in the second part. Then, we see how the partial results can be combined to yield the bivariate resultant. Finally, the experimental results for the implementation are presented.

Related work is discussed separately in Chapters 3 to 6 since the overlap is quite small.

Appendix A contains many of the curve rasterizations that have been produced for the benchmarks in Chapter 3. In addition, the visualization challenges for plane curves are explained in more detail. Plots of the absolute running time of the exact plane curve rasterization algorithm are supplied in Appendix B. Furthermore, equations of curves that have been used in some of the examples in the thesis are provided in Appendix C.

2. Foundations

This chapter briefly explains common concepts, notations, algorithms and their complexities that are used at several places later on. We assume the familiarity of the reader with basic mathematical concepts from algebra, geometry and the analysis of the complexity of algorithms. If not stated otherwise, we denote (commutative) rings with R, integral domains with D, fields with K, ideals with I or J and varieties with V. In addition, we use the notation summarized in Appendix D.

The main source for this chapter is standard literature like [CLO07; GCL92; BPR03]. In most cases, we will only sketch proofs or omit them completely if there exists an appropriate reference. However, there are some minor extensions and original results by the author (namely, Lemmata 2.2.5, 2.4.11, 2.3.35 and 2.3.36 and Propositions 2.4.9 and 2.4.10).

2.1. Complexity of algorithms

In order to analyze the complexity of an algorithm, we have to define some kind of complexity model: Given an algebraic structure $(A, (f_i))$, we count the number of applications of the operations f_i that are necessary to obtain a certain result and the number of elements of A we have to store during the computation. Since this disregards the size of the involved elements of A, it does not give a precise estimation of the complexity. Depending on the algebraic structure, different measures of the size of the objects have to be considered. For integral and rational numbers we assume a binary representation and bound the number of bits necessary to store a number while for polynomials it is convenient to additionally consider their degree. If the bitsize of the involved numbers is taken into account, the complexity is usually referred to as *bit complexity* or *binary complexity*.

Notation 2.1.1 (bitsize). *In order to bound the bitsize, we will use* $\operatorname{bit}(a) = 1 + \lceil \log_2 |a| \rceil \in \mathcal{O}(\log |a|)$ *resp.* $\operatorname{bit}(\frac{a}{b}) = \operatorname{bit}(a) + \operatorname{bit}(b)$ *for* $a, b \in \mathbb{Z} \setminus \{0\}$ *and* $\operatorname{bit}(0) = 1$.

Notation 2.1.2 (degree). *Given a polynomial* $A = \sum_{i=0}^{m} a_i x^i \in R[x]$*, the degree is given by* $\deg(A) = m$*. For multivariate polynomials* $A \in R[x_1, \ldots, x_k]$*, we denote the total degree of* A *using* $\deg_{total}(A)$ *or just* $\deg(A)$*. To consider only a subset of the variables, we use* $\deg_{x_{i_1} \cdots x_{i_l}}(A)$ *to denote the total degree of* A *when viewed in the ring* $R'[x_{i_1}, \ldots, x_{i_l}]$ *where* R' *is the polynomial ring over* R *in the variables* $\{x_1, \ldots, x_k\} \setminus \{x_{i_1}, \ldots, x_{i_l}\}$*. If it is clear from the context, we may omit the subscript of* $\deg(\cdot)$*.*

The degree(s) of a polynomial and the bitsize of its coefficients can be conveniently summarized by the magnitude.

Definition 2.1.3 (magnitude). *The* magnitude *of a polynomial* $A \in R[x_1, \ldots, x_k]$*,* $R \in \{\mathbb{Z}, \mathbb{Q}\}$ *is defined as* $(\tau, n_1, \ldots, n_k)$ *where* τ *is the bitsize of the coefficients of* A *and* $n_1 = \deg_{x_1}(A), \ldots, n_k = \deg_{x_k}(A))$*. We also use* (τ, n) *to denote the magnitude of a polynomial with respect to the total degree* n *of* A*. If we want to distinguish between the degree of the coefficient polynomials* $d = \deg_{x_1 \cdots x_{k-1}}(A)$ *and the degree* $n = \deg_{x_k}(A)$ *when* A *is viewed recursively with outermost variable* x_k*, we use* (τ, d, n)*.*

During the analysis of algorithms, we will usually derive asymptotic bounds on their complexity using the well known Landau symbols $\mathcal{O}, \Omega$ and Θ (see e.g. [Knu97a, Section 1.2.11]). In addition, it is often convenient to disregard (poly-)logarithmic factors and use $f(n) \in \tilde{\mathcal{O}}(g(n))$ as a shorthand for $f(n) \in \mathcal{O}(g(n) \log^k g(n))$ for some $k \in \mathbb{N}$.

Good complexity bounds are frequently derived by multiplying the number of arithmetic operations in the base ring with the maximal cost of one arithmetic operation. We abbreviate this as follows.

Notation 2.1.4. *We use $\mathcal{M}(\tau)$ to bound the number of bit operations necessary to perform arithmetic operations on integral number of bitsize τ.*

Remark. Note that $\mathcal{M}$ is essentially the cost of multiplication, since addition and subtraction are in $\mathcal{O}(\tau)$ and division with remainder is possible in $\mathcal{O}(\mathcal{M}(\tau))$ [BZ10, Theorem 1.4]. Clearly, $\mathcal{M}(\tau) \in \Omega(\tau)$ and $\mathcal{M}(\tau) \in \mathcal{O}(\tau^2)$ using the schoolbook method. These bounds are often sufficient albeit much more sophisticated algorithms exist. Currently, the best bound is $\mathcal{M}(\tau) \in \mathcal{O}(\tau \log \tau 2^{\log^* \tau}) \subset \tilde{\mathcal{O}}(\tau)$ [Für07]. See also Table 2.3 for an overview.

Many of our algorithmic tools are defined in terms of determinants of certain matrices. Although evaluating the determinant directly is typically not the most efficient implementation, it often allows to find precise bounds on the size of the results easily. Such an analysis is usually quite difficult for more sophisticated algorithms. We can utilize the following two results.

Lemma 2.1.5 (Hadamard's inequality). *Let $M \in \mathbb{Z}^{n \times n}$ with columns $c_1, \ldots, c_n$ and rows $r_1, \ldots, r_n$. Then $\det(M) \leq \prod_{i=1}^n ||c_i||_2$ and $\det(M) \leq \prod_{i=1}^n ||r_i||_2$.*

Proof. See [BPR03, Proposition 8.22]. ☐

Corollary 2.1.6 (bitsize of determinants). *Let $M \in \mathbb{Z}^{n \times n}$ with elements of bitsize at most τ. Then $\operatorname{bit}(\det(M)) \leq n(\tau + \operatorname{bit}(n)/2)$.*

Proof. Let us denote the elements of M with m_{ij}. It holds that

$$\left(\sum_{i=1}^{n} m_{ij}^2\right)^{\frac{1}{2}} \leq \left(\sum_{i=1}^{n} 2^{2\tau}\right)^{\frac{1}{2}} = \sqrt{n} 2^{\tau} \leq 2^{\tau + \operatorname{bit}(n)/2}. \tag{2.1}$$

Using Lemma 2.1.5 this yields $\det(M) \leq 2^{n(\tau + \operatorname{bit}(n)/2)}$. ☐

2.2. Basic geometric properties of algebraic curves and surfaces

A few related concepts need to be introduced to aid the visualization of real algebraic curves and surfaces. Some of them can be stated for varieties of arbitrary ideals of $K[x_1, \ldots, x_n]$ and we will do so in order to avoid the distinction between algebraic plane curves, space curves and surfaces, if possible. In this section, we assume the field K to be of characteristic zero.

The following terms help to decompose algebraic curves and surfaces into (geometrically) simpler objects.

Definition 2.2.1 ((algebraic) curve segment). *A* curve segment *is the image of a continuous function $\varphi : I \to K^n$, $n \geq 2$, where I is the open, half-open or closed unit interval. If C is an algebraic curve, then $\varphi : I \to C$ is called an* algebraic curve segment *of C.*

This naturally extends to *(algebraic) surface patches* by replacing $\varphi : I \to K^n$ resp. $\varphi : I \to C$ with $\varphi : I^2 \to K^n$, $n \geq 3$, resp. $\varphi : I^2 \to S$ for an algebraic surface S.

Some points on algebraic varieties can be considered special. Singular points are well known and also the more general concept of a critical point is useful.

Definition 2.2.2 (singular point). *Let $V = V(F_1, \ldots, F_k)$ with $F_1, \ldots, F_k \in K[x_1, \ldots, x_n]$. A point $p \in V$ is called* singular *if*

$$\operatorname{rank}(J(p)) < n - \dim(V), \tag{2.2}$$

where J is the Jacobi matrix $\left(\frac{\partial F_i}{\partial x_j}\right)_{i=1,\ldots,k;j=1\ldots,n}$. Otherwise p is called regular*.*

Clearly, it depends on the defining polynomials if a point of a variety is singular. Every point on the circle $V_1 = V(x^2 + y^2 - 1, z^2)$ is singular while all points of $V_2 = V(x^2 + y^2 - 1, z)$ are regular although $V_1 = V_2$. Note that the Jacobi matrix of a single polynomial F is nothing but the gradient vector ∇F. Hence, $V(F)$ is singular at $P \in V(F)$ if $\nabla F(P) = \vec{0}$.

Definition 2.2.3 (critical point). *A point $p \in V(F)$, $F \in K[x_1, \ldots, x_n]$, is called x_i-*critical *if $\frac{\partial F}{\partial x_j}(p) = 0$ for all $j \neq i$.*

The critical points of an algebraic variety carry crucial geometric information. In order to proof a related fact, we need the following theorem which tells us that an algebraic variety can be covered locally by the image of an explicit function.

Theorem 2.2.4 (implicit function theorem). *Let $F = (F_1, \ldots, F_m) : \mathbb{R}^{n+m} \to \mathbb{R}^m$ be a continuously differentiable function and let us denote the coordinates of $\mathbb{R}^{n+m}$ with $(X, Y) = (x_1, \ldots, x_n, y_1, \ldots, y_m)$. Let further $F(A, B) = C \in \mathbb{R}^m$ for a point $(A, B) \in \mathbb{R}^{n+m}$. If the Jacobi matrix*

$$\begin{pmatrix} \frac{\partial F_1}{\partial y_1}(A,B) & \frac{\partial F_1}{\partial y_2}(A,B) & \ldots & \frac{\partial F_1}{\partial y_m}(A,B) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_m}{\partial y_1}(A,B) & \frac{\partial F_m}{\partial y_2}(A,B) & \ldots & \frac{\partial F_m}{\partial y_m}(A,B) \end{pmatrix} \tag{2.3}$$

is invertible, then there exist open neighborhoods U of A and V of B and a unique continuously differentiable function $G : U \to V$ such that

$$\{(X, G(X)) : X \in U\} = \{(X, Y) \in U \times V : F(X, Y) = C\}. \tag{2.4}$$

Proof. See [Rud98, Theorem 9.28] or [KP02] for an in-depth discussion. ☐

Lemma 2.2.5. *Every bounded maximal* connected component C of $V_{\mathbb{R}}(F)$, $F \in \mathbb{R}[x_1, \ldots, x_n]$, has at least one x_i-critical point.*

Proof. W.l.o.g. we assume $i = n$. Let $p = (p_1, \ldots, p_n) \in C$ be a point with maximal x_n coordinate. Such a point always exists in C: If C contains only one point, then this point is p. Otherwise, there is a continuous path in C with limit p. The limit is in $V_{\mathbb{R}}(F)$ since the roots of a polynomial are continuous functions of its coefficients, which continuously change with the location on the path. Now, $p \notin C$ would contradict the maximality of C.

If p is a singular point of C, it is also critical. Otherwise, $V_{\mathbb{R}}(F)$ is locally the graph of a continuously differentiable function $x_n = G(x_1, \ldots, x_{n-1})$ such that $F(x_1, \ldots, x_{n-1}, G(x_1, \ldots, x_{n-1})) = 0$ due to Theorem 2.2.4. The chain rule for partial derivatives yields

$$\frac{\partial F}{\partial x_j}(p) = \frac{\partial F}{\partial G}(p) \frac{\partial G}{\partial x_j}(p') \tag{2.5}$$

---

*C is maximal connected in the sense that there is no point $P \in V_{\mathbb{R}}(F) \setminus C$ such that $C \cup P$ is still connected.

for $j = 1, \dots, n-1$ and $p' = (p_1, \dots, p_{n-1})$. Since G has a local maximum at p', it follows that $\frac{\partial G}{\partial x_j}(p') = 0$ and therefore $\frac{\partial F}{\partial x_j}(p) = 0$ for $j = 1, \dots, n-1$ proving that p is x_n-critical. ☐

The knowledge of a critical point for each such component is of great value for many geometric algorithms, and visualization algorithms are no exception. Determining that no such component is missing in the visualization is actually one of the major problems in the field. Note that points on unbounded components of plane curves can easily be determined by sampling techniques.

The critical points can also be used to decompose plane curves into monotone segments.

Definition 2.2.6 (strictly monotone curve segment). *A segment* $S \subseteq V_{\mathbb{R}}(F)$, $F \in \mathbb{R}[x_1, x_2]$, *is called* strictly x_i-monotone *if* S *is free of* x_i*-critical points of* $V_{\mathbb{R}}(F)$.

The usual notion of increasing and decreasing monotony can also be applied by viewing the strictly x_i-monotone segment S locally as the graph of a function $x_1 = f(x_2)$ (x_1-monotone) resp. $x_2 = f(x_1)$ (x_2-monotone), which is always possible due to the implicit function theorem (Theorem 2.2.4). Obviously, strictly monotone segments do not have a local minimum or maximum w.r.t. the specified variable between their endpoints. The next corollary follows immediately.

Corollary 2.2.7. *The segments of* $V_{\mathbb{R}}(F)$, $F \in K[x_1, x_2]$, *are decomposed into strictly* x_i*-monotone segments by the* x_i*-critical points of* $V_{\mathbb{R}}(F)$.

Since we are usually working in affine space, we need to cover the concept of asymptotes. We focus on asymptotes with respect to the coordinate axis.

Lemma 2.2.8. *Let* $F \in \mathbb{R}[x_1, \dots, x_n]$. *If* $V_{\mathbb{R}}(F)$ *has an asymptote over* $p = (p_1, \dots, p_{n-1}) \in \mathbb{R}^n$, *then* $\operatorname{lcoeff}_x(F)(p_1, \dots, p_{n-1}) = 0$.

Proof. Assume that $\operatorname{lcoeff}_x(F)(p_1, \dots, p_{n-1}) \neq 0$. Then there is also a neighborhood U of p where $\operatorname{lcoeff}_x(F)$ does not vanish. We can now apply a root bound like e.g. Lemma 2.4.3 to all polynomials $F(Q, x)$, $q \in U$, in order to find a maximal value for this bound. However, this contradicts the fact that $V(F)$ diverges to infinity over U. ☐

Finally, we want to bound the number of critical points of an algebraic plane curve. This can be done by bounding the number of solution of the system $F = \frac{\partial F}{\partial x_i} = 0$.

Theorem 2.2.9 (weak theorem of Bézout). *Let* $F, G \in \mathbb{C}[x_1, x_2]$ *be polynomials of degrees* m *and* n *without common factor. Then* $\#V_{\mathbb{P}^2_{\mathbb{C}}}(A, B) \leq m \cdot n$.

Proof. See [Fis94, p. 25]. ☐

It is important to remark that the bound relates to solutions in $\mathbb{P}^2_{\mathbb{C}}$. Hence, common asymptotes are also covered since $\operatorname{lcoeff}_{x_i}(F)$ and $\operatorname{lcoeff}_{x_i}(\frac{\partial F}{\partial x_i})$ have the same zero set.

Note that we call the above theorem *weak* because Bézout's theorem is actually a more precise statement that incorporates the multiplicity of the solutions which we don't need.

2.3. Resultants, subresultants and the GCD of polynomials

In this section, we will introduce the reader to resultants, subresultants and their applications. They are valuable tools in many of the algorithms developed in this thesis. First, consider polynomials $A, B \in \mathbb{C}[x]$ with $\deg A = m$ and $\deg B = n$. When do A and B have a common root in $\mathbb{C}$? This question can be answered by stating the problem in terms of linear algebra using the following system of linear equations in $x^0, \ldots, x^{m+n-1}$:

$$
\begin{aligned}
0 = a_m x^{m+n-1} + a_{m-1} x^{m+n-2} + \cdots + a_0 x^{n-1} &= x^{n-1} A(x) \\
0 = a_m x^{m+n-2} + \cdots + a_1 x^{n-1} + a_0 x^{n-2} &= x^{n-2} A(x) \\
&\vdots \\
0 = a_m x^m + a_{m-1} x^{m-1} + \cdots + a_0 &= A(x) \\
0 = b_n x^{m+n-1} + b_{n-1} x^{m+n-2} + \cdots + b_0 x^{m-1} &= x^{m-1} B(x) \\
0 = b_m x^{m+n-2} + \cdots + b_1 x^{m-1} + b_0 x^{m-2} &= x^{m-2} B(x) \\
&\vdots \\
0 = b_n x^n + b_{n-1} x^{n-1} + \cdots + b_0 &= B(x)
\end{aligned}
\tag{2.6}
$$

Written in matrix form Equation (2.6) gives

$$
\underbrace{\begin{pmatrix}
a_m & a_{m-1} & \cdots & a_0 & & & \\
 & a_m & a_{m-1} & \cdots & a_0 & & \\
 & & \ddots & & & \ddots & \\
 & & & a_m & a_{m-1} & \cdots & a_0 \\
b_n & b_{n-1} & \cdots & b_0 & & & \\
 & b_n & b_{n-1} & \cdots & b_0 & & \\
 & & \ddots & & & \ddots & \\
 & & & b_n & b_{n-1} & \cdots & b_0
\end{pmatrix}}_{\mathrm{Syl}(A,B)}
\cdot
\begin{pmatrix} x^{m+n-1} \\ \vdots \\ x \\ 1 \end{pmatrix}
=
\begin{pmatrix} x^{n-1} A(x) \\ \vdots \\ A(x) \\ x^{m-1} B(x) \\ \vdots \\ B(x) \end{pmatrix}
= 0
\tag{2.7}
$$

Clearly, if x_0 is a common solution of $A(x) = B(x) = 0$, then $(1, x_0, \ldots, x_0^{m+n-1})$ is also a solution of Equation (2.7) and vice versa. We know from linear algebra that Equation (2.7) has a non-trivial solution if and only if $\det(\mathrm{Syl}(A, B)) = 0$. The matrix $\mathrm{Syl}(A, B)$ is usually referred to as the *Sylvester matrix* but this name is not undisputed (see [Akr93]). This leads to the definition of resultants.

Definition 2.3.1 (resultant). *The* resultant *of two polynomials $A, B \in R[x]$ is the determinant of their Sylvester matrix* $\mathrm{Syl}(A, B)$*, i.e.* $\mathrm{Res}(A, B) = \det(\mathrm{Syl}(A, B))$*. If R is another polynomial domain, we may write* $\mathrm{Res}_x(A, B) = \det(\mathrm{Syl}_x(A, B))$ *in order to identify the outermost variable if this is not clear from the context.*

Lemma 2.3.2 (Sylvester's criterion). *Let $A, B \in D[x]$, D a unique factorization domain (UFD). Then A and B have a non-trivial common factor if and only if* $\mathrm{Res}(A, B) = 0$.

Proof. See [GCL92, p. 288]. ☐

Note that the formulation using the Sylvester matrix is not the only one for resultants. There are several others, e.g. Bézout matrices. See [MS99] for a more general discussion.

For polynomials A, B over a ring R with roots in the algebraically closed field $C \supset R$, the resultant can also be stated in terms of a symmetric function of the roots of A and B.

Theorem 2.3.3. *Let* $A = a_m \prod_{i=1}^{m}(x - x_i), B = b_n \prod_{j=1}^{n}(x - y_j)$, *then*

$$\operatorname{Res}(A,B) = a_m^n b_n^m \prod_{i=1}^{m}\prod_{j=1}^{n}(x_i - y_j). \tag{2.8}$$

Furthermore,

$$\operatorname{Res}(A,B) = a_m^n \prod_{i=1}^{m} B(x_i) = b_n^m \prod_{j=1}^{n} A(y_i). \tag{2.9}$$

Proof. See [BPR03, Theorem 4.26 and Lemma 4.27]. □

Using the definition $\operatorname{Res}(A,B) = \det(\operatorname{Syl}(A,B))$, the remarkable property that $\operatorname{Res}(A,B) \in R$ is obvious. This is not clear from the above theorem since it suggests $\operatorname{Res}(A,B) \in C$. Nevertheless, Theorem 2.3.3 frequently appears in proofs related to resultants.

The resultant has many useful properties in addition to the above facts. Since most of them can be stated similarly for the so-called subresultants, we will first generalize resultants to subresultants to avoid repetitions. In the literature, subresultants have been approached from various sides. The term *signed subresultant* is used in [BPR03], while *subresultants* are defined in [GCL92] and [Yap00]. Furthermore, [GRL98] introduces *Sturm-Habicht polynomials* and in [LR01] *Sylvester-Habicht polynomials* are described. Besides some minor differences regarding some special cases, all definitions are equal up to sign. In this thesis, the signs are only relevant in the context of real root counting of univariate polynomials (see Section 2.4.1). Therefore, we will follow the unified approach of signed subresultants presented in [BPR03].

Definition 2.3.4 (signed subresultant). *For* $0 \leq j \leq n$ *the* j-th signed subresultant *of* $A = \sum_{i=0}^{m} a_i x^i \in R[x]$ *and* $B = \sum_{i=0}^{n} b_i x^i \in R[x]$, $n < m$, *is the polynomial*

$$\operatorname{SRes}_j(A,B) = \det(S_j(A,B)), \tag{2.10}$$

where

$$S_j(A,B) = \left(\underbrace{\begin{matrix} a_m & & \\ a_{m-1} & \ddots & \\ \vdots & \ddots & a_m \\ \vdots & & a_{m-1} \\ \vdots & & \vdots \\ a_{2j-n+2} & \cdots & a_{j+1} \\ x^{n-j-1}A & \cdots & x^0 A \end{matrix}}_{n-j \text{ columns}} \quad \underbrace{\begin{matrix} & & b_n \\ & \iddots & b_{n-1} \\ b_n & \iddots & \vdots \\ b_{n-1} & & \vdots \\ \vdots & & \vdots \\ b_{j+1} & \cdots & b_{2j-m+2} \\ x^0 B & \cdots & x^{m-j-1}B \end{matrix}}_{m-j \text{ columns}} \right) \tag{2.11}$$

is a square matrix of size $(m + n - 2j) \times (m + n - 2j)$. *All coefficients with negative subscripts and the entries above* a_m *and* b_n *are zero. For* $n < j \leq m$ *we define*

$$\operatorname{SRes}_m(A,B) = \operatorname{sign}(a_m^{m-n-1})A, \tag{2.12}$$

$$\operatorname{SRes}_{m-1}(A,B) = \operatorname{sign}(a_m^{m-n+1})B, \tag{2.13}$$

$$\operatorname{SRes}_j(A,B) = 0,\ n < j < m - 1, \tag{2.14}$$

$$\operatorname{SRes}_{-1}(A,B) = 0. \tag{2.15}$$

Note that $\mathrm{SRes}_n(A,B) = \varepsilon_{m-n} b_m^{m-n-1} B$ *where* $\varepsilon_m = (-1)^{\frac{m(m-1)}{2}}$. *In case of* $m = n$, *we set* $\mathrm{SRes}_m(A,B) = B$ *and* $\mathrm{SRes}_j(A,B) = \det(S_j(A,B))$ *for* $0 \le j < n$.

An alternative representation of signed subresultants, which allows to deduce the bound $\deg(\mathrm{SRes}_j) \le j$, is the following.

Corollary 2.3.5. *Let* $A, B \in R[x]$ *be polynomials of degree* m *and* n. *Then for* $0 \le j \le n$

$$\mathrm{SRes}_j(A,B) = \sum_{i=0}^{j} \det(S_{ij}(A,B)) x^i \tag{2.16}$$

with

$$S_{ij}(A,B) = \begin{pmatrix} a_m & & & & & b_n \\ a_{m-1} & \ddots & & & \iddots & b_{n-1} \\ \vdots & \ddots & a_m & b_n & \iddots & \vdots \\ \vdots & & a_{m-1} & b_{n-1} & & \vdots \\ \vdots & & \vdots & \vdots & & \vdots \\ a_{2j-n+2} & \cdots & a_{j+1} & b_{j+1} & \cdots & b_{2j-m+2} \\ a_{i+j-n+1} & \cdots & a_i & b_i & \cdots & b_{i+j-m+1} \end{pmatrix} \in R^{(m+n-2j)\times(m+n-2j)}. \tag{2.17}$$

Proof. Write the last row of the matrix $S_j(A,B)$ as $\sum_{i=0}^{n} v_{ij} x^i$ using $v_{ij} = (a_{i+j-n+1}, \dots, a_i, b_i, \dots, b_{i+j-m+1})$. Since the determinant is a linear function of the last (in fact, of every) row, it follows that $\det(S_j(A,B)) = \sum_{i=0}^{n} x^i \det(S_{ij}(A,B))$ where S_{ij} is equal to S_j with its last row replaced by v_{ij}. In order to show that $\sum_{i=0}^{n} x^i \det(S_{ij}(A,B)) = \sum_{i=0}^{j} x^i \det(S_{ij}(A,B))$, note that the last row of $S_{ij}(A,B)$ is equal to row $(m+n-j-i-1)$ for $i \ge j+1$ (the case $i = j+1$ can easily be visualized in Equation (2.17)). Therefore, $\det(S_{ij}) = 0$ for $i \ge j+1$. ☐

Since it is not completely obvious from the two definitions, we relate resultants to subresultants in the following corollary.

Corollary 2.3.6. $\mathrm{SRes}_0(A,B) = (-1)^{\frac{(m-1)m}{2}} \mathrm{Res}(A,B)$ *is the resultant of* A *and* B *up to sign.*

Proof. First, note that $\mathrm{SRes}_0(A,B) = \det(S_{00}(A,B))$. The matrices $S_{00}(A,B)^T$ and $\mathrm{Syl}(A,B)$ are equal up the reversed order of the rows in the block with coefficients of B. To reverse the order of these m rows, we successively move each of the rows to its desired position by swapping rows. This requires $\sum_{i=1}^{m}(i-1) = m(m-1)/2$ swapping operations, which explains the additional factor $(-1)^{\frac{(m-1)m}{2}}$. ☐

In addition to the signed subresultant, their cofactors SResU and SResV will also prove to be very useful.

Definition 2.3.7 (signed subresultant cofactor). *For* $0 \le j \le n$ *the* j-th signed subresultant cofactors *of* $A = \sum_{i=0}^{m} a_i x^i \in R[x]$ *and* $B = \sum_{i=0}^{n} b_i x^i \in R[x]$, $n < m$, *are the polynomials*

$$\mathrm{SResU}_j(A,B) = \det(M_j(A,B)), \tag{2.18}$$

$$\mathrm{SResV}_j(A,B) = \det(N_j(A,B)), \tag{2.19}$$

where the matrices $M_j(A,B)$ *resp.* $N_j(A,B)$ *are obtained from* $S_j(A,B)$ *by replacing the last row with* $(x^{n-j-1}, \dots, x^0, 0, \dots, 0)$ *resp.* $(0, \dots, 0, x^0, \dots, x^{m-j-1})$.

Corollary 2.3.8. *For $0 \leq j \leq n$ it holds that*

$$\mathrm{SRes}_j(A,B) = \mathrm{SResU}_j(A,B)A + \mathrm{SResV}_j(A,B)B \tag{2.20}$$

with $\deg(\mathrm{SResU}_j(A,B)) \leq n-j$ *and* $\deg(\mathrm{SResV}_j(A,B)) \leq m-j$. *If* $\mathrm{SRes}_j(A,B) \not\equiv 0$, *the representation in Equation* (2.20) *is unique, i.e. if* $\mathrm{SRes}_j(A,B) = UA+VB$ *with* $\deg(U) \leq n-j$ *and* $\deg(V) \leq m-j$, *then* $U = \mathrm{SResU}_j(A,B)$ *and* $V = \mathrm{SResV}_j(A,B)$.

Proof. Expanding the determinant $\det(S_j(A,B))$ along the last row yields the result up to the uniqueness of $\mathrm{SResU}_j(A,B)$ and $\mathrm{SResV}_j(A,B)$. For a proof of this fact we refer to [BPR03, Proposition 8.58]. ☐

Definition 2.3.9. *We will call* SRes_j regular *if* $\deg(\mathrm{SRes}_j) = j$, *and* defective *of degree k if* $\deg(\mathrm{SRes}_j) = k < j$. *A signed subresultant sequence* $\mathcal{S} = [\mathrm{SRes}_0(A,B), \ldots, \mathrm{SRes}_n(A,B)]$ *containing at least one defective subresultant is called defective or non-regular. Otherwise it is called regular.*

Notation 2.3.10. *We denote by* $\mathrm{sres}_j(A,B) = \mathrm{coeff}_j(\mathrm{SRes}_j(A,B))$ *the j-th* principal signed subresultant coefficient *and by* $\overline{\mathrm{sres}}_j(A,B) = \mathrm{lcoeff}(\mathrm{SRes}_j(A,B))$ *the leading coefficient of the signed subresultant. Note that* $\mathrm{sres}_j(A,B) = \overline{\mathrm{sres}}_j(A,B)$ *in the regular case.*

Notation 2.3.11. *We will use the abbreviation* $\mathrm{SRes}_j(A) = \mathrm{SRes}_j(A,A')$ *as well as* $\overline{\mathrm{sres}}_j(A) = \overline{\mathrm{sres}}_j(A,A')$ *and* $\mathrm{sres}_j(A) = \mathrm{sres}_j(A,A')$. *If the polynomials used to create the sequence are clear from the context, we will omit them altogether.*

The definition of the subresultants in terms of determinants has several useful properties as we will see later. However, it is better to use the following theorem to compute the signed subresultants.

Theorem 2.3.12 (structure theorem for signed subresultants). *Let $A, B \in D[x]$ and $0 \leq j < i \leq n$. Suppose* SRes_{i-1} *and* SRes_{j-1} *are non-zero and* $\deg(\mathrm{SRes}_{i-1}) = j$ *and* $\deg(\mathrm{SRes}_{j-1}) = k$, *then*

$$\mathrm{SRes}_{k-1} = -\frac{\mathrm{rem}(\mathrm{sres}_k\,\overline{\mathrm{sres}}_{j-1}\,\mathrm{SRes}_{i-1}, \mathrm{SRes}_{j-1})}{\mathrm{sres}_j\,\overline{\mathrm{sres}}_{i-1}} \tag{2.21}$$

$$= -\frac{\mathrm{sres}_k\,\overline{\mathrm{sres}}_{j-1}\,\mathrm{SRes}_{i-1} - \mathrm{SResQ}_{k-1}\,\mathrm{SRes}_{j-1}}{\mathrm{sres}_j\,\overline{\mathrm{sres}}_{i-1}}, \tag{2.22}$$

where the quotient SResQ_{k-1} *is in $D[x]$ with* $\deg(\mathrm{SResQ}_{k-1}) = j-k$. *Additionally, if $j \leq n$ and $k < j-1$, the polynomials* SRes_k *and* SRes_{j-1} *are proportional with*

$$\overline{\mathrm{sres}}_{j-1}\,\mathrm{SRes}_k = \mathrm{sres}_k\,\mathrm{SRes}_{j-1} \tag{2.23}$$

$$\mathrm{sres}_k = (-1)^{\frac{j(j-1)}{2}} \frac{\overline{\mathrm{sres}}_{j-1}^{\,j-k}}{\mathrm{sres}_j^{\,j-k-1}} \tag{2.24}$$

and the polynomials $\mathrm{SRes}_{j-2}, \ldots, \mathrm{SRes}_{k+1}$ *are identically zero.*

Proof. See [BPR03, Theorem 8.53]. ☐

Corollary 2.3.13. *If* $\deg(\mathrm{SRes}_j) = j$, $0 < j \leq n$ *and* $\deg(\mathrm{SRes}_{j-1}) = k$, $k \leq j-1$, *then*

$$\mathrm{SRes}_{k-1} = -\frac{\mathrm{rem}(\mathrm{sres}_k\,\overline{\mathrm{sres}}_{j-1}\,\mathrm{SRes}_j, \mathrm{SRes}_{j-1})}{\mathrm{sres}_j^2} \tag{2.25}$$

$$= -\frac{\mathrm{sres}_k\,\overline{\mathrm{sres}}_{j-1}\,\mathrm{SRes}_j - \mathrm{SResQ}_{k-1}\,\mathrm{SRes}_{j-1}}{\mathrm{sres}_j^2}. \tag{2.26}$$

Proof. Immediately, using $i = j+1$ in Theorem 2.3.12. □

In other words, a computation of the sequence of signed subresultants is possible as follows: Successively apply Corollary 2.3.13 to already computed subresultants starting with $\mathrm{SRes}_n(A, B)$ and $\mathrm{SRes}_{n-1}(A, B)$ (which needs to be computed elsewise since Corollary 2.3.13 cannot be applied with $j = m$)[†]. As soon as $\deg(\mathrm{SRes}_{k-1}(A, B)) = l < k-1$, we have a defective sequence. Thus, we have to use the proportionality property stated in Theorem 2.3.12 to determine $\mathrm{SRes}_l(A, B)$. The elements $\mathrm{SRes}_{k-2}(A, B), \ldots, \mathrm{SRes}_{l+1}(A, B)$ are identically zero. Now, $\mathrm{SRes}_l(A, B)$ can again be used as a dividend in Corollary 2.3.13 since $\deg \mathrm{SRes}_l(A, B) = l$.

The signed subresultant cofactors SResU and SResV can be computed similarly by utilizing the quotients SResQ_{k-1}. Again, all intermediate results are in $D[x]$.

Lemma 2.3.14. *If* $\deg(\mathrm{SRes}_j) = j$, $0 < j \leq n$ *and* $\deg(\mathrm{SRes}_{j-1}) = k$, $k \leq j-1$, *then*

$$\mathrm{SResU}_{k-1} = -\frac{\mathrm{sres}_k\,\overline{\mathrm{sres}}_{j-1}\,\mathrm{SResU}_j - \mathrm{SResQ}_{k-1}\,\mathrm{SResU}_{j-1}}{\mathrm{sres}_j\,\overline{\mathrm{sres}}_{i-1}}, \tag{2.27}$$

$$\mathrm{SResV}_{k-1} = -\frac{\mathrm{sres}_k\,\overline{\mathrm{sres}}_{j-1}\,\mathrm{SResV}_j - \mathrm{SResQ}_{k-1}\,\mathrm{SResV}_{j-1}}{\mathrm{sres}_j\,\overline{\mathrm{sres}}_{i-1}}, \tag{2.28}$$

where $\mathrm{SResQ}_{k-1} = \mathrm{quot}(\mathrm{sres}_k\,\overline{\mathrm{sres}}_{j-1}\,\mathrm{SRes}_j, \mathrm{SRes}_{j-1})$.

Proof. It is easy to verify by induction that the polynomials SResU_{k-1} and SResV_{k-1} satisfy the relation $\mathrm{SRes}_{k-1} = \mathrm{SResU}_{k-1}\,A + \mathrm{SResV}_{k-1}\,B$. This immediately implies the result due to the uniqueness of this representation (see Corollary 2.3.8). □

See also [BPR03, Algorithm 8.75] for the pseudocode of the algorithm for computing the signed subresultants and their cofactors.

Remark. [Rei97] and [LR01] propose an asymptotically faster algorithm to compute a specific subresultant and its cofactors. It only needs $\mathcal{O}(m \log m)$ arithmetic operations in the coefficient domain instead of the $\mathcal{O}(m^2)$ operations imposed by Theorem 2.3.12. However, if the complete sequence is needed, no improvement of the asymptotic complexity is possible since the number of coefficients in the sequence is in $\Theta(n^2)$.

The structure that arises in the sequence of signed subresultants is often called the *gap structure* due to the gaps that occur in defective sequences. The corresponding diagrams illustrating the sequence of degrees as shown in Figure 2.1 are also known as *Habicht diagrams* due to their occurrence in [Hab48].

In order to better understand Theorem 2.3.12, we connect the signed subresultant sequence to polynomial remainder sequences. We need a few definitions.

[†] Corollary 2.3.13 would be applicable to $j = m$ if we would define $\mathrm{SRes}_m(A, B) = \mathrm{sign}(a_m^{m-n-1})a_m^{-1}A$ for $A, B \in D[x]$. In this case, it is possible that $\mathrm{SRes}_m(A, B) \notin D[x]$. Therefore, we stay close to the definition in [BPR03].

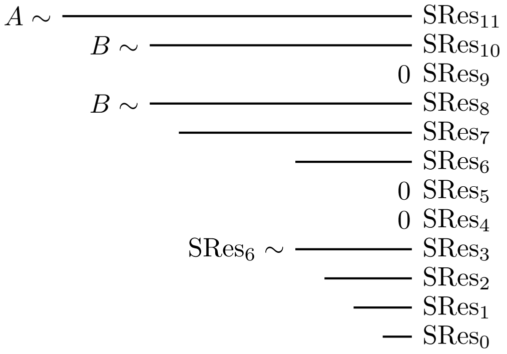


Figure 2.1.: A Habicht diagram illustrating a possible sequence of degrees in a particular sequence of signed subresultants for polynomials A and B with degrees 11 and 8. The length of the lines is chosen according to the number of coefficients, i.e. proportional to $\deg(\mathrm{SRes}_j(A, B)) + 1$. Lines of equal length correspond to subresultants that are similar.

Definition 2.3.15 (similar polynomials). *Two polynomials $A, B \in R[x]$ are* similar*, denoted $A \sim B$, if there exist non-zero $a, b \in R$ such that $aA = bB$.*

Lemma 2.3.16 (pseudodivision). *Let S be a commutative ring and $A, B \in S[x]$ non-zero polynomials of degree m, n. Then there exist $Q, R \in S[x]$ such that*

$$\mathrm{lcoeff}(B)^{\max(m-n+1,0)}A = QB + R \tag{2.29}$$

with $\deg R < n$*. They are unique if* $\mathrm{lcoeff}(B)$ *is not a zero divisor in S. We call* $\mathrm{pquot}(A, B) = Q$ *the* pseudoquotient *and* $\mathrm{prem}(A, B) = R$ *the* pseudoremainder *of the* pseudodivision *of A and B.*

A proof for the existence and uniqueness of the pseudoquotient and -remainder can be found in [Mis93, Theorem 5.2.1]. Note that $\mathrm{pquot}(A, B) = \mathrm{quot}(\mathrm{lcoeff}(B)^{\max(m-n+1,0)}A, B)$ and $\mathrm{prem}(A, B) = \mathrm{rem}(\mathrm{lcoeff}(B)^{\max(m-n+1,0)}A, B)$.

Definition 2.3.17 (polynomial remainder sequence (PRS)). *Let $A_0, A_1 \in R[x]$. Then $[A_k, \ldots, A_1, A_0]$ is called a polynomial remainder sequence (PRS) of A_0 and A_1 if*

$$A_{i+1} \sim \mathrm{prem}(A_{i-1}, A_i), \quad 1 \le i < k, \tag{2.30}$$

such that $\mathrm{prem}(A_{k-1}, A_k) = 0$*.*

An important and well known PRS is the so-called *Euclidean PRS*, which uses $A_{i+1} = \mathrm{rem}(A_{i-1}, A_i)$ and hence requires S to be a field. A variant of this PRS, which we will use in some of our examples as well as for counting the real roots of polynomials, is the *signed Euclidean PRS* where $A_{i+1} = -\mathrm{rem}(A_{i-1}, A_i)$. In [BPR03], this PRS is also called *signed remainder sequence.* The next corollary follows immediately from Theorem 2.3.12.

Corollary 2.3.18. *Each polynomial in the signed subresultant sequence $\mathcal{S} = [\mathrm{SRes}_0(A, B), \ldots, \mathrm{SRes}_m(A, B)]$ is either similar to a polynomial in the signed remainder sequence of A and B or identically zero.*

The signed remainder sequence is, up to sign, equal to the classical Euclidean remainder sequence, which is used in the Euclidean algorithm to compute greatest common divisors (GCDs) of polynomials. Corollary 2.3.18 already suggests that there is a strong connection between the GCD of polynomials and subresultants. This is indeed the case.

Theorem 2.3.19. *Let* $\mathcal{S} = [\mathrm{SRes}_0(A,B), \ldots, \mathrm{SRes}_n(A,B)]$ *be the sequence of signed subresultants of* $A, B \in D[x]$, *D a UFD.* $\mathrm{SRes}_j(A,B)$ *is a GCD of A and B if and only if* $\mathrm{SRes}_j(A,B) \not\equiv 0$ *and* $\mathrm{SRes}_i(A,B) \equiv 0$ *for* $0 \leq i < j$. *Furthermore,* $\mathrm{SRes}_j(A,B)$ *is non-defective, i.e.* $\deg \mathrm{SRes}_j(A,B) = j$.

Proof. See [BPR03, Corollary 4.45, Corollary 8.55]. □

Since $\gcd(A,B)$ divides A and B, it can be used to remove common factors from both polynomials. This is especially useful if $B = A'$.

Definition 2.3.20 (squarefree polynomial). *A polynomial* $A \in D[x]$, *D a UFD, is called* squarefree *if it has no repeated factors, i.e. if there exist no* $B \in D[x]$ *with* $\deg B \geq 1$ *such that* $B^2 | A$. *If* $aA = \prod_{i=0}^{n} A_i^{m_i}$ *with squarefree polynomials* $A_i \in D[x]$ *and a constant* $a \in D$ *then* $\prod_{i=0}^{n} A_i$ *is called the* squarefree part *of A.*

Several algorithms in this thesis require their input to be squarefree. Therefore, we need to be able to compute the squarefree part of a polynomial. Note that the squarefree part is unique up to some multiplicative constant in D.

Lemma 2.3.21. *The polynomial* $\frac{A}{\gcd(A,A')}$ *is squarefree.*

Proof. See [BPR03, Lemma 10.12]. □

We can compute $\gcd(A, A')$ using subresultants according to Theorem 2.3.19 and then perform the division $\frac{A}{\gcd(A,A')}$. By utilizing the subresultant cofactors, the squarefree part can also be expressed as a determinant. Thus, divisions are not necessary to compute the squarefree part.

Lemma 2.3.22. *If* $\deg(\gcd(A,A')) = j$, *then* $\mathrm{SResV}_{j-1}(A,A')$ *is the squarefree part of A.*

Proof. See [BPR03, Corollary 10.14]. □

If the coefficient ring is $\mathbb{Z}$, Theorem 2.3.19 and Lemma 2.3.22 can be extended to further reduce the size of the coefficients.

Lemma 2.3.23. *Let* $\mathrm{SRes}_j = \mathrm{SRes}_j(A,A')$ *and* $\mathrm{SResV}_{j-1} = \mathrm{SResV}_{j-1}(A,A')$ *for* $A \in \mathbb{Z}[x]$. *If* $\deg(\gcd(A,A')) = j$, *then* $\frac{\mathrm{lcoeff}(A)\,\mathrm{SRes}_j}{\overline{\mathrm{sres}}_j}, \frac{\mathrm{lcoeff}(A)\,\mathrm{SResV}_{j-1}}{\mathrm{lcoeff}(\mathrm{SResV}_{j-1})} \in \mathbb{Z}[x]$. *The bitsize of the coefficients of* $\frac{\mathrm{lcoeff}(A)\,\mathrm{SRes}_j}{\overline{\mathrm{sres}}_j}$ *and* $\frac{\mathrm{lcoeff}(A)\,\mathrm{SResV}_{j-1}}{\mathrm{lcoeff}(\mathrm{SResV}_{j-1})}$ *is* $n - j + \tau + \mathrm{bit}(n+1)$.

Proof. See [BPR03, Algorithm 10.17]. □

2.3.1. Size of remainders and subresultants

The size of the remainders in a PRS is of major influence on the time spent in exact algorithms and on the stability of numerical algorithms. Let us first consider the following example taken from [BPR03, Example 8.44]. It illustrates the coefficient growth in the sequence of signed remainders.

Example 2.3.24 *(size of signed remainders).* Let

$$\begin{aligned} A :=& 9x^{13} - 18x^{11} - 33x^{10} + 102x^8 + 7x^7 - 36x^6 \\ &- 122x^5 + 49x^4 + 93x^3 - 42x^2 - 18x + 9. \end{aligned} \tag{2.31}$$

The GCD of A and A' is of degree 5. The leading coefficients of the polynomials in the signed remainder sequence of A and A' are

$$\begin{gathered}\frac{36}{13},\\ -\frac{10989}{16},\\ -\frac{2228672}{165649},\\ -\frac{900202097355}{4850565316},\\ -\frac{3841677139249510908}{543561530761725025},\\ -\frac{6648854900739944448789496725}{6761403525275795353156967 12},\\ -\frac{2001176705547816993081646924785441 84}{180730930229098050132455395887141 5645}.\end{gathered} \tag{2.32}$$

It is also possible to give a bound on the size of the coefficients.

Lemma 2.3.25 (size of signed remainders). *If $A, B \in \mathbb{Z}[x]$ are of degree m and $n < m$ and have coefficients of bitsize at most τ, then the numerators and denominators of the coefficients of the polynomials in the signed remainder sequence of A and B have bitsizes at most $(m+n)(n+1)(\tau + \operatorname{bit}(m+n)) + \tau$.*

Proof. Determine the constants of proportionality of the signed remainders and the signed subresultants. These constants are rational expressions in the subresultant coefficients. Derive the upper bounds by applying Lemma 2.3.27. See [BPR03, Theorem 8.70] for the details. ☐

According to [BPR03, Section 8.3.3], the quadratic growth (in n) of the size of the coefficients is often observed in practice. Furthermore, the coefficients are in $\mathbb{Q}$ although $A \in \mathbb{Z}[x]$. In order to remove this limitation, one could apply pseudodivision instead of Euclidean division. Unfortunately, this leads to an exponential growth of the bitsize of the coefficients in the polynomial remainder sequence (see [GCL92, p. 282]). Applying this method to Example 2.3.24 yields a leading coefficients with 980 decimal digits in the last non-zero pseudoremainder. It would be possible to take the primitive part of the pseudoremainder after each pseudodivision. This method yields the *primitive PRS*. Although the primitive PRS is the PRS with the smallest coefficients possible, it can be quite time consuming to determine the content of the pseudoremainders, especially in the case of multivariate polynomials. Again, subresultants give a nice solution to this problem. We will first revisit Example 2.3.24 and compute its principal signed subresultant coefficients. Afterwards, we state an upper bound for the size of the subresultant coefficients.

Example 2.3.26 *(size of signed subresultants).* Let A be defined as in Example 2.3.24. The principal signed subresultant coefficients $\mathrm{sres}_{11}, \ldots, \mathrm{sres}_5$ of A and A' are

$$\begin{gathered}37908,\\ -72098829,\\ -666229317948,\\ -16635227404 00320,\\ -2181968897553243072,\\ -151645911413926622112,\\ -1651177113027 36225120\end{gathered} \tag{2.33}$$

and $\mathrm{sres}_4, \ldots, \mathrm{sres}_0 \equiv 0$.

Lemma 2.3.27 (size of signed subresultant coefficients). *If $A, B \in \mathbb{Z}[x]$ are of degree m and $n < m$ and have coefficients of bitsizes at most τ, then the bitsize of the coefficients of* $\mathrm{SRes}_j(A,B), \mathrm{SResU}_j(A,B), \mathrm{SResV}_j(A,B)$ *is at most* $(\tau + \mathrm{bit}(m+n-2j))(m+n-2j)$.

Proof. Apply Hadamard's inequality (see Lemma 2.1.5 and Corollary 2.1.6) to the coefficients of $\mathrm{SRes}_j(A,B)$ written in matrix form (see Corollary 2.3.5). See [BPR03, Proposition 8.67] for the details. ☐

The above result can be extended to multivariate polynomials.

Lemma 2.3.28 (degree of signed subresultant coefficients). *If A and B have degrees m and n and coefficients in $R[y_1, \ldots, y_l]$ of total degree d, then* $\deg(\mathrm{coeff}_i(\mathrm{SRes}_j(A,B))) \leq d(m+n-2j)$ *for all* $0 \leq i \leq j$.

Proof. Using the Leibniz formula, the determinants $\det(S_{ij}(A,B))$ (see Corollary 2.3.5) can be written as a sum of products of $m+n-2j$ polynomials having degree at most d. ☐

Lemma 2.3.29 (magnitude of signed subresultant coefficients (multivariate case)). *If A and B have degrees m and n and coefficients in $\mathbb{Z}[y_1, \ldots, y_l]$ of total degree d and bitsize at most τ, then the coefficients of* $\mathrm{SRes}_j(A,B)$ *have bitsizes at most* $(\tau + \mathrm{bit}(m+n))(m+n-2j) + l\,\mathrm{bit}((m+n)d+1)$.

Proof. See [BPR03, Proposition 8.69]. ☐

The above results show that the subresultant PRS gives a method to compute (up to a constant) the GCD, the resultant and the squarefree part of polynomials that is superior to the signed remainder sequence.

Given the size of the subresultant coefficients, it is not hard to derive a bound on the size of the signed subresultant quotient. This bound will be useful in Section 2.4.1.1.

Corollary 2.3.30 (magnitude of signed subresultant quotient coefficients). *Let $A, B \in D[y_1, \ldots, y_l][x]$ be defined as in Lemma 2.3.29 and let $q \in D[y_1, \ldots, y_l]$ be a coefficient of the signed subresultant quotient* SResQ_{k-1} *as defined in Theorem 2.3.12. Then*

$$\deg_{y_1,\ldots,y_k}(q) \leq 3d' \tag{2.34}$$

$$\mathrm{bit}(q) \leq 3\tau' \tag{2.35}$$

where $d' = d(m+n-2(k-1))$ and $\tau' = (\tau + \mathrm{bit}(m+n))(m+n-2(k-1)) + l\,\mathrm{bit}((m+n)d+1)$ are the bounds on $\deg_{y_1,\ldots,y_l}(\mathrm{SRes}_{k-1}(A,B))$ *and the bitsize of the coefficients of* $\mathrm{SRes}_{k-1}(A,B)$.

Proof. By Theorem 2.3.12, it holds that

$$\mathrm{sres}_j \, \overline{\mathrm{sres}}_{i-1} \, \mathrm{SRes}_{k-1} + \mathrm{sres}_k \, \overline{\mathrm{sres}}_{j-1} \, \mathrm{SRes}_{i-1} = \mathrm{SResQ}_{k-1} \, \mathrm{SRes}_{j-1} \, . \tag{2.36}$$

Each coefficient of the product $\mathrm{SResQ}_{k-1} \, \mathrm{SRes}_{j-1}$ is the sum of the product of three coefficients of $\mathrm{SRes}_l(A,B)$, $l = k, j-1, j, i-1$. Since d' is also a bound on $\deg_{y_1,\ldots,y_l}(\mathrm{SRes}_l(A,B))$, $l = k, j-1, j, i-1$, the first result follows. The statement on the bitsize follows by the same line of arguments. We only have to note that the one additional bit caused by the sum can be ignored since τ' overestimates the bitsize of the coefficients of $\mathrm{SRes}_l(A,B)$, $l = k, j-1, j, i-1$. ☐

2.3.2. Specialization properties

We will now examine the behavior of resultants and subresultants under *specialization*. The term specialization arises from computations with polynomials with indeterminate coefficients. The question to answer is the following: Can we interchange the order of specializing the indeterminate coefficients of A and B to values from a concrete ring with the computation of $\mathrm{SRes}_j(A, B)$?

Most of the facts derived in this section originally appeared in [Gon+90; Gon+94]. Nevertheless, we refer to more recent literature where appropriate.

We first define homomorphisms in order to explore the concept of specialization in a more general setting.

Definition 2.3.31 (homomorphism). *Let $(A, (f_i))$ and $(B, (g_i))$ be two algebraic structures and let σ_i be the arity of the operations f_i and g_i. A mapping $\varphi : A \to B$ is called a* homomorphism *with respect to the algebraic structures $(A, (f_i))$ and $(B, (g_i))$ if*

$$\varphi(f_i(a_1, \dots, a_{\sigma_i})) = g_i(\varphi(a_1), \dots, \varphi(a_{\sigma_i})) \tag{2.37}$$

for all i and all $a_1, \dots, a_{\sigma_i} \in A$.

If $\varphi : R \to R'$ is a ring homomorphism, we will usually denote the induced homomorphism between polynomials rings $R[x]$ and $R'[x]$ and between matrices over R and R' by φ, too.

As an example, consider a multivariate polynomial $A \in \mathbb{Z}[y_1, \dots, y_k][x]$. Computations involving $\tilde{A}(x) = A(p_1, \dots, p_k, x)$ with $(p_1, \dots, p_k) \in (\overline{\mathbb{Q}} \setminus \mathbb{Q})^k$, i.e. with A evaluated a point whose coordinates are not-rational algebraic numbers, are mostly extremely difficult. In this case, it is often advantageous to perform the computations directly on the multivariate polynomial A and evaluate the result at $(y_1, \dots, y_k) = (p_1, \dots, p_k)$ afterwards. This type of homomorphism is usually referred to as the *evaluation homomorphism.* Depending on the concrete application, numerical methods can often be utilized in the evaluation step. Clearly, not all computations involving multivariate polynomials can be carried out this way. We will now study the limitations.

Lemma 2.3.32 (specialization property of subresultants). *Let $A, B \in R[x]$ and $\varphi : R \to R'$ a ring homomorphism. If* $\deg(\varphi(A)) = \deg(A)$ *and* $\deg(\varphi(B)) = \deg(B)$*, then*

$$\varphi(\mathrm{SRes}_j(A, B)) = \mathrm{SRes}_j(\varphi(A), \varphi(B)). \tag{2.38}$$

Proof. If the degrees of A and B do not change under φ, the matrices $\varphi(S_j(A, B))$ and $S_j(\varphi(A), \varphi(B))$ are equal and so are their determinants. □

In this thesis, we will often consider evaluation homomorphisms for multivariate polynomials over $\mathbb{Z}$ resp. $\mathbb{Q}$. In the bivariate case, a polynomial $A = \sum_{i=0}^{n} a_i(y)x^i$ might be specialized by the homomorphism $\varphi_c : \mathbb{Q}[y] \to \mathbb{R}$ given by $\varphi_c(a_i(y)) = a_i(c)$ for some constant $c \in \mathbb{R}$. It is clear from the fundamental theorem of algebra (see Theorem 2.4.1) that $a_n(y)$ has at most n complex roots. Therefore, Lemma 2.3.32 is applicable for almost any $c \in \mathbb{R}$. In general, if $A \in \mathbb{Q}[y_1, \dots, y_k][x]$, then $a_n(y_1, \dots, y_k)$ vanishes on a subset of $\mathbb{C}^k$ with codimension at most 1, i.e. the condition of Lemma 2.3.32 is fulfilled for almost any randomly chosen point $c \in \mathbb{C}^k$. However, in practice it is useful to specialize the subresultants also at a point where $\deg(A)$ or $\deg(B)$ decreases under φ.

Lemma 2.3.33. *If* $\deg(\varphi(A)) = m$ *and* $\deg(B) - \deg(\varphi(B)) = k \leq 0$*, then for* $0 \leq j \leq n$

$$\varphi(\mathrm{SRes}(j, A, B)) = \begin{cases} \varphi(a_m)^k \,\mathrm{SRes}(j, \varphi(A), \varphi(B)) & j \leq \deg(\varphi(B)), \\ 0 & \textit{otherwise.} \end{cases} \tag{2.39}$$

Proof. The proof for the case of $j \leq \deg(\varphi(B))$ can be found in [BPR03, p. 278f and Lemma 8.50]. In the remaining case $\deg(\varphi(B)) < j$, the columns $n - j + 1$ and $n - j + 2$ are zero up to the last element. Thus, these columns are linearly dependent. ☐

For brevity we skip the cases, where $j > n$. They can be derived directly from Definition 2.3.4. Finally, we need to examine the case where $\deg(\varphi(A)) < \deg(A)$ in addition to $\deg(\varphi(B)) < \deg(B)$.

Corollary 2.3.34. *If* $\deg(\varphi(A)) < \deg(A)$ *and* $\deg(\varphi(B)) < \deg(B)$*, then* $\varphi(\mathrm{SRes}_j(A, B)) = 0$ *for* $j < n$*.*

Proof. Clearly, all elements in the first row of $S_j(A, B)$ are mapped to zero by φ yielding $\varphi(\det(S_j(A, B))) = \det(\varphi(S_j(A, B))) = 0$. ☐

This last case is often missing in textbooks on computer algebra (e.g. in [BPR03; GCL92; Yap00] while it is stated in [Mis93]). The corollary is not difficult to derive and it will be very useful in Chapter 6.

2.3.3. Polynomial content

Consider subresultants that are computed over a multivariate polynomial ring. In this case, it is often helpful to know their polynomial content or at least some factor of it a priori since each subresultant vanishes completely on the zero set of its content. In the following, we show how certain factors of the coefficients of A and B relate to the content of $\mathrm{SRes}_j(A, B)$.

Lemma 2.3.35. *Let* $A = \sum_{i=0}^{m} a_i x^i$, $B = \sum_{i=0}^{n} b_i x^i \in D[x]$ *and* $c, d \in D$, D *a UFD, with* $c \mid A$ *and* $d \mid B$*. Then* $c^{n-j} d^{m-j} \mid \mathrm{SRes}_j(A, B)$ *for* $0 \leq j < \min(m, n)$*.*

Proof. The matrix $S_j(A, B)$ has $n-j$ columns with entries that are coefficients of A or multiples of A (in the last row). Therefore, these columns are divisible by c and $c^{n-j} \mid \det(S_j(A, B)) = \mathrm{SRes}_j(A, B)$. The same arguments hold for the $m - j$ columns which are divisible by d. ☐

The following result can be used if $d = c$ and c does not divide all coefficients of A and B.

Lemma 2.3.36. *Let* $A, B \in D[x]$ *and* $c \in D$ *as above. If* $c \mid a_m, \ldots, c \mid a_{m-k}, c \mid b_n, c \mid \ldots, b_{n-k}$ *for* $0 \leq k \leq \min(m, n)$ *then* $c^l \mid \mathrm{SRes}_j(A, B)$ *with* $l = \min(k + 1, m + n - 2j - 1)$ *for* $0 \leq j < \min(m, n)$*.*

Proof. The entries of the first $m + n - 2j - 1$ rows of $S_j(A, B)$ are built from the coefficients a_i, b_i and the first $\min(k + 1, m + n - 2j - 1)$ of the rows are built from (a subset of) the coefficients $a_m, \ldots, a_{m-k}, b_n, \ldots, b_{n-k}$ which are divisible by c. ☐

2.3.4. Chain rule

Another important question is the behavior of subresultants under composition, i.e. how are the subresultants of $(A\circ C)(x) = A(C(x))$ and $(B\circ C)(x) = B(C(x))$ connected to the subresultants of $A(x)$ and $B(x)$.

Lemma 2.3.37. *Let* $A, B, C \in D[x]$ *with* $\deg(A) = m, \deg(B) = n$ *and* $\deg(C) = k$ *and denote* $\mathrm{SRes}_j = \mathrm{SRes}_j(A, B)$ *and* $\mathrm{SRes}^*_j = \mathrm{SRes}_j(A \circ C, B \circ C)$*. Then it holds that*

$$\mathrm{SRes}^*_{jk} = \pm \operatorname{lcoeff}(C)^{mnk-j^2k-j} \operatorname{sres}_j^{k-1} \mathrm{SRes}_j \circ C \tag{2.40}$$

and if $k > 1$

$$\mathrm{SRes}^*_{jk-1} = \pm \operatorname{lcoeff}(C)^{mnk-j^2k+j} \operatorname{sres}_j^{k-1} \mathrm{SRes}_{j-1} \circ C \tag{2.41}$$

$$\mathrm{SRes}^*_{jk-2} = \cdots = \mathrm{SRes}^*_{(j-1)k+1} = 0 \tag{2.42}$$

for $0 \le j \le \min(m, n)$*.*

Proof. See [Hon97; Che01]. ☐

In [Hon97; Che01], the chain rule has been derived for Sylvester-Habicht polynomials, which are equal to signed subresultants up to sign. We restrain from the tedious derivation of actual sign of the SRes^* since we do not need it in our later considerations.

2.3.5. Resultants and the extension theorem

We have already seen in Section 2.3 that the resultant can be used to check if two univariate polynomials have a common factor. We will now go one step further and have a look at the multivariate case.

Definition 2.3.38 (elimination ideal). *Given* $I = \langle f_1, \dots, f_s\rangle \subset R[x_1, \dots, x_k]$ *the* l*-th* elimination ideal I_l *is the ideal of* $R[x_{l+1}, \dots, x_k]$ *defined by*

$$I_l = I \cap R[x_{l+1}, \dots, x_k]. \tag{2.43}$$

A similar construction, which is geometric instead of algebraic, is the projection.

Definition 2.3.39 (projection). *The* projection *of a set* $V \in \mathbb{A}^k$ *to an affine subspace* $\mathbb{A}^{k-l}$ *with* $l < k$ *is given by*

$$\pi_l : \mathbb{A}^k \to \mathbb{A}^{k-l}, \quad \pi_l(V) = \{(a_{l+1}, \dots, a_k) : (a_1, \dots, a_k) \in V\}. \tag{2.44}$$

In algebraic geometry, elimination and projection can be connected using the following theorem.

Theorem 2.3.40 (closure theorem). *Let* $V(f_1, \dots, f_s) \subset \mathbb{C}^k$ *and let* I_l *be the* l*-th elimination ideal of* $\langle f_1, \dots, f_s\rangle$*. Then* $V(I_l)$ *is the Zariski closure of* $\pi_l(V(f_1, \dots, f_s)) \in \mathbb{C}^{k-l}$*.*

Proof. See [CLO07, Theorem 3.2.3]. ☐

Therefore, if we want to solve for $f_1 = \cdots = f_s = 0$, we can compute $V(I_1) \in \mathbb{A}^{k-1}$, which is a superset of the projection of $V(I)$ to $\mathbb{A}^{k-1}$, and then try to lift the partial solutions back to $\mathbb{A}^k$. The task of finding the first elimination ideal of an ideal $\langle A, B\rangle$ can be solved by resultants. It will occur frequently in this thesis.

Proposition 2.3.41. *Let* $A, B \in R[x_1, \ldots, x_k]$ *have positive degrees in* x_1. *Then* $\operatorname{Res}_{x_1}(A, B)$ *is in the first elimination ideal* $\langle A, B\rangle \cap R[x_2, \ldots, x_k]$.

Proof. Obviously, $\operatorname{Res}_{x_1}(A, B) \in R[x_2, \ldots, x_k]$ since it is an integer polynomial in the coefficients of A and B which are elements of $R[x_2, \ldots, x_k]$. Furthermore, we know by Corollary 2.3.8 that $\operatorname{Res}_{x_1}(A, B) = UA + VB$ with $U, V \in R[x_1, \ldots, x_k]$. Hence, $UA + VB \in \langle A, B\rangle$. ☐

In order to compute higher elimination ideals of ideals generated by more than two polynomials, more sophisticated methods like Gröbner bases or multipolynomial resultants have to be considered, which is out of the scope of this thesis. The next result can be seen as a generalization of the closure theorem since it gives a condition for $x \in V(I_1) \cap \pi_1(V)$.

Theorem 2.3.42 (extension theorem). *Let* $I = \langle f_1, \ldots, f_s\rangle \subset \mathbb{C}[x_1, \ldots, x_k]$ *and let* I_1 *be the first elimination ideal of* I. *Suppose that there is a partial solution* $(c_2, \ldots, c_k) \in V(I_1)$. *If* $(c_2, \ldots, c_k) \notin V(\operatorname{lcoeff}_{x_1}(f_1), \ldots, \operatorname{lcoeff}(f_s))$. *Then there exists a* $c_1 \in \mathbb{C}$ *such that* $(c_1, c_2, \ldots, c_k) \in V(f_1, \ldots, f_s)$.

Proof. See [CLO07, Theorem 3.6.4]. ☐

The extension theorem tells us that every partial solution of a polynomial system of equations extends to a solution if we are working over $\mathbb{C}^k$ provided the leading coefficients of the defining polynomials do not vanish at the partial solution. The latter restriction can be dropped if we move over to the projective extension theorem and allow solutions from $\mathbb{P}^1_{\mathbb{C}} \times \mathbb{C}^{k-1}$. Since we are mainly interested in real solutions, we have to face the additional problem that a partial solution from $\mathbb{R}^{k-1}$ will not necessarily extend to a solution in $\mathbb{R}^k$ even if the condition on the leading coefficients is met.

2.4. Polynomial real root counting

For polynomials with coefficients in the complex numbers, we have the following well known statement about its roots.

Theorem 2.4.1 (Fundamental theorem of algebra). *Every non-constant polynomial* $A \in \mathbb{C}[x]$ *has a root in* $\mathbb{C}$.

Proof. See e.g. [BPR03, Theorem 2.14 and Remark 2.21]. ☐

While the total number of complex roots of a real polynomial is, due to the fundamental theorem of algebra, immediately clear by its degree, there is no such simple rule for its number of real roots. Hence, counting the number of real roots of a real polynomial is one of the fundamental problems in algorithmic real algebraic geometry. The known methods can be roughly grouped into two categories: Algorithms that allow to directly retrieve the number of real roots over any interval and methods that have to apply some kind of subdivision scheme to find the exact number of real roots. For the first class, we will review *Sturm chains* and its extension based on subresultants. As an instance of the second class, we will look at a subdivision method that is based on *Descartes' rule of signs*. We will introduce some useful notation before going into the details.

Definition 2.4.2 (number of real roots). *We denote by*

$$\#\mathrm{rr}(A, I) = |\{x \in I : A(x) = 0\}| \tag{2.45}$$

the number of distinct real roots *of A within $I \subseteq \mathbb{R}$. We take multiplicities into account by using*

$$\#\mathrm{rrm}(A, I) = \sum_{x \in I \wedge A(x)=0} m(A, x), \tag{2.46}$$

where $m(A, x)$ denotes the multiplicity of x as a root of A. The case $I = \mathbb{R}$ may be abbreviated by $\#\mathrm{rr}(A) = \#\mathrm{rr}(A, \mathbb{R})$ and $\#\mathrm{rrm}(A) = \#\mathrm{rrm}(A, \mathbb{R})$.

In order to count the total number of real roots of a real polynomial, we can utilize the following bound to find an interval containing all real roots. It is shown in [Slu70] that this bound is nearly optimal.

Lemma 2.4.3 (Fujiwara root bound). *Let $A = \sum_{i=0}^{m} a_i x^i \in \mathbb{C}$ and $r \in \mathbb{C}$ be a root of A. Then*

$$|r| < 2 \max\left(\left|\frac{a_{m-1}}{a_m}\right|, \left|\frac{a_{m-2}}{a_m}\right|^{\frac{1}{2}}, \ldots \left|\frac{a_1}{a_m}\right|^{\frac{1}{m-1}}, \left|\frac{a_0}{2a_m}\right|^{\frac{1}{m}} \right). \tag{2.47}$$

Proof. See [Fuj16]. □

Finally, two of the presented methods for real root counting rely on the counting scheme for the number of sign variations in a sequence of numbers introduced below.

Definition 2.4.4 (number of sign variations). *The* number of sign variations*, $\mathrm{Var}(\mathcal{A})$, of a nonempty sequence $\mathcal{A} = [a_0, a_1, \ldots, a_k]$ of nonzero real numbers $a_i \in \mathbb{R} \setminus \{0\}$ is defined inductively by*

$$\mathrm{Var}(a_0) = 0 \tag{2.48}$$

$$\mathrm{Var}(a_0, a_1, \ldots, a_k) = \mathrm{Var}(a_0, a_1, \ldots, a_{k-1}) + \begin{cases} 1 & \text{for } a_{k-1}a_k < 0, \\ 0 & \text{for } a_{k-1}a_k > 0. \end{cases} \tag{2.49}$$

If $\mathcal{A}$ contains zeros, we set $\mathrm{Var}(\mathcal{A}) = \mathrm{Var}(\tilde{\mathcal{A}})$ where $\tilde{\mathcal{A}}$ is created from $\mathcal{A}$ by removing all zero elements.

2.4.1. Sturm chains and signed subresultant sequences

Definition 2.4.5 (Sturm chain). *Let $A \in \mathbb{R}[x]$ be a nonzero polynomial and let $\mathcal{A} = [A_k, \ldots, A_1, A_0]$ be a PRS of $A_0 = A$ and $A_1 = A'$. $\mathcal{A}$ is called a* Sturm chain *(or* Sturm sequence*) if for all $i = 1, \ldots, k-1$ it holds that*

$$\beta_i A_{i+1} = \alpha_i A_{i-1} + Q_i A_i \tag{2.50}$$

for $\alpha_i, \beta_i \in \mathbb{R}$, $Q_i \in \mathbb{R}[x]$ such that $\alpha_i \beta_i < 0$.

We have already encountered Sturm chains above: The signed Euclidean PRS of A and A' is a Sturm chain since $A_{i+1} = -(A_{i-1} - \mathrm{quot}(A_{i-1}, A_i)A_i) = -A_{i-1} + Q_i A_i$. The famous result by Sturm is the following.

Theorem 2.4.6 (Sturm's theorem). *Let* $\mathcal{A}(x) = [A_k(x), \dots, A_1(x), A_0(x)]$ *be a Sturm sequence of* $A = A_0 \in \mathbb{R}[x]$ *and let* $a < b$ *be elements of* $\mathbb{R}$ *that are not roots of* A*. Then*

$$\#\mathrm{rr}(A, [a, b]) = \mathrm{Var}(\mathcal{A}(a)) - \mathrm{Var}(\mathcal{A}(b)). \tag{2.51}$$

Proof. The original proof (in French) can be found in [Stu29]. For a proof in a more general context consider [Mis93, Corollary 8.4.4] or [BPR03, Theorem 2.56]. ☐

We have already stated some drawbacks of the Euclidean PRS. The coefficient domain needs to be a field and the coefficient growth is quadratically in the degree. In addition, the Euclidean PRS does not behave well under specialization. Since each element in the signed subresultant sequence is proportional to an element in the Sturm sequence, it seems natural that the same information about the roots can be extracted from the sequence of signed subresultants. This is indeed the case, as shown in [GRL98]. First, we need to define another function to count sign variations in a sequence of real numbers.

Definition 2.4.7 (modified number of sign variations). *Let* $\mathcal{P} = [P_0, P_1, \dots, P_n]$*,* $P_i \in \mathbb{R}[x]$*, be an arbitrary sequence of polynomials and* a *be an element of* $\mathbb{R}$*. Then* $\mathrm{MVar}(\mathcal{P}, a)$*, the* modified number of sign variations *of* $\mathcal{P}$ *at* a*, is defined as follows:*

1. *Obtain a list* $\mathcal{Q} = [Q_0, Q_1, \dots, Q_m]$ *from* $\mathcal{P}$ *by deleting all polynomials identically zero.*
2. *Obtain a list* $\mathcal{Q}(a) = [Q_0(a), Q_1(a), \dots, Q_m(a)]$ *by evaluating the elements of* $\mathcal{Q}$ *at* a*.*
3. *Count sign variations in groups of consecutive elements of* $\mathcal{Q}(a)$ *using the following rules:*
    - *Count* 1 *sign variation for the groups* $[-, +]$*,* $[+, -]$*,* $[-, 0, +]$*,* $[+, 0, -]$*,* $[-, 0, 0, +]$ *and* $[+, 0, 0, -]$*.*
    - *Count* 2 *sign variations for the groups* $[-, 0, 0, -]$ *and* $[+, 0, 0, +]$*.*

We may write $\mathrm{MVar}(\mathcal{P})$ *if* $\mathcal{P}$ *is already a sequence of numbers, and count the sign variations using step 3.*

Now we can state the result on real root counting using the signed subresultants.

Theorem 2.4.8 (Real root counting using signed subresultants). *Let* $\mathcal{S} = [\mathrm{SRes}_0, \dots, \mathrm{SRes}_n]$ *be the sequence of signed subresultants for* $A, A' \in \mathbb{R}[x]$ *and* $a, b \in \mathbb{R}$ *with* $a < b$ *and* $A(a)A(b) \neq 0$*. Then*

$$\#\mathrm{rr}(A, [a, b]) = \mathrm{MVar}(\mathcal{S}, a) - \mathrm{MVar}(\mathcal{S}, b). \tag{2.52}$$

Sign combinations other than listed in Definition 2.4.7 can not occur.

Proof. See [GRL98] or [BPR03, Theorem 9.30]. In both references, the more general Cauchy index is considered whereof real root counting is an application. ☐

The above theorem tells us nothing if the interval endpoints are roots of the polynomial A. The result can be extended in the following way.

Proposition 2.4.9. *Let* $\mathcal{S} = [\mathrm{SRes}_0(A), \dots, \mathrm{SRes}_n(A)]$ *be the sequence of signed subresultants for* $A \in \mathbb{R}[x]$ *and* $a, b, \varepsilon \in \mathbb{R}$ *with* $a < b$ *and* $\varepsilon > 0$ *such that* $[a - \varepsilon, a + \varepsilon] \setminus \{a\}$ *and* $[b - \varepsilon, b + \varepsilon] \setminus \{b\}$ *do not contain real roots of* A*. Then*

$$\#\mathrm{rr}(A, [a, b]) = \mathrm{MVar}(\mathcal{S}, a - \varepsilon) - \mathrm{MVar}(\mathcal{S}, b + \varepsilon), \tag{2.53}$$

$$\#\mathrm{rr}(A, (a, b)) = \mathrm{MVar}(\mathcal{S}, a + \varepsilon) - \mathrm{MVar}(\mathcal{S}, b - \varepsilon). \tag{2.54}$$

Proof. Since $\#\mathrm{rr}(A, [a-\varepsilon, a)) = \#\mathrm{rr}(A, [b, b+\varepsilon)) = 0$, it is clear that

$$\begin{aligned}\#\mathrm{rr}(A, [a,b]) &= \#\mathrm{rr}(A, [a-\varepsilon, a)) + \#\mathrm{rr}(A, [a,b]) + \#\mathrm{rr}(A, (b, b+\varepsilon]) && (2.55)\\ &= \#\mathrm{rr}(A, [a-\varepsilon, b+\varepsilon]) && (2.56)\\ &= \mathrm{MVar}(\mathcal{S}, a-\varepsilon) - \mathrm{MVar}(\mathcal{S}, b+\varepsilon). && (2.57)\end{aligned}$$

Similarly, we have $\#\mathrm{rr}(A, (a,b)) = \#\mathrm{rr}(A, [a+\varepsilon, b-\varepsilon])$. ☐

Remark. The previous proposition is easily extended to half open intervals.

In practice, it is hard to find a suitable ε that allows the application of Proposition 2.4.9. Therefore, we will use a sequence which is slightly different from the signed subresultant sequence and which allows to retrieve $\mathrm{MVar}(S, a \pm \varepsilon)$ by evaluating it at a. To that end, we need another proposition, which is an adaptation of [BPR03, Proof of Theorem 9.30].

Proposition 2.4.10. *Let $a \in \mathbb{R}$ be a simple real root of $A \in \mathbb{R}[x]$ and $\varepsilon > 0$ such that $[a-\varepsilon, a+\varepsilon] \setminus \{a\}$ does not contain roots of the polynomials in $\mathcal{S}_{0,\ldots,n-1} = [\mathrm{SRes}_0(A), \ldots, \mathrm{SRes}_{n-1}(A)]$. Then*

$$\mathrm{MVar}(\mathcal{S}_{0,\ldots,n-1}, a) = \mathrm{MVar}(\mathcal{S}_{0,\ldots,n-1}, a \pm \varepsilon). \tag{2.58}$$

Proof. Assume that a is a root of SRes_{j-1}, $j < n-1$. Since its multiplicity is one, it can not be a root of $A' = \mathrm{SRes}_{n-1}$. Applying $\mathrm{SRes}_{j-1}(a) = 0$ to Corollary 2.3.13 yields

$$\mathrm{sres}_j^2 \, \mathrm{SRes}_{k-1}(a) = -\,\mathrm{sres}_k \, \overline{\mathrm{sres}}_{j-1} \, \mathrm{SRes}_j(a). \tag{2.59}$$

Thus, $\mathrm{SRes}_{k-1}(a)$ and $\mathrm{SRes}_j(a)$ are both equal to zero or not equal to zero. If both are zero, then all subresultant polynomials are zero at a. But since a is a simple root of A, it is not a root of the greatest common divisor of A and A', which is, due to Theorem 2.3.19, the last non identically zero polynomial in the sequence. Thus, $\mathrm{SRes}_{k-1}(a)$ and $\mathrm{SRes}_j(a)$ are non-zero. For the signs we get

$$\begin{aligned}\mathrm{sign}(\mathrm{sres}_k \, \overline{\mathrm{sres}}_{j-1}) &= -\,\mathrm{sign}(\mathrm{SRes}_{k-1}(a)\,\mathrm{SRes}_j(a))\\ &= -\,\mathrm{sign}(\mathrm{SRes}_{k-1}(a \pm \varepsilon)\,\mathrm{SRes}_j(a \pm \varepsilon)),\end{aligned} \tag{2.60}$$

since SRes_{k-1} and SRes_j have no roots in $[a-\varepsilon, a+\varepsilon] \setminus \{a\}$.

We will now examine the sign situation around SRes_{j-1}. If SRes_{j-1} is non-defective, i.e. $k = j-1$, then

$$\mathrm{MVar}([\mathrm{SRes}_{j-2}, \mathrm{SRes}_{j-1}, \mathrm{SRes}_j], a) = \mathrm{MVar}([\mathrm{SRes}_{j-2}, \mathrm{SRes}_{j-1}, \mathrm{SRes}_j], a \pm \varepsilon) = 1, \tag{2.61}$$

since $\mathrm{SRes}_{j-2}(a)\,\mathrm{SRes}_j(a) < 0$. If SRes_{j-1} is defective of degree k, then

$$\begin{aligned}&\mathrm{MVar}([\mathrm{SRes}_{k-1}, \mathrm{SRes}_k, \mathrm{SRes}_{j-1}, \mathrm{SRes}_j], a)\\ &\quad= \mathrm{MVar}([\mathrm{SRes}_{k-1}, \mathrm{SRes}_k, \mathrm{SRes}_{j-1}, \mathrm{SRes}_j], a \pm \varepsilon)\\ &\quad= \begin{cases} 2 & \mathrm{SRes}_j(a)\,\mathrm{SRes}_{k-1}(a) > 0,\\ 1 & \mathrm{SRes}_j(a)\,\mathrm{SRes}_{k-1}(a) < 0.\end{cases}\end{aligned} \tag{2.62}$$

Note that SRes_{j-1} and SRes_k are proportional and therefore zero at a. At $a \pm \varepsilon$, the proportionality also gives

$$\mathrm{sign}(\overline{\mathrm{sres}}_{j-1}\,\mathrm{sres}_k) = \mathrm{sign}(\mathrm{SRes}_{j-1}(a \pm \varepsilon)\,\mathrm{SRes}_k(a \pm \varepsilon)) \tag{2.63}$$

at a	at $a \pm \varepsilon$	MVar
$[+, 0, 0, +]$	$[+, +, -, +], [+, -, +, +]$	2
$[+, 0, 0, -]$	$[+, +, +, -], [+, -, -, -]$	1
$[-, 0, 0, +]$	$[-, +, +, +], [-, -, -, +]$	1
$[-, 0, 0, -]$	$[-, +, -, -], [-, -, +, -]$	2

Table 2.1.: Possible signs occurring in Equation (2.62).

and therefore

$$\operatorname{sign}(\mathrm{SRes}_{j-1}(a \pm \varepsilon)\,\mathrm{SRes}_k(a \pm \varepsilon)) = -\operatorname{sign}(\mathrm{SRes}_{k-1}(a \pm \varepsilon)\,\mathrm{SRes}_j(a \pm \varepsilon)). \tag{2.64}$$

Now, the relations between the $[\mathrm{SRes}_{k-1}, \mathrm{SRes}_k, \mathrm{SRes}_{j-1}, \mathrm{SRes}_j]$ at a and $a \pm \varepsilon$ are clear and Equation (2.62) (and with it the proposition) follows by examining MVar for all possible sign combinations (see Table 2.1). ☐

Lemma 2.4.11. *Let $a \in \mathbb{R}$ be a simple real root of $A \in \mathbb{R}[x]$ and $\varepsilon > 0$, such that $[a-\varepsilon, a+\varepsilon]\setminus\{a\}$ does not contain roots of polynomials in $\mathcal{S} = [\mathrm{SRes}_0(A), \ldots, \mathrm{SRes}_n(A)]$. Further denote by $\mathcal{S}_\pm = [\mathrm{SRes}_0(A), \ldots, \mathrm{SRes}_{n-1}(A), \pm\mathrm{SRes}_{n-1}(A)]$ sequences derived from $\mathcal{S}$ by replacing the $(n+1)$-th element by $\pm\mathrm{SRes}_{n-1}(A)$. Then it holds that*

$$\mathrm{MVar}(\mathcal{S}, a \pm \varepsilon) = \mathrm{MVar}(\mathcal{S}_\pm, a). \tag{2.65}$$

Proof. First, note that $\mathrm{SRes}_n(A) = A$ and $\mathrm{SRes}_{n-1}(A) = A'$. Since a is a simple root of A, we have $A'(a) \neq 0$. If $A'(a) > 0$, then A is monotone increasing at a and $\operatorname{sign}(A(a \pm \varepsilon)) = \pm\operatorname{sign}(A'(a))$. If $A'(a) < 0$, then A is monotone decreasing at a and $\operatorname{sign}(A(a \pm \varepsilon)) = \pm\operatorname{sign}(A'(a))$, too. Thus,

$$\mathrm{MVar}([\mathrm{SRes}_{n-1}, \mathrm{SRes}_n], a \pm \varepsilon) = \mathrm{MVar}([\mathrm{SRes}_{n-1}, \pm\mathrm{SRes}_{n-1}], a). \tag{2.66}$$

The equality for the remaining sequence is evident by Proposition 2.4.10. ☐

As soon as $\mathrm{SRes}_n(A)(a) = 0$ and $\mathrm{SRes}_{n-1}(A)(a) \neq 0$ is detected during the computation, the element SRes_n can be replaced by $+\mathrm{SRes}_{n-1}$ or $-\mathrm{SRes}_{n-1}$, depending on whether the interval endpoint a is included in the interval or not. Using Lemma 2.4.11, we are able to count the number of real roots of a squarefree polynomial $A \in \mathbb{R}[x]$ on any open, half-open or closed interval of $\mathbb{R}$. Note that it is easy to include the interval endpoints $\pm\infty$ by employing that $\lim_{t\to\pm\infty} \operatorname{sign}(\mathrm{SRes}_j(t)) = (\pm 1)^j \,\mathrm{sres}_j$.

2.4.1.1. Evaluation of signed resultant sequences

To count the number of real roots of a polynomials, we need to determine the signs of the signed subresultants. We have already seen in Corollary 2.3.5 that $\deg(\mathrm{SRes}_j(A, B)) \leq j$. Evaluating each of the $\mathrm{SRes}_j(A, A')(a)$ directly would result in $\mathcal{O}(\deg(A)^2)$ arithmetic operations in the coefficient ring. Since this evaluation is likely to be performed at several positions along the real line, it is desirable to reduce the cost of a single evaluation.

As a consequence of Corollary 2.3.13, the subresultant polynomials satisfy the division relation

$$\mathrm{SRes}_{k-1} = -\frac{\mathrm{sres}_k\,\overline{\mathrm{sres}}_{j-1}\,\mathrm{SRes}_j - \mathrm{SResQ}_{k-1}\,\mathrm{SRes}_{j-1}}{\mathrm{sres}_j^2}. \tag{2.67}$$

for $\deg(\mathrm{SRes}_j) = j$ and $\deg(\mathrm{SRes}_{j-1}) = k$. If $\mathrm{SResQ}_{k-1}(a)$, all sres_i, $\overline{\mathrm{sres}}_i$ and two elements of $\{\mathrm{SRes}_{k-1}(a), \mathrm{SRes}_{j-1}(a), \mathrm{SRes}_j(a)\}$ are known, it is easy to determine the value of the third element using the above formula. Instead of storing the elements of the PRS, we store the quotients SResQ_i, the principal and leading signed subresultants coefficients and $\{\mathrm{SRes}_n, \mathrm{SRes}_{n-1}, \mathrm{SRes}_l\}$, such that SRes_l is the last non-zero element of the signed subresultant sequence, i.e. $\mathrm{SRes}_k \equiv 0$ for all $k < l$. We call this new sequence a *polynomial quotient sequence (PQS).*

If $j - 1 = l$ in Theorem 2.3.12, then $\mathrm{SRes}_{k-1} \equiv 0$ and Equation (2.67) simplifies to $\mathrm{sres}_k \overline{\mathrm{sres}}_{j-1} \mathrm{SRes}_j = \mathrm{SResQ}_{k-1} \mathrm{SRes}_{j-1}$. This allows to evaluate all signed subresultants using the quotient sequence, either in decreasing order using SRes_n and SRes_{n-1} or in increasing order starting with SRes_l.

We will now estimate the complexity of the evaluation of the signed subresultants using the quotients.

Lemma 2.4.12. *The sum of the degrees in the sequence of quotients is bounded by* $\deg(A)$*.*

Proof. Let $\mathcal{S} = [\mathrm{SRes}_{i_0}, \dots, \mathrm{SRes}_{i_k}]$, $i_0 < \cdots < i_k$, be the sequence of signed subresultants which are computed using the division rule in Equation (2.67) and $\mathrm{SRes}_{i_{k+1}} = \mathrm{SRes}_{n-1}$ and $\mathrm{SRes}_{i_{k+2}} = \mathrm{SRes}_n$. Note that in this case $\mathrm{SRes}_{i_0} = \mathrm{SRes}_{l-1} \equiv 0$ and $\mathrm{SRes}_{i_1} = \mathrm{SRes}_l \not\equiv 0$. The degree of SResQ_{i_j}, $0 \le j \le k$, is exactly $\deg(\mathrm{SRes}_{i_{j+2}}) - \deg(\mathrm{SRes}_{i_{j+1}})$. Finally,

$$\begin{aligned}
\sum_{i=0}^{k} \deg(\mathrm{SResQ}_{i_j}) &= \sum_{i=0}^{k} \Big(\deg(\mathrm{SRes}_{i_{j+2}}) - \deg(\mathrm{SRes}_{i_{j+1}}) \Big) \\
&= \deg(\mathrm{SRes}_{i_{k+2}}) - \deg(\mathrm{SRes}_{i_1}) \qquad (2.68) \\
&= \deg(A) - \deg(\mathrm{SRes}_{i_1}) \le \deg(A)
\end{aligned}$$

□

Lemma 2.4.13. *Evaluating all signed subresultants of polynomials* $A, A' \in \mathbb{Z}[x]$ *with magnitude* (τ, n) *at* $a \in \mathbb{Z}$ *with* $\mathrm{bit}(a) = \nu$ *using the sequence of quotients needs at most* $\mathcal{O}(n\mathcal{M}(n(\tau + \mathrm{bit}(n)) + \nu))$ *bit operations.*

Proof. It is clear from Lemma 2.4.12 that we need $\mathcal{O}(n)$ arithmetic operations in $\mathbb{Z}$ involving the coefficients of the quotients and a. The size of the subresultant and the quotient coefficients is bounded by $\mathcal{O}(n(\tau + \mathrm{bit}(n)))$ according to Lemma 2.3.27 and Corollary 2.3.30. Evaluating a subresultant directly using e.g. Horner's scheme yields intermediate (and final) results of size at most $\mathcal{O}(n(\tau + \mathrm{bit}(n)) + n + \nu) = \mathcal{O}(n(\tau + \mathrm{bit}(n)) + \nu)$. Due to Equation (2.67), the same must be true if the quotients are used since the results are identical. □

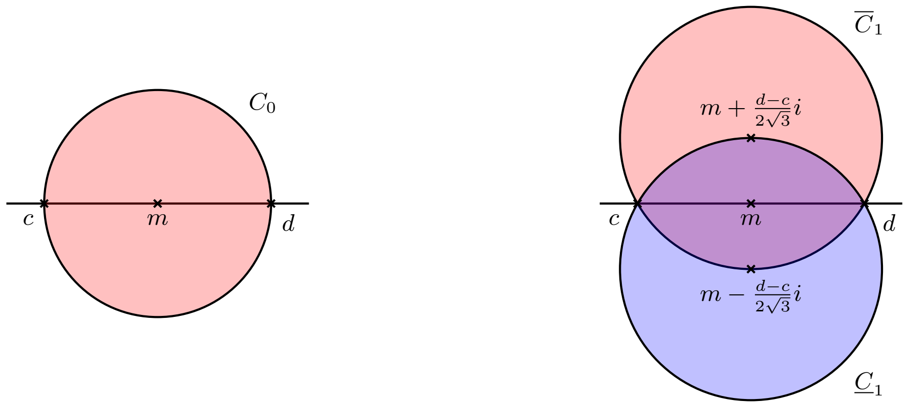
2.4.2. Descartes' rule of signs

Theorem 2.4.14 (Descartes' rule of signs). *Let* $A = \sum_{i=0}^{n} a_i x^i \in \mathbb{R}[x]$ *be a polynomial with real coefficients. Then*

$$\#\mathrm{rrm}(A, (0, \infty)) = \mathrm{Var}(a_0, \dots, a_n) - 2m, \qquad (2.69)$$

for some $m \in \mathbb{N} \cup \{0\}$*, i.e.* $\mathrm{Var}(a_0, \dots, a_n)$ *exceeds the number of positive real roots of* A *(counted with multiplicities) by a nonnegative even integer.*

Proof. See [Obr63, §13]. □



(a) The Obreshkov circle C_0.

(b) The Obreshkov circles $\overline{C}_1$ and $\underline{C}_1$.

Figure 2.2.: Given the interval (c, d) with midpoint $m = (c + d)/2$, consider open discs in the complex plane bounded by the circles C_0, $\overline{C}_1$ and $\underline{C}_1$. If the polynomial $A \in \mathbb{R}[x]$ has no (complex) root in C_0 then $\mathrm{Var}(A, (c, d)) = 0$. If A has exactly one root in $\overline{C}_1 \cup \underline{C}_1$ then $\mathrm{Var}(A, (c, d)) = 1$. Note that if $a + ib \in (\overline{C}_1 \cup \underline{C}_1) \setminus \mathbb{R}$ is a root of A, then $a - ib$ is a also root of A since $A \in \mathbb{R}[x]$. Therefore, $\mathrm{Var}(A, (c, d)) \geq 2$ in this case.

Clearly, if $\mathrm{Var}(a_0, \dots, a_n)$ is odd, there is at least one positive real root. If $\mathrm{Var}(a_0, \dots, a_n) = 0$ or $\mathrm{Var}(a_0, \dots, a_n) = 1$, we even know that the exact number of positive real roots is zero or one. The converse is not true in general but the following results due to Obreshkov give necessary conditions which have to be satisfied as illustrated in Figure 2.2. In order to bound the number of real roots in an arbitrary interval, we use the following transformation.

Proposition 2.4.15. *Let $A \in \mathbb{R}[x]$ be a polynomial with real coefficients and*

$$T_{c,d}(x) = (x + 1)^{\deg(A)} A \left(\frac{dx + c}{x + 1} \right) \tag{2.70}$$

for $c, d \in \mathbb{R}$. Then $\#\mathrm{rrm}(T_{c,d}, (0, \infty)) = \#\mathrm{rrm}(A, (c, d))$.

Proof. The continuous mapping $x \mapsto \frac{dx+c}{x+1}$ maps the interval $(0, \infty)$ in a bijective manner to (c, d). Therefore, the rational function $A(\frac{dx+c}{x+1})$ has as many positive roots as the polynomial $A(x)$ has in (c, d). The factor $(x + 1)^{\deg(A)}$ cancels out the denominators so that $T_{c,d}(x)$ is a polynomial. Since $(x + 1)$ has no positive root, the result follows. ☐

Notation 2.4.16. *Let $A = \sum_{i=0}^{n} a_i x^i \in \mathbb{R}[x]$ be a real polynomial. Then we write* $\mathrm{Var}(A) = \mathrm{Var}([a_0, \dots, a_n])$ *and* $\mathrm{Var}(A, (c, d)) = \mathrm{Var}(T_{c,d})$.

Theorem 2.4.17. *Let $A \in \mathbb{R}[x]$ and $I = (c, d) \in \mathbb{R}$ an interval with midpoint $m = (c + d)/2$.*

One circle theorem *If the open disc in the complex plane, bounded by the circle C_0 centered at m and passing through c and d, contains no complex root of A, then* $\mathrm{Var}(A, I) = 0$.

Two circle theorem *If the union of open discs in the complex plane, bounded by the circles $\overline{C}_1$ and $\underline{C}_1$ centered at $m \pm \frac{d-c}{2\sqrt{3}}i$ and passing through c and d, contains exactly one simple complex root of A, then* $\mathrm{Var}(A, I) = 1$.

Proof. See [Ost50; Obr25]. ☐

Now, we are able to isolate the real roots of a squarefree real polynomial A in an interval (c, d) by repeated bisection. Usually, the polynomial $T_{c,d}(x)$ is not computed by directly transforming A in each step. Instead, one scales and shifts A so that the interval of interest is $(0, 1)$ which can be mapped to $(0, \infty)$ easily. The cost of all further transformations during the bisection is dominated by the so-called *Taylor shift* $x \mapsto x + 1$. The classical algorithm for the Taylor shift finishes in $\mathcal{T}(\tau, n) = \mathcal{O}(n^2\mathcal{M}(\tau + n))$ [GG97; JKR05]. More sophisticated variants (also known as *asymptotically fast Taylor shifts*) finish in $\mathcal{T}(\tau, n) = \mathcal{O}(\mathcal{M}(n^2 + n\tau) \log n)$ [GG97]. The overall termination of the isolation algorithm is ensured by Theorem 2.4.17 since one of the base cases $\mathrm{Var} = 0$ and $\mathrm{Var} = 1$ occurs once the considered interval is small enough.

Several new bounds on the size of the recursion tree of the bisection process have been established. This led to bounds on the complexity of the isolation of all real roots of integer polynomials using Descartes' rule of signs. [EMT08; ESY06] state a bound of $\mathcal{O}(n(\tau + \log n)\mathcal{T}(n^2(\tau + \mathrm{bit}(n)), n))$ for isolating the real roots of a polynomial $A \in \mathbb{Z}[x]$ of magnitude (τ, n). This also includes the cost for determining the squarefree part of A. Most recently, this bound has been improved in [Sag12] by employing the locally convergent Newton iteration in order to achieve a higher order of convergence:

Lemma 2.4.18 (Bit complexity of polynomial real root isolation). *Isolating the real roots of a polynomial* $F(x) \in \mathbb{Z}[x]$ *of magnitude* (τ, n) *needs* $\tilde{\mathcal{O}}(n^3\tau)$ *bit operations.*

Proof. See [Sag12]. □

Note that much more literature about various algorithmic aspects related to Descartes' rule of signs is available. See e.g. [CA76; Vin36; TE06] and others.

Despite the simplicity of Descartes' rule of signs one has to keep in mind that the termination of the algorithm can only be ensured for simple roots while Sturm's theorem disregards the multiplicities. This is of great importance if the coefficients of a polynomial are not known exactly but only approximately (with arbitrary precision), i.e. if they arise by specialization of a multivariate polynomial. If the gap structure of the signed subresultants is known in advance, we may still be able to determine $\#\mathrm{rr}(A, [c \pm \varepsilon_1, d \pm \varepsilon_2]) = \mathrm{MVar}(\mathcal{S}, c \pm \varepsilon_1) - \mathrm{MVar}(\mathcal{S}, d \pm \varepsilon_2)$ for appropriately chosen $\varepsilon_1, \varepsilon_2$ where $\mathcal{S}$ is the sequence of signed subresultants of A and A'. This seems to be more difficult for Descartes' rule of signs. To the author's knowledge, the only solution is to determine the squarefree part of the polynomial which is computationally intense since the coefficients are real algebraic numbers. However, the issue has been solved for the case of a single multiple root in [Ker06] with the aid of signed subresultants.

2.5. Numerical filtering

Implementations of exact algorithms are often considered to be slow in practice. This is especially true if one has to deal with real (algebraic) numbers but it also happens for computations over $\mathbb{Z}$ and $\mathbb{Q}$. A valuable technique to attenuate this issue is to use numerical filtering based on fixed precision floating point computations where appropriate. Clearly, the reduced precision can lead to wrong results. A possible solution is to use interval arithmetic to validate the outcomes of a computation. In case of insufficient quality of the result, the precision is increased or the algorithm switches to exact arithmetic. In this section, we will briefly introduce interval arithmetic and demonstrate how to choose the precision for the interval computations.

2.5.1. Interval arithmetic

The purpose of interval arithmetic is to perform self-validating numerical computations. Instead of using exact numbers, one uses intervals that enclose the exact results. The interval endpoints are usually fixed precision floating point numbers, but rational numbers are also possible. Note that the interval representation of intermediate results is often sufficient to compute exact final results like in the very important case of sign determination, e.g. $\operatorname{sign}([1,2]) = +$.

In order to employ interval arithmetic algorithmically, it is necessary to define the basic arithmetic operations.

Definition 2.5.1. *Given two closed intervals* $[a,b],[c,d] \subseteq \mathbb{R}$ *we define*

$$[a,b]+[c,d] = [a+c, b+d], \tag{2.71}$$

$$[a,b]-[c,d] = [a-d, b-c], \tag{2.72}$$

$$[a,b] \cdot [c,d] = [\min(ac,ad,bc,bd), \max(ac,ad,bc,bd)], \tag{2.73}$$

$$[a,b] \div [c,d] = \left[\min\left(\tfrac{a}{c},\tfrac{a}{d},\tfrac{b}{c},\tfrac{b}{d}\right), \max\left(\tfrac{a}{c},\tfrac{a}{d},\tfrac{b}{c},\tfrac{b}{d}\right)\right], \tag{2.74}$$

where $0 \notin [c,d]$ *for the division.*

It is easy to see that the intervals on the right-hand side include all possible outcomes of arithmetic operations on numbers from the intervals on the left. Note that the addition and multiplication operations are commutative, associative, but not distributive even if exact representations for the interval endpoints are used. Instead, they are what is called sub-distributive: For intervals $A, B, C \subset \mathbb{R}$ it holds that

$$A \cdot (B+C) \subseteq A \cdot B + A \cdot C. \tag{2.75}$$

This also illustrates the so-called dependency problem of interval arithmetic. On the left-hand side of Equation (2.75), the multiplication with A is performed only once while it is done twice on the right-hand side. In practice, this increases the overestimation of the range of a function. Note that rounding errors also contribute significantly when working with fixed precision interval endpoints. It is important to take this into consideration when designing algorithms based on interval arithmetic.

Definition 2.5.1 can easily be extended to allow for open and half-open intervals and to cover elementary functions. In this treatment, we refrain from further discussing the various properties and extensions of interval arithmetic such as affine arithmetic and refer the interested reader to [Moo79; Kul08].

2.5.2. Choice of precision

There are various strategies for adjusting the precision of floating point interval endpoints. The following works well if the result is in $\mathbb{Z}$ or $\mathbb{Q}$ and if we can bound its size. We start with a low floating point precision, e.g. the size of a machine word, and then double the precision each time the numerical filtering fails. As soon as we reach a mantissa length that is approximately equal to the worst case bound on the result, we switch to exact computations in $\mathbb{Z}$ respectively $\mathbb{Q}$. As we will see next, this filtering strategy does not add significant overhead to the asymptotic cost of an algorithm even if we always fall back to exact arithmetic. Hence, the cost only increases by a constant factor.

Proposition 2.5.2. *Let $\mathcal{C}(\tau) : \mathbb{R} \to \mathbb{R}$, $\mathcal{C}(\tau) \in \Omega(\tau)$, be a monotonically increasing function. Then there exists a $\tau_0 > 0$ such that*

$$\mathcal{C}(\tau_1) + \mathcal{C}(\tau_2) \le \mathcal{C}(\tau_1 + \tau_2) \tag{2.76}$$

for all $\tau_1, \tau_2 > \tau_0$.

Proof. For τ larger than some τ_0, we can write $\mathcal{C}(\tau) = \tau\tilde{\mathcal{C}}(\tau)$ with a monotonically increasing function $\tilde{\mathcal{C}}$. Therefore,

$$\begin{aligned} \mathcal{C}(\tau_1) + \mathcal{C}(\tau_2) &= \tau_1\tilde{\mathcal{C}}(\tau_1) + \tau_2\tilde{\mathcal{C}}(\tau_2) \le \tau_1\tilde{\mathcal{C}}(\tau_1 + \tau_2) + \tau_2\tilde{\mathcal{C}}(\tau_1 + \tau_2) && (2.77)\\ &= (\tau_1 + \tau_2)\tilde{\mathcal{C}}(\tau_1 + \tau_2) = \mathcal{C}(\tau_1 + \tau_2) && (2.78) \end{aligned}$$

for all $\tau_1, \tau_2 > \tau_0$. □

Lemma 2.5.3. *Let $\mathcal{C}(\tau) : \mathbb{R} \to \mathbb{R}$, $\mathcal{C}(\tau) \in \Omega(\tau) \cap \mathcal{O}(\tau^c)$ for some constant c, be a monotonically increasing function. Then*

$$\sum_{i=0}^{\lceil \log_2 \tau \rceil} \mathcal{C}(2^i) \in \mathcal{O}(\mathcal{C}(\tau)). \tag{2.79}$$

Proof. Applying Proposition 2.5.2 yields

$$\sum_{i=0}^{\lceil \log_2 \tau \rceil} \mathcal{C}(2^i) \le \sum_{i=0}^{\lceil \log_2 \tau_0 \rceil} \mathcal{C}(2^i) + \mathcal{C}\left(\sum_{i=\lceil \log_2 \tau_0 \rceil + 1}^{\lceil \log_2 \tau \rceil} 2^i \right) \in \mathcal{O}\left(\mathcal{C}\left(\sum_{i=\lceil \log_2 \tau_0 \rceil}^{\lceil \log_2 \tau \rceil} 2^i \right) \right) = \mathcal{O}(\mathcal{C}(2\tau)). \tag{2.80}$$

Due to $\mathcal{C}(\tau) \in \mathcal{O}(\tau^c)$ it follows that $\mathcal{O}(\mathcal{C}(2\tau)) = \mathcal{O}(\mathcal{C}(\tau))$. □

We can assume that the bound $\mathcal{M}(\tau)$ for multiplying two τ bit numbers is such a monotonically increasing function. Clearly, $\mathcal{M}(\tau) \in \Omega(\tau)$ and $\mathcal{M}(\tau) \in \mathcal{O}(\tau^2)$. Furthermore, addition, subtraction and integral division are in $\mathcal{O}(\mathcal{M}(\tau))$ (cf. Table 2.3). This allows us to apply the above result to all arithmetic operations we need. At least from the asymptotic point of view, no cost is added by the numerical filtering if we repeatedly double the precision in case of a failure and stop the iteration once the precision reaches the bound τ on the size of the exact result.

2.6. Summary of algorithm complexities

In the following, we summarize the asymptotic complexities of some important algorithms on integral numbers, matrices and polynomials. The size of the input to the algorithms is as follows: $a, b, c, d \in \mathbb{Z}$ with $\operatorname{bit}(a), \operatorname{bit}(b) \le \tau$ and $\operatorname{bit}(c), \operatorname{bit}(d) \le \sigma$; $M \in \mathbb{Z}^{n \times n}$ with entries of bitsize at most τ; $A, B \in \mathbb{Z}[x]$ of magnitude (τ, n), $F, G \in \mathbb{Z}[y_1, \ldots, y_k][x]$ of magnitude (τ, d, n). Additionally, we use the shorthand $\nu = \operatorname{bit}(n)$.

	Algorithm	Bit complexity		Reference
$\mathbb{Z}$	Multiplication $c \cdot d$	$\mathcal{M}(\tau)$	$\tilde{\mathcal{O}}(\tau)$	
	Naive	$\mathcal{O}(\tau^2)$	$\tilde{\mathcal{O}}(\tau^2)$	
	Toom-Cook	$\mathcal{O}(\tau^{1+\varepsilon})$	$\tilde{\mathcal{O}}(n^{1+\varepsilon})$	[Coo66]
	Schönhage-Strassen	$\mathcal{O}(\tau \log \tau \log \log \tau)$	$\tilde{\mathcal{O}}(\tau)$	[SS71]
	Fürer	$\mathcal{O}(\tau \log \tau \, 2^{\mathcal{O}(\log^* \tau)})$	$\tilde{\mathcal{O}}(\tau)$	[Für07]
	Division with remainder $c = d \cdot \mathrm{quot}(c, d) + \mathrm{rem}(c, d)$			
	Naive	$\mathcal{O}(\tau^2)$	$\tilde{\mathcal{O}}(\tau^2)$	
	Newton-Raphson	$\mathcal{O}(\mathcal{M}(\tau))$	$\tilde{\mathcal{O}}(\tau)$	[BZ10]
$\mathbb{Z}^{n \times n}$	Matrix determinant $\det(M)$			
	Dogson-Jordan-Bareiss	$\mathcal{O}(n^3 \mathcal{M}(\tau n + \nu))$	$\tilde{\mathcal{O}}(\tau n^4)$	[Dod66; BPR03]
$\mathbb{Z}[x]$	Multiplication $A \cdot B$			
	Mapping to integers	$\mathcal{O}(\mathcal{M}(\tau n))$	$\tilde{\mathcal{O}}(\tau n)$	[Sch82]
	Evaluation $d^n A(\frac{c}{d})$			
	Horner / Naive	$\mathcal{O}(n \mathcal{M}(\tau + n\sigma))$	$\tilde{\mathcal{O}}(n^2 \sigma + n\tau)$	[BPR03]
	Translation $b^n A(x - \frac{a}{b})$			
	Naive	$\mathcal{O}(n^2 \mathcal{M}(n\tau))$	$\tilde{\mathcal{O}}(\tau n^3)$	[BPR03]
	Asymptotically fast	$\mathcal{O}(\mathcal{M}(n^2 + n\tau) \log n)$	$\tilde{\mathcal{O}}(n^2 + n\tau)$	[GG97]
	Signed subresultant sequence $\mathrm{SRes}_n(A, B), \ldots, \mathrm{SRes}_0(A, B)$			
	Matrix determinants	$\mathcal{O}(n^4 \mathcal{M}((\tau + \nu)n))$	$\tilde{\mathcal{O}}(\tau n^5)$	
	SRes Structure Thm.	$\mathcal{O}(n^2 \mathcal{M}((\tau + \nu)n))$	$\tilde{\mathcal{O}}(\tau n^3)$	[BPR03]
	Resultant $\mathrm{Res}(A, B)$, greatest common divisor $\gcd(A, B)$, GCD-free part† $A/\gcd(A, B)$ and signed subresultant quotients $\mathrm{SResQ}_n(A, B), \ldots, \mathrm{SResQ}_0(A, B)$			
	SRes Structure Thm.	$\mathcal{O}(n^2 \mathcal{M}((\tau + \nu)n))$	$\tilde{\mathcal{O}}(\tau n^3)$	[BPR03]
	Half-GCD idea	$\mathcal{O}(n \log n \, \mathcal{M}((\tau + \nu)n))$	$\tilde{\mathcal{O}}(\tau n^2)$	[Rei97; LR01]
$\mathbb{Z}[y_1, \ldots, y_k][x]$	Multiplication $F \cdot G$			
	Mapping to $\mathbb{Z}[x]$	$\mathcal{O}(\mathcal{M}(\tau n d^k))$	$\tilde{\mathcal{O}}(\tau n d^k)$	[Pan94]
	Signed subresultant sequence $\mathrm{SRes}_n(F, G), \ldots, \mathrm{SRes}_0(F, G)$			
	Matrix determinants	$\mathcal{O}(n^4 \mathcal{M}((\tau + \nu)n^{k+1} d^k))$	$\tilde{\mathcal{O}}(\tau n^{k+5} d^k)$	[Dod66]
	SRes Structure Thm.	$\mathcal{O}(n^2 \mathcal{M}((\tau + \nu)n^{k+1} d^k))$	$\tilde{\mathcal{O}}(\tau n^{k+3} d^k)$	[Rei97; BPR03]
	Resultant $\mathrm{Res}(F, G)$, greatest common divisor $\gcd(F, G)$ and GCD-free part $F/\gcd(F, G)$			
	SRes Structure Thm.	$\mathcal{O}(n^2 M((\tau + \nu)n^{k+1} d^k))$	$\tilde{\mathcal{O}}(\tau n^{k+3} d^k)$	[Rei97; BPR03]
	Half-GCD idea	$\mathcal{O}(n \log n \, \mathcal{M}((\tau + \nu)n^{k+1} d^k))$	$\tilde{\mathcal{O}}(\tau n^{k+2} d^k)$	[Rei97]

---

† For $B = A'$ this becomes the squarefree part.

Table 2.3.: Summary of algorithm complexities.

3. Exact rasterization of real algebraic plane curves

3.1. Motivation

Rendering a curve defined implicitly by an equation $F(x, y) = 0$ is a frequently occurring task in computer graphics and computer aided design. Curves of this type often arise from projections of space curves such as silhouette curves or intersection curves of surfaces. An example is given in Figure 3.1. Visualizations of implicit curves are also of great use in many fields of mathematics as they can assist the process of understanding certain mathematical concepts. Consequently, many computer algebra systems include routines for plotting implicit curves. In general, a global rational parametrization of an algebraic curve only exists if its genus is zero (see e.g. [SWP07] for an in-depth treatment of the subject). More sophisticated representations like formal power series and Puiseux series allow to parametrize any real algebraic curve locally. Unfortunately, such exact representations are hard to generate and to handle since they involve costly computations with algebraic numbers (see e.g. [Fis94, Chapter 6 and 7] for an introduction). Therefore, most visualization algorithms for implicit curves (including the approach presented here) operate directly on the implicit form.

In what follows, we solve the task of rasterizing a real algebraic plane curve $V_{\mathbb{R}}(F)$ defined by a polynomial $F \in \mathbb{Z}[x, y]$. Although many algorithms exist for rendering such a curve, only a few of them guarantee the correctness of the output. The term *correctness* is often interpreted as topological correctness, i.e. the graph induced by the rendering is isotopic to $V_{\mathbb{R}}(F)$. When rasterizations are considered, this definition is no longer applicable since the topology of the curve cannot be recognized below pixel level. We follow the definition of [Lab10b].

Notation 3.1.1. *A* pixel *with coordinates* $(x, y) \in \mathbb{Z}^2$ *is the subset* $[x, x + 1] \times [y, y + 1] \subset \mathbb{R}^2$. *We call* $(x, y) \in \mathbb{Z}^2$ *a* raster position *or* pixel coordinate *for* $(x_0, y_0) \in \mathbb{R}^2$*, if* $(x_0, y_0) \in [x, x + 1] \times [y, y + 1]$. *We may also speak of raster positions referring only to one of the two coordinates.*

Definition 3.1.2 (exact rasterization). *In an exact rasterization a pixel* P *is painted if and only if* $V(F) \cap P \neq \emptyset$.

In the above notion of correctness, it is important to note that adjacent pixels overlap at their boundary. This is one way to ensure that an algorithm is able to produce symmetric images of symmetric curves. It has to be remarked that other definitions of exact curve rasterizations may be valid, too, due to the gap between the exact mathematical definition of the curve based on the zero set of a continuous function and its mapping to a discrete raster display.

An example of a correctly rasterized curve is given in Figure 3.2. The figure also shows some commonly occurring visualization errors of inexact algorithms. Knowing the pixels where the curve has critical points provides a lot of useful information about the curve. Isolated points of the curve are covered as well as real components smaller than one pixel. Those parts of the curve are frequently overlooked by approaches that do not guarantee correctness (as illustrated

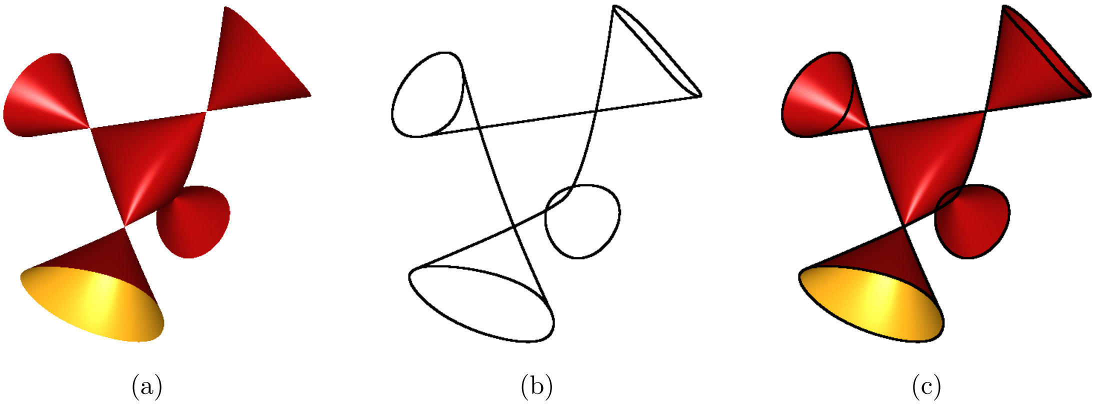

(a) (b) (c)

Figure 3.1.: The silhouette curve of a surface is the intersection of the surface and its polar surface with respect to the viewers position. In (a), a rendering of a Cayley surface is shown. The surface is clipped to a sphere. In (b), the projection of its silhouette to the plane is rendered. Note that the intersection of the Cayley surface and the clipping sphere have been included. Also, the silhouette curve has been clipped to exclude parts outside of the sphere. Finally, (c) shows an overlay of the surface rendering and its silhouette. Projections of silhouettes can greatly assist surface rendering as they partition the image plane into regions where the surface is smooth. In addition, the number of real points of the surface above a fiber in the direction of projection does not change within such a region.

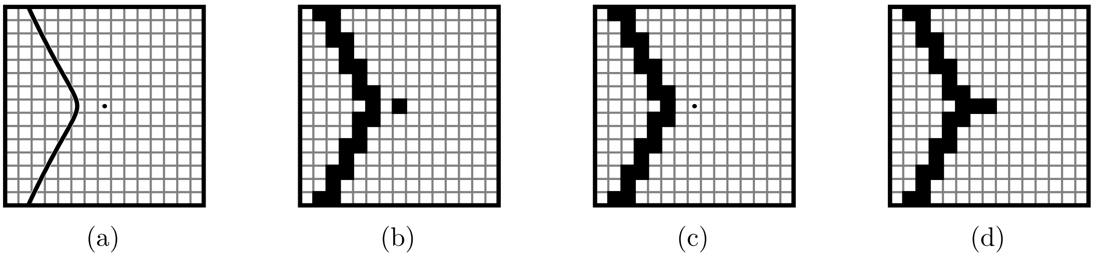

(a) (b) (c) (d)

Figure 3.2.: Correct and wrong rasterizations of the curve $y^2 + x^2(x+1) = 0$ (scaled and translated appropriately). (a) shows the grid of pixels and the curve to be rasterized. A correct rasterization with respect to Definition 3.1.2 is shown in (b) while the rasterizations in (c) and (d) feature missing respectively additional pixels, which are often observed in images rasterized with inexact methods. The above illustrations are inspired by very similar ones presented in [Lab10b].

in Figure 3.2). Additionally, the direction of the curve segments between the critical pixels is almost clear. We will see how the segments can be traced efficiently as soon as the raster positions of their start and end points are known.

A preliminary version of the new approach has been published in [Stu11]. Before outlining the method, we will discuss related work about implicit curve visualization. Although bivariate polynomial systems have to be solved during the preprocessing of the curve, the vast amount of literature related to this task will not be reviewed. The presented investigations only cover the actual rendering and utilize polynomial system solving only as a tool. Therefore, we restrict to the short overview given in Section 3.4.1.

3.2. Related work

The visualization of algebraic curves has been a field of active research for the last decades. Most algorithms are based on (real) root approximation resp. isolation, space covering techniques or continuation methods. Furthermore, exact methods often rely on a topological analysis of the curve.

An exact but rather costly algorithm yielding exact rasterizations is proposed in [Lab10b]. It can be summarized as follows.

Algorithm 3.1 (trivial exact plane curve rasterization). *Determine the raster position of all real solutions of* $F = \frac{\partial F}{\partial y} = 0$*, thus, of the critical points of the curve. Then, compute the intersections of* $V_{\mathbb{R}}(F)$ *with the boundary of each pixel.*

Here, the term “trivial” refers to the trivial approach of checking each pixel. In contrast, determining the raster positions of the critical points $F = \frac{\partial F}{\partial y} = 0$ is more involved. Although not explicitly stated, the computation of the intersection of $V_{\mathbb{R}}(F)$ with the boundary of each pixel can be considered as a sampling technique combined with some form of real root isolation. Examining each pixel during the rasterization seems to be rather wasteful. Therefore, we may consider the following improved version.

Algorithm 3.2 (improved trivial exact plane curve rasterization). *Determine the raster position of all real solutions of* $F = \frac{\partial F}{\partial y} = 0$*, thus, of the critical points of the curve. Then, compute the raster positions of the intersections of* $V_{\mathbb{R}}(F)$ *with the boundary of each row and column of the image.*

A full implementation of the above algorithm is provided in this thesis and compared with the new method. See Sections 3.7.5 and 3.9. An approach similar to Algorithm 3.2 has been used in [Lip10]. Instead of trying to provide exact results its author aims for speed by implementing the real root isolation on graphics processing hardware. Hence, the method may overlook small components of the curve and is prone to numerical errors.

Space covering techniques recursively subdivide the initial region of interest into smaller cells until a desired precision is reached. Cells that do not contain the curve need to be discarded and this should happen as early as possible. One of the most commonly applied techniques is range estimation of F over an interval. In [FS96], affine arithmetic, a special type of interval arithmetic, is used for this task. [Dok+05] improves upon affine arithmetic by introducing the so-called recursive Taylor method for algebraic curves and surfaces. There, the range of F is estimated by bounding the range of a linear Taylor expansion of F. The approximation error is bounded by recursively applying the Taylor method at most $\deg(F)$ times. [LOF01] also uses interval arithmetic but applies a heuristic and adaptive criterion based on curvature estimation in order to stop the subdivision at a reasonable level. In [OF00], interval arithmetic is also used to visualize offsets to a curve. [Mar+02] gives a detailed survey on different forms of interval arithmetic and discusses their application to the problem of plane curve visualization. In [Tau94], the distance between the origin and the zeros of F is bounded from below in order to exclude some of the cells during the subdivision. The used test is a simplified version of [Tau93] that works equally well in the vicinity of singular points of the curve. All proposed tests only work as an exclusion test: $V_{\mathbb{R}}(F)$ does not intersect any of the discarded cells. However, the remaining cells do not necessarily contain any part of $V_{\mathbb{R}}(F)$. The quality of the approximation usually improves with the level of subdivision. An exact visualization is reached in the limit when the subdivision depth goes to infinity. In order to achieve a termination after a finite number of steps, the above methods would have to be combined with an inclusion test: If $V_{\mathbb{R}}(F)$ intersects a cell, then the inclusion test would confirm this intersection after a certain number of subdivisions. This is done in [PV04] in a completely numerical way for bounded, non-singular curves (and surfaces). In [Bur+08], this approach is extended to unbounded curves with isolated singularities. First, the singular points are computed using a homotopy method. Then, the non-singular cells are subdivided using interval arithmetic as an exclusion and evaluation bounds as an inclusion test. Using the intersections of the curve and the cells, they are able to provide an arbitrary close approximation of the curve by line segments. Unfortunately, no analysis of the worst case complexity is provided.

The class of continuation methods computes initial points on the curve and then gradually determines new points on the curve based on the already known ones. In most cases, the points on the curve are not represented exactly. A sufficiently good approximation is used instead. [Cha88] traces the curve using the change of sign of F when $V_{\mathbb{R}}(F)$ crosses a pixel boundary. Their algorithm can also produce correct results for some non algebraic curves but fails for challenging curves like those with densely packed curve segments. [Bre65; Pra85] rasterize lines and quadrics using a scheme of finite differences. This is extended to arbitrary real algebraic plane curves in [Hob90]. In order to deal with the problem of undersampling of the curve a theorem of Budan and Fourier is combined with a threshold. Unfortunately, the details for the computation of an appropriate threshold are only worked out for cubic curves. Other continuation methods are based on predictor and corrector steps. In [MY95; FYK97], the next point on the curve is predicted by moving a predefined step size into the tangent direction of the curve at the current point. The approximation (x', y') is then improved in the corrector step by applying a multivariate Newton iteration that utilizes the gradient of F at (x', y'). The iteration terminates as soon as the absolute or relative error is below some predefined value. The approach in [Baj+88] is similar but makes use of higher order approximations to the curve using local curvature estimation in addition to the tangent. [MG04] avoids calculating derivatives by using the so called angular false position method. It is an extension to the regula falsi method for approximating the zeros of $F(\alpha) = F(x(\alpha), y(\alpha))$, where $(x(\alpha), y(\alpha))$ is a parametrization of a specific circle. [EBS09] present an exact curve tracing that rasterizes individual curve segments. The direction of curve progression is computed by excluding directions using interval arithmetic at pixel boundaries combined with pixel subdivision. The segments are determined using a topological analysis of the curve. We will compare the herein presented algorithm to [EBS09] in Section 3.9.

Some authors also propose hybrid techniques. [RR05] applies interval arithmetic and subdivision first and switches to curve tracing as soon as a cell with a single smooth curve segment is detected. In [AM07], the plane is first divided into cells without singular points of the curve. This is either done numerically or using a rational univariate representation

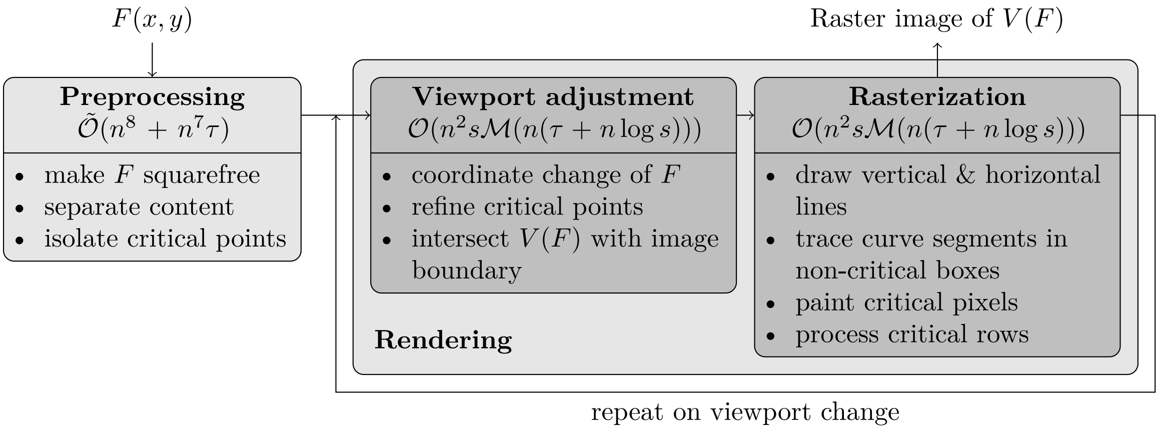

$F(x, y)$
Raster image of $V(F)$
Preprocessing
$\tilde{\mathcal{O}}(n^8 + n^7\tau)$
make F squarefree
separate content
isolate critical points
Viewport adjustment
$\mathcal{O}(n^2 s \mathcal{M}(n(\tau + n \log s)))$
coordinate change of F
refine critical points
intersect $V(F)$ with image boundary
Rasterization
$\mathcal{O}(n^2 s \mathcal{M}(n(\tau + n \log s)))$
draw vertical & horizontal lines
trace curve segments in non-critical boxes
paint critical pixels
process critical rows
Rendering
repeat on viewport change

Figure 3.3.: Outline of the exact rasterization algorithm for real algebraic plane curves. The shown complexities of the different stages assume an image of size $s \times s$ and the polynomial F to be of magnitude (τ, n). Note that the result of the preprocessing can be reused for different viewports.

of the critical points of F. Each cell is subdivided until all contained segments are x or y monotone. Then, the intersections of each cell boundary and the curve are computed using real root isolation. Intersections belonging to the same curve segment are connected by line segments by taking their tangent directions into account.

Most modern exact methods utilize numerical range estimation and continuation methods to improve speed. In contrast, the approach of [Arn83] relies almost entirely on symbolic calculations. There, a topology graph is computed that identifies each node, edge and cell using simultaneous polynomial inequalities. The graph already provides a coarse approximation to the curve. In order to refine the approximation, the edges of the graph are successively subdivided using computations with algebraic numbers.

As we will see next, the algorithm developed in this chapter can be classified as a continuation or curve tracing method. Once a point on each real component of the curve has been identified, such methods are considered to be quite efficient since they stay close to the curve instead of spending much computation time on regions that do not contain real zeros of F.

3.3. Algorithm outline

As shown in Figure 3.3, the new algorithm consists of a preprocessing stage and a rendering stage. In the first stage, which is detailed in Section 3.4, some symbolic computations are performed. F is made squarefree and its content is separated from the primitive part in order to cope with vertical and horizontal lines. Then, we determine the critical points of the curve, which also include the singularities. It is enough to know their position up to pixel level. They are used to divide the image plane into rows with critical points and blocks of rows without critical points. To further simplify the structure of the non-critical blocks, the rows where the curve intersects the image boundary are added to the rows of critical points.

In the second stage, a coordinate change is applied to the curve since the rendering is always performed in the viewport $[0, w] \times [0, h]$. In an implementation, it would also be possible to choose a different default viewport and skip the change of coordinates but this would require a more complicated notation. The actual rasterization in the critical and non-critical rows is described in detail in Section 3.5. The pixels of a critical row that contain a critical point are known from the preprocessing. As we will see in Section 3.5.2, the connections between the critical pixels in a critical row can be detected by a few simple tests.

Within the non-critical blocks, the curve has no self crossings and intersects the block boundary only at the bottom and top. The monotony of each curve segment can easily be calculated from these intersections. The non-critical blocks are now processed row by row. Two kinds of tests are used to detect the pixels occupied by the curve within the current row. In case of a good separation of the curve segments, a simple test for a sign change of $F(x, y)$ at the upper row boundary suffices. In difficult cases, real root counting is used to find the number of segments that are leaving the row at a certain pixel. Utilizing the occupied pixels of the previous row and the known monotony of the segments, the algorithm examines only those pixels containing the curve, but not a single empty pixel. Tests are only performed at the corners of the pixels. No pixel subdivision is necessary. An in-depth explanation is given in Section 3.5.1 and Section 3.5.2.

3.4. Precomputations

In order to prepare for the rasterization stage, we need to compute the squarefree part of F and the critical points of $V_{\mathbb{R}}(F)$. As we have already seen in Lemma 2.3.22, the first can be done by computing the subresultant cofactor $\mathrm{SResV}_{j-1}(F, F')$. Therefore, we will assume that F is squarefree. To avoid the occurrence of some special cases we will also remove vertical and horizontal segments from the curve and rasterize them separately.

Lemma 3.4.1. *If the line segment $\alpha \times [a, b]$ with $\alpha, a, b \in \mathbb{R}$, $a < b$, is contained in $V(F)$ then the whole vertical line $\alpha \times \mathbb{R}$ is part of $V(F)$.*

Proof. The polynomial $F_\alpha(y) = F(\alpha, y)$ has infinitely many roots in $[a, b]$. Due to the Fundamental Theorem of Algebra this is only possible if $F_\alpha(y) \equiv 0$ and thus $F(\alpha, y) = 0$ for all $y \in \mathbb{R}$. ☐

By a linear change of coordinates we can transform any direction into the vertical one. Therefore, Lemma 3.4.1 can be extended to line segments of any direction.

Lemma 3.4.2. *The vertical line $V(x - \alpha)$ is contained in $V(F)$ with $\deg(F) > 0$, if and only if $(x - \alpha)$ divides $\mathrm{cont}_y(F)$.*

Proof. If $x = \alpha$ is a vertical line in $V(F)$, then $F_\beta(x) = F(x, \beta)$ has a root at $x = \alpha$ for all $\beta \in \mathbb{R}$ and thus $(x-\alpha)$ divides every F_β and consequently F. Since $(x-\alpha)$ is a polynomial in x, it can only divide the coefficients of $F \in \mathbb{R}[x][y]$ and thus it divides $\mathrm{cont}_y(F)$. If $(x - \alpha)$ divides $\mathrm{cont}_y(F)$, then $x = \alpha$ is a root of every coefficient of $F \in \mathbb{R}[x][y]$ and hence $F(\alpha, y) \equiv 0$. ☐

To remove the vertical and horizontal lines from $V(F)$, we compute $\mathrm{cont}_y(F)$, $\mathrm{cont}_x(\mathrm{pp}_y(F))$ and proceed using the primitive part $\mathrm{pp}_x(\mathrm{pp}_y(F))$.

3.4.1. Isolation of critical points

Due to symmetry, we will focus on the computation of x-critical points of the curve $V_{\mathbb{R}}(F)$, i.e. of the real solutions of the bivariate polynomial system $F(x, y) = \frac{\partial F}{\partial y}(x, y) = 0$. Although we are only interested in the raster positions of the solutions, there is (to the knowledge of the author) no algorithm that is able to compute them without isolating the solutions of the system. For our purpose, any method that solves bivariate polynomial systems suffices, but (sub-)resultant based methods are preferred, because the subresultant sequence is reused frequently during the rasterization. Since the new algorithm currently builds upon the work of [EKW07], this method will be explained in more detail. The algorithm presented therein is based on projection techniques. The projections of the solutions of the system to the x axis are obtained as the roots of the polynomial resultant $\mathrm{Res}_y(F, \frac{\partial F}{\partial y}) \in \mathbb{Z}[x]$. The projected solutions are isolated using polynomial real root finding and then lifted to solutions in the plane. The latter relies on certified root isolation based of Descartes' rule of signs for polynomials with approximate coefficients (see Section 2.4.2 and [Eig08]), i.e. of arbitrary precise approximations of $F(\alpha, y)$ where $\alpha \in \mathbb{R}$ is a root of the resultant $\mathrm{Res}_y(F, \frac{\partial F}{\partial y})$. In order to cope with a single multiple root along the fiber $\alpha \times \mathbb{R}$, they take advantage of the GCD property of subresultants (see Theorem 2.3.19). If the direction of projection was not sufficiently generic (i.e. a fiber over a projection contains more than one critical point), the lifting phase fails and another direction of projection is chosen by shearing F and $\frac{\partial F}{\partial y}$. The process repeats until all solutions to the systems have been found. Note that the main interest in [EKW07] is the computation of the topology of an algebraic curve whereof the computation of critical points is just the first step. The additional topology information is not necessary for the rasterization.

The cost of the symbolic precomputation is not negligible, but efficient parallelization of the computation of bivariate resultants is possible in practice as shown in [Eme10a; Eme10b] and [SS12] (see also Chapter 6). The latter one supplies a complete GPU based implementation of the resultant computation by repeated polynomial division (of homomorphic univariate modular images of the bivariate input polynomials). This parallel resultant computation can easily be extended to output the subresultant polynomials or at least their degrees, which yield sufficient additional information for the rasterization algorithm presented below.

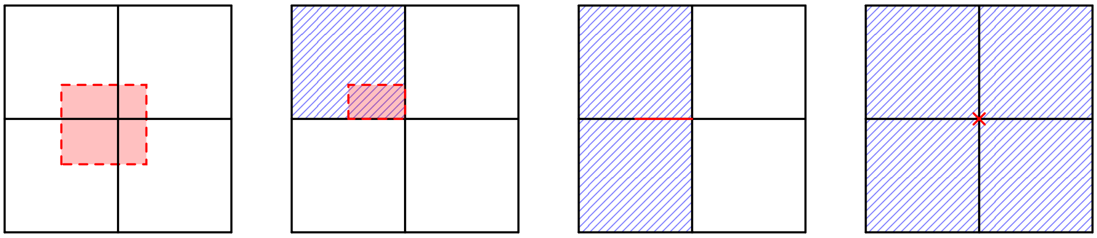
3.4.2. Viewport specific precomputations

The symbolic precomputation as well as the isolating box representation of the critical points can be reused for different viewports. In contrast, raster positions are only valid with respect to a specific viewport. Therefore, the following calculations have to be redone when the viewport changes. The complexity of this step is analyzed in Section 3.7.2.

3.4.2.1. Refinement of the critical points

For further processing, we need to know the positions of the critical points up to pixel level. For each critical point, we refine its isolating box up to a side length smaller than the side length of a pixel. If the isolating box B for a critical point $C \in \mathbb{R}^2$ is contained in a single pixel, the refinement is completed. Otherwise, B overlaps with at most 4 pixels and we split B along the sides of these pixels. It suffices to perform an additional refinement step for the isolating box B of C using the raster positions as the splitting points. This does not involve the computationally intense comparison of two algebraic numbers since the raster position is an integer coordinate. The final number of pixel assigned to C might be 1, 2 or 4, depending on whether C is located on a pixel boundary or corner as illustrated in Figure 3.4.

Several critical points may be contained in the same pixel. Once their raster positions are computed, there is no further need to distinguish between them. Therefore, we will frequently refer to critical pixels instead of critical points.



(a) Isolating box B refined below the size of a pixel.

(b) Critical point within pixel (not on boundary): 1 pixel.

(c) Critical point on pixel boundary (not on corner): 2 pixels.

(d) Critical point on pixel corner: 4 pixels.

Figure 3.4.: Possible configuration of pixels (up to symmetry) assigned to a critical point C represented by an isolating box B. Pixels are shown as □ (unassigned) and ▨ (assigned), B is shown as ⬚.

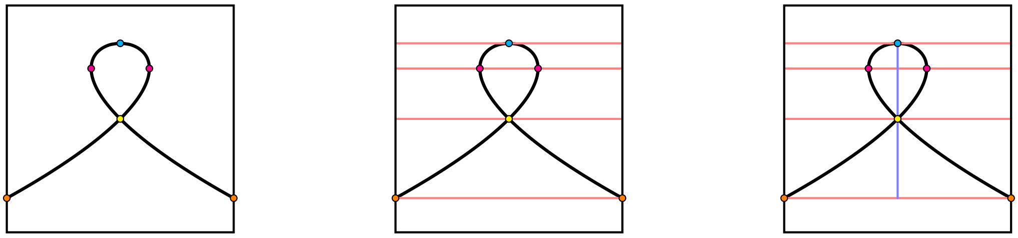
3.4.3. Intersections with the viewport boundary

The critical points allow us to partition the plane into subsets where the curve has no local minimum or maximum with respect to the coordinate directions. Since we want to render the curve within a specific viewport, we have to determine where the curve leaves the viewport to the left and right. This is computed by real root counting of F specialized at the viewport boundary, i.e. at $x \in \{0, w\}$ and $y \in \{0, h\}$, using the signed subresultants (see Section 2.4.1). Note that we do not need to isolate the roots. The real root counting can be combined with a bisections scheme that uses only pixel coordinates as bisections points. The subdivision stops as soon as it reaches the level of one pixel. No further refinement, e.g. down to the separation bound of the roots, is necessary. The intersections with the viewport boundary can also be treated as critical points since a curve segment starts resp. ends there with respect to the viewport. This unification also simplifies the rasterization process since the viewport becomes irrelevant once the start and end pixels of all curve segments are added to the set of critical pixels.

3.5. Rasterization

By means of the precomputation, we assured that the curve polynomial $F \in \mathbb{Z}[x, y]$ is squarefree and that $V(F)$ is free of vertical resp. horizontal lines. The raster positions of the critical points in the viewport have been computed and the raster positions of the intersections of the curve with the left and right viewport boundary have been added to the list of critical rows.

In what follows, we will focus on the rasterization of the curve in the non-critical slices of the image (Section 3.5.1) and then proceed with the critical rows (Section 3.5.2). The rasterization of vertical and horizontal lines is trivial. By Lemma 3.4.2, these lines correspond to the real roots of $\mathrm{cont}_x(F)$ resp. $\mathrm{cont}_y(F)$, thus of univariate polynomials, which have been divided out of F during the preprocessing. Determining their raster positions is equivalent to the computation of the raster positions of the curve intersections with the viewport boundary (see Section 3.4.3). Once the raster positions are known, all pixels in the respective columns or rows of the image are painted.



(a) Critical points and intersections with viewport boundary.

(b) Non-critical boxes.

(c) Monotone non-critical boxes.

Figure 3.5.: Decomposition of the viewport into monotone non-critical boxes using x-critical (●) and y-critical points (●) (which can be singular (●)) as well as the intersections of the curve with the viewport boundary (●). Note that the non-critical boxes (separated by –) are built on the basis of the y coordinates of the critical points. The additional decomposition into monotone boxes (by |) happens by comparing the monotony of adjacent curve segments within a non-critical box. Only raster positions are used for that purpose. Hence, the red and blue lines in the illustrations correspond to critical rows and columns in the raster image.

3.5.1. Non-critical slices and boxes

We will start with the definition of the subject and then have a closer look at some of its properties. See also Figure 3.5.

Definition 3.5.1. *A* slice $S = \mathbb{R} \times [b, t]$, $b, t \in \mathbb{R}$, *is called a* non-critical slice *with respect to a curve* $V(F)$, $F \in \mathbb{Z}[x, y]$, *if* S *does contain neither critical points nor horizontal asymptotes of* $V(F)$, *i.e.* $S \cap (V(F) \cap (V(\frac{\partial F}{\partial x}) \cup V(\frac{\partial F}{\partial y})) = S \cap V(\mathrm{lcoeff}_x(F)) = \emptyset$.

Lemma 3.5.2. *Let* $S = \mathbb{R} \times [b, t]$, $b, t \in \mathbb{R}$, *be a non-critical slice with respect to* $V(F)$ *and* $f_\alpha(x) = F(x, \alpha)$ *for any fiber* $\alpha \in [b, t]$. *Then*

1. $\deg(f_\alpha) = \deg_x(F)$,
2. *all real roots of* f_α *are simple,*
3. *the number of distinct real roots of* f_α *is independent of* α, *thus constant over* $[b, t]$.

Proof. 1. We have excluded horizontal asymptotes from non-critical slices. By Lemma 2.2.8, $\mathrm{lcoeff}_x(F)$ has no real root in $[b, t]$. 2. Assume $x = \beta$ would be multiple root of f_α, then (β, α) would be a critical point of $V(F)$. This violates the definition of a non-critical slice. 3. Due to 1, the number of complex roots of f_α is constant for $\alpha \in [b, t]$. Because complex roots appear in conjugate pairs for polynomials with real coefficients and the roots are continuous functions of the coefficients, a real root is only created if the imaginary part of two conjugate complex roots vanishes for some α. A double real root is formed, which voilates 2. □

The following is clear from the above lemma and from the definitions of non-critical slices and strictly monotone curve segments.

Corollary 3.5.3. *The number of strictly monotone curve segments of* $V(F)$ *in the non-critical slice* S *is equal to the number of real roots of* f_α *for any* $\alpha \in [c, d]$.

We will now introduce the notion of non-critical boxes.

Definition 3.5.4. *A subset* $B = [l, r] \times [b, t] \subset S$*,* $l, r \in \mathbb{R}$*, of a non-critical slice* $S = \mathbb{R} \times [b, t]$*,* $b, t \in \mathbb{R}$*, with respect to* $V(F)$ *is called a* non-critical box *if* $V(F)$ *does not intersect the left and right box boundary, i.e. more precisely* $((l \cup r) \times (b, t)) \cap V(F) = \emptyset$ *where* (b, t) *denotes the open interval between* b *and* t*. A non-critical box is called* monotone *if all curve segments in the box have the same monotony.*

In the presented algorithm, non-critical boxes are obtained by restricting the non-critical slices to the viewport $[0, w] \times [0, h]$ and dividing them along the raster positions of the real roots of $f_0(y) = F(0, y)$ and $f_w(y) = F(w, y)$. See Figure 3.5c. Note that the curve may cut a corner of a non-critical box.

To draw the curve within a non-critical box $B = [l, r] \times [b, t]$, we first determine the raster positions of the start and end points of all segments in B. This is done by using real root counting and pixel-based bisection applied to $f_b(x) = F(x, b)$ and $f_t(x) = F(x, t)$. Due to the definition of B, all real roots of these polynomials are simple. The i-th real root at $y = b$ and the i-th real root at $y = t$ belong to the same segment by Corollary 3.5.3.

Due to Corollary 2.2.7, the curve segments in a non-critical slice are strictly monotone but they do not necessarily have the *same* monotony. Comparing the raster positions of the start and end point of a segment suffices to identify its direction of monotony with respect to the rasterization. If the raster positions have the same x coordinate, any monotony can be assigned, because the slope of the curve is not recognizable at the chosen resolution so that the segment will be rasterized as a vertical line segment. Adjacent segments of the same monotony are grouped together into monotone non-critical boxes. W.l.o.g. we will only consider the rasterization of monotone non-critical boxes with monotone increasing segments since the coordinate transformation $y \mapsto -y$ suffices to turn the monotone decreasing segments into monotone increasing ones.

3.5.1.1. Tracing the curve segments in a monotone non-critical box

The curve segments are now traced from row to row in increasing y-direction. From the previous row (which may be critical, see Section 3.5.2), we know the painted pixels and how many curve segments pass through each of them into the current row. We will now detect the pixels where the segments leave the current row by examining the upper pixel boundaries. Due to the monotony of the segments, the curve will only proceed in positive x- and y-direction. Therefore, we do not need to look at pixels to the left (i.e. negative x-direction) of the occupied pixels in the previous row. Hence, we gradually step pixel by pixel to the right. Let now the y-coordinate of the upper boundary of the current row be $r_j \in \mathbb{Z}$ and $f_{r_j}(x) = F(x, r_j)$ be the polynomial associated with the line $y = r_j$. We will denote the x coordinates of the columns by $c_i, c_{i+1}, \ldots \in \mathbb{Z}$. See Figure 3.6 for an illustration of what follows.

At each pixel, we test how many segments pass through it into the row above. We distinguish two cases: Either, the segments are well separated and we know that exactly one segment is contained in the current pixel, or the segments are densely packed and the pixel contains more than one segment.

First, we have to remark that if a segment passes into the row at the point $(c_i, r_{j-1}) \in \mathbb{Z}^2$, it will not escape through the point $(c_i, r_j) \in \mathbb{Z}^2$, because all segments are strictly monotone and vertical lines have been removed. Note that $f_{r_j}(c_i) = 0$ means that a segment to the left of the current one leaves the current row in (c_i, r_j), i.e. we have the case of densely packed segments and cannot apply the following rule. If only one segment proceeds through the current pixel $[c_i, c_{i+1}] \times [r_{j-1}, r_j]$, then $f_{r_j}(x)$ has at most one simple real root in $(c_i, c_{i+1}]$. Due to the

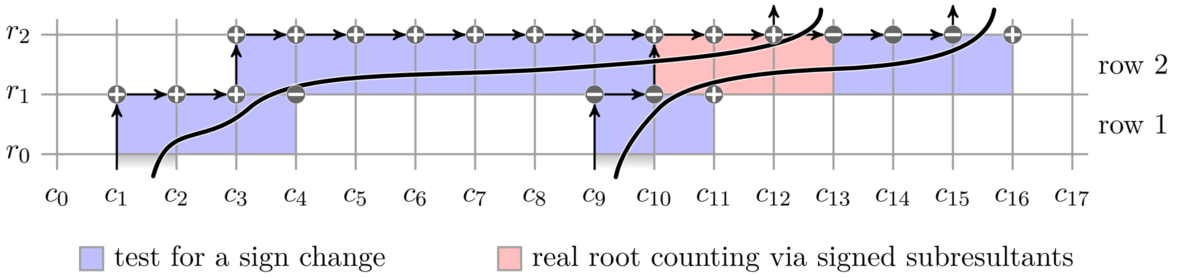

r_2
r_1
r_0
row 2
row 1
c_0 c_1 c_2 c_3 c_4 c_5 c_6 c_7 c_8 c_9 c_{10} c_{11} c_{12} c_{13} c_{14} c_{15} c_{16} c_{17}
test for a sign change
real root counting via signed subresultants

Figure 3.6.: Example of tracing two curve segments in a monotone, non-critical box. The first segment enters the box in $(c_1, c_2]$ of r_0. We also know that the second segment starts in $(c_9, c_{10}]$ and that it is monotone increasing. Therefore, we can use the simple test for a sign change of $f_{r_1}(x) = F(x, r_1)$ to find the raster position where the first segment leaves row 1 as long as we are below c_9. The sign change of $f_{r_1}(x)$ is found to be between c_3 and c_4. We process the second segment in the same way and the sign change is found between c_{10} and c_{11}. In row 2, we can again start with the simple test for a sign change to trace the first segment. But as soon as we reach c_{10}, we know that there is a second segment in the pixel and we move over to the real root counting via signed subresultant sequences. The first segment leaves the row between $(c_{12}, c_{13}]$ and we switch back to the simple test for a sign change for pixels to the right of c_{13}.

intermediate value theorem, this leads to

$$\#\mathrm{rr}(f_{r_j}, (c_i, c_{i+1}]) = \begin{cases} 1 & f_{r_j}(c_i) f_{r_j}(c_{i+1}) \leq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

The segment passes from $p_{i,j-1}$ to $p_{i,j}$ if and only if $\#\mathrm{rr}(f_{r_j}, (c_i, c_{i+1}]) = 1$, and this is easily determined by checking for a sign change of f_{r_j} in $(c_i, c_{i+1}]$. This simple test suffices for almost all pixels of a large class of curves rasterizations and thus makes the rasterization algorithm very efficient if the curve is sufficiently easy, i.e. the segments are not densely packed. It is applied to all of the blue pixels in Figure 3.6. There, the polynomial f_{r_1} has no sign change in $(c_1, c_2]$ and $(c_2, c_3]$ but in $(c_3, c_4]$. Thus, the first segment escapes the current row through $(c_3, c_4]$ and we proceed by processing the next segment, which enters the current row in $(c_9, c_{10}]$.

If the current pixel contains more than one segment, $f_{r_j}(c_i) f_{r_j}(c_{i+1})$ will be non-positive in case of an even number of segments leaving the row in $(c_i, c_{i+1}]$. In this case, the simple test gives no definitive answer about the exact number of escaping segments. If this happens as in the red pixels in Figure 3.6, we obtain $\#\mathrm{rr}(f_{r_j}, (c_i, c_{i+1}])$ by evaluating the signs of the signed subresultant sequence $\mathcal{S}$ of f_{r_j} at c_i and c_{i+1}. Note that due to the definition of non-critical slices, all real roots of f_{r_j} are simple. This allows to perform real root counting even in presence of a root at c_i or c_{i+1} by applying Lemma 2.4.11, i.e.

$$\#\mathrm{rr}(f_{r_j}, (c_i, c_{i+1}]) = \begin{cases} \mathrm{MVar}(\mathcal{S}_+, c_i) - \mathrm{MVar}(\mathcal{S}, c_{i+1}) & \text{for } f_{r_j}(c_i) = 0, f_{r_j}(c_{i+1}) \neq 0, \\ \mathrm{MVar}(\mathcal{S}, c_i) - \mathrm{MVar}(\mathcal{S}_+, c_{i+1}) & \text{for } f_{r_j}(c_i) \neq 0, f_{r_j}(c_{i+1}) = 0, \\ \mathrm{MVar}(\mathcal{S}_+, c_i) - \mathrm{MVar}(\mathcal{S}_+, c_{i+1}) & \text{for } f_{r_j}(c_i) = 0, f_{r_j}(c_{i+1}) = 0, \\ \mathrm{MVar}(\mathcal{S}, c_i) - \mathrm{MVar}(\mathcal{S}, c_{i+1}) & \text{otherwise.} \end{cases} \tag{3.2}$$

Thus, we can determine $\#\mathrm{rr}(f_{r_j}, (c_i, c_{i+1}])$ without subdividing $(c_i, c_{i+1}]$ in any case.

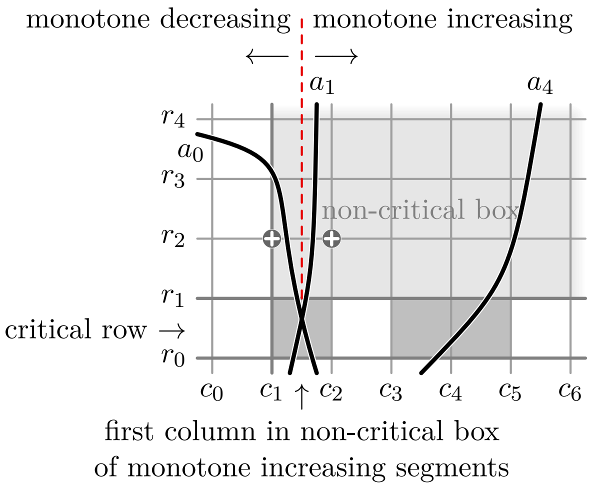


Figure 3.7.: If we build non-critical boxes based on the raster positions of the intersections of monotone segments with the lower and upper boundary of their non-critical slice, the box will not necessarily be monotone. This is due to the overlap of adjacent non-critical boxes if two segments of different monotony start at the same raster position. In the above example, the method for tracing curve segments in a monotone non-critical box assumes that there is only one monotone increasing segment a_1 entering into the non-critical box in its first (leftmost) column. Therefore, it will apply the simple test for a sign change. This may fail since the monotone decreasing segment a_0 (which is attributed to the adjacent non-critical box to the left) also enters the non-critical box for the monotone increasing segments at its first column. In the example, no sign change is detected between (c_1, r_2) and (c_2, r_2) although the monotone increasing segment a_1 leaves the first row at the first column of the box. Computating a rational coordinate (¦) that separates a_0 and a_1 involves repeated pixel subdivision and may be costly.

Since the number of segments in the current pixel is always known, the algorithm may switch between both tests several times within one row, depending on the local segment separation. This also happens in Figure 3.6, where the algorithm switches back to the simple test in the columns to the right of c_{13} in row r_2.

As soon as the rightmost pixel of the rightmost segment has been determined w.r.t. the current row, the algorithm moves on to the next row with upper boundary r_{j+1}. Note that only those pixels are examined, that are covered by the curve and thus painted. Empty pixels between the last pixel of the current segment and the start pixel of the next are safely discarded. This desirable property of curve rendering algorithms is fulfilled by the presented method.

3.5.1.2. The first column in a non-critical box

Until now, we ignored a special case that can occur. The described algorithm only works correctly if applied to a monotone non-critical box. However, the previous construction does not yield monotone non-critical boxes in all cases. This is best explained using Figure 3.7. The large gray box is a non-critical box. Its size has been computed by determining the raster positions of the intersections of all segments in a group of adjacent segments with the same monotony. In the example, the monotone increasing segments a_1 and a_4 start somewhere within $[c_1, c_2] \times r_1$ and $[c_4, c_5] \times r_1$, respectively. The monotone decreasing segment a_0 also

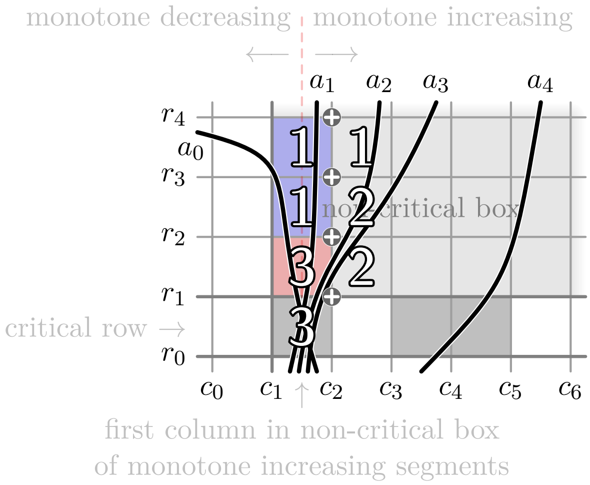


Figure 3.8.: Special treatment of the first column in a non-critical box that is monotone increasing except for its first (leftmost) column. The figure is a continuation of Figure 3.7 with the two monotone increasing segments a_2 and a_3 added. In order to rasterize monotone increasing segments, the algorithm needs to know the correct number of monotone increasing segments in each pixel. We know that the three segments a_1, a_2 and a_3 enter the non-critical box in the first column. The segment a_0 does not count since it is monotone decreasing. Thus, $t_{1,1} = 3 > 1$ and we have to apply the real root counting (🟥) in order to determine that a_2 and a_3 leave the first column in (r_1, r_2). Then $t_{2,1} = \#\mathrm{rr}(F(x, r_1), [c_2, c_3)) + \#\mathrm{rr}(F(c_2, y), (r_1, r_2)) = 0 + 2 = 2$ and $t_{1,2} = t_{1,1} - \#\mathrm{rr}(F(c_2, y), (r_1, r_2)) = 3 - 2 = 1$. The value of $t_{1,2}$ is sufficient to complete the rasterization of the current row while $t_{2,1}$ will be used for the special treatment of the first column in the next row below r_3. Since $t_{2,1} = 1$, the algorithm can return to the simple test for a sign change (🟦).

starts in $[c_1, c_2] \times r_1$, i.e. there is a one column overlap of adjacent non-critical boxes. Therefore, the non-critical box is not monotone due to its first column. It can not be processed by the algorithm described so far.

To avoid these problems, one might compute a rational coordinate, that separates the groups. This would involve the subdivision of the image below pixel level until the roots of f_{r_1} in $[c_1, c_2]$ have been separated. The row below r_1 is a critical one and the root separation near critical or even singular points is likely to be bad. For this reason, we will avoid the subdivision.

During the construction of the non-critical box, we have determined the raster positions of the start and end points of all of its segments. Thus, when we start to rasterize the curve in the current (putative) monotone non-critical box, we are able to detect if there is an overlap with an adjacent non-critical box which contains segments of different monotony. If this is the case, we apply a special treatment. See Figure 3.8 for a continuation of the example given in Figure 3.7. In what follows, we will investigate the special treatment in general.

Consider the rasterization of the curve in the row below r_j. Let $P_{i,j-1} = [c_i, c_{i+1}] \times [r_{j-1}, r_j]$ be the first pixel in the current row of a non-critical box which is monotone up to its first column, $P_{i,j}$ the pixel above $P_{i,j-1}$ and $P_{i+1,j-1}$ the pixel to the right of $P_{i,j-1}$. Furthermore, let $t_{i,j-1}$ be the total number of monotone increasing segments in $P^{\circ}_{i,j-1} = P_{i,j-1} \setminus (c_i, r_j)^{\dagger}$.

[†] We have to ignore the upper left pixel corner (c_i, r_j) of $P_{i,j-1}$ since monotone increasing segments passing

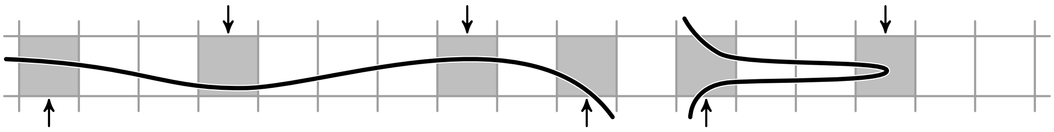


Figure 3.9.: In order to rasterize the curve in a critical row, we first paint all its critical points (↓) and intersections with the row boundaries (↑).

Now, the total number $t_{i+1,j-1}$ of monotone increasing segments in $P^{\circ}_{i+1,j-1}$ (the pixel to the right of $P^{\circ}_{i,j-1}$) is given by

$$t_{i+1,j-1} = \#\mathrm{rr}(F(x, r_j), [c_{i+1}, c_{i+2})) + \#\mathrm{rr}(F(c_{i+1}, y), (r_{j-1}, r_j)). \tag{3.3}$$

and the total number $t_{i,j}$ of monotone increasing segments in $P^{\circ}_{i,j}$ (the pixel above $P^{\circ}_{i,j-1}$) is given by

$$t_{i,j} = t_{i,j-1} - \#\mathrm{rr}(F(c_{i+1}, y), (r_{j-1}, r_j)). \tag{3.4}$$

Having this information at hand, we can proceed with the tracing of the curve segments to the right of $P^{\circ}_{i,j-1}$ as normal, i.e. as presented in Section 3.5.1.1. Once we moved over to the row above r_j, we can use $t_{i,j}$ to start the same special treatment again until all segments of increasing monotony left the first column.

Note that $t_{i+1,j-1}$ and $\#\mathrm{rr}(F(x, r_j), [c_{i+1}, c_{i+2}))$ are known a priori. They have been computed when we determined the raster positions of the start and end points of the segments in a non-critical box. It remains to calculate $\#\mathrm{rr}(F(c_{i+1}, y), (r_{j-1}, r_j))$. Due to the monotony property, we can reuse the techniques discussed above, i.e. we apply the test for a sign change in simple and real root counting in difficult cases. The process is best illustrated in Figure 3.8.

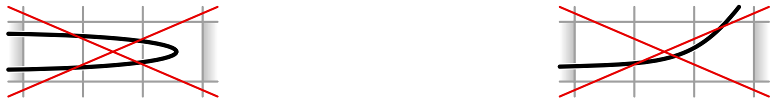
3.5.2. Rows with critical points

Within a critical row, we first paint all the pixels, that contain critical points. They have been isolated during the preprocessing, but may need refinement with respect ot the current viewport. Then we paint the pixels containing intersections of the curve with the row boundary. They have been determined during the creation of the non-critical slices. Having these pixels painted, we obtain a situation like in Figure 3.9.

Let us now examine the blocks of pixels between two already painted pixels in a critical row. Within these blocks, the curve does not have self crossings, local extreme values and intersections with the row boundary. Thus, curve segments may only proceed from left to right. Figure 3.10 shows the possible and impossible configurations of curve progression. It suffices to test the left (or right) boundary of such a block for an intersections with the curve to determine, whether to paint all pixels of the block or none. An odd number of intersections can be detected by testing the left (or right) boundary for a sign change of $F(x, y)$. For an even number of intersections we use real root counting as described in Section 2.4.1. By construction, the curve does not cross the corners of the blocks between the critical pixels of a critical row. No special cases need to be considered.

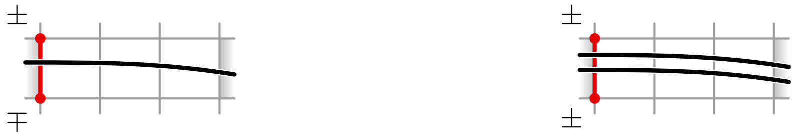
---

through (c_i, r_j) are attributed to $P_{i-1,j-1}$. If we would not ignore (c_i, r_j), we would count such segments twice.



(a) Loops are impossible since they contain a critical point.

(b) Pixels where the curve leaves the critical row have already been painted.



(c) Sign change: Odd number of segments.

(d) No sign change: Even number of segments.

Figure 3.10.: Since critical points and intersections with the boundary of the critical row are already painted, the configurations shown in (a) and (b) cannot occur between the already painted pixels. All segments (if any) proceed from left to right through the remaining blocks of pixels. Therefore, we can first perform the test for a sign change (see (c)) and in the case of an even number of segments (see (d)) we move over to the real root counting via signed subresultants. The test at the left boundary of each block suffices to determine if all or no further pixels have to be painted.

3.6. Numerical filtering

The proposed curve rendering algorithm uses subresultant based real root counting of polynomials to solve difficult cases. We have already discussed exact methods in Section 2.4.1. These are often considered to be inefficient in practice because their bit complexity is quite high, e.g. in the worst case $\mathcal{O}(n\mathcal{M}(n(\tau + \mathrm{bit}(n)) + \sigma)) \subset \tilde{\mathcal{O}}(n^2\tau + n\sigma)$ bit operations are needed to determine the signs of the signed subresultants of a polynomial of magnitude (τ, n) at an integer coordinate with σ bits using the sequence of quotients. Since only $\mathcal{O}(n)$ arithmetic operations are performed, numerical filtering can give a speedup of up to two orders of magnitude in this example. We will now investigate some techniques that often allow to work with approximate rather than exact numbers.

First, it is important to note that exact results can often be obtained using numerical approximations during the intermediate computations. In our case, the exact result is the sign of a real number while the exact value of the number is not of interest. In order to validate the exactness of the result, we use interval arithmetic as described in Section 2.5. If an exact result could not be obtained due to round-off errors at a certain level of precision, we double the precision and repeat the respective part of the computation. This numerical approach works quite well if we use it to replace exact computations where only integers are involved since any integer can also be represented as a binary floating point number. If we are going to replace computations with rationals, this is not true (as in the case of $\frac{1}{3}$) and the successive increase of the precision might not terminate. Therefore, one has to fall back to rational arithmetic once a certain level of precision is reached. In the above rasterization algorithm, there is no need for rational numbers since the sampling points as well the coefficients of $F(x, y)$ are integral numbers.

3.6.1. Numerical computation of signed subresultant quotient sequences

In general, the numerical computation of Sturm sequences for real root counting is quite ill conditioned since the bitsize of the coefficients of the polynomials in the sequence grows quadratically (see Lemma 2.3.25). Subresultants perform better due to the linear growth of the coefficient size (see Lemma 2.3.27). However, additional aspects have to be considered.

During the numerical computation of the subresultant quotient sequence on the basis of Corollary 2.3.13, one needs to know the exact degrees of the already known polynomials in order to perform the polynomial division, since this relies on the leading coefficient of the divisor. One possibility is to guess the degree of the remainder of a division by treating leading coefficients smaller than some threshold as zero. Guessing a degree that is too small will render the remainder useless. Guessing a degree that is too large might introduce severe numerical errors in subsequent divisions, since they are performed with a divisor, whose leading coefficient is likely to be almost as small as the threshold. Additionally, it is unclear if the resulting sequence is still useful for real root counting.

In the proposed rasterization method, we can avoid such problems in most cases using the specialization property of signed subresultants. During the preprocessing stage of the algorithm, we computed the two signed subresultant sequences for the bivariate polynomial $F \in \mathbb{Z}[x, y]$ and its partial derivatives. One sequence used x as the outermost variables, while the other sequence used y. Due to Lemma 2.3.32, these sequences give a quite good upper bound on the degrees of the polynomials in the signed subresultant sequences of F specialized at a certain value of y or x (the respective innermost variable). The upper bound is met with probability one since the degenerate case only occurs if we specialize at a root of one of the $\mathrm{sres}_j(F, \frac{\partial F}{\partial x})$ resp. $\mathrm{sres}_j(F, \frac{\partial F}{\partial y})$, which are univariate polynomials having only a finite number of roots. To compute the quotients at one of these roots, one can simply switch to exact integer arithmetic.

This approach creates a numerical approximation of the quotient sequence or an exact sequence in some rare cases. Since we will use this sequence for real root counting only, numerical inexactness in the sequence does not cause trouble as long as it still allows to determine the exact sign of the signed subresultants at a certain position.

3.6.2. Numerical real root counting

In this section, we assume that the sequence of quotients has been created successfully, e.g. the degree of each signed subresultant polynomial is known. As mentioned earlier, real root counting relies on sign determination. Due to the numerical evaluation, we might not be able to detect all signs correctly. We denote the signs by '+', '−' and '0' as usual and by '?' in the uncertain case. Because of Definition 2.4.7 and Theorem 2.4.8, we know that only the subsequences $[+, +]$, $[+, -]$, $[+, 0, -]$, $[+, 0, 0, +]$, $[+, 0, 0, -]$ and their symmetric counterparts obtained by interchanging '+' and '−' can occur in a valid sign sequence of evaluated signed subresultants. Thus, we can exploit some combinatorial arguments in uncertain cases to avoid switching to a higher precision. The sign pattern $[+, ?, -]$ may actually be $[+, +, -]$, $[+, -, -]$ or $[+, 0, -]$, all of which have $\mathrm{MVar} = 1$. The exact sign of '?' is not relevant to the real root counting in this case. This technique can be applied to many sign patterns. Table 3.1 provides a complete list of all possible cases of uncertain sign patterns with a minimal length of 3 elements that satisfy the following conditions. Let $\mathcal{T}$ be a sequence of signs, then

1. $\mathcal{T}$ starts with a '+'

sign sequence $\mathcal{S}$	possible resolutions	$\mathrm{MVar}(\mathcal{S})$
$[+,?,+]$	$[+,+,+],[+,-,+]$	$0,2$
$[+,?,-]$	$[+,+,-],[+,-,-],[+,0,-]$	1
$[+,0,0,?]$	$[+,0,0,+],[+,0,0,-]$	$2,1$
$[+,0,?,+]$	$[+,0,-,+],[+,0,0,+]$	2
$[+,0,?,-]$	$[+,0,-,-],[+,0,0,-]$	1
$[+,?,0,+]$	$[+,-,0,+],[+,0,0,+]$	2
$[+,?,0,-]$	$[+,+,0,-],[+,0,0,-]$	1
$[+,0,0,?,+]$	$[+,0,0,+,+],[+,0,0,-,+]$	2
$[+,0,0,?,-]$	$[+,0,0,+,-],[+,0,0,-,-]$	$3,1$
$[+,0,?,0,+]$	$[+,0,-,0,+]$	2
$[+,?,0,0,+]$	$[+,+,0,0,+],[+,-,0,0,+]$	2
$[+,?,0,0,-]$	$[+,+,0,0,-],[+,-,0,0,-]$	$1,3$
$[+,0,0,?,0,+]$	$[+,0,0,-,0,+]$	2
$[+,0,0,?,0,-]$	$[+,0,0,+,0,-]$	3
$[+,0,?,0,0,+]$	$[+,0,-,0,0,+]$	2
$[+,0,?,0,0,-]$	$[+,0,-,0,0,-]$	3
$[+,0,0,?,0,0,+]$	$[+,0,0,+,0,0,+],[+,0,0,-,0,0,+]$	$4,2$
$[+,0,0,?,0,0,-]$	$[+,0,0,+,0,0,-],[+,0,0,-,0,0,-]$	3

Table 3.1.: Possible resolutions for all uncertain sign patterns with a length of 3 to 7 elements, that start with a '+' and contain a single '?' inside the sequence (not at the beginning or end). For many of them, the modified number of sign changes can be determined in spite of an uncertain sign if $\mathrm{MVar}(\mathcal{S})$ is the same for all possible resolutions. Due to symmetry, the list of sequences starting with '$-$' is obtained by interchanging the roles of '+' and '$-$'.

2. $\mathcal{T}$ contains a single '?' inside the sequence but not at the beginning or end
3. $\mathcal{T}$ is not reducible, i.e. $\mathcal{T}$ does not contain the sign patterns $[+,+]$, $[-,-]$, $[+,-]$, $[-,+]$, $[+,0,-]$, $[-,0,+]$, $[+,0,0,+]$, $[-,0,0,-]$, $[+,0,0,-]$ and $[-,0,0,-]$, which can be resolved by Definition 2.4.7.

Note that a sequence $\mathcal{T}$ with a length greater than 7 can not occur. Since only one uncertain sign is present, longer sequences contain at least one reducible subsequence and can be shortened by applying the $\mathrm{MVar}(\cdot)$ operator to the subsequence. To illustrate the reduction, an example with fewer signs suffices:

$$\mathrm{MVar}([+,?,-,0,+]) = \mathrm{MVar}([+,?,-]) + \mathrm{MVar}([-,0,+]) \tag{3.5}$$

$$= 1 + 1 = 2. \tag{3.6}$$

If a full sign sequence $\mathcal{S} = [\mathrm{SRes}_0(c), \ldots, \mathrm{SRes}_{n-1}(c), \mathrm{SRes}_n(c)]$ for an evaluation point $c \in \mathbb{Z}$ contains several uncertain signs, $\mathrm{MVar}(S)$ can still be calculated as long as $\mathcal{S}$ can be decomposed into overlapping certain sign patterns and resolvable uncertain sign patterns, e.g. $[+,?,-,?,+]$ has two uncertain signs and can be resolved as

$$\mathrm{MVar}([+,?,-,?,+]) = \mathrm{MVar}([+,?,-]) + \mathrm{MVar}([-,?,+]) \tag{3.7}$$

$$= 1 + 1 = 2. \tag{3.8}$$

y	signs in $\mathcal{Q}(y)$	$\Leftrightarrow$	resolution of `?` signs	$\mathrm{MVar}(\mathcal{Q}(y))$
-0.12	`+---+-++++-+---+-+-+-+-+-+-`	$\Leftarrow$	no ‘?’	19
0.24	`-?+-+++-+++++-+------------`	$\Leftarrow$	$\mathrm{MVar}([-,?,+]) = 1$	8
0.36	`??+-+++-+++++-+------------`	$\Leftarrow$	evaluate ‘$\Rightarrow$’	
	`--+-+++-+++++-+------------`	$\Rightarrow$	no ‘?’	8
6.00	`??+-+++-++++-++------------`	$\Leftarrow$	evaluate ‘$\Rightarrow$’	
	`+++-+++-++++-++------??????`	$\Rightarrow$	merge with result of ‘$\Leftarrow$’	
	`+++-+++-++++-++------------`	$\Leftrightarrow$	no ‘?’	7

Table 3.2.: Resolution of uncertain signs during the real root counting of the “Bundle” curve $B(x,y) = 0$ at $x = 0.01$ using the signed subresultant PQS. The list of signs corresponds to the evaluations of the sequence of signed subresultants $\mathcal{Q}(y) = [\mathrm{SRes}_n(y), \mathrm{SRes}_{n-1}(y), \ldots, \mathrm{SRes}_0(y)]$ at the corresponding value for y at a floating point precision of 256 bit. The arrows specify the direction of evaluation of the sequence of quotients. In case of ‘$\Leftarrow$’, the recursive evaluation started from SRes_0, while in case of ‘$\Rightarrow$’, the elements SRes_n and SRes_{n-1} have been used. ‘$\Leftrightarrow$’ marks the merged result of ‘$\Leftarrow$’ and ‘$\Rightarrow$’. Using combinatorial arguments and recursive evaluation of the PQS in both directions often allows to determine $\mathrm{MVar}(\mathcal{Q}(y))$ in spite of uncertain signs. This effectively avoids the need to switch to a higher numerical precision. Note that in practice we can even stop the evaluation of the ‘$\Rightarrow$’ direction as soon as all signs have been resolved. In the case of $y = 0.36$ and $y = 6.00$, the evaluation of $\mathrm{SRes}_n(y)$ and $\mathrm{SRes}_{n-1}(y)$ would be sufficient but we displayed the whole sequence for clarity.

In practice, the most frequent cases are $[+,?,+]$ and $[+,?,-]$ resp. $[-,?,-]$ and $[-,?,+]$, since a ‘0’ only occurs at the finitely many zeros of the subresultant polynomials.

During the evaluation of the quotient sequence, round-off errors will usually accumulate and reach their maximum value in the last element. To alleviate these effects, the evaluation can be started from SRes_l, which in the squarefree case is SRes_0, the resultant, and thus a constant for all evaluations. As soon as an uncertain signs pattern cannot be resolved into a certain number of sign changes, the evaluation is restarted from SRes_n and SRes_{n-1} as explained in Section 2.4.1.1. If the modified number of sign changes can still not be determined, the numerical precision is increased until the final result is known.

The practical relevance of the above consideration is emphasized by the following example.

Example 3.6.1. Consider the curve $B(x,y) = 0$ of degree 26 defined by the polynomial shown in Equation (C.1) (see Appendix C). For the purpose of illustration, the sequence of signed subresultant quotients has been created for the specializations of $B(x,y)$ at $x = 0.01$. The signs of the signed subresultants have been determined at $y \in \{-0.12, 0.24, 0.36, 6.00\}$ using a floating point precision of 256 bit. Table 3.2 shows the results and the occurring problems due to uncertain signs, which can be resolved by applying the techniques explained above. This prevents the algorithm from switching to a higher numerical precision, e.g. 512 bit, which would slow down the computations considerably.

3.7. Analysis of the asymptotic complexity

The presented algorithm allows to compute exact rasterizations of real algebraic plane curves using symbolical and exact computations combined with numerical filtering techniques. The exactness is guaranteed for all inputs. The time needed for the computation splits up into the precomputation and the actual rasterization, which can be repeated using different viewports. As a consequence of the exactness criterion, the running time of the algorithm depends not only on the number of arithmetic operations performed, but also on the size of the involved numbers. Therefore, we will derive the worst case bounds on the bit complexity of all steps of the new algorithm. A bound on the improved trivial algorithm (Algorithm 3.2) is provided in Section 3.7.5, and it is shown how the new algorithm improves upon it.

During the analysis, we will work on a polynomial $F(x, y) \in \mathbb{Z}[x, y]$ of magnitude (τ, n) as stated in Definition 2.1.3. For reasons of simplicity, we assume that the bound on the bitsize does not change if the curve is considered within a different viewport, i.e. if T is a mapping that transforms the coordinates to the new viewport, then we use τ as the maximum of the bitsizes of the coefficients of $F(x, y)$ and $F(T(x, y))$. Furthermore, the viewport is $[0, 0] \times [s, s]$ for $s \in \mathbb{Z}$. We assume $s > n$ since $s \leq n$ does not provide reasonable rasterization detail for many curves.

We already discussed that numerical filtering is of great importance in practice. Due to Lemma 2.5.3, we can ignore its asymptotic cost if we choose the precisions as stated in Section 2.5.

The results of the analysis can be summarized as follows:

Theorem 3.7.1 (bit complexity of the exact real algebraic curve rasterization). *The preprocessing of the curve has a bit complexity bounded by* $\tilde{\mathcal{O}}(n^8+n^7\tau)$. *Afterwards, the algorithm is able to compute a rasterization within a given viewport using* $\mathcal{O}(n^2 s \mathcal{M}(n(\tau + n \operatorname{bit}(s)))) \subset \tilde{\mathcal{O}}(n^3\tau s + n^4 s)$ *bit operations.*

3.7.1. Analysis of the preprocessing stage

First, the new algorithm requires the removal of vertical and horizontal lines, i.e. $\operatorname{cont}_y(F)$ and $\operatorname{cont}_x(F)$. A trivial implementation can compute each content using at most n GCD computations of univariate polynomials of degree at most n and coefficients of bitsize at most τ. Using the subresultant algorithm this, needs at most $n\mathcal{O}(n^2\mathcal{M}(n\tau)) \subseteq \tilde{\mathcal{O}}(n^4\tau)$ bit operations (cf. Table 2.3) which is good enough for our purpose.

During the rasterization stage, we need to be able to compute the raster positions of the critical points using their precomputed representation, which we assume to be as follows.

Assumption 3.7.2. The real solutions of a bivariate polynomial system $F = G = 0$ are represented using three lists: disjoint isolating intervals for $x_0, \ldots, x_k$, $k \leq n^2$, the real roots of $\operatorname{Res}_y(F, G)$; disjoint isolating intervals for $y_0, \ldots, y_l$, $l \leq n^2$, the real roots of $\operatorname{Res}_x(F, G)$; a list of pairs (i, j), such that (x_i, y_j) is a solution of the system. The lists of isolating intervals are sorted. Furthermore, the sign of the squarefree part of $\operatorname{Res}_y(F, G)$ is known for each interval endpoint. The same is assumed for the real roots $y_0, \ldots, y_l$ of $\operatorname{Res}_x(F, G)$. Finally, we require the occurring rational interval endpoints to be in reduced form, i.e. for $\frac{m}{n} \in \mathbb{Q}$ we require $\gcd(m, n) = \pm 1$.

The authors of [DET09] presented a bound of $\tilde{\mathcal{O}}(n^{12} + n^{10}\tau)$ for the task of computing the above mentioned interval representation of the real solutions of a bivariate polynomial system. Most recently, this bound has been improved in [ES12] by four orders of magnitude to $\tilde{\mathcal{O}}(n^8 + n^7\tau)$. Both methods also compute the resultants with respect to x and y and the squarefree part if necessary. Furthermore, the sequence of signed subresultant quotients and therefore also its degrees appear as a byproduct of the resultant computation. The cost of these symbolic computations is in $\tilde{\mathcal{O}}(n^4\tau)$ using the fast subresultant algorithm (see Table 2.3). Implementations are usually based on Corollary 2.3.13, which yields a complexity of $\tilde{\mathcal{O}}(n^5\tau)$. From a theoretical point of view, this in any case negligible compared to the total preprocessing cost of $\tilde{\mathcal{O}}(n^8 + n^7\tau)$. Note that the squarefree part of F has coefficients of bitsize at most $\tau' \leq \tau + n + \operatorname{bit}(n)$. We ignore that during the analysis since substituting τ by $\tau' \in \mathcal{O}(\tau + n)$ does not change the final result.

3.7.2. Adjustment of the viewport

In order to perform the rasterization algorithm on a new viewport, we need to transform the curve and determine the raster positions of the critical points. These processes can be done one after another for both coordinate directions. In this section, we focus on the x direction and derive the following bound:

Theorem 3.7.3 (bit complexity for the viewport adjustment). *The total cost for transforming F to a new viewport and for computing the raster positions of all critical points of the curve within the viewport is bounded by $\mathcal{O}(sn^2\mathcal{M}(n(\tau + n\operatorname{bit}(s))))$.*

The coordinate transformation T is of type $x \mapsto \frac{x-a}{b}$ resp. $x \mapsto (x-a)b$, i.e. a translation about a followed by a scale about a factor b with $a, b \in \mathbb{Z}$. Note that $V(F(\frac{x}{b}, y)) = V(b^n F(\frac{x}{b}, y))$, but $b^n F(\frac{x}{b}, y) \in \mathbb{Z}[x, y]$.

Proposition 3.7.4. *Applying the coordinate transformation $T(x) = \frac{x-a}{b}$ to the curve $V(F)$, i.e. determining the coefficients of the polynomial $G(x, y) = b^n F(T(x), y) \in \mathbb{Z}[x, y]$, where* $\operatorname{bit}(T)$ *and the bitsize of the coefficients of F, G are in $\mathcal{O}(\tau)$, is possible within $\mathcal{O}(n^2 s\mathcal{M}(\tau))$ bit operations.*

Proof. In order to apply the translation to $F(x, y) = \sum_{i=0}^{n} f_i(x)y^i$, we need to translate the $n + 1$ coefficient polynomials $f_i(x)$ of degree at most n. Using the trivial algorithm (see Table 2.3), this takes $(n + 1)\mathcal{O}(n^2\mathcal{M}(n\tau))$. But as stated in Section 3.7, we assume that the bitsize of the result is also smaller than τ and that $s > n$. Therefore, the shift is possible in $(n + 1)\mathcal{O}(n^2\mathcal{M}(\tau)) \subseteq \mathcal{O}(n^2 s\mathcal{M}(\tau))$ bit operations. The cost of scaling F is in $\mathcal{O}(n^2\mathcal{M}(\tau))$ by the same argument. ☐

The next step is to bound the number of operations needed to determine the new raster positions of the critical points. In practice, this relies heavily on the method used to isolate the critical points. Since we have not fixed this algorithm, we state one (not necessarily optimal) possibility that matches the final complexity bound of the rasterization stage.

The calculation of the raster positions is identical for both coordinate directions. Therefore, we focus on the x coordinates again. In order to relate the isolating intervals to the raster positions of the new viewport, one should not apply a coordinate transformation to the isolating intervals. The bitsize of the interval endpoints is bounded by the bitsize of the separation bound of the real roots, which is $\mathcal{O}(n^2 \cdot n(\tau + \operatorname{bit}(n)))$ for the resultant [EMT08]. Instead, we transform the raster positions $0, \ldots, s$ of the new viewport into the coordinate system of the default viewport. The bitsize of the raster positions transformed by T^{-1} is at most $\mathcal{O}(\operatorname{bit}(s) + \tau)$.

The raster positions are now determined as follows: Let $[a_0, b_0], \dots, [a_k, b_k]$ be the isolating intervals for the roots of the resultant as stated in Assumption 3.7.2. First, we merge the sorted list of intervals $[a_0, b_0, \dots, a_k, b_k]$ and the sorted list of transformed raster positions $[T^{-1}(0), \dots, T^{-1}(s)]$ into a combined sorted list. Then, we determine which isolating intervals contain which raster positions. This possibly requires to split isolating intervals at transformed raster positions.

Lemma 3.7.5. *Merging the sorted lists of intervals $[a_0, b_0, \dots, a_k, b_k]$ and transformed raster positions $[T^{-1}(0), \dots, T^{-1}(s)]$ into one sorted list is possible using at most $\mathcal{O}((n^2+s)\mathcal{M}(\operatorname{bit}(s)+\tau))$ bit operations.*

Proof. Merging the two sorted lists requires $\mathcal{O}(n^2 + s)$ comparisons since there are at most n^2 isolating intervals and $s + 1$ raster positions. Let $\frac{c}{d} \in \mathbb{Q}$, $\gcd(c, d) = 1$, be an endpoint of an isolating interval and let $\frac{p}{q} \in \mathbb{Q}$ be a transformed raster position.

The test for $\frac{c}{d} = \frac{p}{q}$ reduces to the test for $cp = dp$. Since $\frac{c}{d}$ is in reduced form, $\frac{c}{d} \neq \frac{p}{q}$, if $\frac{c}{d}$ has a bitsize greater than $\frac{p}{q}$. Therefore, the products cp and dp are only computed if their bitsize is in $\mathcal{O}(\operatorname{bit}(s) + \tau)$.

Assuming that $c, d, p, q > 0$, we can test for $\frac{c}{d} < \frac{p}{q}$ by testing for $\frac{c}{d}q < p$. The result is clear as soon as $\operatorname{bit}(p) \in \mathcal{O}(\operatorname{bit}(s) + \tau)$ bits of $\frac{c}{d}q$ are known. A sufficient approximation of the quotient $\frac{c}{d}$ can be computed in $\mathcal{O}(\mathcal{M}(\operatorname{bit}(s) + \tau))$ using only the necessary upper bits of c and d. If $c, d, p, q > 0$ does not hold, we have to change the relation symbol appropriately. ☐

Remark. Note that the above algorithm to compare $\frac{c}{d}q$ and p is considerably faster than computing all the $\mathcal{O}(n^3(\tau + \operatorname{bit}(n)) + \operatorname{bit}(s))$ bits in the test for $cq < dp$.

The merging algorithm yields a sorted list and this allows to determine which isolating intervals span which raster positions easily. Isolating intervals that are contained in the interval $(T^{-1}(r), T^{-1}(r+1))$ for some raster position $r \in \{0, \dots, s\}$ can be immediately assigned to raster position r. If an isolating interval $[a_i, b_i]$ contains $T^{-1}(r), \dots, T^{-1}(r+l)$, $l \in \mathbb{N}$, then $[a_i, b_i]$ has to be split at some of the $T^{-1}(t+j)$, $0 \leq j \leq l$, until it can be clearly assigned to a raster position. The total number of interval splits for all isolating intervals is bounded by $s + 1$ since they do not overlap. In practice, the number of splits can usually be reduced by using some kind of bisection scheme on the raster positions.

In order to split an isolating interval at a transformed raster position $T^{-1}(r)$, $r \in \{0, \dots, s\}$, contained in an isolating interval $[a_i, b_i]$ of a root $x_i \in \mathbb{R}$, we compare the signs of the squarefree part of $\operatorname{Res}_y(F, G)$ at the positions a_i, b_i and $T(r)$. Note that the signs at a_i and b_i are known from the precomputation. Therefore, the cost solely depends on the evaluations at $T(r)$, $r \in \{0, \dots, s\}$.

Lemma 3.7.6. *An evaluation of the squarefree part of $\operatorname{Res}_y(F, G)(x)$ at $T^{-1}(r)$ for some $r \in \{0, \dots, s\}$ is computable using $\mathcal{O}(n^2\mathcal{M}(n(\tau + n\operatorname{bit}(s)))$ bit operations.*

Proof. The squarefree part of the resultant is of magnitude $\mathcal{O}(n\tau + n\operatorname{bit}(n) + n^2, n^2)$ (see Lemma 2.3.23). Evaluating it at $T^{-1}(r)$, which has coefficients of bitsize $\mathcal{O}(\operatorname{bit}(s) + \tau)$, involves numbers of bitsize $\mathcal{O}(\tau n + n^2\operatorname{bit}(s))$ (cf. Table 2.3). Since $\operatorname{Res}_y(F, G)$ has at most $\mathcal{O}(n^2)$ coefficients, the evaluation needs no more than $\mathcal{O}(n^2\mathcal{M}(n(\tau + n\operatorname{bit}(s)))$ bit operations. ☐

Lemma 3.7.6 has to be applied to at most $s + 1$ raster positions. This yields the total complexity of $\mathcal{O}(sn^2\mathcal{M}(n(\tau + n\operatorname{bit}(s))))$ for the adjustment of the viewport as stated in Theorem 3.7.3.

3.7.3. Analysis of the rasterization stage

In the worst case, the algorithm needs to perform the real root counting via the signed subresultant sequence for all $\mathcal{O}(s)$ rows of the image. This happens for curves with many densely packed segments. For example, $V(F^2 - \varepsilon)$ shows two badly separated segments for each segment of $V(F)$. The main result of the analysis is:

Theorem 3.7.7 (bit complexity for the rasterization stage). *The bit complexity for the rasterization of the curve in the non-critical slices and the critical columns is* $\mathcal{O}(n^2 s\mathcal{M}(n(\tau + n\,\mathrm{bit}(s))))$.

We will derive the theorem in what follows.

Lemma 3.7.8. *The total cost to compute the quotients for the signed subresultant sequences for all* $s+1$ *rows resp. columns of the image is bounded by* $\mathcal{O}(n^2 s\mathcal{M}(n(\tau+n\,\mathrm{bit}(s))))$ *bit operations if the structure theorem for signed subresultants is applied, and by* $\mathcal{O}(n\log ns\mathcal{M}(n(\tau+n\,\mathrm{bit}(s))))$ *bit operations if the asymptotically fast subresultant algorithm is used.*

Proof. $F(x, y)$ evaluated at $x = r$ resp. $y = r$ for $r \in \{0, \ldots, s\}$ has coefficients of bitsize $\mathcal{O}(\tau + n\,\mathrm{bit}(s))$ (cf. Table 2.3). The bitsize of the coefficients in the subresultant quotient sequence is therefore bounded by $\mathcal{O}(n(\tau + n\,\mathrm{bit}(s)))$ due to Lemma 2.3.27 and Corollary 2.3.30. The number of arithmetic operations is bounded by $\mathcal{O}(n^2)$ using the structure theorem resp. by $\mathcal{O}(n\log n)$ using the asymptotically fast method. Therefore, computing the quotients for all $s+1$ rows resp. columns needs $(s+1)\mathcal{O}(n^2\mathcal{M}(n(\tau+n\,\mathrm{bit}(s))))$ resp. $(s+1)\mathcal{O}(n\log n\mathcal{M}(n(\tau + n\,\mathrm{bit}(s))))$ bit operations. ☐

In the worst case, we have to perform the real root counting for each pixel in the image covered by the curve. If we derive a bound for a single row and multiply it by the number of rows, the bound would be too weak since the curve may cover all pixels of a row. Thus, we need to bound the total number of covered pixels in the image.

Proposition 3.7.9. *A curve of total degree* n *covers at most* $\mathcal{O}(ns)$ *pixels of an image of size* $s \times s$.

Proof. First, note that by Theorem 2.2.9, the curve has up to n^2 x-critical and y-critical points, which is in $\mathcal{O}(ns)$ using $s > n$. If these points are painted, all subpixel components of the curve are covered, and it suffices to investigate the intersections of the curve at the boundary of each pixel. A vertical or horizontal line with a x or y coordinate equal to a raster position covers all $2s$ pixels in the respective adjacent columns or rows. Since there are at most n such lines, this contributes at most $\mathcal{O}(ns)$ pixels. Let $x = r$ be a raster position where F does not have a vertical line. Then $F(r, y) \not\equiv 0$ by Lemma 3.4.1. By the fundamental theorem of algebra (Theorem 2.4.1), $F(r, y)$ has at most n real roots. Applying this to the $s+1$ raster position with respect to rows and columns yields the result of $\mathcal{O}(ns)$ covered pixels that have an intersection with $V(F)$ at their boundary. ☐

Remark. A slightly closer look at the number of pixels contributed by each step of the algorithm suggests that the tight upper bound is located between $3ns$ and $6ns$ painted pixels. In order to find the true upper bound, one has to identify pixels, that are counted more than once. However, the exact value is not of importance in this analysis.

Lemma 3.7.10. *The number of bit operations necessary to perform real root counting for all pixels in the rasterization of the curve, that do not have a multiple root at a pixel corner, is bounded by* $\mathcal{O}(n^2 s\mathcal{M}(n(\tau + n\,\mathrm{bit}(s))))$.

Proof. By Lemma 2.4.13, the signed subresultant quotient sequence of a polynomial of magnitude (τ', n) can be evaluated in $\mathcal{O}(n\mathcal{M}(n(\tau' + \operatorname{bit}(n)) + \sigma))$, where σ is the bitsize of the evaluation point. In our case, $\tau' \in \mathcal{O}(\tau + n \operatorname{bit}(s))$ (cf. Lemma 3.7.8) and $\sigma \in \mathcal{O}(\operatorname{bit}(s))$. Since the real root counting is never performed on an interval which has a multiple root at one or both of its endpoints, we can always apply Lemma 2.4.11. Thus, no further cost arises from interval subdivision or the computation of the squarefree part of the respective univariate polynomials. The result follows by applying Proposition 3.7.9. ☐

We can now complete Theorem 3.7.7.

Proof of Theorem 3.7.7. Lemma 3.7.8 and Lemma 3.7.10 can immediately be applied to the non-critical slices as they do not contain multiple roots of the appearing univariate polynomials. All pixels in a critical column that contain critical points are known after the viewport has been adjusted (see Theorem 3.7.3). Thus, the remaining pixels do not have multiple roots at their boundary. In order to find the pixels, where the curve escapes a critical row between critical pixels, we can safely apply Lemma 2.4.11 and therefore also Lemma 3.7.8 and Lemma 3.7.10. The same is true, when we want to resolve the connections between all the pixels in the critical rows that have been painted so far (see Section 3.5.2). The number of pixels we have to check does not exceed $\mathcal{O}(ns)$ by Proposition 3.7.9 and the complexity of $\mathcal{O}(ns \cdot n\mathcal{M}(n(\tau + n \operatorname{bit}(s))))$ follows. Ignoring logarithmic factors yields the bound $\tilde{\mathcal{O}}(n^3\tau s + n^4 s)$ ☐

3.7.4. The improved trivial algorithm

To the knowledge of the author, there is no detailed analysis of other exact rasterization algorithms for real algebraic plane curves. For comparison, we will derive the complexity of the improved trivial algorithm (Algorithm 3.2). It is out of the scope of this work to derive bounds for the yet unknown complexity of more sophisticated algorithms like [EBS09].

The trivial algorithms also compute the raster positions of the critical points. Therefore, we can reuse the bounds on the preprocessing stage and the adjustment of the viewport. The only interesting stage is the rasterization stage. The improved trivial algorithm isolates the real roots of F along each row and column. This leads to the following lemma.

Lemma 3.7.11 (bit complexity of the rasterization stage using the improved trivial algorithm). *The rasterization needs no more than $\tilde{\mathcal{O}}(n^3\tau s + n^4 s)$ bit operations.*

Proof. By Lemma 2.4.18, the real root isolation of a polynomial of magnitude (τ', n) demands for $\tilde{\mathcal{O}}(n^3\tau')$ bit operations. The bitsize of the coefficients of F evaluated at one of the raster positions is $\tau' = \tau + n \operatorname{bit}(s)$. Hence, isolating the real roots of one such polynomial is in $\tilde{\mathcal{O}}(n^4\tau)$. Applying this to all $2(s + 1)$ raster positions of the rows and columns in the viewport results in a complexity of $\tilde{\mathcal{O}}(n^3(\tau + n \operatorname{bit}(s))s) = \tilde{\mathcal{O}}(n^3\tau s + n^4 s)$. ☐

3.7.5. Summary and comparison

We have seen that the precomputation takes at most $\tilde{\mathcal{O}}(n^8 + n^7\tau)$ bit operations and that the adjustment of the viewport and the rasterization need no more than $\mathcal{O}(n^2 s\mathcal{M}(n(\tau+n \operatorname{bit}(s)))) \subset \tilde{\mathcal{O}}(n^3\tau s + n^4 s)$ bit operations. The bound on the precomputation clearly outweighs the rest of the computation. As the rasterization is usually done repeatedly for different viewports, the bound of $\tilde{\mathcal{O}}(n^3\tau s + n^4 s)$ is still of major importance.

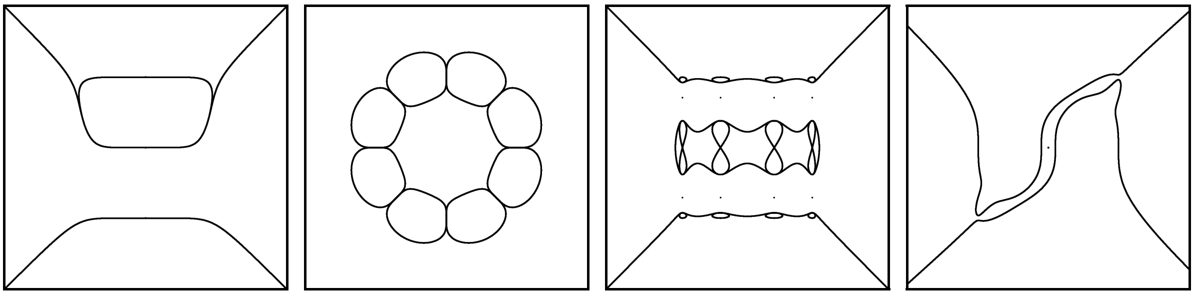
This bound is equal to the bound given in Lemma 3.7.11 on the rasterization stage of the improved trivial algorithm if logarithmic factors are ignored. Note that the bound of $\tilde{\mathcal{O}}(n^3\tau)$ for the asymptotic complexity of exact polynomial real root isolation is a rather recent one published in [Sag12]. The previous bound for this task using Descartes' rule of signs or signed subresultant sequences was, to the knowledge of the author, in $\tilde{\mathcal{O}}(n^4\tau^2)$ (see [EMT08]). Thus, the bit complexity of the rasterization stage of the improved trivial algorithm was $\tilde{\mathcal{O}}(s(n^4(\tau + n\,\mathrm{bit}(s))^2)) = \tilde{\mathcal{O}}(n^4\tau^2 s + n^6 s)$ prior to the results of [Sag12]. This is a difference of 2 orders of magnitude. Furthermore, [Sag12] uses fast Taylor shifts in order to perform the transformation $x \mapsto x + 1$ during the real root isolation (see also Section 2.4.2). Ignoring logarithmic factors, classical fast Taylor shifts are about $\mathcal{O}(n)$ slower than asymptotically fast Taylor shifts, which are not used in practice for any reasonable size of the input. According to [GG97], asymptotically fast Taylor shifts are worthwhile for polynomial degrees above some threshold between 256 and 512. Due to the high cost of $\tilde{\mathcal{O}}(n^8 + n^7\tau)$ for the preprocessing, the exact rasterization of curves given by polynomial of similarly high degrees seem out of reach for current implementations. Therefore, a worst case bound of $\tilde{\mathcal{O}}(n \cdot (n^3\tau s + n^4 s))$ for the improved trivial exact rasterization algorithm seems more realistic from a practical point of view.

A closer look at the fast real root isolation algorithm of [Sag12] reveals that $\mathcal{O}(n \log(n\tau'))$ Taylor shifts are necessary to isolate the real roots of a polynomial of magnitude (τ', n). This is due to the size of the recursion tree of the interval subdivision process. The involved numbers have a bitsize of $\mathcal{O}(n\tau')$ as in the algorithm presented above (here $\tau' = \tau + n\,\mathrm{bit}(s)$). Therefore, the new rasterization method improves upon the improved trivial exact rasterization algorithm by at least a factor of $\mathcal{O}(\log(n\tau')) = \mathcal{O}(\log(n(\tau + n\,\mathrm{bit}(s))))$ even if the fast Taylor shift is used in the new real root isolation algorithm of [Sag12]. Due to the $\tilde{\mathcal{O}}$ notation, this factor is not obvious.

We have not discussed the unimproved trivial algorithm (Algorithm 3.1) yet. It repeats the real root isolation resp. real root counting for each pixel. There are $\mathcal{O}(s^2)$ pixels. Using signed subresultants, each pixel demands for $\mathcal{O}(1)$ evaluations of the sequence. By Lemma 2.4.13 this yields a bit complexity of $\mathcal{O}(ns^2\mathcal{M}(n(\tau + n\,\mathrm{bit}(s)))) \subset \tilde{\mathcal{O}}(n^2 s^2 \tau + n^3 s^2)$, i.e. a factor of n is replaced by $s > n$. In order to derive a bound on Algorithm 3.1 when the real root isolation of [Sag12] is used, we would need further investigations since it is not obvious how the size of the recurvsion tree changes if we apply the real root isolation to the boundary of the $\mathcal{O}(s^2)$ pixels. We skip this analysis since Algorithm 3.2 is preferable over Algorithm 3.1 in any case. Furthermore, we have shown how the new rasterization algorithm improves upon Algorithm 3.2.

3.8. Implementation

The described approach has been implemented on top of the CGAL 3.7 library [CGAL12]. Besides various implemented algorithms related to linear computational geometry, it provides methods for computing with (multivariate) polynomials over the integers and rationals and for isolating the real zeros of univariate polynomials and systems of bivariate polynomials. The implementation of the latter algorithm is based on [EKW07; EK08]. The computation of signed subresultants subresultants has been extended to include the quotients and to allow for numerical filtering as described in Section 3.6. The algorithm has been tested for its efficiency. The results are shown in the Section 3.9.



(a) $f^2_{8,2,-}$ (b) $C_{2,8}$ (c) $FTT^2_{2,2,4,-}$ (d) $SAA_{3,2,10^{-2}}$

Figure 3.11.: Rasterizations produced by the new algorithm. The equations of the shown curves are taken from the list of challenging curves provided in [Lab10b]. More rasterized curves can be found in Appendix A. Note that the curves have been rasterized exactly at a high resolution in order to obtain a high detail. Afterwards each pixel is replaced by a circle of width larger than one pixel. This yields a smooth visualization.

3.9. Benchmarks of challenging curves

In order to test the practical efficiency of the new algorithm, we use the list of real algebraic plane curves presented in [Lab10b]. Its author emphasizes that the given curves are challenging for rasterization algorithms. In Appendix A, we will briefly review and discuss each type of challenging curve and show images of some of the curves created with the new algorithm. A few of them are also shown in Figure 3.11. Here, we will restrict to compare the runtime of the proposed algorithm to the runtime of the improved trivial algorithm (Algorithm 3.2) and to the runtime of the algorithm introduced in [EBS09] by means of density and violin plots. For brevity, the scatter plots of the absolute runtimes have been moved to Appendix B. In [Lab10b], constructions for curves of arbitrary degree are provided. Since some of these construction require more advanced computer algebra tools than available in the implementation of the new algorithm, we use the list of precomputed polynomials supplied at [Lab10a].

For all tested algorithms, the time that is occupied by the precomputation becomes infeasible for curves of high degree. Therefore, we restrict to curves $V(F)$ with $\deg(F) \leq 30$. This still includes 2696 curves from the list.

We will use the following abbreviations: ‘S’ for the new algorithm, ‘EBS’ for the algorithm of [EBS09] and ‘T’ for the improved trivial algorithm. Note that ‘S’ and ‘EBS’ have been chosen to reflect the initial letters of the authors of the respective algorithms.

3.9.1. Preprocessing

We first look at the time for the precomputation of the three algorithms. The speedup is illustrated in summarized form in Figure 3.12 and separately for each challenge in Figure 3.13. There are two noticeable peeks in the diagrams: one at a speedup of approximately $\frac{1}{2}$ and one at a speedup of approximately 1. The peek at $\frac{1}{2}$ is due to the fact that algorithm S computes both, the x critical and y critical points while the other two algorithm only need one direction (which can be chosen arbitrarily). The peek at 1 and most of the speedups smaller than $\frac{1}{2}$ occur since for many curves the x critical points are easier to compute than y critical points (or vice versa). For some curves the speedup is also greater than 1. The main reason is that algorithm S removes vertical and horizontal lines and rasterizes them separately. See for

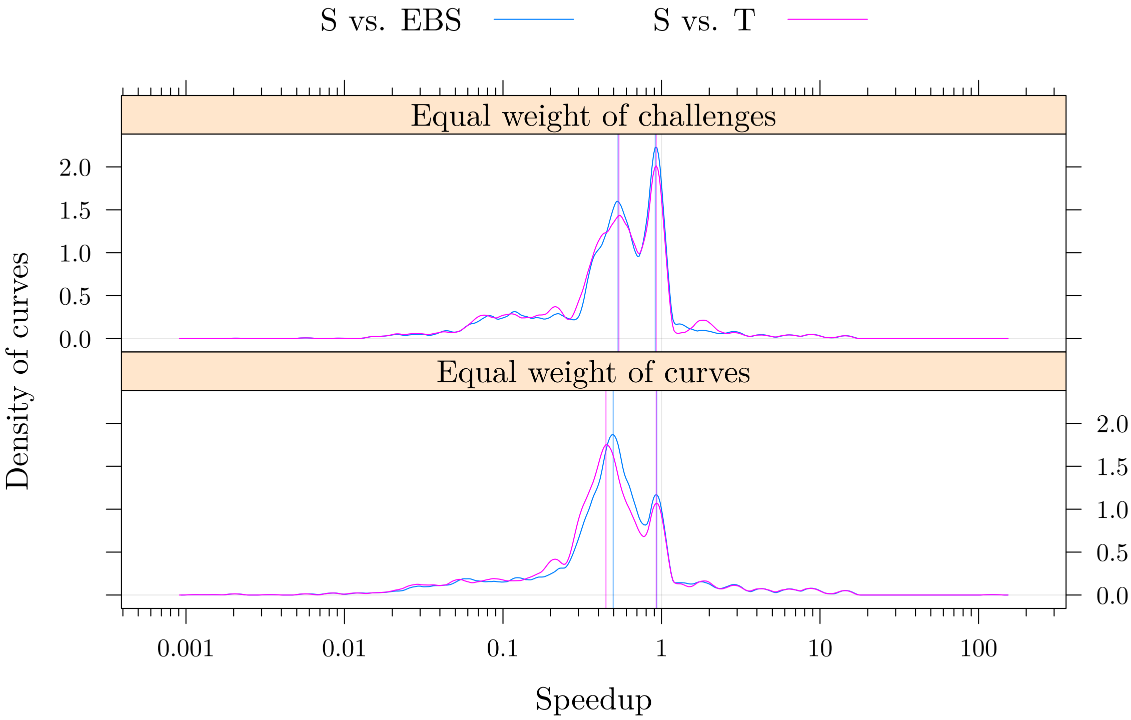


Figure 3.12.: Density plot of the speedup with respect to the preprocessing. Since some challenges contain much more curves than others, the densities are weighted by challenge and by curve. In the case of equal weight of curves, all determined speedups contribute equally to the density. In the case of equal weight of challenges, a curve contributes by the reciprocal of the number of curves in its challenge. Thus, challenges with fewer curves are not discriminated. Prominent peeks are marked with vertical lines.

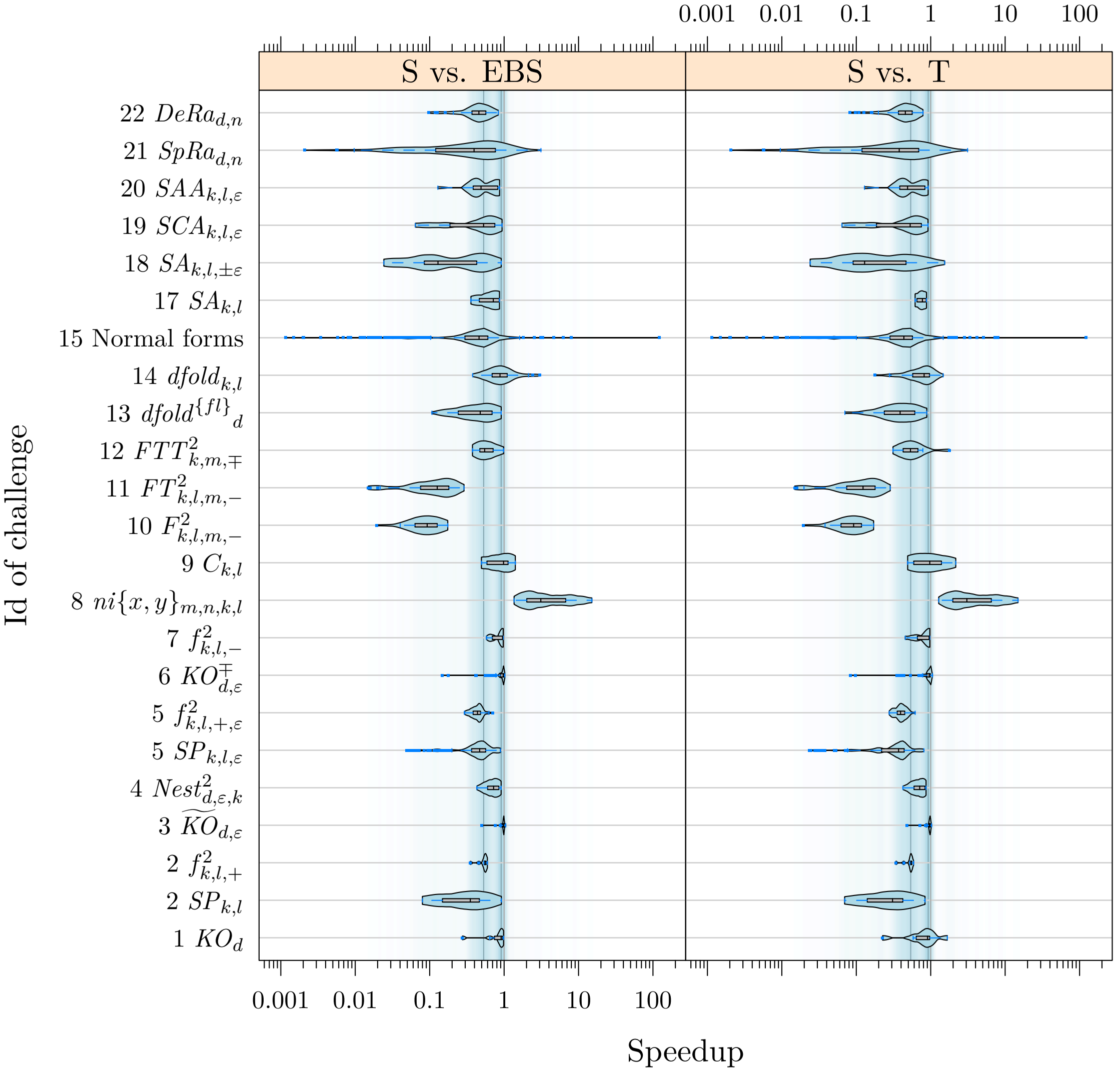


Figure 3.13.: Violin plot of the speedup with respect to the preprocessing. The "violins" show how the speedups are distributed over the different challenges, i.e. the thickness of a violin relates to the relative number of curves with that speedup. Note that the gray bars enclosed in the violins illustrate the lower and upper quartile and the location of the median. The prominent peeks of Figure 3.12 are also included in this diagram as vertical lines. This allows to analyze the contribution of a specific challenge to the respective peek. Furthermore, the approximation of the overall density summarized for all challenges is shown behind the violins. More saturated colors imply higher density.

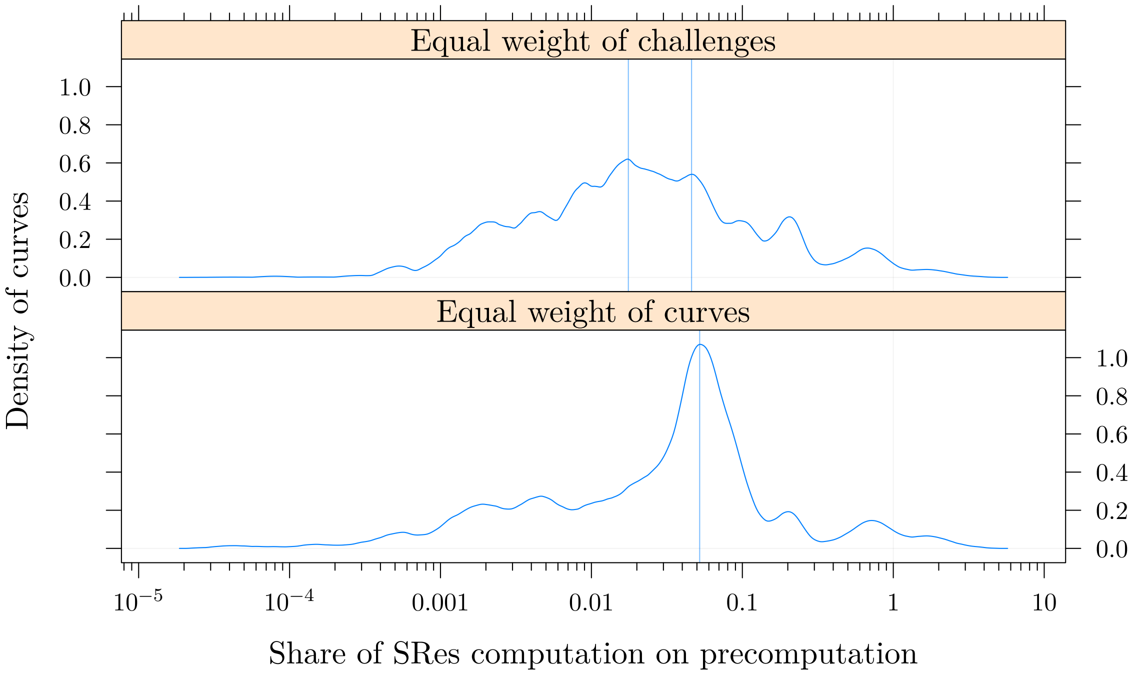


Figure 3.14.: Density plot of the share of the bivariate signed subresultant computation on the total precomputation time.

example Challenge 8. Removing the lines does not only simplify the curve but also reduces its degree, which is an important factor in polynomial system solving. It is hard to clearly analyze the remaining speedups. Two additional facts are probably related. First, the EBS algorithm also computes the topology of the curve, which might add considerable overhead in some cases. Secondly, the method for bivariate system solving [EK08] relies on the randomized real root isolation of polynomials with approximate coefficients presented in [Eig08] to ensure exact results. The randomization might only be of minor influence but it should not be ignored.

3.9.1.1. Share of the computation of bivariate signed subresultants

Multivariate signed subresultants are often considered to be a costly symbolic operation. In the implementation, the signed subresultants are computed during the calculation of the critical points. Afterwards, they are just queried from the underlying data structure. Their degrees are used during the numerical computation of the univariate signed subresultant quotient sequences (see Section 3.6). To determine the actual share of the computation of the bivariate subresultants on the precomputation, they have been computed separately in the experiments. The results are shown in Figures 3.14 and 3.15. For some curves the computation of the subresultants takes almost as much time as the whole precomputation. This is especially true for the curves of Challenges 4 and 13. Nevertheless, for many of the tested curves, the share is around 5%. If we consider equal weight of challenges, then the share is even lower. Therefore, we consider the cost of this symbolic computation to be negligible compared to the total time spend for isolation of the critial points in most cases.

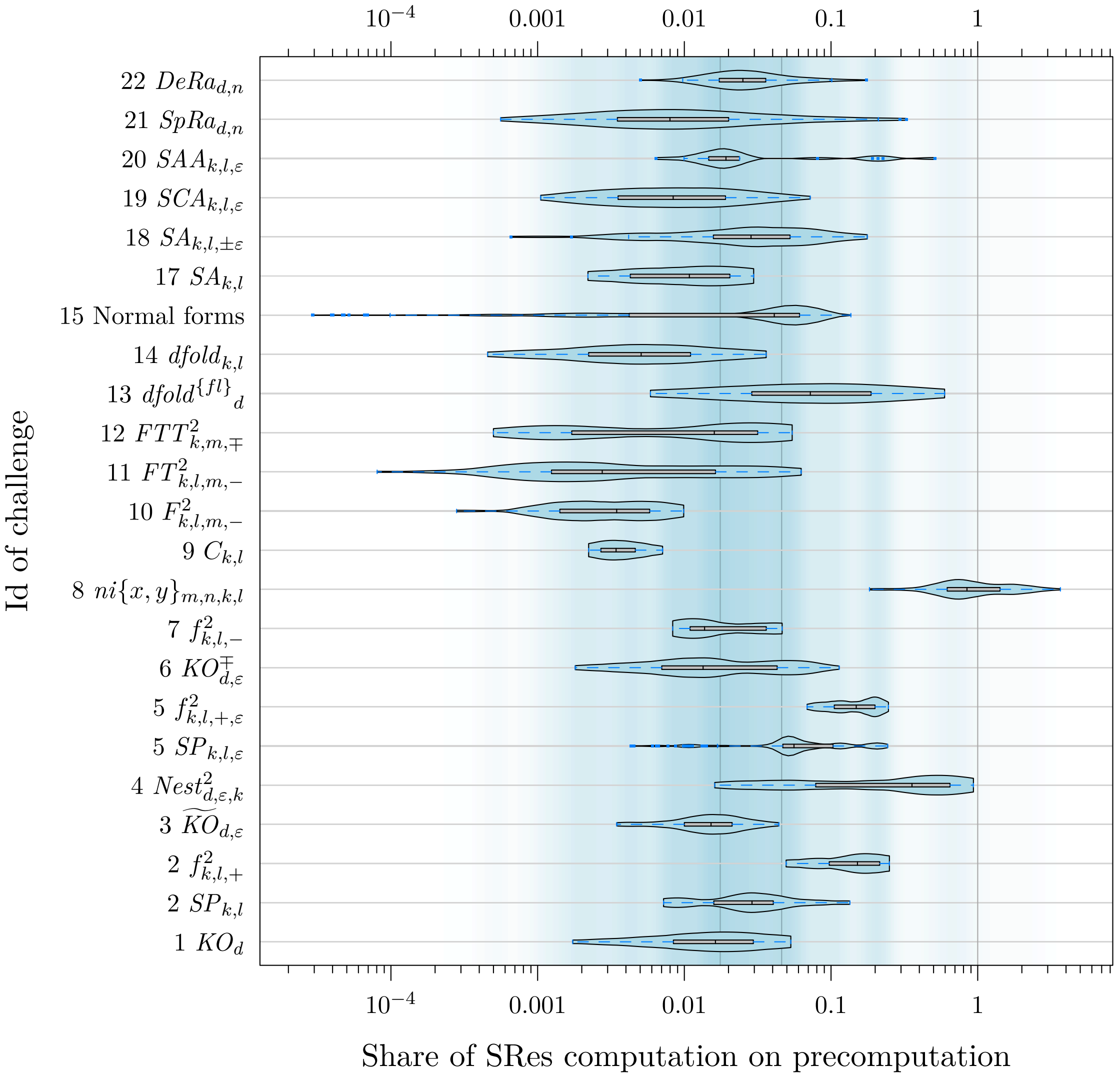


Figure 3.15.: Violin plot of the share of the bivariate signed subresultant computation on the total precomputation time.

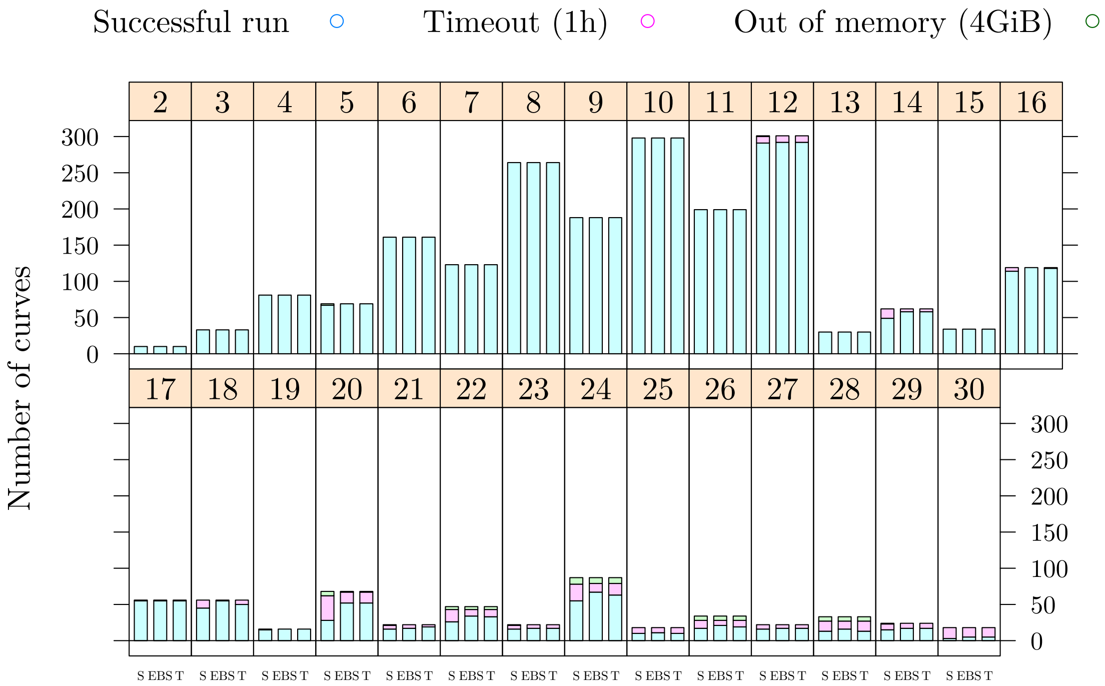


Figure 3.16.: A plot of the number of curves for each total degree. The bars are partitioned in order to illustrated the number of successful runs, timeouts and out of memory errors.

Remark. For some of the curves of Challenge 8, the share is above 100%. These curves are not squarefree and they contain vertical and horizontal lines. This has not been taken into account during the separate computation of the share of the subresultant computation i.e. the subresultants have been determined for the unreduced curves.

3.9.1.2. Unsuccessful runs

For some curves, the precomputation did not finish within a reasonable amount of time. The threshold was set to one hour. In some cases, the precomputation also allocated all the memory available to the system so that the application crashed. Both problems also occurred for curves of relatively low degree. The number of curves that could not be rendered properly is illustrated in Figure 3.16. The rendering failed for 264 of the 2696 tested curves. The EBS and T algorithms perform better as they only compute the critical points with respect to one direction, i.e. 119 resp. 140 renderings failed. Note that the mentioned issues are related to the algorithm for the analysis of real algebraic plane curves provided by the CGAL library which is called by the author's implementation.

3.9.2. Adjustment of the viewport

In Section 3.7.2, we detailed an algorithm for the adjustment of the viewport which has a good worst case complexity. However, a simplified approach has been used in the implementation. It is built on top of CGAL's mechanism for comparing real algebraic numbers and rational numbers (e.g. transformed raster position). Therefore, the algorithms used to compute the

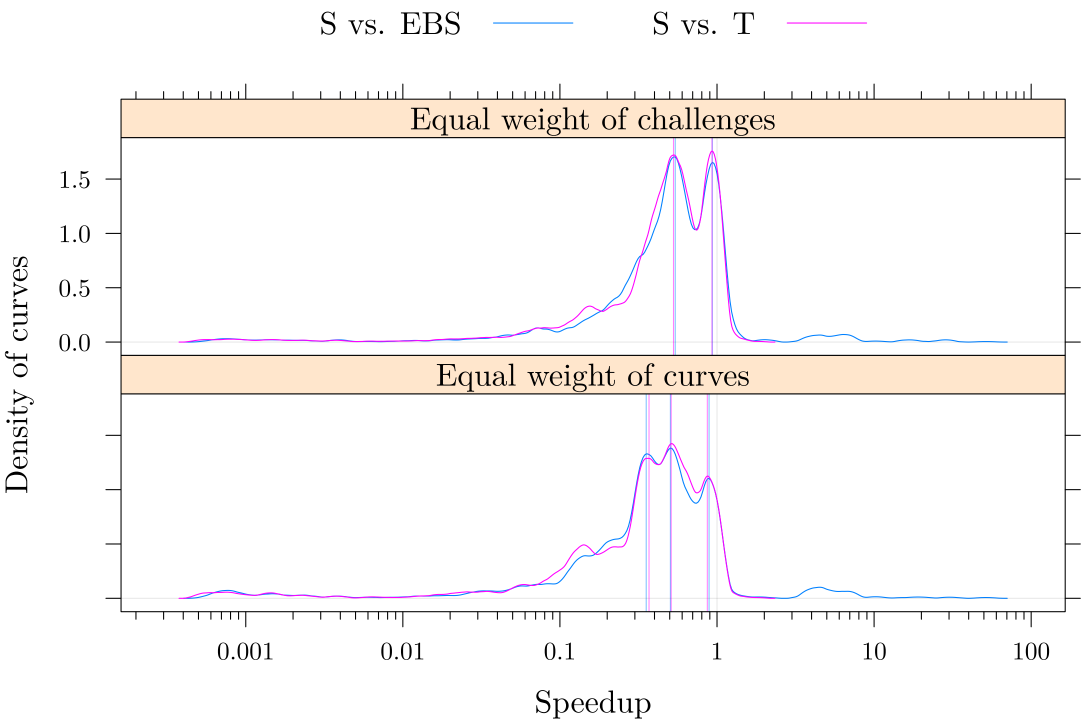


Figure 3.17.: Density plot of the speedup with respect to the adjustment of the viewport.

raster positions of the critical points are very similar in all three tested algorithms. The results shown in Figures 3.17 and 3.18 are comparable to the results of the preprocessing, i.e. there is one peek at a speedup of approximately $\frac{1}{2}$ and one peek at a speedup of approximately 1 for the same reasons presented in Section 3.9.1. The benchmarks only reflect the first adjustment of the viewport after the critical points are isolated. Changing to a different viewport afterwards did not take any considerable amount of time for most of the curves from the randomly chosen subset it has been tested on. This might be due to the excellent caching mechanism in CGAL. Therefore, no additional benchmark results are included for this task.

3.9.3. Rasterization

We now examine the speedups for the main contribution: the rasterization. The results are illustrated in Figures 3.19 and 3.20 for two different resolutions of the viewport. Algorithm S outperforms the EBS and T algorithms for most of the tested curves. The speedups are scattered over the approximate range of $[-\frac{1}{2}, 100]$ (ignoring the small amount of outliers). The density plot based on the equal weight of the challenges (which probably allows a fairer evaluation) shows the highest density at the following approximate speedups: 13.6 (T,1024×1024), 9.4 (T,512×512), 2 (EBS,1024×1024) and 1.5 (EBS, 512×512). Since the data for the rasterization is scattered over a wide range, the numerical values of the quartiles are also supplied in Table 3.3.

The results emphasize that the more sophisticated algorithms S and EBS are usually preferable over the improved trivial algorithm T. The speedup even increases from the 512×512 resolution to the 1024×1024 resolution. The EBS algorithm is also considerably slower than the new

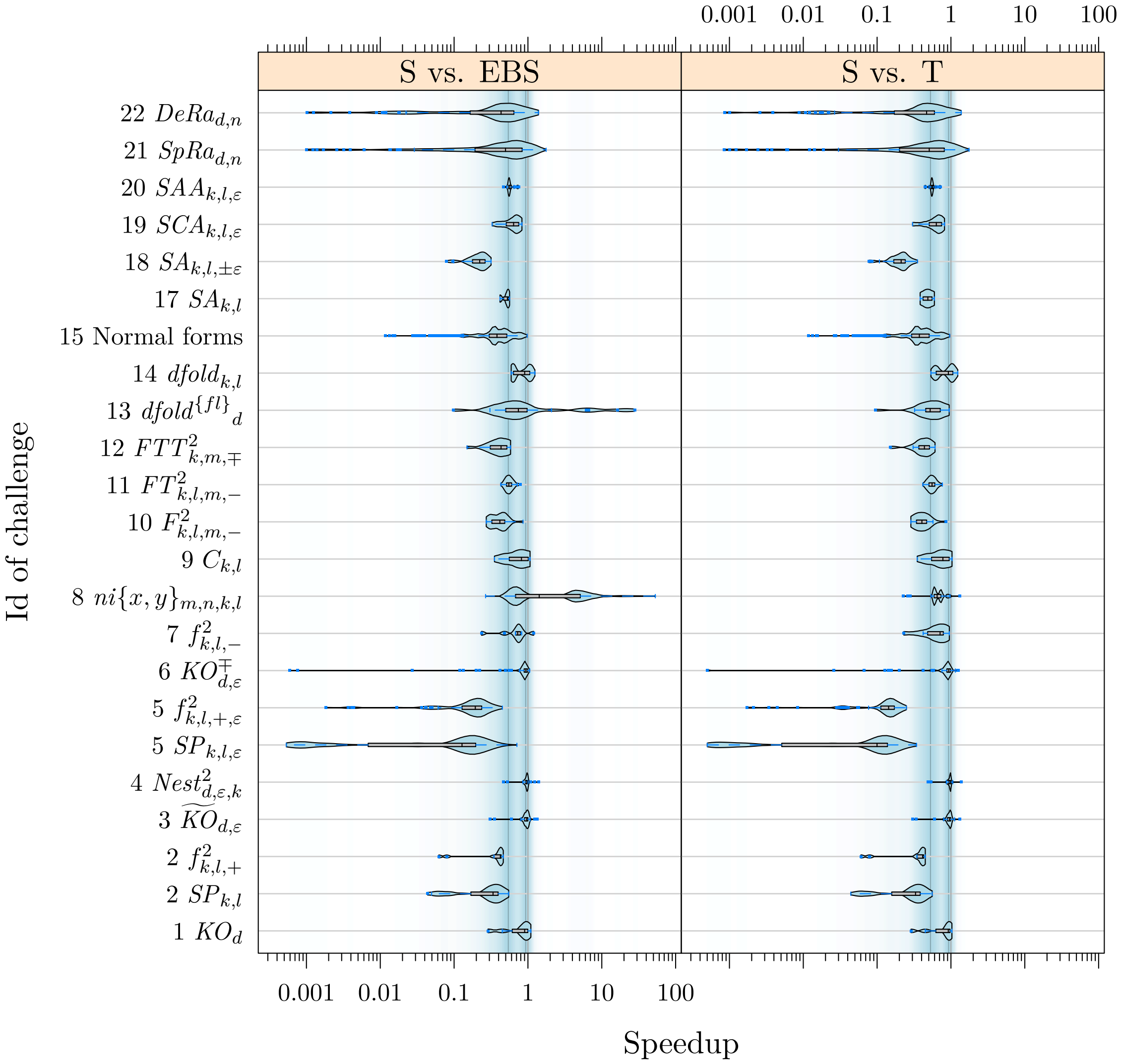


Figure 3.18.: Violin plot of the speedup with respect to the adjustment of the viewport.

	512		1024	
	S vs. EBS	S vs. T	S vs. EBS	S vs. T
Lower quartile	1.31	4.52	1.35	5.96
Median	2.00	8.16	2.01	11.82
Upper quartile	3.22	12.27	2.95	17.65

Table 3.3.: Lower quartile, upper quartile and median of the speedups with respect to the rasterization. The lower and upper quartiles are also shown graphically in Figure 3.19.

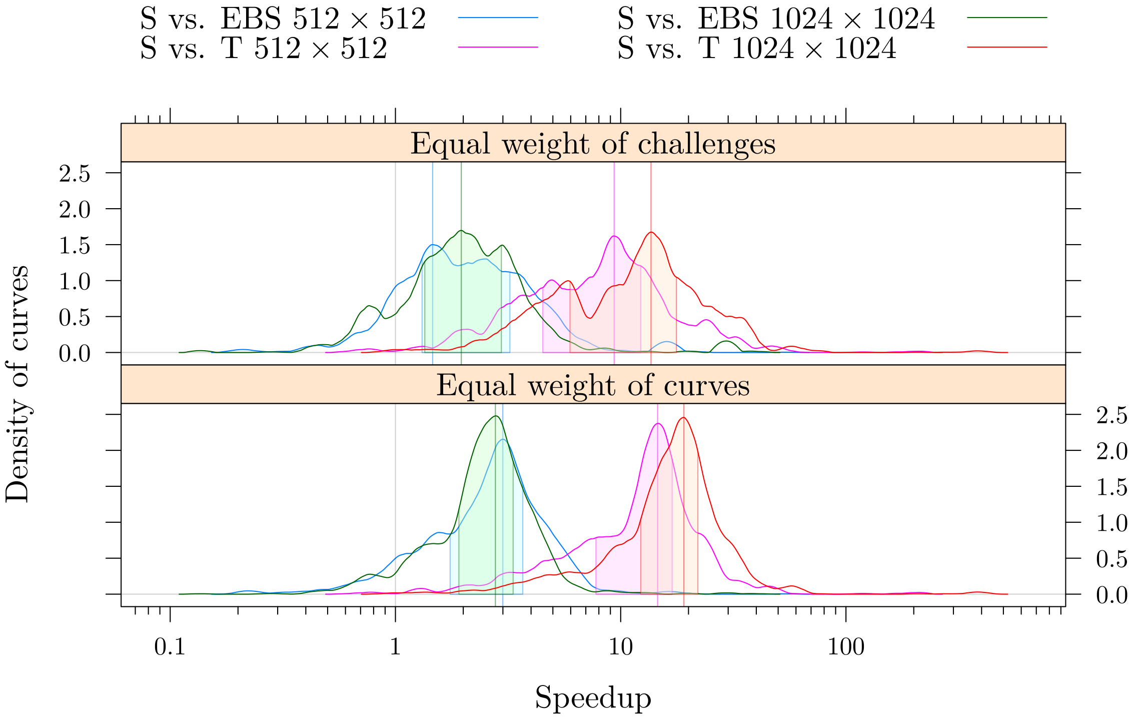


Figure 3.19.: Density plot of the speedup with respect to the rasterization. The colored regions below the curves outline the speedups of 50% of the curves of which the first half is below and the seond half is above the median.

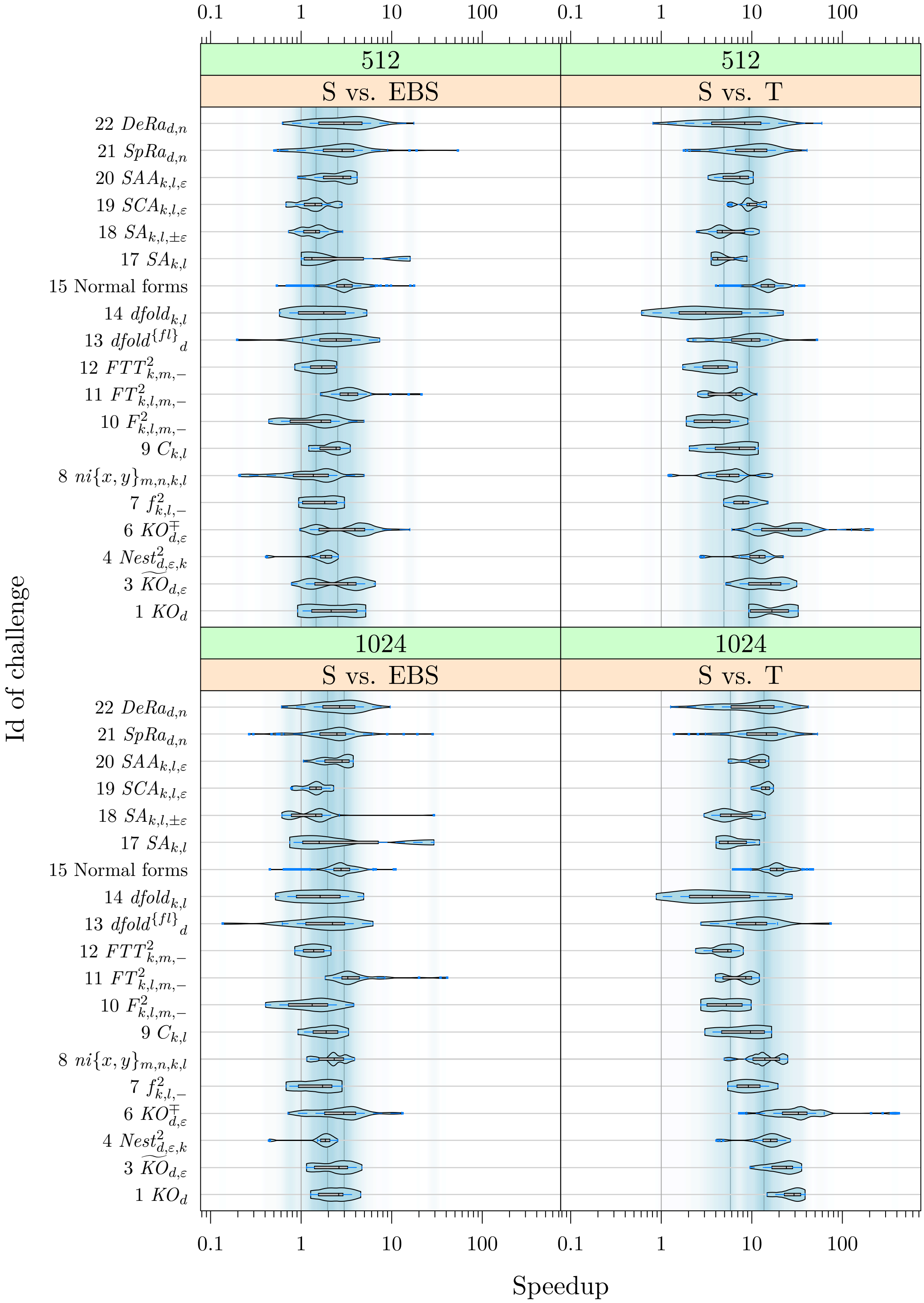


Figure 3.20.: Violin plot of the speedup with respect to the rasterization.

algorithm S although the difference is much smaller than in the case of algorithm T. The speedup is approximately the same for both tested resolutions. There are no significant changes in the speedup with respect to the individual challenges as illustrated in Figure 3.20. This suggests that algorithm S and EBS exhibit a similiar complexity in practice. Clearly, this statement cannot be verified given the speedups at only two different resolutions. Further tests would be necessary. Due to the time consuming precomputation for curves of higher degree, a repetition of the benchmarks is out of the scope of this thesis. Apart from that, the speedups are hovering around the value two for both tested resolutions. A plausible explanation for this behavior is the difference in the precomputation. While algorithm S computes critical points with respect to both coordinate directions, the EBS algorithm uses only one direction. Therefore, the EBS algorithm has to check more possible directions of curve progression during the curve rasterization. It would be interesting to verify this assumption by modifying the EBS algorithm to make use of the additional set of critical points we use in algorithm S.

3.10. Conclusions and future work

A new curve tracing based rasterization method for real algebraic plane curves has been presented. With respect to the above mentioned definition of correctness, it yields exact results in all cases. The algorithm gains its efficiency from a simple test for a sign change of the defining polynomial between pixel corners. In tough cases, it uses real root counting based on signed subresultants in order to avoid repeated subdivision of the image space below pixel level. Numerical filtering for both, the test for a sign change and the real root counting, additionally allows to avoid most of the exact computations without violating the exactness of the rasterization.

We have shown that the new algorithm is superior to the trivial, real root isolation based algorithms from a theoretical point of view (although the improvement is only small). If we restrict the real root isolation methods to those actually used in practice, i.e. which do not use asymptotically fast Taylor shifts, the complexity of the new algorithm is at least one order of magnitude faster than the improved trivial one.

The practical efficiency of the new method, the improved trivial one and the method described in [EBS09] has been tested by means of a large number of challenging curves. It has been demonstrated that the new algorithm outperforms the other ones during the rasterization at the cost of a more expensive precomputation. Therefore, the new method can be recommended in cases where the curve is explored interactively, i.e. if the curve is rendered repeatedly with different viewports. In such cases, the additional time spent for the precomputation might not pose a limitation. Furthermore, for curves of sufficiently small degree (e.g. 10) the precomputation usually finishes within a few seconds or even milliseconds.

Currently, the implementation of the rasterization stage uses the degrees of the bivariate signed subresultants for numerical filtering which have been created during the precomputation. Although it has been shown in Section 3.9.1.1 that this computation has only a small share on total cost in the current implementation, future algorithms for solving bivariate systems might not utilize the signed subresultants. In this case, computing the bivariate subresultant sequence separately would impose an additional overhead.

As we are only interested in the degrees of the subresultants with respect to the outermost variable, several different approaches could be investigated. Let us consider $\mathrm{SRes}_j(A, B)$ for $A, B \in \mathbb{Z}[x][y]$ with $\deg_y(B) \leq \deg_y(A) = n$. A specialization of $\mathrm{SRes}_j(A, B)$ will usually have the same degree of the outermost variable as $\mathrm{SRes}(A, B)$. The degree only decreases if $\overline{\mathrm{sres}}_j(A, B)(x) = 0$, which has only a finite number of roots. Hence, a specialization at a random evaluation point will in most cases result in a univariate subresultant polynomial of correct degree. However, it is hard to guarantee the correctness without testing a sufficiently large number of random evaluation points, i.e. more than the maximum possible number of real roots of the leading coefficient of the bivariate subresultant. A better approach might be to deterministically choose an evaluation point which is definitely no real root. We can bound the bitsize of the coefficients of $\overline{\mathrm{sres}}_j(A, B)$ by Lemma 2.3.29 and consequently also the absolute value of its roots (see Lemma 2.4.3). Therefore, we choose an evaluation point $a \in \mathbb{Z}$ larger than the real root bound for all $\overline{\mathrm{sres}}_j(A, B)$ and compute $\mathcal{S} = [\mathrm{SRes}_0(A(a, y), B(a, y)), \ldots, \mathrm{SRes}_n(A(a, y), B(a, y))]$. The univariate polynomials in $\mathcal{S}$ have the same y degree as the bivariate signed subresultants. Computing the bivariate sequence is hence superfluous. This removes the limitation on the choice of algorithms for bivariate system solving.

4. Robust graphical display of real algebraic space curves

4.1. Motivation

In this chapter, we will investigate the rendering of algebraic space curves given as the intersection of two algebraic surfaces $A = B = 0$, $A, B \in \mathbb{Q}[x, y, z]$. Such intersections are of interest in Computer Aided Design and Geometric Modeling. Furthermore, rendering algebraic space curves provides the link between the visualization of real algebraic plane curves and real algebraic surfaces. The intersection of the surface and its so-called polar surface gives an important curve on the surface called the silhouette curve or apparent contour. This space curve contains all the singularities and consequently also all one dimensional real components of the surface.

The herein proposed method for rendering real algebraic space curves is based on projection and lifting. It partially builds upon the approach taken in [DMR08] for the computation of the topology of real algebraic space curves. Although the newly presented method does not compute the topology, it borrows a fast lifting method that can be applied in situations where the curve is sufficiently generic, i.e. there is a birational map between $V(A, B)$ and its projection in the direction of the z axis for all but a finite number of points.

The main contribution presented in this chapter is the weakening of the genericity constraints. Common one-dimensional asymptotes of $V(A)$ and $V(B)$ do not pose a problem anymore so that the lifting method of [DMR08] can be applied. More tough violations of the genericity condition of [DMR08] are solved by applying a different lifting method. The new method has the advantage that the space curve is rendered in the exact same system as it is defined in. The approach works for all but finitely many points on the space curve. Hence, it is possible to create arbitrarily close approximations of any one dimensional component of the space curve. Before introducing the new algorithm, we will review related work in the next section.

4.2. Related work

We will distinguish between algorithms which are designed for computing the topology and algorithms which are used for rendering algebraic space curves in the first place.

Knowing the topological configuration of a space curve usually simplifies its graphical display. This is especially the case if the topology information is coupled with geometric information such as the location of critical and singular points as well as at least one point on each segment in between. Often, the methods applied for computing such points can easily be extended for rendering the space curve. Although this approach is not necessarily efficient, some good examples such as [DMR08] exist. There, the curve is sheared into generic position so that no two one-dimensional curve components have the same projection when mapped to the x, y-plane. This allows to rationally parametrize the space curve points with respect to the coordinates of their projection based on the coefficients of certain subresultant polynomials.

The basis for [DMR08] is [GN02] where the computation of the topology of plane curves is studied. In [AR05], the topology is determined using a symbolic-numeric algorithm that is based on approximate GCDs of polynomials. The correctness of the result is not guaranteed in all cases. The method of [Gat+05] recovers the topology of a reduced space curve in generic position using two projections in different directions. [El 08] propose a method to analyze algebraic space curves defined by polynomials $A_1, \dots, A_k \in \mathbb{Q}[x, y, z]$. Their method requires the space curve to be in reduced form and therefore incorporates the computation of radical ideals and Gröbner bases. The method described in [Mou+06] also assumes a reduced curve but additionally restricts to curves defined by only two polynomials. This poses a problem in practice if the basis of the computed radical ideal has more than two elements. Furthermore, it is required that the space curve has no asymptote in the direction of projection and that no two critical points of the space have the same projection. The method assumes exact arithmetic and utilizes linear algebra tools similar to subresultants and Eigenvalue computations for the lifting. It is left open how the Eigenvalues of fiber polynomials above a sampling point of the projection can be computed efficiently in practice since these polynomials have algebraic number coefficients.

In some cases, visualization methods for space curves can be derived from rendering methods for plane curves. Some space covering approaches are especially easy to extend. Good examples are [Tau94] and most of the algorithms mentioned in the survey [Mar+02]. The used tests for excluding regions of the plane directly yield exclusion tests for regions of three-dimensional space. Curve tracing algorithms like [MY95; FYK97], which utilize the tangents of the plane curve, can be extended by using the tangents of the space curve given by the cross product of the gradients of A and B. This simple approach breaks down if the two gradient vectors are linearly dependent. Although this problem also occurs in [Baj+88], they are able to trace curves which have a singularity at the origin. The approach in [GW89] is similar to the one in [DMR08] in the sense that a rational map between a projection and the space curve is derived. However, the change of coordinates required to ensure the genericity condition requires to move over to projective space. Furthermore, no topology information is computed.

In contrast to space covering methods, sampling algorithms for plane curves are usually not applicable for space curves since space curves have codimension two. Nevertheless, many methods for the visualization of plane curves can serve as a basis for space curve visualization algorithms that are based on projections of the space curve to the plane such as [GW89; Gat+05; DMR08] as well as the approach presented here.

4.3. Algorithm outline

This section provides a brief overview of the new algorithm for rendering real algebraic space curves implicitly defined by the common zero set of polynomials $A = \sum_{i=0}^{m} a_i(x, y) z^i, B = \sum_{i=0}^{n} b_i(x, y) z^i \in \mathbb{Q}[x, y, z]$, $m \leq n$. We assume that $V(A, B)$ is of dimension one, i.e. $\gcd(A, B) = 1$. This overview is also summarized graphically in Figure 4.1.

The algorithm is based on the computation of the signed subresultant sequence of A and B with respect to z. The projection of $V(A, B)$ to the (x, y) plane is given by the resultant $\mathrm{SRes}_0(A, B)$. In order to lift the projection into three dimensional space, we search for appropriate surfaces containing the space curve. To that end, we compute a squarefree factorization $\prod_{i=1}^{m} \Delta_i(x, y)^i$ of the resultant, i.e. the $\Delta_i(x, y)$ are squarefree polynomials. As a result of the GCD property of subresultants (Theorem 2.3.19), a lifting surface for all but

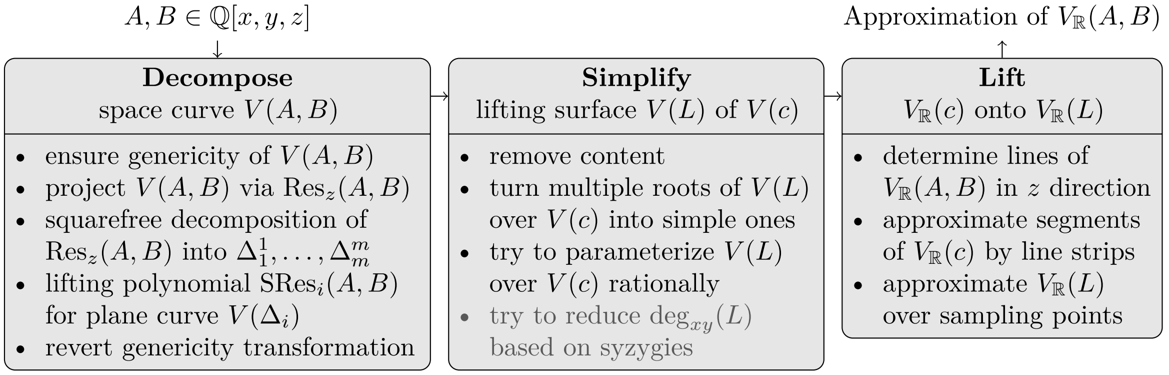

$A, B \in \mathbb{Q}[x, y, z]$
Approximation of $V_{\mathbb{R}}(A, B)$
Decompose
space curve $V(A, B)$
ensure genericity of $V(A, B)$
project $V(A, B)$ via $\mathrm{Res}_z(A, B)$
squarefree decomposition of $\mathrm{Res}_z(A, B)$ into $\Delta_1^1, \ldots, \Delta_m^m$
lifting polynomial $\mathrm{SRes}_i(A, B)$ for plane curve $V(\Delta_i)$
revert genericity transformation
Simplify
lifting surface $V(L)$ of $V(c)$
remove content
turn multiple roots of $V(L)$ over $V(c)$ into simple ones
try to parameterize $V(L)$ over $V(c)$ rationally
try to reduce $\deg_{xy}(L)$ based on syzygies
Lift
$V_{\mathbb{R}}(c)$ onto $V_{\mathbb{R}}(L)$
determine lines of $V_{\mathbb{R}}(A, B)$ in z direction
approximate segments of $V_{\mathbb{R}}(c)$ by line strips
approximate $V_{\mathbb{R}}(L)$ over sampling points

Figure 4.1.: Outline of the rendering algorithm for real algebraic space curves.

finitely many points of $V(\Delta_i)$ is given by $\mathrm{SRes}_i(A, B)(x, y, z)$. Unfortunately, this construction breaks down if $V(A)$ and $V(B)$ have a common asymptote over $V(c)$ where c is a factor of $\mathrm{Res}_z(A, B)$. In this case, c is also a factor of all $\mathrm{SRes}_i(A, B)$, $0 \leq i \leq n$ so that none of the $\mathrm{SRes}_i(A, B)$ can be selected as a lifting surface. In order to solve this issue, we perform a simple coordinate transformation of A and B which maps $z = \pm\infty$ onto a real value such that the transformed varieties do not share a common one-dimensional asymptote in z direction. The transformation is reverted after the lifting surfaces have been determined so that the final lifting takes place in the original coordinate system.

Lifting a component $V(c)$ of the projection is done using two different approaches: The first one is applicable if the lifting surface $V(L)$ has only a single (possibly multiple) root above the plane curve $V(c)$ except for the points in the finite set $V(c, \mathrm{lcoeff}(L))$. In this case, the z coordinate of a point p on the corresponding real component $V_{\mathbb{R}}(c, L) \setminus (V_{\mathbb{R}}(\mathrm{lcoeff}(L)) \times \mathbb{R})$ of the space curve $V(A, B)$ is obtained as a rational function of its projection. This is the lifting method proposed in [DMR08]. If this approach fails, we use certified real root counting of polynomials with approximate coefficients. The approximation is based on interval arithmetic and is refined if the precision is not sufficient in order to determine exact isolating intervals for the real roots. Since the known algorithms for this task only terminate if the underlying real polynomial is squarefree, we need to compute its squarefree part. The exact coefficients are real algebraic numbers so that it is costly to do this for each point on $V(c)$. This is solved by applying subresultants a second time: For a factor c of $\mathrm{Res}(A, B)$ and its associated lifting polynomial L, we decompose c into factors c_j such that $\mathrm{SResV}_{j-1}(L, \frac{\partial}{\partial z} L)(x, y, z)$ is a lifting polynomial for $V(c_j)$ which is squarefree w.r.t the z direction for all but finitely many specializations $(x, y) \in V(c_j)$. Hence, the certified numerical real root isolation method can be applied for almost all points on the curve.

Although we have seen in Section 2.3.1 that the coefficients of subresultants have moderate size compared to other methods, the coefficients are still quite large. In Section 4.6.4, we will investigate heuristic methods for the reduction of the size of the coefficients by searching for lifting polynomials of lower degree. These heuristics significantly speed up the symbolic precomputation and the subsequent lifting process in some cases.

In the following sections, we will describe and discuss the above method in detail. The different steps and approaches will be illustrated by means of several examples. Unfortunately, it is not easy to find a single example which covers all cases, has moderate complexity and yields nice and easy to comprehend visual results at the same time. The surface and plane curve visualizations have been computed using the program SURF [End+10] while the visualizations containing only space curves have been created with the new algorithm. Since the space curves are usually highlighted on the surfaces, it is important to stress that in most cases a-priori knowledge of the geometry of the space curves was necessary to render correct images using SURF. In addition, several surface images have been edited manually since SURF failed to produce correct results. The images produced by the algorithms presented here do not require any a-priori knowledge and have not been edited in any way.

4.4. Squarefree factorization of the projected curve

We already know from Section 2.3.5 that the projection of $V(A,B)$ onto the (x,y) plane is contained in the zero set of $\mathrm{Res}_z(A,B)$. Since several components of $V(A,B)$ may have the same projection, it is possible that $\mathrm{Res}_z(A,B)$ contains multiple factors. In addition, there might be space curve components of higher multiplicity. Under some genericity assumptions, a squarefree factorization of $\mathrm{Res}_z(A,B) \in \mathbb{Q}[x,y]$ can be stated easily in terms of the following polynomials, which are determined on the basis of the sequence of principal signed subresultant coefficients of A and B.

Definition 4.4.1. *Consider the sequence* $\mathcal{S}(A,B) = [\mathrm{SRes}_0(A,B), \ldots, \mathrm{SRes}_m(A,B)]$ *of signed subresultants and let* $h(x,y)$ *be the squarefree part of* $\mathrm{SRes}_0(A,B)$*. Then we define*

$$\Theta_0(x,y) = h(x,y), \tag{4.1}$$

$$\Delta_0(x,y) = 1, \tag{4.2}$$

and for $1 \leq i \leq m$

$$\Theta_i(x,y) = \gcd(\Theta_{i-1}(x,y), \mathrm{sres}_i(x,y)), \tag{4.3}$$

$$\Delta_i(x,y) = \frac{\Theta_{i-1}(x,y)}{\Theta_i(x,y)}. \tag{4.4}$$

Theorem 4.4.2. *If* $\Theta_m(x,y) = 1$*, the sequence* $\mathcal{D} = [\Delta_1, \ldots, \Delta_m]$ *is a decomposition of* $h(x,y)$ *into factors such that*

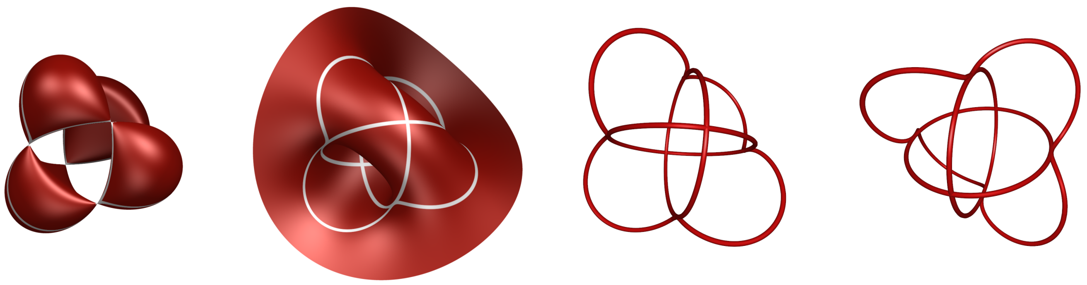
$$h(x,y) = \prod_{i=1}^{m} \Delta_i(x,y) \tag{4.5}$$

and

$$\mathrm{Res}_z(A,B) = c \prod_{i=1}^{m} \Delta_i(x,y)^i, \tag{4.6}$$

for some constant $c \in \mathbb{Q}$ *i.e.* $\mathcal{D}$ *provides a squarefree factorization of* $\mathrm{Res}_z(A,B)$.

Proof. Obviously, $\prod_{i=1}^{m} \Delta_i = \prod_{i=1}^{m} \frac{\Theta_i}{\Theta_{i-1}} = \frac{\Theta_0}{\Theta_m} \prod_{i=1}^{m-1} \frac{\Theta_i}{\Theta_i} = \Theta_0 = h$. This proves the first result. To prove the second part of the theorem, consider the GCD property shown in Theorem 2.3.19 and the specialization properties of Lemmata 2.3.32 and 2.3.33. By construction, $\Delta_i \mid \mathrm{sres}_{i'}$ for $0 \leq i' < i$ but $\Delta_i \nmid \mathrm{sres}_i$. Therefore, the polynomial SRes_i is a GCD of degree i of A and B for each specialization at a point in $V(\Delta_i) \setminus V(\mathrm{sres}_i)$, which means that A and B have i common solutions over $V(\Delta_i) \setminus V(\mathrm{sres}_i)$. Since $\gcd(\Delta_i, \mathrm{sres}_i) = 1$, the polynomial Δ_i occurs as a factor of multiplicity i in $\mathrm{Res}_z(A,B)$. □



(a) $V(A_1)$ (b) $V(B_1)$ (c) $V(A_1, B_1)$ (d) $V(A_1, B_1)$

Figure 4.2.: Visualizations of the surfaces and intersection curves as defined in Example 4.4.3. (c) and (d) show two different views of the intersection curve.

As outlined in the above proof, the surface $V(\mathrm{SRes}_i(A, B))$ contains the common roots of A and B over $V(\Delta_i) \setminus V(\mathrm{sres}_i)$. Thus, $V(\mathrm{SRes}_i(A, B))$ can serve as a lifting surface for $V(\Delta_i)$ except for the finite set of points $V(\Delta_i, \mathrm{sres}_i)$. Note that Theorem 4.4.2 is only applicable if $\Theta_m(x, y) = 1$. Unfortunately, this does not hold true if the GCD of $\mathrm{lcoeff}(A)$ and $\mathrm{lcoeff}(B)$ is non-constant. This issue is addressed in the next section. Prior to that we will have a look at an example.

Example 4.4.3. Consider the polynomials

$$A_1 = z^4 + (2x^2 + 2y^2 - 10)z^2 + 8xyz + x^4 + 2x^2y^2 - 10x^2 + y^4 - 10y^2 + 25, \tag{4.7}$$

$$B_1 = \frac{\partial}{\partial z} A_1 = 4z^3 + (4x^2 + 4y^2 - 20)z + 8xy, \tag{4.8}$$

whose vanishing sets and intersections are shown in Figures 4.2a and 4.2b. The subresultant sequence of A_1 and B_1 is $\mathrm{SRes}_4(A_1, B_1) = A_1$, $\mathrm{SRes}_3(A_1, B_1) = B_1$ and

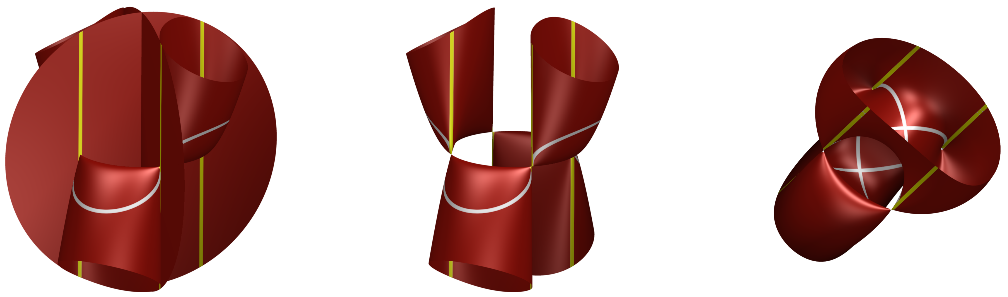
$$\begin{aligned} \mathrm{SRes}_2(A_1, B_1) &= 16((x^2 + y^2 - 5)z^2 + (6xy)z \tag{4.9}\\ &\quad + (x^4 + 2x^2y^2 - 10x^2 + y4 - 10y^2 + 25)), \tag{4.10}\\ \mathrm{SRes}_1(A_1, B_1) &= 256((9x^2y^2)z + (2x^5y + 4x^3y^3 - 20x^3y + 2xy^5 - 20xy^3 + 50xy)), \tag{4.11}\\ \mathrm{SRes}_0(A_1, B_1) &= 4096(4x^8y^2 + 12x^6y^4 + 12x^4y^6 + 4x^2y^8 - 60x^6y^2 - 147x^4y^4 \\ &\quad - 60x^2y^6 + 300x^4y^2 + 300x^2y^4 - 500x^2y^2). \tag{4.12} \end{aligned}$$

The squarefree part of $\mathrm{SRes}_0(A_1, B_1)$ is

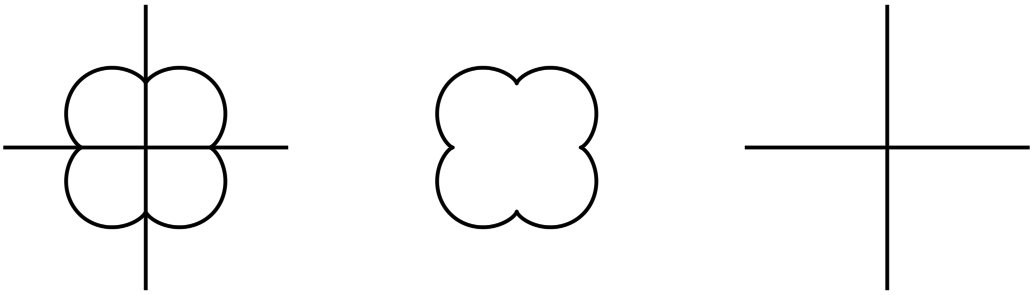
$$\begin{aligned} h(x, y) &= x^7y + 3x^5y^3 + 3x^3y^5 + xy^7 - 15x^5y - \tfrac{147}{4}x^3y^3 \tag{4.13}\\ &\quad - 15xy^5 + 75x^3y + 75xy^3 - 125xy. \tag{4.14} \end{aligned}$$

Applying Definition 4.4.1 yields the following Θ and Δ polynomials:

$$\begin{aligned} &\Theta_4(A_1, B_1) = 1 && \Delta_4(A_1, B_1) = 1 \tag{4.15}\\ &\Theta_3(A_1, B_1) = 1 && \Delta_3(A_1, B_1) = 1 \tag{4.16}\\ &\Theta_2(A_1, B_1) = 1 && \Delta_2(A_1, B_1) = xy \tag{4.17}\\ &\Theta_1(A_1, B_1) = xy && \Delta_1(A_1, B_1) = x^7y + 3x^5y^3 + 3x^3y^5 + xy^7 - 15x^5y \\ & && \quad - \tfrac{147}{4}x^3y^3 - 15xy^5 + 75x^3y + 75xy^3 - 125xy \tag{4.18}\\ &\Theta_0(A_1, B_1) = h(A_1, B_1) && \Delta_0(A_1, B_1) = 1. \tag{4.19} \end{aligned}$$



(a) $V(\mathrm{SRes}_1(A_1, B_1))$ (b) $V\left(\frac{\mathrm{SRes}_1(A_1,B_1)}{\Theta_1}\right)$ (c) $V(\mathrm{SRes}_2(A_1, B_1))$



(d) $V(\mathrm{SRes}_0(A_1, B_1))$ (e) $V(\Delta_1(A_1, B_1))$ (f) $V(\Delta_2(A_1, B_1))$

Figure 4.3.: Visualization of the squarefree factorization of $\mathrm{SRes}_0(A_1, B_1)$ into Δ_i factors and the respective subresultant lifting surfaces following Example 4.4.3. The space curve above each $V(\Delta_i)$ is plotted in white. The yellow lines (in the direction of projection) illustrate the fibers above the intersections of the leading coefficient of the lifting surface with $V(\Delta_i)$. Along these fibers the lifting surface can not be used to lift Δ_i. Note that different rotations of the lifting surfaces have been chosen in order to present the geometry in the best possible way.

By definition, $\Theta_i(A_1, B_1)$ divides $\mathrm{sres}_i(A_1, B_1)$ and therefore also $\mathrm{SRes}_i(A_1, B_1)$. Thus, we can easily remove some of the content of $\mathrm{SRes}_1(A_1, B_1)$. $\mathrm{SRes}_1 / \Theta_1$ provides a better lifting polynomial then SRes_1 due to its lower total degree. See Figure 4.3 for a visualization of the zero set of the involved polynomials.

4.5. Non-constant leading coefficients

As outlined in Section 4.3, we want to lift the space curve above the component $V(\Delta_i)$ of its projection $V(\mathrm{Res}_z(A, B))$ using the surface $V(\mathrm{SRes}_i(A, B))$. If the leading coefficients of A resp. B are non-constant polynomials, the procedure can fail for several reasons which we are going to cover in this section.

4.5.1. The content

First of all, note that we assumed $\gcd(A, B) = 1$. Then the content $\mathrm{cont}_z(A, B)$ only poses a problem if $\deg_z A = \deg_z B$. In this case, A does not occur as an element of the subresultant sequence since $\mathrm{SRes}_m(A, B) = \mathrm{SRes}_n(A, B) = B$ is the last element (cf. Definition 2.3.4). By Lemma 2.3.35, $\mathrm{cont}_z(A)$ and $\mathrm{cont}_z(B)$ are factors of $\mathrm{Res}_z(A, B)$. The squarefree part of

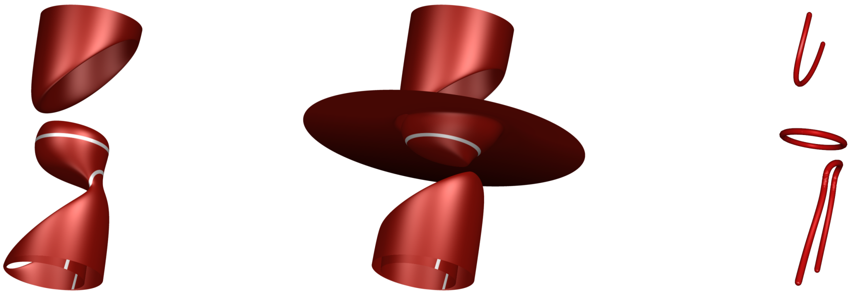

(a) $V(A_2)$ (b) $V(B_2)$ (c) $V(A_2, B_2)$

Figure 4.4.: Visualizations of the surfaces and the space curves discussed in Example 4.5.1. The surfaces $V(A_2)$ and $V(B_2)$ both have asymptotes over $V(x^2+y^2-1)$ and $V(x^2+y^2-2)$ and intersect at $z=0$ in $V(x^2+y^2-1)$.

$\operatorname{cont}_z(A)$ does not divide B since $\gcd(A,B)=1$. Hence, B serves as the lifting polynomial for $V(\operatorname{cont}_z(A))$. In contrast, $\operatorname{cont}_z(B) \mid B$ so that the squarefree part of $\operatorname{cont}_z(B)$ is a factor of Θ_m. This prevents the application of Theorem 4.4.2.

The solution is simple. We just add A to the end of the subresultant sequence. $\operatorname{cont}_z(B) \nmid A$ so that A is correctly determined as the lifting polynomial for $V(\operatorname{cont}_z(B))$. Note that $\operatorname{cont}_z(A)^n \operatorname{cont}_z(B)^m$ is a factor of $\operatorname{Res}_z(A,B)$. In order to speed up the computation of $h(x,y)$, we can also determine it as the squarefree part of

$$\operatorname{cont}_z(A)\operatorname{cont}_z(B)\operatorname{Res}_z(\operatorname{pp}_z(A),\operatorname{pp}_z(B)), \tag{4.20}$$

where $\operatorname{pp}_z(P)=\frac{P}{\operatorname{cont}_z(P)}$ is the primitive part of a polynomial $P \in R[x,y,z]$ with respect to z.

4.5.2. Asymptotes

By Lemma 2.3.36, $c=\gcd(\operatorname{lcoeff}(A),\operatorname{lcoeff}(B))$ appears as a factor of each principal subresultant coefficient. Therefore, it is also a factor of Θ_{m-1} and $\operatorname{lcoeff}(A)$. In this case $c \mid \Theta_m$, i.e. c is not a factor of any of the Δ_i so that no lifting surface is associated to the plane curve $V(c)$.

Geometrically, a vanishing leading coefficient $\operatorname{lcoeff}(A)$ of a polynomial $A \in \mathbb{R}[x,y][z]$ is equivalent to $V(A)$ having an asymptote in the z direction over $V(\operatorname{lcoeff}(A))$ (cf. Lemma 2.2.8). Hence, for a space curve $V(A,B)$, the set $V(c)=V(\operatorname{lcoeff}(A),\operatorname{lcoeff}(B))$ defines the common asymptotes of $V(A)$ and $V(B)$, i.e. the common solutions at infinity in z direction. If there are additional solutions of $A=B=0$ for $z \in \mathbb{R}$ over $V(c)$, these cannot be determined since no lifting surface is associated with c.

Example 4.5.1. Consider the surfaces defined by the polynomials

$$A_2=(x^2+y^2-2)((x^2+y^2-1)z^4+xz^3)+z^2+(x^2+y^2-1)z+(x^2+y^2-1), \tag{4.21}$$

$$B_2=\frac{\partial}{\partial z}A_2=4(x^2+y^2-2)((x^2+y^2-1)z^3+3xz^2)+2z+(x^2+y^2-1). \tag{4.22}$$

These surfaces have two different intersections at infinity: One above the circle $x^2+y^2-2=0$ and one above the circle $x^2+y^2-1=0$. In addition, there is the solution $V(x^2+y^2-1,z)$. Due to $\gcd(\operatorname{lcoeff}(A_2),\operatorname{lcoeff}(B_2))=x^2+y^2-1$ each of the $\operatorname{SRes}_i(A_2,B_2)$, $0 \le i < n$, is divisible by the polynomial x^2+y^2-1 so that no lifting surface is associated with it. See Figure 4.4 for a visualization of $V(A_2)$, $V(B_2)$ and $V(A_2,B_2)$.

In order to solve the issues related to asymptotes, we perform a change of coordinates. Unlike many other algorithms (see, e.g., [DMR08; GW89]), we do not shear or rotate the coordinate system. Instead, we map the z axis onto itself. The transformation is reverted after the symbolic precomputation so that the numerical approximation of the space curve is generated in the original coordinate system.

The following method transforms the polynomials A and B such that they do not intersect in a curve at infinity. First, we perform the translation

$$A_{z-\lambda}(x,y,z) = A(x,y,z-\lambda) \qquad B_{z-\lambda}(x,y,z) = B(x,y,z-\lambda). \tag{4.23}$$

The parameter $\lambda \in \mathbb{Z}$ is chosen such that $\gcd(\operatorname{coeff}_0(A_{z-\lambda}), \operatorname{coeff}_0(B_{z-\lambda})) = 1$. The following lemma ensures that there is a deterministic algorithm in order to compute a suitable λ.

Lemma 4.5.2. *The number of values for* $\lambda \in \mathbb{R}$ *such that* $\gcd(\operatorname{coeff}_0(A_{z-\lambda}), \operatorname{coeff}_0(B_{z-\lambda})) \neq 1$ *is bounded by* $2m^2$.

Proof. The GCD condition is violated if $A(x,y,\lambda)$ and $B(x,y,\lambda)$ have a common factor, i.e. $V(A)$ and $V(B)$ have a common one dimensional component at $z = \lambda$. Each of these components occurs as a factor of $\operatorname{Res}_z(A,B)$. By Lemma 2.3.28, we know that $\deg \operatorname{Res}_z(A,B) \leq 2m^2$ giving an upper bound on the number of possible values for λ. ☐

Note that the GCD property is fulfilled with probability one for a random choice of λ. We do not even need to compute the full expansion of $A(x,y,z-\lambda)$ resp. $B(x,y,z-\lambda)$ to check the GCD property. It suffices to determine the GCD of

$$\operatorname{coeff}_0(A(x,y,z-\lambda)) = \sum_{k=0}^{m} (-\lambda)^k a_k(x,y), \tag{4.24}$$

$$\operatorname{coeff}_0(B(x,y,z-\lambda)) = \sum_{k=0}^{n} (-\lambda)^k b_k(x,y). \tag{4.25}$$

The condition $\gcd(\operatorname{coeff}_0(A_{z-\lambda}), \operatorname{coeff}_0(B_{z-\lambda})) = 1$ ensures that $A_{z-\lambda}$ and $B_{z-\lambda}$ do not exhibit a one dimensional intersection at $z = 0$. We now apply a second transformation which interchanges the roles of $z = 0$ and $z = \pm\infty$. This is given by

$$A_{\frac{1}{z}-\lambda}(x,y,z) = z^{\deg_z A} A_{z-\lambda}\left(x,y,\tfrac{1}{z}\right), \tag{4.26}$$

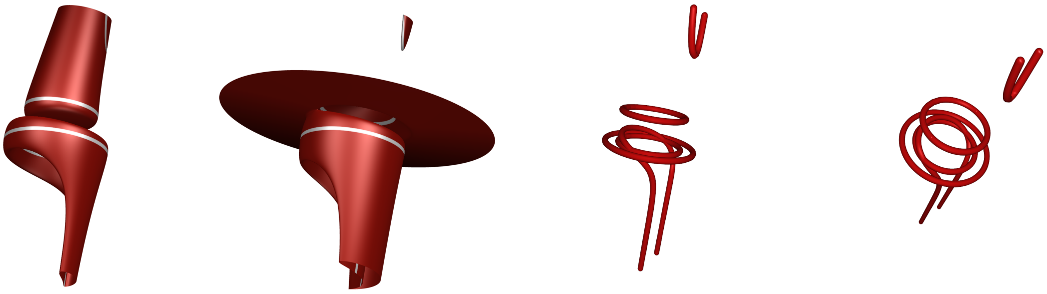
$$B_{\frac{1}{z}-\lambda}(x,y,z) = z^{\deg_z B} B_{z-\lambda}\left(x,y,\tfrac{1}{z}\right). \tag{4.27}$$

Clearly, $\gcd(\operatorname{lcoeff}(A_{\frac{1}{z}-\lambda}), \operatorname{lcoeff}(B_{\frac{1}{z}-\lambda})) = 1$ as required for applying Theorem 4.4.2. Note that the transformation of Equations (4.26) and (4.27) is algorithmically trivial: It can be carried out by reversing the order of the coefficients of $A_{z-\lambda}$ and $B_{z-\lambda}$.

The following result shows that the transformation does not affect the projection of the space curve onto the (x,y) plane.

Lemma 4.5.3. $\operatorname{Res}_z\left(A_{\frac{1}{z}-\lambda}, B_{\frac{1}{z}-\lambda}\right) = \pm \operatorname{Res}_z(A,B)$.

Proof. The chain rule for (sub-)resultants applied to $\operatorname{Res}_z(A(x,y,z-\lambda), A(x,y,z-\lambda))$ yields $\operatorname{Res}_z(A,B) = \pm \operatorname{Res}_z(A_{z-\lambda}, B_{z-\lambda})$ (cf. Lemma 2.3.37). The Sylvester matrix $\operatorname{Syl}\left(A_{\frac{1}{z}-\lambda}, B_{\frac{1}{z}-\lambda}\right)$ is now obtained from $\operatorname{Syl}(A_{z-\lambda}, B_{z-\lambda})$ by reversing the order of its $m+n$ columns and of its first n and last m rows. Hence, the determinants of both matrices are equal up to sign. ☐



(a) $V(A_{2,\frac{1}{z}+1})$ (b) $V(B_{2,\frac{1}{z}+1})$ (c) $V(A_{2,\frac{1}{z}+1}, B_{2,\frac{1}{z}+1})$ (d) $V(A_{2,\frac{1}{z}+1}, B_{2,\frac{1}{z}+1})$

Figure 4.5.: Visualization of the transformed surfaces and of their intersection curves for $\lambda = -1$ as discussed in Example 4.5.1. The two circles at infinity have been mapped to $z = 0$ so that all three circles now have a finite z coordinate.

The above lemma seems natural since the transformation $z \mapsto \frac{1}{z} - \lambda$ only maps the (common) roots of A and B to a different position on the z axis. No solution in $\mathbb{R}^2 \times (\mathbb{R} \cup \infty)$ is added or lost.

If the polynomials A and B do not satisfy $\gcd(\mathrm{lcoeff}(A), \mathrm{lcoeff}(B)) = 1$, we perform the above substitution and apply the algorithm to $A = A_{\frac{1}{z}-\lambda}$ and $B = B_{\frac{1}{z}-\lambda}$. Once a lifting surface is determined for each of the $\Delta_i(x, y)$, we revert the substitution. In order to simplify notation, assume that $L_{\frac{1}{z}-\lambda} \in \mathbb{Q}[x, y, z]$ is the lifting polynomial for a plane curve $V(c)$ in the transformed system. Then

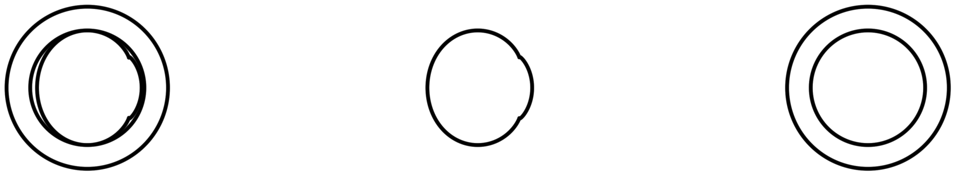
$$L = z^{deg_z L_{\frac{1}{z}-\lambda}} L_{\frac{1}{z}-\lambda}\left(x, y, \frac{1}{z+\lambda}\right) \tag{4.28}$$

provides the lifting polynomial in the original system. The solutions at $z = \pm\infty$ are still present in the lifting surfaces. If $\gcd(\mathrm{lcoeff}(L), c)$ is non-constant, then the lifting surface L shows asymptotes over a one dimensional component of $V(c)$. The vanishing leading coefficient of $L(\alpha, \beta, z)$ for $(\alpha, \beta) \in V(\gcd(\mathrm{lcoeff}(L), c))$ poses a problem for numerical methods when solving $L(\alpha, \beta, z) = 0$ since the point (α, β) is only know approximately. We will see in the next section how this issue can be solved by applying additional symbolical methods. Beforehand, we will continue our current example.

Example 4.5.1 *(continuing from p. 77).* Due to $\gcd(\mathrm{lcoeff}(A_2), \mathrm{lcoeff}(B_2)) = (x^2+y^2-1)(x^2+y^2-2)$, we apply the transformation. Since also $\gcd(\mathrm{coeff}_0(A_2), \mathrm{coeff}_0(B_2)) = \gcd(\mathrm{coeff}_0(A_{2,z-0}), \mathrm{coeff}_0(B_{2,z-0})) = x^2 + y^2 - 1$, we can not choose $\lambda = 0$. It turns out that $\lambda = -1$ is an appropriate choice so that $\gcd(\mathrm{coeff}_0(A_{2,z+1}), \mathrm{coeff}_0(B_{2,z+1})) = 1$. Visualizations of the zero sets of $A_{2,\frac{1}{z}+1}$ and $B_{2,\frac{1}{z}+1}$, which are determined from $A_{2,z+1}$ and $B_{2,z+1}$ by reversing the order of coefficients, are shown in Figure 4.5. The transformed space curve is illustrated there, too.

4.5.3. Curves at infinity of different multiplicity

Let $V(L)$ be the lifting surface associated with the plane curve $V(c)$. We know that $\mathrm{lcoeff}(L)$ vanishes for all points in $V(\gcd(\mathrm{lcoeff}(L), c))$. Therefore, we can lift $V(\gcd(\mathrm{lcoeff}(L), c))$ and $V(c/\gcd(\mathrm{lcoeff}(L), c))$ separately using different surfaces. In order to deal with all special cases, we use the following systematic approach.



(a) $V(\mathrm{SRes}_0(A_{2,\frac{1}{z}+1}, B_{2,\frac{1}{z}+1}))$ (b) $V(\Delta_1(A_{2,\frac{1}{z}+1}, B_{2,\frac{1}{z}+1}))$ (c) $V(\Delta_2(A_{2,\frac{1}{z}+1}, B_{2,\frac{1}{z}+1}))$

Figure 4.6.: Visualization of the squarefree factorization of $\mathrm{SRes}_0(A_{2,\frac{1}{z}+1}, B_{2,\frac{1}{z}+1})$ into Δ_i factors following Example 4.5.1.

Definition 4.5.4. *With* $L = \sum_{i=0}^{k} l_i(x,y)z^i \in \mathbb{Q}[x,y,z]$ *and* $c \in \mathbb{Q}[x,y]$ *let*

$$\begin{aligned} c_{k+1}(x,y) &= c(x,y), && (4.29)\\ c_j(x,y) &= \gcd(c_{j+1}(x,y), l_j(x,y)), && (4.30)\\ d_j(x,y) &= \frac{c_{j+1}(x,y)}{c_j(x,y)} && (4.31)\\ L_j(x,y,z) &= \sum_{i=0}^{j} l_i(x,y)z^i && (4.32) \end{aligned}$$

for $0 \le j \le k$.

Lemma 4.5.5. *If* $\gcd(c, L) = 1$ *it holds that* $\prod_{j=0}^{k} d_j(x,y) = c(x,y)$.

Proof. First of all, $c_0 = \gcd(c, L) = 1$. Therefore, $\prod_{j=1}^{k} d_j = \prod_{j=0}^{k} \frac{c_{j+1}}{c_j} = \frac{c_{k+1}}{c_0} \prod_{j=1}^{k} \frac{c_j}{c_j} = c$. □

By definition, d_j divides $l_k, \dots, l_{j+1}$ but the set $V(d_j, l_j) = V(d_j, \mathrm{lcoeff}(L_j))$ is finite. The construction of L and c by Definition 4.4.1 also ensures $\gcd(L, c) = 1$. Hence, we use L_j to lift the space curve above $V(d_j)$. By Lemma 4.5.5, it is clear that no factor of c is lost.

Remark. It is possible that d_0 is non-constant. Since $\deg_z L_0 = 0$, the polynomial L has no real roots over d_0 but only asymptotes.

Example 4.5.1 *(continuing from p. 79).* $\mathrm{SRes}_0(A_{2,\frac{1}{z}+1}, B_{2,\frac{1}{z}+1})$ has two non-constant Δ factors as visualized in Figure 4.6. We skip the processing of Δ_1 and focus on the asymptotes over $V(\Delta_2)$. We get

$$c = \Delta_2(A_{2,\frac{1}{z}+1}, B_{2,\frac{1}{z}+1}) = (x^2+y^2-2)(x^2+y^2-1) \tag{4.33}$$

$$\begin{aligned} L &= z^2\,\mathrm{SRes}_2(A_{2,\frac{1}{z}+1}, B_{2,\frac{1}{z}+1})\left(x, y, \frac{1}{z-1}\right)\\ &= (x^2+y^2-2)(x^2+y^2-1)\tilde{l}_2 z^2 + (x^2+y^2-2)\tilde{l}_1 z + (x^2+y^2-1)\tilde{l}_0 \end{aligned} \tag{4.34}$$

for some $\tilde{l}_2, \tilde{l}_1, \tilde{l}_0 \in \mathbb{Q}[x,y]$, which we leave unspecified for brevity. The actual value of the $\tilde{l}_i$ is

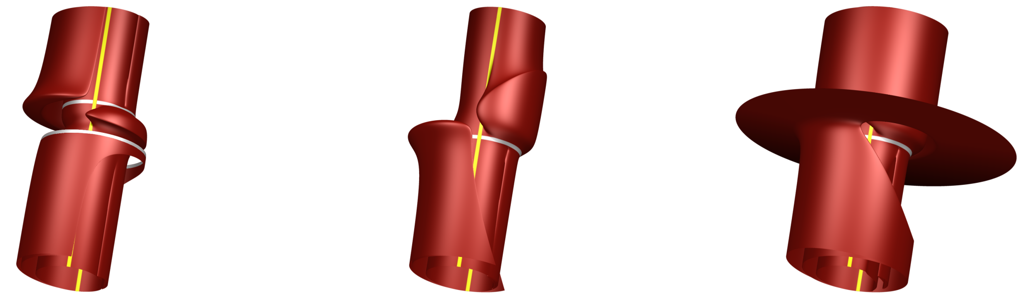

(a) $V(\mathrm{SRes}_2(A_{2,\frac{1}{z}+1}, B_{2,\frac{1}{z}+1}))$ (b) $V(L_2) = V(L)$ (c) $V(L_1)$

Figure 4.7.: Visualization of the lifting surfaces associated with $V(\Delta_2(A_{2,\frac{1}{z}+1}, B_{2,\frac{1}{z}+1}))$. In contrast to (b), (a) shows the lifting surface before the transformation has been reverted (see the intersections at $z = 0$ which correspond to the common asymptotes of $V(A_2)$ and $V(B_2)$). $V(L)$ has several asymptotes over components of $V(\Delta_2(A_{2,\frac{1}{z}+1}, B_{2,\frac{1}{z}+1}))$. In Example 4.5.1, we have seen how to further decompose $\Delta_2(A_{2,\frac{1}{z}+1}, B_{2,\frac{1}{z}+1})$ and L to solve this problem. (c) shows the only remaining lifting surface $V(L_1)$ intersecting with the factor $x^2 + y^2 - 1$ at a finite z coordinate (except for the exceptional fibers shown in yellow).

not relevant for the construction. Applying Definition 4.5.4 yields

$$c_3 = c \tag{4.35}$$

$$c_2 = c \qquad d_2 = 1 \qquad L_2 = L \tag{4.36}$$

$$c_1 = x^2 + y^2 - 2 \qquad d_1 = x^2 + y^2 - 1 \qquad L_1 = (x^2 + y^2 - 2)\tilde{l}_1 z + (x^2 + y^2 - 1)\tilde{l}_0 \tag{4.37}$$

$$c_0 = 1 \qquad d_0 = x^2 + y^2 - 2 \qquad L_0 = (x^2 + y^2 - 1)\tilde{l}_0. \tag{4.38}$$

Thus, the asymptotes over c have been factored out so that only the space curve $V(d_1, L_1)$ remains, i.e. the circle $x^2 + y^2 - 1 = 0$ at $z = 0$. See also Figure 4.7.

4.6. Simplification of the lifting surfaces

We have seen how to decompose the resultant of A and B into factors so that each factor c has an associated lifting surface L with

$$V(L, c) \setminus (V(\mathrm{lcoeff}(L), c) \times \mathbb{R}) \subseteq V(A, B). \tag{4.39}$$

The set $V(\mathrm{lcoeff}(L), c)$ is finite by construction. We will now try to simplify the lifting surface L in an algebraic and geometric sense in order to prepare for the numerical approximation of the space curve.

4.6.1. Eliminating the content

It happens quite often that $\mathrm{cont}_z(L)$ is non-constant. We should divide L by its content in order to speed up subsequent computations, but we have to save it for later reference since it helps to compute a suitable segmentation of the projected curve (see Section 4.7.2). Since $\mathrm{cont}_z(L)$ is the GCD of the coefficients of L, its computation can be costly. If L is one of the subresultants of A and B, we sometimes know a factor of $\mathrm{cont}_z(L)$ in advance: In Definition 4.4.1, the polynomial Θ_i divides sres_i by construction. Thus, it also divides $\mathrm{SRes}_i(A, B)$. This case occurs for $\mathrm{SRes}_1(A_1, B1)$ in Example 4.4.3 (see also Figures 4.3a and 4.3b). It often allows to speed up the computation of the content. In addition, the exceptional set $V(c, \mathrm{lcoeff}(L)/\mathrm{cont}_z(L))$ contains fewer points than $V(c, \mathrm{lcoeff}(L))$ if $\deg(\mathrm{cont}_z(L)) > 0$.

4.6.2. Rational parametrization

The following argument has been utilized in [DMR08] (based on the investigations on plane curves in [GN02]):

Proposition 4.6.1. *If a polynomial* $f = \sum_{i=0}^{k} f_i x^i \in C[x]$ *of degree* k *has a root of multiplicity* k*, then this root is* $\tilde{x} = -\frac{(k-i+1)f_{i-1}}{if_i}$ *for* $0 < i \leq k$*.*

Proof. Let $\tilde{x}$ be the root of multiplicity k. Then $f_i = \binom{i}{k}(-\tilde{x})^{k-i}$ since $f = f_k(x-\tilde{x})^k = \sum_{i=0}^{k}\binom{i}{k}x^i(-\tilde{x})^{k-i}$. It follows that

$$\frac{f_{i-1}}{f_i} = \frac{\binom{k}{i-1}(-\tilde{x})^{k-(i-1)}}{\binom{k}{i}(-\tilde{x})^{k-i}} = -\frac{k!(k-i)!i!}{k!(k-(i-1))!(i-1)!}\tilde{x} = -\frac{i}{k-i+1}\tilde{x} \tag{4.40}$$

for $0 < i \leq k$. ☐

If we can certify that the lifting surface $V(L)$ for $L = \sum_{i=0}^{k} l_i(x,y)z^i$ has only a single root of multiplicity k over $V(c)$ except for finitely many points, then we can set $i = k$ in Proposition 4.6.1 and use the polynomial $L_{\text{linear}} = kl_k(x,y)z + l_{k-1}(x,y)$ instead of L. Hence, a rational parametrization of the space curve above $V(c) \setminus V(\mathrm{lcoeff}(L))$ is given by

$$z(x,y) = -\frac{l_{k-1}(x,y)}{kl_k(x,y)}. \tag{4.41}$$

The following test allows to check if the above is applicable. Although it has been discovered independently by the author of this thesis, it has been published first in [DMR08]. By Proposition 4.6.1, the polynomial L has a root of multiplicity k over a point $(\alpha, \beta) \in V(c) \setminus V(\mathrm{lcoeff}(L))$ if and only if

$$L(\alpha,\beta,z) = \sum_{i=0}^{k} l_i(\alpha,\beta)z^i = l_k(\alpha,\beta)\left(z + \frac{l_{k-1}(\alpha,\beta)}{kl_k(\alpha,\beta)}\right)^k. \tag{4.42}$$

Expanding the right hand side by applying the binomial formula yields

$$\sum_{i=0}^{k} l_i(\alpha,\beta)z^i = l_k(\alpha,\beta)\sum_{i=0}^{k}\binom{k}{i}\left(\frac{l_{k-1}(\alpha,\beta)}{kl_k(\alpha,\beta)}\right)^{k-i} z^i. \tag{4.43}$$

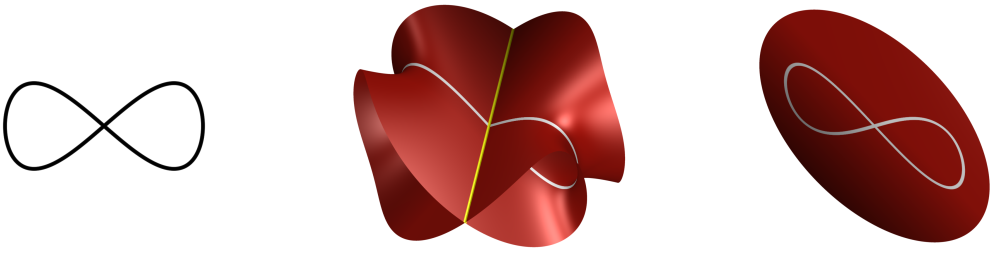
By recursive substitution of coefficients we get

$$l_i(\alpha,\beta) = l_k(\alpha,\beta)\binom{k}{i}\left(\frac{l_{k-1}(\alpha,\beta)}{kl_k(\alpha,\beta)}\right)^{k-i} = \frac{i+1}{k-i}\frac{l_{k-1}(\alpha,\beta)}{kl_k(\alpha,\beta)}l_{i+1}(\alpha,\beta) \tag{4.44}$$

for $0 \leq i < k$. Since $l_k(\alpha,\beta) \neq 0$, this is identical to

$$k(k-i)l_k(\alpha,\beta)l_i(\alpha,\beta) - (i+1)l_{k-1}(\alpha,\beta)l_{i+1}(\alpha,\beta) = 0. \tag{4.45}$$

Finally, we generalize in order to ensure the above condition for all $(\alpha,\beta) \in V(c) \setminus V(\mathrm{lcoeff}(L))$.



(a) $V(c)$ (b) $V(L)$ (c) $V(L_{\text{linear}}/\operatorname{cont}_z(L_{\text{linear}}))$

Figure 4.8.: Visualization of the surface $V(L)$ which intersects tangentially with the cylinder over $V(c)$ as discussed in Example 4.6.3. The polynomial L can be simplified into the polynomial L_{linear} which is linear with respect to z. In (c), the content of L_{linear} has been removed for the visualization.

Theorem 4.6.2. *The polynomial* $L(\alpha, \beta, z)$ *has a root of multiplicity* $\deg_z L = k$ *over all points* $(\alpha, \beta) \in V(c) \setminus V(\operatorname{lcoeff}(L))$ *if*

$$k(k-i)l_k(x,y)l_i(x,y) - (i+1)l_{k-1}(x,y)l_{i+1}(x,y) \equiv 0 \mod c(x,y) \tag{4.46}$$

for all $i \in 0, \ldots, k-1$.

Details on the proof of Theorem 4.6.2 and its converse can be found in [DMR08].

Example 4.6.3. The zero set of the polynomial

$$L = (48x^2 - 64y^2)z^2 + (64xy^2 - 48x^3)z + (9x^2 - 12y^2 - 16x^2y^2) \tag{4.47}$$

intersects tangentially with the cylinder over the zero set of

$$c = 4x^4 - 3x^2 + 4y^2. \tag{4.48}$$

We use Theorem 4.6.2 to check that L has two roots above $V(c) \setminus V(\operatorname{lcoeff}(L))$. For $i = 1$, we have

$$2(48x^2 - 64y^2)(64xy^2 - 48x^3) - 2(64xy^2 - 48x^3)(48x^2 - 64y^2) = 0 \tag{4.49}$$

and for $i = 0$ we have

$$\begin{aligned} &4(48x^2 - 64y^2)(9x^2 - 12y^2 - 16x^2y^2) - (64xy^2 - 48x^3)^2 \\ &\qquad = -192(4x^4 - 3x^2 + 4y^2)(3x^2 - 4y^2). \end{aligned} \tag{4.50}$$

A reduction modulo $c = 4x^4 - 3x^2 + 4y^2$ yields zero in both cases. Therefore, the test is positive and we can use

$$L_{\text{linear}} = 2(48x^2 - 64y^2)z - x(48x^2 - 64y^2) \tag{4.51}$$

instead of L to lift $V(c)$. Obviously, L_{linear} can be simplified further by dividing out its content. This yields

$$\frac{L_{\text{linear}}}{\operatorname{cont}_z(L_{\text{linear}})} = 2z - x. \tag{4.52}$$

The initial setting and the final result are shown in Figure 4.8.

4.6.3. Computing the squarefree part

If c and L do not pass the test of Theorem 4.6.2, then L has several distinct roots above at least one one-dimensional component of $V(c) \setminus V(\mathrm{lcoeff}(L))$. In this case, we want to use certified real root isolation as a tool for lifting. To the knowledge of the author, all real root isolation algorithms that rely on (arbitrarily close) approximations of the coefficients of $L(\alpha, \beta, z)$ for $(\alpha, \beta) \in V(c) \setminus V(\mathrm{lcoeff}(L))$ require $L(\alpha, \beta, z)$ to be squarefree. We know how to compute the squarefree part of an univariate polynomial using subresultants by Lemma 2.3.22. In this section we study how to apply this lemma to our multivariate problem.

First of all, we compute the signed subresultant sequence of L and $\frac{\partial}{\partial z}L$ with respect to z. By construction, $V(c)$ and $V(\mathrm{lcoeff}(L))$ intersect only in a finite number of points. Due to Lemma 2.3.32, this subresultant sequence is valid for all specialization points in $V(c) \setminus V(\mathrm{lcoeff}(L))$. All specialized subresultants except those equal to L and $\frac{\partial}{\partial z}$ are identically zero on $V(c, \mathrm{lcoeff}(L))$ by Corollary 2.3.34. These facts allow to factorize c further.

Definition 4.6.4. *Consider the sequence* $\mathcal{S} = [\mathrm{SRes}_0, \ldots, \mathrm{SRes}_k]$ *of signed subresultants of* L *and* $\frac{\partial}{\partial z}L$ *with* $\deg_z(L) = k$. *Then we define* $\Psi_{-1}(x, y) = c(x, y)$ *and for* $0 \le i < k$

$$\Psi_i(x, y) = \gcd(\Psi_{i-1}(x, y), \mathrm{sres}_i(x, y)), \tag{4.53}$$

$$\Xi_i(x, y) = \frac{\Psi_{i-1}(x, y)}{\Psi_i(x, y)}. \tag{4.54}$$

The approach is similar to what we have done before. It is trivial to verify that $c = \prod_{i=0}^{k-1} \Xi_i$ since $\Psi_{k-1} = 1$. No factor is lost or added during the construction.

Lemma 4.6.5. *Given the notation of Definition 4.6.4, the squarefree part* $\tilde{L}_i(\alpha, \beta, z)$ *of* $L(\alpha, \beta, z)$ *for all specializations at* $(\alpha, \beta) \in V(\Xi_i) \setminus V(\mathrm{sres}_i)$ *is given by*

$$\tilde{L}_i(\alpha, \beta, z) = \begin{cases} L(\alpha, \beta, z) & i = 0, \\ \mathrm{SResV}_{i-1}(\alpha, \beta, z) & 0 < i < k. \end{cases} \tag{4.55}$$

Proof. Since Ξ_0 does not divide SRes_0, the polynomial L has no multiple roots over $V(\Xi_0) \setminus V(\mathrm{SRes}_0)$. This proves the first part. For the second part, note that Ξ_i is constructed so that the signed subresultants $\mathrm{SRes}_0, \ldots, \mathrm{SRes}_{i-1}$ vanish on $V(\Xi_i)$ but SRes_i vanishes only on $V(\Xi_i, \mathrm{sres}_i)$. This allows to apply Lemma 2.3.22 so that the subresultant cofactor SResV_{i-1} is the squarefree part of L for all specializations at $V(\Xi_i) \setminus V(\mathrm{sres}_i)$ and for $0 < i < k$. ☐

Our original construction is based on Definition 4.4.1. Therefore, the lifting surface $V(L)$ has the same number of roots (counted with multiplicity) over all points of $V(c) \setminus V(\mathrm{lcoeff}(L))$. Nevertheless, the structure of the roots might be different for different factors of c. Consider the case $deg_z L = 3$. The polynomial L has three distinct roots over $V(\Xi_0)$, a double and a simple root over $V(\Xi_1)$ and a triple root over $V(\Xi_2)$ (disregarding the exceptional points).

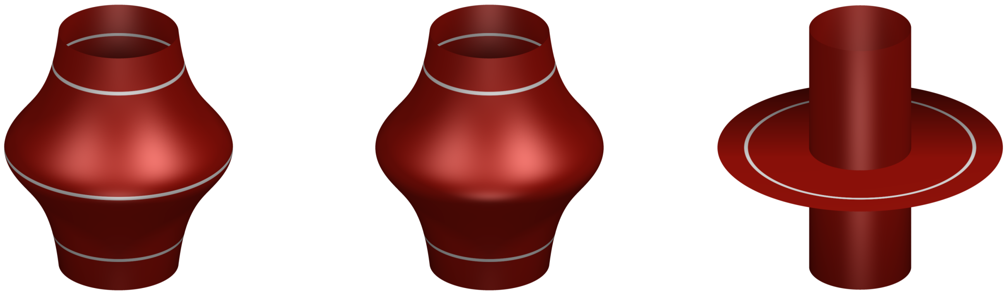
Example 4.6.6. We illustrate the construction of the factorization based on

$$c = (x^2 + y^2 - \tfrac{1}{3})(x^2 + y^2 - 1) \qquad L = (15x^2 + 15y^2 - 3)z^2 + (2x^2 + 2y^2 - 2). \tag{4.56}$$

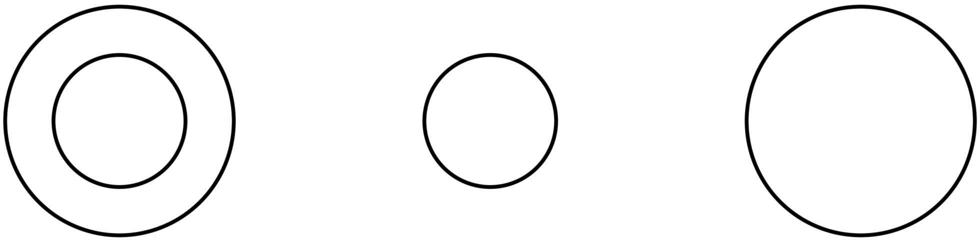
The relevant signed subresultants are

$$\mathrm{SRes}_0(L, \tfrac{\partial}{\partial z}L) = 72(x^2 + y^2 - 1)(5x^2 + 5y^2 - 1) \tag{4.57}$$

$$\mathrm{SRes}_1(L, \tfrac{\partial}{\partial z}L) = -18(5x^2 + 5y^2 - 1)z. \tag{4.58}$$



(a) $V(L)$ (b) $V(\tilde{L}_0)$ (c) $V(\tilde{L}_1)$



(d) $V(c)$ (e) $V(\Xi_0)$ (f) $V(\Xi_1)$

Figure 4.9.: Visualization of the curves and surfaces involved in Example 4.6.6. Subresultants and their cofactors can be used to compute the squarefree part of the polynomial L with respect to the zero set $V(c)$. The factorization of c into Ξ_0 and Ξ_1 and the associated lifting surfaces (with simple roots above $V(\Xi_{\{0,1\}})$) are shown in (e) and (f) respectively (b) and (c).

Applying Definition 4.6.4 and Lemma 4.6.5 yields

$$\Psi_{-1} = c \tag{4.59}$$

$$\Psi_0 = x^2 + y^2 - 1 \qquad \Xi_0 = x^2 + y^2 - \tfrac{1}{3} \qquad \tilde{L}_0 = L \tag{4.60}$$

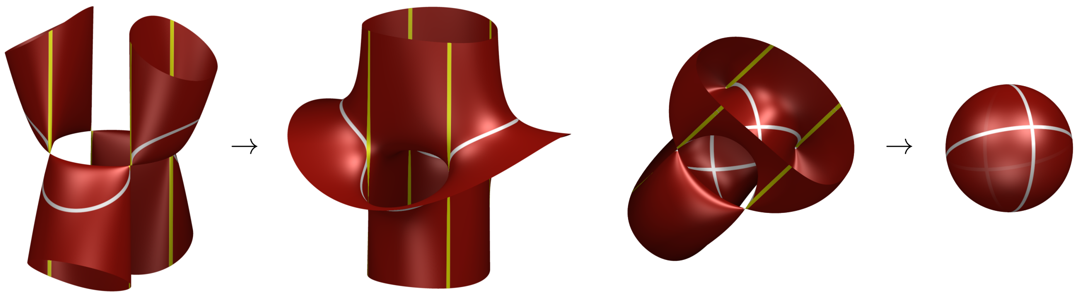
$$\Psi_1 = 1 \qquad \Xi_1 = x^2 + y^2 - 1 \qquad \tilde{L}_1 = \mathrm{SRes}_1(L, \tfrac{\partial}{\partial z} L). \tag{4.61}$$

Thus, it is clear that L has a double root over the larger circle $\Xi_1 = x^2 + y^2 - 1$ and two simple roots over $\Xi_0 = x^2 + y^2 - \frac{1}{3}$. See Figure 4.9 for a visualization of the involved curves and surfaces.

4.6.4. Syzygies

We will now try to reduce the degree of L based on a heuristic approach. While keeping c fixed, we may modify L as long as we do not significantly change $V(L, c)$. For us, it is acceptable to change the set of exceptional points on $V(c)$ as long as it is still finite. Hence, the modified lifting polynomial $\tilde{L}$ has to vanish on the exact same zero set over $V(c)$ up to a finite number of exceptional points on $V(c)$.

Since the lifting polynomial L is derived from a subresultant (cofactor), we may follow an approach based on Lemma 2.3.23. In the univariate case over $\mathbb{Z}$, the lemma allows to significantly reduce the coefficients of a subresultant SRes_i if it is a GCD of A and B. In this case, the polynomials $\frac{\mathrm{lcoeff}(A)\,\mathrm{SRes}_i}{\mathrm{sres}_i}$ and $\frac{\mathrm{lcoeff}(A)\,\mathrm{SResV}_{i-1}}{\mathrm{lcoeff}(\mathrm{SResV}_i)}$ are elements of the ring $\mathbb{Z}[x]$, i.e. the



(a) $V(L_1)$ and $V(\tilde{L}_1)$ (b) $V(L_2)$ and $V(\tilde{L}_2)$

Figure 4.10.: Results of the syzygy based reduction of the coefficient size for Example 4.4.3. This approach allows to reduce $L_1 = (9xy)z + (2x^4 + 4x^2y^2 - 20x^2 + 2y^4 - 20y^2 + 50)$ to $\tilde{L}_1 = (2x^2+2y^2-10)z+(3xy)$ and $L_2 = (x^2+y^2-5)z^2+(6xy)z+(x^4+2x^2y^2-10x^2+y^4-10y^2+25)$ to $\tilde{L}_2 = z^2 + (x^2 + y^2 - 5)$, i.e. $\deg \tilde{L}_1 = \deg L_1 - 1$ resp. $\deg \tilde{L}_2 = \deg L_2 - 2$.

remainder of the division is zero. Unfortunately, our problem is a multivariate one so that L is only a GCD of A and B over the plane curve $c = 0$. In addition, the construction breaks down at some exceptional points on $V(c)$. For these reasons, there is in general no $\operatorname{lcoeff}(L)^{-1}$ so that $\operatorname{lcoeff}(L) \operatorname{lcoeff}(L)^{-1} \equiv 1 \mod c$.

Nevertheless, we can relax the conditions and try to solve the modified division equation

$$\operatorname{lcoeff}(L)\tilde{L} = a \operatorname{lcoeff}(A)L + b\,c \tag{4.62}$$

for $a \in \mathbb{Q}[x, y]$ $\tilde{L}, b \in \mathbb{Q}[x, y, z]$ so that $\deg_z \tilde{L} = \deg_z L$ and $\gcd(\operatorname{lcoeff}(\tilde{L}), c) = 1$. This ensures $V(\tilde{L}(\alpha, \beta, z)) = V(L(\alpha, \beta, z))$ for all points $(\alpha, \beta) \in V(c) \setminus (V(\operatorname{lcoeff}(\tilde{L})) \cup V(\operatorname{lcoeff}(L)))$. We may refer to $\tilde{L}$ as some kind of pseudoquotient of $\operatorname{lcoeff}(A)L$ and $\operatorname{lcoeff}(L)$ under the additional condition $c = 0$. The case $a = 1$ would be the exact division, which does usually not occur since we are computing in the coefficient ring $\mathbb{Q}[x, y]$ instead of $\mathbb{Z}$.

A solution $\tilde{L}, a, b$ to Equation (4.62) is what is called a *syzygy*. See [GP02; Mis93] for details on syzygies since we only use the fact that most computer algebra systems are able to compute them. Obviously, there is more than one solution to Equation (4.62) and computer algebra systems accommodate this fact by determining a set of generators so that each solutions is a polynomial combination of these generators.

In order to reduce the coefficients of L, we select one of the generators where $\tilde{L}$ has minimal total degree in x and y and unchanged degree in z. Using this approach, we are often able to simplify the lifting polynomial as shown in Figure 4.10. The best results have been achieved on the basis of term orderings that consider the degree first when comparing monomials such as degree reverse lexicographical ordering. The time for the computation of the syzygies differed from almost negligible to several hours without result in the tested cases. For this reason and since we do not always find a polynomial of lower degree, this simplification method is only a heuristic. In an implementation, the user might specify a threshold for the execution time. This threshold can also be set in relation to the time spent for the computation of the subresultants so that more time is spent for optimizing complicated instances.

4.7. Numerical approximation

In this section, we assume that the factors $c_i \in \mathbb{Q}[x,y]$ of the resultant $\mathrm{Res}_z(A,B)$ and the lifting polynomials $L_i \in \mathbb{Q}[x,y,z]$ for $i = 1,\dots,k$ have been computed so that $V(L_i)$ has only simple roots and no asymptotes over $V(c_i) \setminus V(\mathrm{lcoeff}(L_i))$. Due to this decomposition, we are able to lift the space curve above $V_{\mathbb{R}}(\mathrm{Res}_z(A,B)) \setminus E$ where the set

$$E = \bigcup_{i=1}^{k} V_{\mathbb{R}}(c_i, \mathrm{lcoeff}(L_i)) \in \mathbb{R}^2 \tag{4.63}$$

of exceptional points is finite. Along the z fiber above a point $P \in E$ the two polynomials A and B have either a finite number of common roots or are identically zero. Hence, the curve $V(A,B)$ contains a line above P in the latter case. Our goal is to draw all one dimensional components of the space curve including lines in z direction. We will examine this important case first. To this end, assume that we want to approximate the space curve up to a maximum error $\varepsilon > 0$.

4.7.1. Lines in the direction of projection

The set of real lines in z direction is given by $V_{\mathbb{R}}(F) \times \mathbb{R}$ where

$$F = \langle a_0(x,y),\dots,a_m(x,y),b_0(x,y),\dots,b_n(x,y)\rangle. \tag{4.64}$$

Note that $V_{\mathbb{R}}(F)$ is finite since $\gcd(A,B) = 1$. Over the real numbers, the solutions of

$$a_0(x,y) = \ldots = a_m(x,y) = b_0(x,y) = \ldots = b_n(x,y) = 0 \tag{4.65}$$

are identically to the solutions of

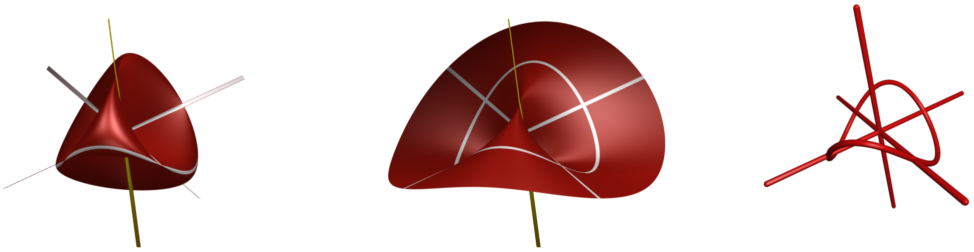
$$F_1 = a_0(x,y)^2 + \cdots + a_m(x,y)^2 + b_0(x,y)^2 \cdots + b_n(x,y)^2 = 0. \tag{4.66}$$

In order to compute the real solutions using a resultant based projection approach (cf. e.g. [BES11; SW05; CGL09; GN02]), we need another polynomial which vanishes on $V_{\mathbb{R}}(F)$. A good choice is to select the $F_2 \in \{a_0(x,y),\dots,a_m(x,y),b_0(x,y),\dots,b_n(x,y)\}$ with minimal degree, e.g. first minimize by the total degree and then by the degree of the individual variables. In case the minimal F_2 turns out to be a constant, the set $V_{\mathbb{R}}(F) = V_{\mathbb{R}}(F_1,F_2)$ is empty. Otherwise, we compute certified approximations of $V_{\mathbb{R}}(F_1,F_2)$ with an interval size smaller than ε using one of the previously mentioned methods and draw the lines above these approximations. The fact that $\deg_{xy}(F_1) = 2\max(\deg_{xy}(A),\deg_{xy}(B))$ is not a big issue in practice due to the choice of F_2. Furthermore, the cost for approximating $V_{\mathbb{R}}(F_1,F_2)$ is usually outweighed by the analysis of the factors of $\mathrm{Res}_z(A,B)$, which are of degree $\mathcal{O}(m^2)$ in the worst case.

Example 4.7.1. Consider the zero sets of

$$A_3 = (x^2+y^2)z^2 - (xy)z + (x^2y^2), \tag{4.67}$$

$$B_3 = \frac{\partial}{\partial z}A_3 = (2x^2+2y^2)z - (xy). \tag{4.68}$$



(a) $V(A_3)$ (b) $V(B_3)$ (c) $V(A_3, B_3)$

Figure 4.11.: Visualization of the (a) Steiner surface, (b) its derivative with respect to z and (c) the intersection curve of both. This illustrates that the described algorithm is able to deal with lines in the direction of projection. Note that the lines in (a) and (b) have been added manually. The yellow line is the line in the direction of projection where the subresultant based lifting method fails, i.e. all subresultant polynomials are identically zero on $V(x, y)$. The method described in Section 4.7.1 circumvents this problem.

Both, $V(A_3)$ and $V(B_3)$ contain the three line $V(x, y)$, $V(x, z)$ and $V(y, z)$. The line $V(x, y)$ is in the direction of projection and we compute it using

$$F_1 = (x^2 + y^2)^2 + (-xy)^2 + (x^2y^2)^2 \tag{4.69}$$

$$= x^4y^4 + x^4 + 3x^2y^2 + y^4, \tag{4.70}$$

$$F_2 = -xy, \tag{4.71}$$

i.e. F_2 is the coefficient of A_3 and B_3 with the lowest degree. Solving for the real zero set yields $V(F_1, F_2) = \{0, 0\}$. The involved surfaces and the intersection curve (computed by the presented algorithm) are shown in Figure 4.11.

4.7.2. Approximating the projections and lifting

We can now state the common real zero set of A and B in terms of our decomposition:

$$V_{\mathbb{R}}(A, B) \setminus S = \bigcup_{i=1}^{k} (V_{\mathbb{R}}(c_i, L_i) \setminus (V_{\mathbb{R}}(\mathrm{lcoeff}(L_i)) \times \mathbb{R})) \cup (V_{\mathbb{R}}(F) \times \mathbb{R}). \tag{4.72}$$

Here $S = V_{\mathbb{R}}(A, B) \cap (((E \setminus V_{\mathbb{R}}(F)) \times \mathbb{R}))$ is the finite set of points that is missing due to the exceptional set E except for $V_{\mathbb{R}}(F) \times \mathbb{R}$, which is again the set of real lines in z direction. Since we already computed these lines, the next step is to approximate the plane curves $V_{\mathbb{R}}(c_i)$. This can be done using one of the approaches detailed in Chapter 3. The only difference is that we need to be able to dynamically adjust the accuracy of the approximation on a per sample point basis, which is easy to implement. Note that we have to choose the resolution for the rasterization of $V_{\mathbb{R}}(c_i)$ based on ε. Using this approach, we can determine a piecewise linear approximation to segments of the curve $V_{\mathbb{R}}(c_i)$. Since we have no topological information at hand we are not able to connect the segments properly. In order to create a high quality rendering, we increase the precision in the neighborhood of the start and end point of each segment so that the disconnection is visually imperceptible.

In order to lift the projection $(\alpha,\beta) \in V_{\mathbb{R}}(c_i) \setminus V_{\mathbb{R}}(\operatorname{lcoeff}(L_i))$, we compute an interval approximation of $L_i(\alpha,\beta,z)$. Since we have ensured in Section 4.6.3 that $L_i(\alpha,\beta,z)$ is squarefree, we can apply certified real root isolation methods for polynomials with approximate coefficients like [Eig08; MS11], which are based on Descartes' rule of signs (cf. Section 2.4.2). These algorithms either compute correct isolating intervals for the real roots of $L_i(\alpha,\beta,z)$ or request an improved approximation of $L_i(\alpha,\beta,z)$ due to insufficient precision. It is proven that this loop terminates. See [Eig08; MS11] for the details.

It remains to exclude the points from the set $V_{\mathbb{R}}(c_i, \operatorname{lcoeff}(L_i))$. This is simply done by adding the real roots of $\operatorname{Res}_y(c_i, \operatorname{lcoeff}(L_i))$ to the set of critical coordinates during the rasterization of the plane curve $V_{\mathbb{R}}(c_i)$. Note that $V_{\mathbb{R}}(c_i)$ must be decomposed further using the original content of L_i (see Section 4.6.1) so that the rasterized segments of $V_{\mathbb{R}}(c_i)$ are free of exceptional points.

4.8. A note on asymptotic complexity

In this section, a short summary of the complexity of the preprocessing steps of the proposed algorithm is given. No details on the lifting will be presented since there were to many open questions at the time of writing. This is mostly due to the fact that the algorithm depends on a certified method for isolating and approximating the real roots of a polynomial whose coefficients are (arbitrary close approximations of) algebraic numbers (cf. [Eig08; MS11]). Further investigations are necessary to provide useful results in a manner similar to Section 3.7. See also Section 4.11.

Theorem 4.8.1. *Given two polynomials* $A, B \in \mathbb{Z}[x,y,z]$ *of magnitude* (τ, n) *the algorithm computes the* (c_i, L_i) *decomposition of* $V(A,B)$ *using no more than* $\tilde{\mathcal{O}}(\tau n^{11})$ *bit operations.*

The theorem can easily be derived from the following results.

Proposition 4.8.2. *Computing* $\lambda \in \mathbb{Z}$ *such that* $\gcd(\operatorname{coeff}_0(A_{z-\lambda}), \operatorname{coeff}_0(B_{z-\lambda})) = 1$ *is possible in* $\tilde{\mathcal{O}}(\tau n^6)$ *bit operations.* $A_{z-\lambda}$ *and* $B_{z-\lambda}$ *are of magnitude* $\mathcal{O}(\tau + \operatorname{bit}(n), n)$*.*

Proof. By Lemma 4.5.2 we know that there are at most n^2 constants $\lambda \in \mathbb{Z}$ that violate $\gcd(\operatorname{coeff}_0(A_{z-\lambda}), \operatorname{coeff}_0(B_{z-\lambda})) = 1$. Hence, a good choice of λ has bitsize at most $\mathcal{O}(\operatorname{bit}(n))$. Computing $\operatorname{coeff}_0(A_{z-\lambda})$ and $\operatorname{coeff}_0(B_{z-\lambda})$ then needs no more than $\tilde{\mathcal{O}}(\tau n^3)$ bit operations using the trivial shift algorithm for polynomials. The GCD is computable in $\mathcal{O}(\tau n^4)$. After at most n^2 tries, this sums up to at most $\tilde{\mathcal{O}}(\tau n^6)$ bit operations.

The bound on the magnitude of $A_{z-\lambda}$ and $B_{z-\lambda}$ can easily be verified by writing the shift operation $P(x,y,z-\lambda)$ as the determinant $\det \operatorname{Syl}(P(x,y,t), t-(z-\lambda))$ and applying Hadamard's bound (cf. Lemma 2.1.5). ☐

Once the genericity of the space curve is established, we continue with the (possibly repeated) computation of trivariate signed subresultants to decompose $\operatorname{Res}_z(A,B)$ into the factors c_i.

Proposition 4.8.3. *Computing* $S_i = \operatorname{SRes}_i(A,B)$ *for* $0 \le i \le n$ *and* $\operatorname{SRes}_{j_i}(S_i, \frac{\partial}{\partial z} S_i)$ *as well as its cofactors* $\operatorname{SResV}_{j_i}(S_i, \frac{\partial}{\partial z} S_i)$ *for* $0 \le j_i \le i$ *needs no more than* $\tilde{\mathcal{O}}(\tau n^8)$ *bit operations.*

Proof. Computing the sequence of signed subresultants for two trivariate polynomials of magnitude (τ, n) costs $\tilde{\mathcal{O}}(\tau n^7)$ bit operations (cf. Table 2.3). Since we compute at most $\mathcal{O}(n)$ of them, the bound follows. Note that the increased bitsize of S_i does not increase the cost for computing the $\operatorname{SRes}_{j_i}$ sequence since it is compensated by the smaller degree of S_i. ☐

Finally, we bound the cost for the actual decomposition.

Proposition 4.8.4. *The polynomial* $\mathrm{Res}_z(A,B)$ *can be decomposed into the factors* c_i *using* $\tilde{\mathcal{O}}(\tau n^{11})$ *bit operations.*

Proof. The decomposition is based on a series of GCD computations of bivariate polynomials of magnitude $\tilde{\mathcal{O}}(\tau n, n^2)$. Hence, a GCD costs at most $\tilde{\mathcal{O}}(\tau n^9)$ bit operations. There are two different kinds of GCDs. First, the GCDs of $\mathrm{Res}_z(A,B)$ and the $\mathcal{O}(n^2)$ principal subresultant coefficients are computed. Here, we ignore that each GCD operation might allow to split up an additional factor of $\mathrm{Res}_z(A,B)$ since this does not increase the asymptotic complexity. Secondly, the GCDs used to determine the asymptotes of the lifting surfaces as described in Section 4.5.3. Each such asymptote occurs as a factor of the resultant. Hence, their number and therefore also the number of GCDs is bounded by $\mathcal{O}(n^2)$ due to the degree of the resultant. □

A slight modification of the algorithm allows to reduce the bound of $\tilde{\mathcal{O}}(\tau n^{11})$ stated in Theorem 4.8.1: If we use the polynomials $\tilde{A} = A^2 + B^2$ and $\tilde{B} = \frac{\partial}{\partial z}\tilde{A}$ instead of A and B, there is no need to compute the subresultant sequence of each $\mathrm{SRes}_j(\tilde{A},\tilde{B})$ and its derivative. We used this construction to determine the squarefree part of a lifting surface with respect to some factor of the resultant. But now, $\tilde{B}$ is the derivative of $\tilde{A}$ and $V_{\mathbb{R}}(A,B) = V_{\mathbb{R}}(\tilde{A},\tilde{B})$. Hence, if $c \mid \mathrm{SRes}_0(\tilde{A},\tilde{B}),\ldots,c \mid \mathrm{SRes}_{k-1}(\tilde{A},\tilde{B})$ but $c \nmid \mathrm{SRes}_k(\tilde{A},\tilde{B})$ for $c \in \mathbb{Q}[x,y]$, then $\mathrm{SResV}_{k-1}(\tilde{A},\tilde{B})$ is already the squarefree part of the lifting polynomial $\mathrm{SRes}_k(\tilde{A},\tilde{B})$ with respect to c. This yields the following complexity:

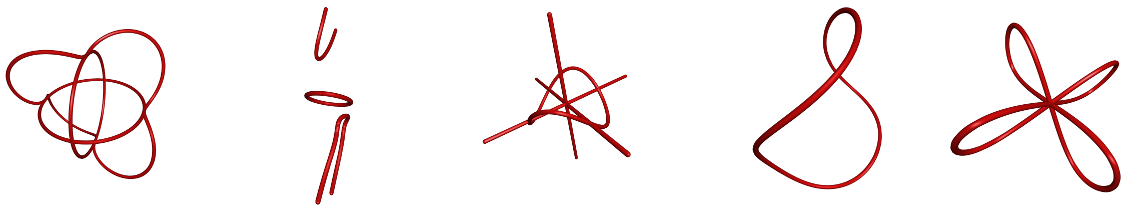
Theorem 4.8.5. *Given two polynomials* $A,B \in \mathbb{Z}[x,y,z]$ *of magnitude* (τ,n) *the above algorithm computes the* (c_i,L_i) *decomposition of* $V(\tilde{A},\tilde{B})$ *using no more than* $\tilde{\mathcal{O}}(\tau n^{10})$ *bit operations.*

Proof. We only need to improve on Proposition 4.8.4. The magnitude of the involved polynomials increases only by a constant factor if we use $(\tilde{A},\tilde{B})$ instead of (A,B). We do not need to compute subresultants of subresultants due to the modification. Hence, the total number of principal subresultant coefficients is in $\mathcal{O}(n)$ and not $\mathcal{O}(n^2)$. In the modified construction, all lifting polynomials are squarefree with respect to their associated component of the projection $V(c)$. In the transformed (generic) coordinate system, the one-dimensional asymptotes of $\tilde{A}$ and $\tilde{B}$ are solutions at $z = 0$ and their multiplicity must be one, too. After transforming back into the original system, this statement remains true for asymptotes. Hence, for a lifting polynomial L of degree k, we only need to compute $\gcd(c, \mathrm{lcoeff}(L))$ since $\gcd(c, \mathrm{coeff}_{k-1}(L)) = 1$. The number of lifting polynomials and therefore also the number of GCDs is bounded by the number of subresultants, which is in $\mathcal{O}(n)$. Using again that the cost for one GCD operation is bounded by $\tilde{\mathcal{O}}(\tau n^9)$, the result follows. □

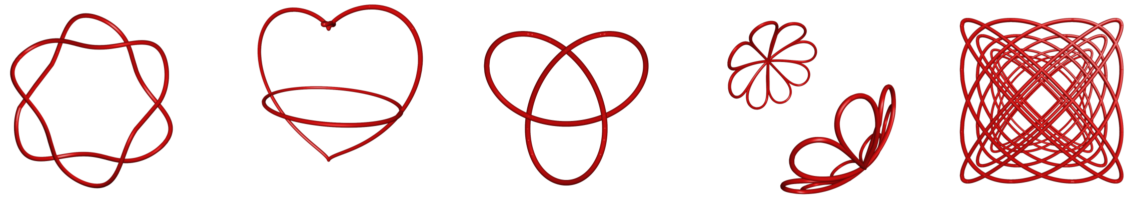
In the modified construction, the degree of the input doubles, which yields a fourfold increase of the degree of the resultant. For this reason, the modified algorithm shows bad performance in practice. In order to create an accurate sampling of the space curve, we analyze the projection for critical points. We will see in Theorem 4.8.6 and Section 4.10 that this is the theoretical as well as the practical bottleneck of the preprocessing in any case. Therefore, the previous improvement is more of theoretical interest.

Theorem 4.8.6. *Finding all critical points of the projected curve components* $V_{\mathbb{R}}(c_i)$ *needs no more than* $\tilde{\mathcal{O}}(\tau n^{15} + n^{16})$ *bit operations.*

Proof. Real solving a system of two polynomials in $\mathbb{Z}[x,y]$ of magnitude (τ,n) is possible in $\tilde{\mathcal{O}}(\tau n^7 + n^8)$ as discussed in Section 3.7.1. The result follows directly since the product of the c_i factors yields the squarefree part of the resultant, which is of magnitude $\tilde{\mathcal{O}}(\tau n + n^2, n^2)$. □



Example 1 Example 2 Example 3 Example 4 Example 5



Example 6 Example 7 Example 8 Example 9 Example 10

Figure 4.12.: Renderings of the examples utilized in Table 4.1. The visualizations have been created by the new algorithm. They are based on line strip approximations to the segments of each curve. The line segments are displayed as cylinders of small height and diameter.

Obviously, this part of the algorithm dominates the previous computations. We will now take a look at an implementation and at some experimental results.

4.9. Implementation

The described approach has been implemented on top of the CGAL 3.8 library [CGAL12], which already provides many of the necessary subalgorithms. This includes arithmetic with polynomials, subresultants, GCDs and the method for isolating the real roots of polynomials with approximate coefficients described in [Eig08]. In order to compare the new algorithm with [DMR08], their test of generic position has also been implemented. The input curve is repeatedly sheared until the genericity condition that is imposed by Theorem 4.6.2 is achieved. The result is then processed further with the algorithm presented in this chapter since both methods are identical for a curve that is generic with respect to [DMR08].

In order to determine sampling points on the projections of the space curve components, we follow Algorithm 3.2. Although this might not be the most efficient approach, it has the advantage that it is quite easy to implement and that we can utilize much of CGAL's infrastructure for refining the approximations of the sampling points. Finally, the lifting is based on the real root isolation method of [Eig08] as stated above.

4.10. Results and discussion

The implementation has been tested with a couple of curves to illustrate the practical behaviour of the algorithm. To the author's knowledge, no benchmark sets are available for space curve rendering algorithms. Although one can possibly build upon the work for plane curves (see [Lab10b]), this task is out of the scope of this work. Therefore, we restrict to the small set of space curves illustrated in Figure 4.12. The time spent for generating these images is listed in

Table 4.1. The set of curves covers all special cases the presented algorithm has to deal with. Often, this becomes explicit on the basis of the defining equations. These can be found in Appendix C.2. Some of the examples also occurred previously during the description of the new method. Most importantly, all tested curves violate the genericity condition used in [DMR08] so that their algorithm has to perform a change of coordinates. This allows to compare the two approaches since both algorithms are equivalent for curves that are in generic position according to [DMR08]. We will denote the latter algorithm by 'DMR'.

The results show that ensuring generic position by a shear transformation of type $(x, y, z) \mapsto (x + \lambda_1 z, y + \lambda_2 z, z)$ often leads to a drastic increase of the running time in practice. The test for genericity is quite complex for the DMR algorithm. It requires to compute the Θ and Δ polynomials introduced in Definition 4.4.1. This involves the computation of subresultants of trivariate polynomials and of several bivariate GCDs. Afterwards, the number of different roots of the lifting polynomial $\mathrm{SRes}_i(A, B)$ above $V(\Delta_i)$ has to be checked using Theorem 4.6.2. In contrast, the algorithm described above only requires the computation of a single bivariate GCD (see Section 4.5.2).

Usually, the test for genericity is not the most expensive part of the computation. The shear operation separates the projections of space curve components that previously had the same projection. Hence, the projection becomes more difficult to analyze. The results show that this is the main reason for the long running times of the DMR algorithm. Due to Lemma 4.5.3, we can be sure that the algorithm proposed in this chapter does not suffer from this problem.

Another disadvantage of separating the space curve components in the projection is that it is hard to identify which curve segments in the sheared coordinate system are stacked on top of each other in the original coordinate system. This is of major importance in an application, where the curve needs to be rendered exactly from a specified direction.

4.11. Conclusion and future work

We have seen that the new algorithm is able to produce high quality visualizations of real algebraic space curves. Since the result is a line strip approximation of the curve segments, the space curve can be viewed interactively. We have also shown that achieving some notion of generic position by some kind of shear transformation often adds a major overhead to the running time. We avoid such coordinate changes so that we are able to outperform the previous algorithm we build upon. In general, the computation time is reasonable for curves of moderate complexity such as the examples used in Section 4.10. Nevertheless, it seems hard to achieve a reliable visualization in real time as long as the preprocessing steps involve symbolic computations. We will see in Chapter 6 how parallel processing may reduce running time in the future. But since the complexity of the algorithm grows faster than linear with the problem size, it remains hard to deal with curves defined by high degree polynomials. Even if we would consequently base an algorithm on numerical methods, we would have to utilize arbitrary-precision arithmetic to validate the results.

In the current implementation, numerical methods have only been employed during the approximation of the sampling points and during their lifting. The numerical lifting process currently does not utilize the coherence of the z coordinate between adjacent sampling points. In the new method, the tangent of $V(c_i, L_i) \setminus (V(\mathrm{lcoeff}(L_i)) \times \mathbb{R})$ is well defined by $(\frac{\partial}{\partial x} c_i, \frac{\partial}{\partial y} c_i, 0) \times (\frac{\partial}{\partial x} L_i, \frac{\partial}{\partial y} L_i, \frac{\partial}{\partial z} L_i)$ so that it seems to be possible to construct an efficient curve tracing algorithm. Exploiting these facts appears to be a crucial ingredient for further speed up. Together with an

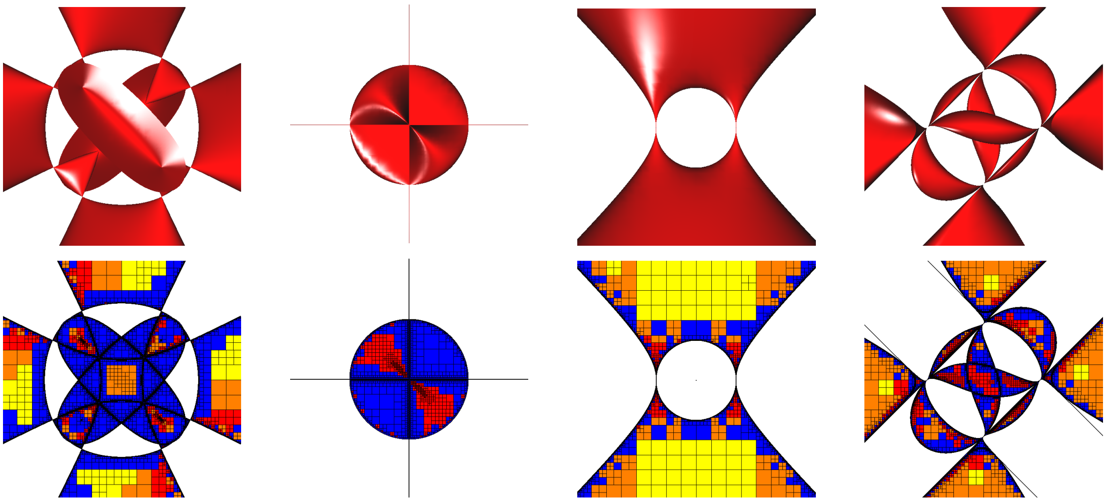
	deg A			deg B			Preprocessing 3D			Preprocessing 2D			Lifting		# of points		Points/s		
Id	xy	xyz	z	xy	xyz	z	DMR (s)	S (s)	×	DMR (s)	S (s)	×	DMR (s)	S (s)	DMR	S	DMR	S	×
1	4	4	4	2	3	3	0.236	0.020	**11.6**	32.35	2.035	**15.9**	90.31	10.38	8888	3066	98.41	295.29	**3.0**
2	4	8	4	4	7	3	18.709	2.282	**8.2**	583.24	13.458	**43.3**	97.53	17.09	1746	888	17.90	51.95	**2.9**
3	4	4	2	2	3	1	0.049	0.007	**6.7**	1.01	0.410	**2.5**	6.06	1.79	2453	1744	404.92	976.15	**2.4**
4	2	2	2	2	2	2	0.009	0.004	**2.3**	0.88	0.164	**5.4**	3.77	1.79	1316	1044	348.74	582.29	**1.7**
5	2	2	2	4	4	0	0.009	0.003	**3.3**	1.60	0.260	**6.2**	7.83	2.32	2504	2016	319.73	870.31	**2.7**
6	4	4	4	4	4	3	2.425	1.946	**1.2**	84.68	34.941	**2.4**	121.19	81.64	4510	4106	37.21	50.30	**1.4**
7	6	6	6	4	5	5	130.880	1.585	**82.6**	969.04	2.270	**426.8**	67.64	9.10	2020	1700	29.86	186.89	**6.3**
8	8	8	6	3	3	1	227.780	8.310	**27.4**	—	66.811	∞	—	80.91	—	3074	—	37.99	∞
9	16	16	8	2	2	2	321.070	1.624	**197.7**	—	86.956	∞	—	147.22	—	18360	—	124.71	∞
10	8	8	8	0	7	7	—	7.604	∞	—	35.716	∞	—	289.60	—	82688	—	285.52	∞

Table 4.1.: Computation times for creating the space curve visualizations shown in Figure 4.12. The timings have been determined on a machine with a 2.4 GHz Intel Core 2 Q6600 Quad CPU having 4 GB of RAM. The "Id" refers to the number of the example in Figure 4.12 defined by the polynomials $A_{\mathrm{Id}}, B_{\mathrm{Id}}$ given in Appendix C.2. The printed degrees are the total degrees with respect to the shown variables. The computations have been split into three stages: "Preprocessing 3D" includes all symbolic computations needed to determine the (c_i, L_i) decomposition. "Preprocessing 2D" summarizes the analysis of the plane curves $V_{\mathbb{R}}(c_i)$ and the generation of sampling points on each curve segment based on a minimum grid resolution of 256 × 256. The final number of sampling points highly depends on the actual space curve. During the "Lifting", the z coordinates of the sampling points are determined to a minimum precision of 10^{-4}. However, the lifting procedure often requires a successive refinement of the coordinates in the plane which eventually causes a higher precision in the final result. The timings are given for the two algorithms we abbreviate with DMR (based on [DMR08]) and S (the new algorithm designed by the author). The column "×" denotes the speedup of algorithm S compared to DMS. The sign "—" denotes a timeout of the respective algorithm after 1 h and likewise ∞ denotes the speedup where algorithm DMS timed out. Note that the column "Points/s" summarizes "Lifting" and "# of points" since both algorithms usually create a different number of points due to the different coordinate systems they work in.

in-depth analysis of the lifting process and the syzygy based simplification idea this provides an interesting area for future research.

Another extension of the method would be to ensure correct topology. Although we provide line strip approximations to the curve segments, these segments are not connected. The segment endpoints are approximated with high precision, but in case a curve segment has a steep tangent with respect to the z direction, the visualization might still suggest a wrong topology. However, such problems only occur for specific examples so that the new approach can be regarded as a significant contribution to the subject of real algebraic space curve visualization.

Given that we are able to reliably visualize space curves, we could go one step further and visualize algebraic surfaces in the same manner. If we fix an eye point $P = (x_p, y_p, z_p) \in \mathbb{Q}^3$, the most important curve on a real algebraic surface $V_\mathbb{R}(A)$ for $A \in \mathbb{Q}[x, y, z]$ is the space curve commonly referred to as silhouette, apparent contour or polar curve given by $V_\mathbb{R}(A, B)$ with $B(x, y, z) = ((x, y, z) - P) \cdot \nabla A(x, y, z)$, where ∇A denotes the gradient of A. Informally, the silhouette curve contains all points of $V_\mathbb{R}(A)$ that appear as the boundary when viewed from P. Additionally, it includes the singular points of $V_\mathbb{R}(A)$ due to the vanishing gradient of A. In order to visualize $V_\mathbb{R}(A)$, we may render $V_\mathbb{R}(A, B)$ and then fill up the empty space by exploiting that each patch of $V_\mathbb{R}(A) \setminus V_\mathbb{R}(A, B)$ is smooth. The basics for such an approach have been carried out in the Diploma thesis of P. Hiesinger (see [Hie11]). There, he investigated different methods such as interval arithmetic, curvature estimation and inclusion tests known from algebraic plane curves rendering (cf. [Tau94]) in order to fill such regions in a reliable way without creating a very dense sampling. Although his preliminary implementation tends to be quite slow due to the heavy usage of exact arithmetic, the outcomes are quite promising. Some results of his work that has been created in collaboration with the author of this thesis are shown in Figure 4.13. Note that the image of the Steiner surface in Figure 4.13b correctly displays the vertical and horizontal line. As illustrated in the examples above, this even works if $V_\mathbb{R}(A)$ contains a more complicated space curve in its real part.



(a) Kummer surface (b) Steiner surface (c) Cassini surface (d) Stagnaro surface

Figure 4.13.: Surface visualizations created with the program implemented in [Hie11]. It first decomposes the image plane based on a rendering of the projection of the silhouette curve. This information is stored in a quadtree. The quadtree is gradually refined based on various heuristic approaches. In each quadtree cell, the surface is then approximated by two triangles. The lower images show the final subdivision tree for each surface. The different colors represent the different tests that caused the subdivision of that cell. Since the algorithm currently only deals with polynomials over $\mathbb{Q}$, the coefficients of the defining polynomials of (a) and (d) have been approximated since they involve square roots.

5. Interactive visualization of blowups of the plane

The investigations in this chapter have been published in [SS13], which is a joint work of the author of this thesis and P. Schenzel, who contributed the results on elimination.

5.1. Motivation

Let us start with an example to motivate the visualization of blowups of the plane. Consider the cubic plane curve defined as the zero set of the polynomial $C = u^2 - v^2(v+1) \in \mathbb{R}[u,v]$ shown in red on the lower left of Figure 5.1. This curve has a well defined tangent at all points except for the origin $(0,0)$, where two lines are tangent to C, i.e the point $(0,0)$ of the curve is singular. In general, it is much easier to understand the properties of an algebraic variety (e.g. an algebraic curve) if it is non-singular.

In order to smooth the singularities of an algebraic variety, blowups are an essential technique [Hir64; Har83]. By replacing the singularities of any complex algebraic variety V by an appropriately chosen sequence of blowups, one eventually obtains a non-singular variety $\tilde{V}$. That this process finishes with a non-singular model was proven by Hironaka (see [Hir64]).

Let us illustrate this concept using the curve C restricted to a disc

$$\mathbb{D} = \{(u,v) \in \mathbb{A}^2_{\mathbb{R}} | u^2 + v^2 \leq \rho^2\} \tag{5.1}$$

of radius ρ as reproduced in the lower part of Figure 5.1. This restriction has some advantages for the visualization as we will see later on. In order to blow up the disc $\mathbb{D}$ in the origin, we first move over to the affine 3-space by mapping a given point $(u,v) \in \mathbb{D}$, $v \neq 0$, to $(u, v, \frac{u}{v}) \in \mathbb{R}^3$.

This yields a parametrization of a surface $\mathbb{S}_0$ over the disc excluding the line $v = 0$. The vertical line through the origin becomes part of the topological closure of $\mathbb{S}_0$. The resulting closed surface $\mathbb{S}$ is shown in the upper left part of Figure 5.1.

Now, we apply the mapping $(u,v) \mapsto (u, v, \frac{u}{v})$ to $(\mathbb{D} \setminus V(v)) \cap V(C)$, i.e. we lift the plane curve C onto $\mathbb{S}_0$. The closure of the resulting space curve (illustrated in red color in the top left in Figure 5.1) is non-singular.

The construction excludes all the points on the v axis since we cannot divide by zero. Therefore, we may swap the roles of u and v and repeat the process by mapping $(u,v) \in \mathbb{D}$, $u \neq 0$ to $(u, v, \frac{v}{u})$ obtaining the surfaces $\tilde{\mathbb{S}}_0$ and $\tilde{\mathbb{S}}$. Mathematically, it is more convenient to work with only one surface $\mathbb{B}$ in $\mathbb{A}^2_{\mathbb{R}} \times \mathbb{P}$ (using the projective instead of the real line) given by the closure of $\mu(\mathbb{D} \setminus \{(0,0)\})$, $\mu : (u,v) \mapsto (u, v, (u : v))$. We call $\mathbb{B}$ the *blowup* of the disc in the origin and $\mathbb{S}$ and $\tilde{\mathbb{S}}$ are called the *two affine charts* of $\mathbb{B}$. Since we blew up the disc based on the rational map $(u,v) \mapsto (u : v)$, we use the more precise notation $\mathbb{B}_{u,v}$, where appropriate.

In order to visualize the surface $\mathbb{B}$, we could visualize its two affine charts. But this is not a satisfying solutions since both, $\mathbb{S}$ and $\tilde{\mathbb{S}}$, are infinite surfaces. Only a small part of the geometry is visualizable in a finite viewport. A nice approach is to use an appropriate embedding of $\mathbb{B}$

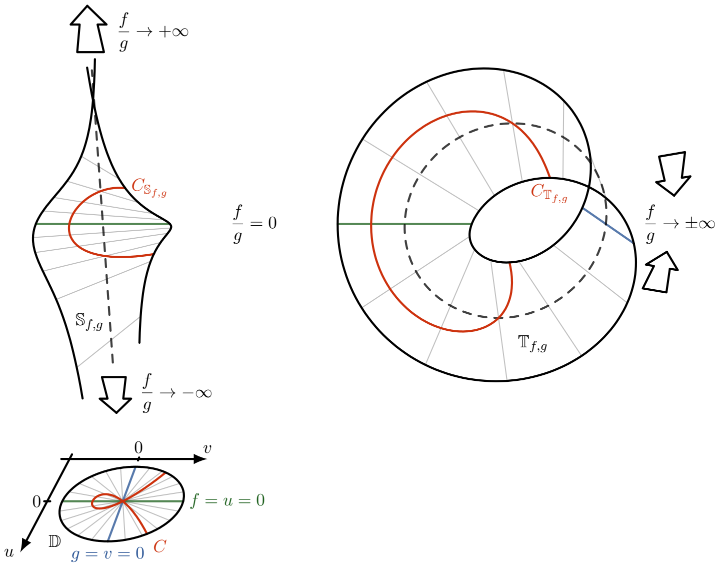


Figure 5.1.: The singular plane curve C in the disc $\mathbb{D}$ (lower left) becomes a smooth space curve when lifted to the blowup $\mathbb{B}_{f,g}$ of $\mathbb{D}$ in the origin given by the intersection of $f = u = 0$ and $g = v = 0$. The first affine chart $\mathbb{S}_{f,g}$ of the surface $\mathbb{B}_{f,g}$ is shown in the upper left. To cover $\mathbb{B}_{f,g}$ completely, another affine chart $\tilde{\mathbb{S}}$ is needed (not shown). Furthermore, $\mathbb{S}_{f,g}$ and $\tilde{\mathbb{S}}_{f,g}$ are infinite surfaces, so that most of their geometry can not be visualized directly. The transformation of the infinite cylinder over $\mathbb{D}$ into a torus yields a suitable embedding of the blowup in affine 3-space as shown on the right.

into 3-space, such as the one utilized in [Bro08] (see also [Bro95] for a full length treatment in German). That is, a certain type of a stereographic projection is used in order to transform the projective lines over each point of $\mathbb{D}$ into circles. Therefore, the infinite solid cylinder over $\mathbb{D}$ is mapped into a solid torus. Applying this transformation to $\mathbb{B}$ yields a compact model $\mathbb{T}$ of $\mathbb{B}$ in 3-space. We call the surface $\mathbb{T}$ the *toroidal blowup*. In our concrete example, the chart $\mathbb{S}_{u,v}$ of $\mathbb{B}_{u,v}$ is transformed into the so-called Möbius strip as shown on the right side of Figure 5.1. Note that the curve C transforms into a non-singular curve on the Möbius strip.

The generators $u = 0, v = 0$ are a good choice in order to obtain an appropriate blowup of the disc in the origin that resolves the singularities of C. However, plane curves with several or more complicated singularities require different generators, e.g. so that the disc is blown up in more than one point. We will investigate the concept of blowups of the disc in a finite number of points defined by the vanishing of two polynomials in the next section.

5.1.1. The definition of blowups

In general, the blowup of the real affine plane in a finite set of points $X = V(f,g) = \{P_1, \ldots, P_r\} \subset \mathbb{A}^2_{\mathbb{R}}$ is obtained as follows: Consider the map

$$\begin{aligned} \mu : \mathbb{A}^2_{\mathbb{R}} \setminus X &\to \mathbb{A}^2_{\mathbb{R}} \times \mathbb{P}^1_{\mathbb{R}}, \\ (u,v) &\mapsto ((u,v), (f(u,v) : g(u,v))), \end{aligned} \tag{5.2}$$

where $f, g \in \mathbb{R}[u,v]$ are polynomials with the zero set $X = V(f,g)$. Note that f and g do not have a common factor. Then the blowup $\mathbb{B}_{f,g}$ of $\mathbb{A}^2_{\mathbb{R}}$ in X is defined as the closure of $\mu(\mathbb{A}^2_{\mathbb{R}} \setminus X)$

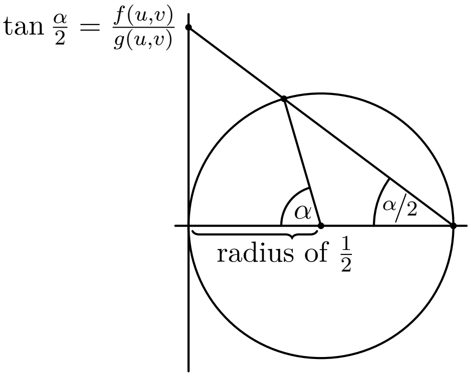


Figure 5.2.: Computation of the angle α for a point of height $\frac{f(u,v)}{g(u,v)}$ over $\mathbb{D}$ in order to map this point onto a circle as proposed in [Bro95]. Note that α only depends on the height $\frac{f(u,v)}{g(u,v)}$ of the point but not on its (u, v) coordinates in the disc.

in $\mathbb{A}^2_{\mathbb{R}} \times \mathbb{P}^1_{\mathbb{R}}$. There is a natural map $\pi : \mathbb{B}_{f,g} \to \mathbb{A}^2_{\mathbb{R}}, (u, v, (s : t)) \mapsto (u, v)$. The preimage $\pi^{-1}(X) \subset \mathbb{B}_{f,g}$ is called the *exceptional fiber* $\mathbb{E}$. It consists of the union of projective lines over the points $P_i \in X, i = 1, \ldots, k$. Moreover, π induces an isomorphism $\mathbb{A}^2_{\mathbb{R}} \setminus X \cong \mathbb{B}_{f,g} \setminus \mathbb{E}$. For further technical details, we refer to a textbook on Algebraic Geometry, e.g. [Har83] and [CLO07, pp. 506-508].

5.1.2. The embedding into a torus

An interesting question is how a blowup of the plane in a finite number of points $X = V(f, g)$ looks like. As mentioned in the initial example, the embedding of the blowup of the disc $\mathbb{D}$ into affine 3-space by transforming the infinite cylinder over $\mathbb{D}$ into a solid torus is a promising approach.

This idea has been carried out in detail in [Bro95]. The composition of the stereographic projection with a certain diffeomorphism provides the following parametrization $P(u, v)$ of the image of $\mathbb{D} \setminus V(f, g)$ in $\mathbb{D} \times \mathbb{P}^1_{\mathbb{R}}$ embedded as a solid torus with central radius $r \in \mathbb{R}$, $r > \rho$ in $\mathbb{A}^3_{\mathbb{R}}$:

$$(u, v) \mapsto (u, (v - r) \cos \alpha, (r - v) \sin \alpha). \tag{5.3}$$

Following Figure 5.2, the angle α is defined as

$$\alpha = \begin{cases} 2 \arctan \frac{f(u,v)}{g(u,v)} & g(u, v) \neq 0, \\ \pi & g(u, v) = 0. \end{cases} \tag{5.4}$$

This can be simplified by applying the double angle formulas of sine and cosine to Equation (5.3). It yields a rational parametrization $P(u, v) = (x, y, z) \in \mathbb{A}^3_{\mathbb{R}}$ with

$$\begin{aligned} x &= u, \\ y &= (r - v)\frac{f^2 - g^2}{f^2 + g^2}, \\ z &= (r - v)\frac{2fg}{f^2 + g^2}. \end{aligned} \tag{5.5}$$

The image of $\mathbb{D} \setminus V(f,g)$ is now contained in the torus with equation $x^2 + (r - \sqrt{y^2+z^2})^2 = \rho^2$, which we call the $\mathbb{D}$-torus. The set of points we have to visualize is $\mathbb{T}_{f,g}$, the closure of $\{P(u,v) \in \mathbb{A}^3_{\mathbb{R}} : (u,v) \in \mathbb{D} \setminus V(f,g)\}$ (in the Zariski topology) restricted to the $\mathbb{D}$-torus. Under the transformation into the torus, the exceptional fiber $\mathbb{E}$ is mapped to a family of circles, one circle for each point of $V(f,g)$. Note that in contrast to the original parametrization in [Bro95], the torus is centered at the origin in order to realize additional symmetry.

5.1.3. On the drawbacks of the parametrization

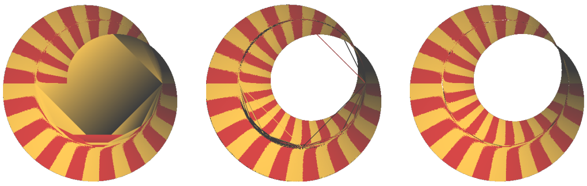
For the case of toroidal blowups of the plane in $X = \{(0,0)\}$, a few static renderings have been realized in [Bro95]. This includes for example the Möbius strip ($f = u, g = v$) and the Whitney double umbrella ($f = u^2, g = v^2$). The case of four points $X = \{(\pm 1, \pm 1)\} = V(u^2-1, v^2-1)$ is illustrated as an excellent hand drawing (see [Bro95, Figures 11 and 12]).

The first naive idea for the visualization of a parametrically defined surface is to create a triangle mesh based on an appropriate set of parameters $(u_i, v_i) \in \mathbb{D}, i = 1, \ldots, N$. This approach works well for parametrizations of non-singular surfaces without parametric singularities. The following problems arise in the case of toroidal blowups of the plane:

- The numerical computation of $P(u,v)$ becomes unstable in the neighborhood of the zero set $V(f,g)$.
- The parametric normal $\vec{n}(u,v) = (\frac{\partial P}{\partial u} \times \frac{\partial P}{\partial v})(u,v)$ necessary for the shading is a rational expression containing f and g in the eighth power. Cancellation errors frequently occur near $V(f,g)$.
- The constructed triangles might have extremal sizes, e.g. large edges and very small height. Triangles that span a large range of angles in the torus must be avoided. Otherwise holes and visualization artifacts might occur.
- For sufficiently satisfactory images, N needs to be rather large. Moreover, the points (u_i, v_i) have to be chosen very carefully to provide a good approximation of the surface.
- Points $P(u,v)$ close to the exceptional fiber must be appropriately connected with the exceptional fiber above $V(f,g)$.
- For given $f, g \in \mathbb{R}[u,v]$ the zero set $V(f,g)$ is not known a priori. Its computation requires additional effort.
- The triangulation has to be recomputed for any deformation of f and g.

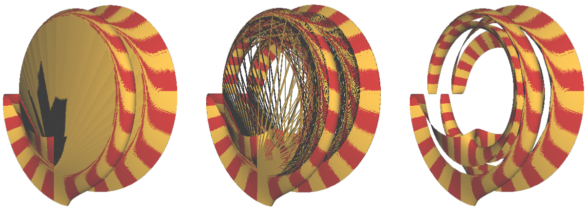
Visualizations of two toroidal blowups based on the parametric form and a uniform subdivision of the parameter space are reproduced in Figures 5.3 and 5.4. This is the approach carried out in [Bro95]. It is easy to see that the visualization of simple toroidal blowups can be achieved by omitting all (u,v) close to $X = V(f,g)$ (Figure 5.3). This solution does not extend to more complicated polynomials f and g, which becomes apparent in Figure 5.4.

In order to create triangulations of a wider range of toroidal blowups, a more sophisticated method has been studied by A. Prager in his diploma thesis [Pra11]. He created high quality polygonal meshes of toroidal blowups using methods from differential geometry to trace the level curves of the blowup $\mathbb{B}_{f,g}$ over the disc. The surface is formed by connecting the approximations of the level curves while taking special care of the regions close to the exceptional



(a) Filled (b) Wireframe (c) Faulty triangles removed

Figure 5.3.: Rendering of the parametric form of the toroidal blowup using $f = u, g = v$ (Möbius strip) and one million vertices. The triangles generated near $X = V(f, g) = \{(0, 0)\}$ span across the torus, thus, pretending a different surface geometry as shown in (a) and (b). Removing all triangles with $\sqrt{u^2 + v^2} < \varepsilon$ results in (c).



(a) Filled (b) Wireframe (c) Faulty triangles removed

Figure 5.4.: The approach of Figure 5.3 applied to $f = v^2 - u(u - 1)(u + 1)$ and $g = v^2$. Many more wrong triangles are generated near $X = V(f, g) = \{(-1, 0), (0, 0), (1, 0)\}$. Additionally, cancellation errors occur during the computation of the normals (colored in black). Removing all wrong triangles as shown in (c) yields large gaps in the mesh. Note that Figure 5.11b shows a correct visualization created by the method presented in this chapter.

fibers. With this idea, he has been able to visualize for example the blowup in the four points $X = \{(\pm 1, \pm 1)\}$ mentioned above. The improved visualization is done at the expense of time consuming precomputations. Therefore, this method is inappropriate for interactive deformations of the generators f and g.

5.1.4. Implicitization based real-time visualization

About 20 years of development in the field of computer graphics have passed since Brodmann's approach. A big step during the last years was the introduction of GPU programming. This allows an interactive visualization and parametric deformation of implicit surfaces in real time by ray casting methods (e.g. see the program RealSurf by the author [Stu09]). A technical overview and samples of visualizations can be found in [Stu07; Stu09; SS11; RS08; Kno+09].

In this work, it is demonstrated that these are suitable approaches for the visualization of toroidal blowups and their deformations by utilizing an implicit form of the toroidal blowup rather than a parametric one. A major advantage of the implicit form is that the points of the exceptional fiber are part of the surface. In contrast to the parametric form, no special treatment is necessary for these points.

It is well known that an implicit (algebraic) equation $\mathfrak{F} = 0$ for any rationally parametrized surface can be derived by elimination techniques. In the case of a toroidal blowup, three variables need to be eliminated from a system of four algebraic equations (see Section 5.2). This can be accomplished by Gröbner bases methods [CLO07] or multipolynomial resultants [MC92; SA84]. The polynomials f and g occur quadratically in Equation (5.5). This suggests a high degree of $\mathfrak{F}$ as a result of the elimination process. Therefore, this approach seems to be computationally rather complex at first glance. A closer examination based upon some techniques of commutative algebra surprisingly shows that the elimination can be done by the computation of a single, very simple resultant of two polynomials. Furthermore, we will see that the result, i.e. the polynomial $\mathfrak{F}$, has an unexpected low degree.

Once the implicit form $\mathfrak{F}$ has been obtained, we are able to visualize the surface $V_{\mathbb{R}}(\mathfrak{F})$ and its deformation based on the deformation of f and g in real time. Since $\mathbb{T}_{f,g} \subset V_{\mathbb{R}}(\mathfrak{F})$, additional clipping techniques have to be employed to display exactly the toroidal blowup $\mathbb{T}_{f,g}$ but not $V_{\mathbb{R}}(\mathfrak{F}) \setminus \mathbb{T}_{f,g}$. Different texturing schemes can be applied to the surface to get a better understanding of the relation between the toroidal blowup and the disc $\mathbb{D}$.

5.2. Derivation of the implicit form

For the impatient reader, the mathematical results of this section can be summarized as follows:

- The implicit equation of the toroidal blowup is
$$\mathfrak{F} = \mathrm{Res}_v(F, H)/z = \mathrm{Res}_v(G, H)/z = 0, \tag{5.6}$$
where $F = fz - ((r - v) + y)g$, $G = gz - ((r - v) - y)f$, $H = y^2 + z^2 - (r - v)^2$ and u is substituted by x in f and g.
- The degree and the time for the computation of $\mathfrak{F}$ grow linearly with the degree of f and g and not quadratically as one would expect.
- Parameters of f and g used for the deformation of the blowup occur as parameters of $\mathfrak{F}$. The implicit form $\mathfrak{F}$ is independent of the order of elimination and the specialization of the parameters as long as f and g intersect in a finite number of points.

Since the following results on elimination are valid in a more general context than the field of real numbers, we switch from $\mathbb{R}$ to an arbitrary infinite ordered field K of characteristic $\neq 2$ during the derivation of the above results.

5.2.1. Algebraic preparations

The substitution $u = x$ in the polynomials $f(u, v)$ and $g(u, v)$ provides the following parametrization of the toroidal blowup

$$y = (r - v)\frac{f^2 - g^2}{f^2 + g^2}, \qquad z = 2(r - v)\frac{fg}{f^2 + g^2}, \tag{5.7}$$

where now $f, g \in K[x, v]$. Therefore, we have to eliminate the variable v from the previously given rational parametrization to obtain the implicit equation of the surface. Consequently, we have to eliminate the variables v, w from the ideal

$$I = \langle A, B, 1 - w(f^2 + g^2)\rangle \tag{5.8}$$
$$A = y - (r - v)w(f^2 - g^2) \tag{5.9}$$
$$B = z - 2(r - v)wfg \tag{5.10}$$

in the polynomial ring $K[x, y, z, v, w]$ with $f, g \in K[x, v]$ (see e.g. [CLO07]). The auxiliary variable w and the equation $1 - w(f^2 + g^2) = 0$ allow to work with a system of polynomials instead of rational functions without changing the set of solutions.

Proposition 5.2.1. *With the previous notation, define*

$$F = fz - ((r-v)+y)g, \tag{5.11}$$

$$G = gz - ((r-v)-y)f. \tag{5.12}$$

Then $I = \langle F, G, 1 - w(f^2+g^2)\rangle$.

Proof. We define $J = \langle F, G, 1 - w(f^2+g^2)\rangle \subset K[x,y,z,v,w]$. We first show that $F, G \in I$, so that $J \subseteq I$. By subtracting $\pm((r-v) - (r-v)w(f^2+g^2)) \in I$ from $A \in I$ it follows that

$$F' = (r-v) - y - 2(r-v)wg^2 \in I, \tag{5.13}$$

$$G' = (r-v) + y - 2(r-v)wf^2 \in I. \tag{5.14}$$

Then $F = Bf - G'g \in I$ and $G = Bg - F'f \in I$, as required.

In order to show $I \subseteq J$, first note that we have $B = w(Ff + Gg) + z(1 - w(f^2+g^2)) \in J$. It follows that $A = -w(Fg - Gf) + y(1 - w(f^2+g^2)) \in J$, which finally completes the proof. □

For the use in Section 5.2.2, we need an elementary statement about a certain ideal.

Proposition 5.2.2. *With the previous notation, define the ideal* $J = \langle F, G, H\rangle \subset K[x,y,z,v]$, *where* $H = y^2 + z^2 - (r-v)^2$.

1. *The ideal* J *is generated by the* 2×2*-minors of the matrix*

$$\begin{pmatrix} f & z & (r-v)+y \\ g & (r-v)-y & z \end{pmatrix}. \tag{5.15}$$

2. *There are the equalities*

$$J \cap \langle (r-v)+y, z\rangle = \langle F, H\rangle, \tag{5.16}$$

$$J \cap \langle (r-v)-y, z\rangle = \langle G, H\rangle. \tag{5.17}$$

Proof. The statement 1 is easy to check. For the proof of statement 2, consider the equalities

$$\begin{aligned} J \cap \langle (r-v)+y, z\rangle &= \langle F, H, \langle (r-v)+y, z\rangle \cap \langle G\rangle\rangle \\ &= \langle F, H, \langle (r-v)+y, z\rangle G\rangle. \end{aligned} \tag{5.18}$$

The last equality follows since $\langle (r-v)+y, z\rangle$ is a prime ideal and $G \notin \langle (r-v)+y, z\rangle$. But now $\langle (r-v)+y, z\rangle G \subset \langle F, H\rangle$, as follows by the trivial relations among the 2×2-minors of the matrix given in statement 1. This proves Equation (5.16). The proof of Equation (5.17) follows by the same line of arguments. □

5.2.2. Elimination

In order to find the defining equation of the surface obtained by the toroidal blowup, we have to find the elimination ideal of $I \subset K[x,y,z,v,w]$ in $K[x,y,z]$ for given $f, g \in K[x,v]$. Of course, this might be done using a computer algebra system which supports Gröbner basis. Since the worst case complexity of computing a Gröbner basis for a given ideal is double-exponential, the main objective is the reduction of computational complexity. Even for polynomials f, g of low degree, a computer algebra system like Singular runs out of memory when eliminating the variables u, v, w from the ideal I as given in Equation (5.8)

We begin with the elimination of the variable w. Afterwards, we reduce the whole elimination process to the computation of a single resultant with respect to v.

Lemma 5.2.3. *With the previous notation, let* $I = \langle F, G, 1 - w(f^2 + g^2)\rangle \subset K[x, y, z, v, w]$*. Then* $I \cap K[x, y, z, v] = \langle F, G, H\rangle$*.*

Proof. In order to prove the equality of the two ideals, we use several facts from commutative algebra (see [BH98] for reference). We will use the abbreviations $S = K[x, y, z, v, w]$ and $R = K[x, y, z, v]$.

First of all, note that $S/I \simeq R[1/h]/\langle F, G\rangle$, where $h = f^2 + g^2$. Therefore, $I \cap R = (\langle F, G\rangle)^{\text{sat}}$, where the saturation is taken with respect to h. That is, $I \cap R = \langle F, G\rangle : h^n$ for $n \gg 0$. We continue by proving the equality

$$\langle F, G\rangle = \langle f, g\rangle \cap J, \tag{5.19}$$

where $J = \langle F, G, H\rangle$. Clearly, $\langle f, g\rangle \cap J = \langle F, G, \langle f, g\rangle \cap \langle H\rangle\rangle$. But now $\{f, g\}$ forms a regular sequence since $f, g \in K[x, v]$ do not have a common component by our general assumptions. Therefore, $\{f, g, H\}$ forms also a regular sequence and $\langle f, g\rangle : H = \langle f, g\rangle$. That implies

$$\langle F, G, \langle f, g\rangle \cap \langle H\rangle\rangle = \langle F, G, \langle f, g\rangle H\rangle. \tag{5.20}$$

But now $fH = Fz + G((r - v) + y) \in \langle F, G\rangle$ and $gH = Gz + F((r - v) - y) \in \langle F, H\rangle$. This finally proofs Equation (5.19).

By virtue of Equation (5.19), it is enough to show that the ideal $\langle F, G, H\rangle$ is saturated with respect to $h = f^2 + g^2$. That is, we have to prove that $\langle F, G, H\rangle : h = \langle F, G, H\rangle$. Because H is irreducible, it follows that $J = \langle F, G, H\rangle$ is an ideal with height $J \geq 2$. By the presentation of J as the determinantal ideal in Proposition 5.2.2, we get height $J \leq 2$ and therefore height $J = 2$. By the Hilbert-Burch Theorem, it follows that R/J is a perfect ideal. So any associated prime ideal P of R/J is of height 2.

Now suppose there is an associated prime ideal P of R/J such that $h = f^2 + g^2 \in P$. Because of $(f^2 + g^2)z - 2(r - v)fg = Ff + Gg \in J \subseteq P$, it follows also that $2(r - v)fg \in P$. Since $r - v \notin P$, we get either $f \in P$ or $g \in P$. Because of $f^2 + g^2 \in P$, this implies in both cases $f, g \in P$. Due to $H \in J \subseteq P$, this shows that $f, g, H \in P$. Since $\{f, g, H\}$ is a regular sequence, it yields that height $P \geq 3$. But this contradicts the fact that height $P = 2$ as a consequence of the unmixedness of J and height $J = 2$. Whence, there is no associated prime ideal P of R/J such that $h = f^2 + g^2 \in P$. Therefore, $J : h = J$, which proves that the ideal is saturated. ☐

The equation of the implicit surface of the blowup is now given by the following corollary.

Corollary 5.2.4. *The defining equation* $\mathfrak{F}$ *of the implicit surface of the toroidal blowup is given by the elimination ideal*

$$\langle \mathfrak{F}\rangle = \langle F, G, H\rangle \cap K[x, y, z]. \tag{5.21}$$

Proof. By our above consideration, the equation of $\mathfrak{F}$ is given by the elimination of the variable v in the ideal $J \subset K[x, y, z, v]$. ☐

5.2.3. Resultants

One way to compute the elimination ideal is to use an elimination ordering and to compute the corresponding Gröbner bases (see e.g. [CLO07]). A simpler method is based on resultants, which can be used to eliminate a single variable in a system of two polynomials. In this section, we will reduce the computation of the polynomial $\mathfrak{F}$ to the calculation of a single, rather simple resultant.

Theorem 5.2.5. *The defining equation* $\mathfrak{F}$ *of the elimination ideal* $\langle\mathfrak{F}\rangle$ *of* $I \cap K[x,y,z]$ *can be computed by*

1. $\mathfrak{F} = \mathfrak{F}(F,H) = \operatorname{Res}_v(F,H)/z$,
2. $\mathfrak{F} = \mathfrak{F}(G,H) = \operatorname{Res}_v(G,H)/z$.

Here, $F = fz - ((r-v)+y)g$, $G = gz - ((r-v)-y)f$, $H = y^2 + z^2 - (r-v)^2$ *and* $f, g \in K[x,v]$ *are obtained from the original* $f, g \in K[u,v]$ *by substituting* $u = x$.

Proof. From Corollary 5.2.4, it follows that the elimination ideal $\langle\mathfrak{F}\rangle$ is given by $J \cap T$, where $J = \langle F, G, H\rangle$ and $T = K[x,y,z]$. Now we apply Proposition 5.2.2 and obtain

$$(J \cap T) \cap (\langle (r-v)+y, z\rangle \cap T) = \langle F, H\rangle \cap T, \tag{5.22}$$

$$(J \cap T) \cap (\langle (r-v)-y, z\rangle \cap T) = \langle G, H\rangle \cap T. \tag{5.23}$$

This implies that $\langle\mathfrak{F}\rangle \cdot \langle z\rangle = \langle \operatorname{Res}_v(F,H)\rangle$ and $\langle\mathfrak{F}\rangle \cdot \langle z\rangle = \langle \operatorname{Res}_v(G,H)\rangle$. To this end, recall that $\langle (r-v)+y, z\rangle \cap T = \langle (r-v)-y, z\rangle \cap T = \langle z\rangle$. □

As a consequence of Theorem 5.2.5, we are able to determine the implicit form of the toroidal blowup by computing a single resultant, which is subsequently divided by the variable z. Clearly, one should compute the resultant, which involves the polynomials of lower degree.

We continue with a consideration of the complexity of the calculation of the resultant. It turns out that this resultant is easy to compute.

Lemma 5.2.6. *The computation of the polynomial* $\mathfrak{F}$ *requires* $\mathcal{O}(\max(\deg_v f, \deg_v g))$ *operations in* $K[x,y,z]$.

Proof. We only show the case of $\mathfrak{F} = \operatorname{Res}_v(F,H)/z = (-1)^{\deg F}\operatorname{Res}_v(F,-H)/z$. For this purpose, we use the notation $F = \sum_{i=0}^{d} f_i v^i$ and $-H = v^2 + (-2r)v + (r^2 - y^2 - z^2)$. Let $\tilde{F} = \operatorname{rem}_v(F, -H)$. Then

$$\operatorname{Res}_v(F,-H) = \operatorname{lcoeff}(-H)^{d-\deg_v \tilde{F}} \operatorname{Res}_v(\tilde{F},-H) = \operatorname{Res}_v(\tilde{F},-H) \tag{5.24}$$

as follows from the properties of the resultant (see [GCL92, Theorem 9.4]). Although the polynomials have their coefficients in $K[x,y,z]$, the remainder $\tilde{F}$ can be easily computed by polynomial long division with respect to v. Recall that $-H$ is a monic polynomial in the variable v. Because of $\deg H = 2$, we have $\tilde{F} = \tilde{f}_1 v + \tilde{f}_0$ with $\tilde{f}_0, \tilde{f}_1 \in K[x,y,z]$. Thus, the final result is obtained by

$$\operatorname{Res}_v(\tilde{F},-H) = \begin{vmatrix} \tilde{f}_1 & \tilde{f}_0 & 0 \\ 0 & \tilde{f}_1 & \tilde{f}_0 \\ 1 & -2r & r^2 - y^2 - z^2 \end{vmatrix} \tag{5.25}$$

$$= \tilde{f}_0^2 + 2r\tilde{f}_1\tilde{f}_0 + (r^2 - y^2 - z^2)\tilde{f}_1^2. \tag{5.26}$$

The number of arithmetic operations in $K[x,y,z]$ for the computation of $\tilde{F}$ is bounded by $\mathcal{O}(d)$, where $d = \max(\deg_v f, \deg_v g)$. The costs for the division $\operatorname{Res}_v(\tilde{F},-H)/z$ are insignificant. □

The proof of Lemma 5.2.6 provides an efficient, easily implementable algorithm for the computation of $\mathfrak{F}$. In fact, only basic arithmetic of polynomials is necessary.

5.2.4. Degree bounds

In general, the result of an implicitization has a degree that is quadratic in the degree of the underlying rational parametrization (see e.g [SA84]). Thus, even for low degree parametric surfaces, the efficient visualization using the implicit form might become infeasible. In the following, we will see that the degree bounds for the implicit surface of the toroidal blowup depend only linear on the degrees of $f, g \in K[u, v]$. These bounds are not only of theoretical interest. They provide significant information on the computational effort for the visualization.

Lemma 5.2.7. *The degrees of x, y and z in $\mathfrak{F}$ are bounded by*

$$\deg_x \mathfrak{F} \leq 2\max(\deg_x f, \deg_x g) + 2 \tag{5.27}$$

$$\deg_y \mathfrak{F} \leq 2\max(\deg_v f, \deg_v g) + 4 \tag{5.28}$$

$$\deg_z \mathfrak{F} \leq 2\max(\deg_v f, \deg_v g) + 3. \tag{5.29}$$

The total degree of $\mathfrak{F}$ is bounded by

$$\deg \mathfrak{F} \leq 4\max(\deg f, \deg g) + 3. \tag{5.30}$$

Proof. With $t \in \{x, y, z, \mathrm{total}_{xyz}\}$, we have the following bounds on the degrees occurring in the resultant (see Lemma 2.3.28 and [Win96, p. 97]):

$$\deg_t \mathrm{Res}_v(F, H) \leq \deg_t F \deg_v H + \deg_t H \deg_v F. \tag{5.31}$$

The result is obtained by inserting the respective degrees of F and H, repeating the process with G and H and selecting the smallest of both bounds. For the total degree and the degree in z we take into account that the resultant is divisible by z. ☐

These bounds show that the degree of the implicit equation of the toroidal blowup does not explode as one would expect from the original definition in Equation (5.8). The key ingredient for the linear bound is the change of the generating set of polynomials from A, B, $1 - w(f^2 + g^2)$ to F, G, H, where H does not depend on f and g. Together with the reduction of the computation to a single resultant, we get the linear degree bounds. As a side effect, the polynomials f and g appear only linear in F and G, while they appear with its squares in A, B and $1 - w(f^2 + g^2)$. This allows to reduce the degree bound by another constant factor.

Remark. The actual total degree of the examples that we consider in Section 5.4 is even much lower than the worst case bound we obtained in Lemma 5.2.7. It seems that the total degree does not exceed the degree of the individual variables. However, it might be possible to construct polynomials that hit the degree bound.

5.2.5. A simplified geometric construction

The previous derivation of the implicit form was a purely algebraic one based on Brodmann's parametrization given in Equation (5.5). We will now study a simplified, geometric approach with utilizes the construction of the angle α. In Figure 5.2, we used a circle of fixed radius $\frac{1}{2}$ as Brodmann suggested. Then, by the computed angle α, Brodmann parametrizes a circle of radius $r - v$ in order to perform the embedding into the torus (see Section 5.1.2). This yields

$$(x(\alpha), y(\alpha), z(\alpha)) = (u, (v - r)\cos\alpha, (r - v)\sin\alpha). \tag{5.32}$$

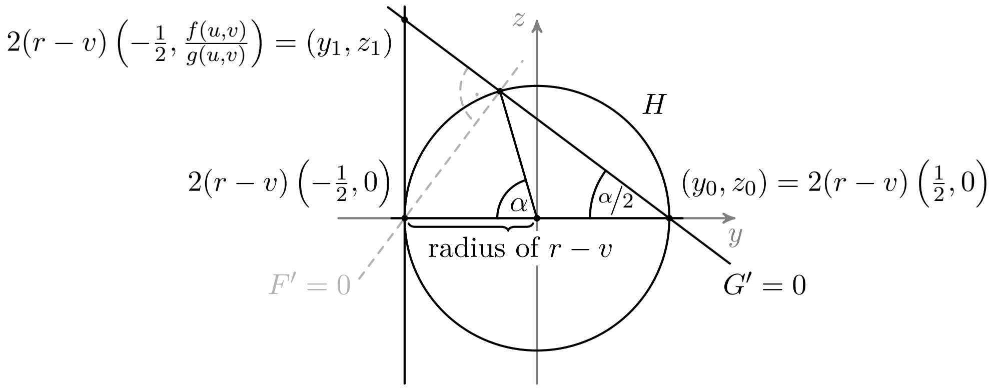

$2(r-v)\left(-\frac{1}{2},\frac{f(u,v)}{g(u,v)}\right)=(y_1,z_1)$
z
H
$2(r-v)\left(-\frac{1}{2},0\right)$
α
$\alpha/2$
$(y_0,z_0)=2(r-v)\left(\frac{1}{2},0\right)$
y
radius of $r-v$
$F'=0$
$G'=0$

Figure 5.5.: Transformation of Brodmann's construction of the angle α to the final (y,z) coordinate system with appropriate scaling of $2(r-v)$ such that the radius of the circle becomes $r-v$ as in Equation (5.3).

Instead of computing the angle based on the circle of radius $\frac{1}{2}$, we could also transform the construction such that the circle is congruent to the one used in the parametrization. That is, the circle is centered at the origin of the (y,z) plane and scaled to have a radius of $r-v$. This is shown in Figure 5.5. The whole construction in the $y-z$ plane is parametrized by u and v (assuming that $r\in\mathbb{R}$ is a constant). Provided that $g\neq 0$, the embedding of $(u,v,f:g)$ into the circle is now given as the intersection of the circle $H=y^2+z^2-(r-v)^2=0$ and the line G' through the points (y_0,z_0) and (y_1,z_1). The implicit equation of G' in the (y,z) plane written in normal form is

$$\begin{aligned}0&=(y-y_0,z-z_0)\cdot(z_1-z_0,-(y_1-y_0))=(y-y_0)(z_1-z_0)-(y_1-y_0)(z-z_0) &(5.33)\\&=(y-(r-v))\left(2(r-v)\frac{f}{g}-0\right)-(-(r-v)-(r-v))(z-0) &(5.34)\\&=2(r-v)(y-(r-v))\frac{f}{g}+2(r-v)z. &(5.35)\end{aligned}$$

Dividing by $2(r-v)$ results in

$$G'=(y-(r-v))\frac{f}{g}+z=0. \tag{5.36}$$

The division is valid since in the case $r-v=0$ the circle is equal to the origin and so is the intersection of the circle and the line. In order to provide a complete embedding of $(u,v,f:g)$, we multiply G' by g, which yields the line

$$G=g(u,v)z-((r-v)-y)f(u,v)=0. \tag{5.37}$$

If $g\neq 0$, we have $V(G)=V(G')$. Otherwise, $V(G)$ is the vertical line $y=r-v$ provided that $f\neq 0$. The intersection of $y=r-v$ and H is equal to (y_0,z_0), which is consistent with Brodmann's choice of $\alpha=\pi$ for $g=0$ in Equation (5.4).

We now compute the intersection of G and H. The substitution $u=x$ given by Brodmann's construction provides the extension from the (y,z) plane to the (x,y,z) coordinate space. The remaining variable v can be eliminated from the two equations $G=H=0$ using the resultant $\operatorname{Res}_v(G,H)$. Clearly, there are at least two (not necessarily different) intersections of $V(G)$

and $V(H)$. If $f = g = 0$, then also $G = 0$, i.e. the whole circle appears as a solution since $V(G, H) = V(H)$. These circles correspond to the exceptional fiber. The degenerate case where f and g have a common factor is examined in detail in Section 5.2.7. Otherwise, there are exactly two intersection of the line $G = 0$ and the circle $H = 0$. The first intersection is always equal to $(y_0, z_0) = (r - v, 0)$, i.e. for any $v \in \mathbb{C}$ there exists a solution with $z = 0$. It follows that $\operatorname{Res}_v(G, H)$ contains a factor z^l for $l \in \{1, 2\}$. Assume that $l = 2$. Then, the line G has a second intersection with the circle at $z = 0$ for any value of v, i.e. G must be horizontal or vertical for any v. This is only possible if $f \equiv 0$ (G is horizontal) or $g \equiv 0$ (G is vertical), which we excluded a priori. Therefore, $l = 1$. Since we are only interested in the second (nontrivial) solution which has been used in Brodmann's constructions to determine the angle α, we divide out the factor z. The remaining polynomial $\mathfrak{F} = \operatorname{Res}_v(G, H)/z$ is the implicit definition of the toroidal blowup as already shown in Theorem 5.2.5 using the algebraic approach.

Note that the polynomial $F = fz - ((r - v) + y)g$, which we used in Theorem 5.2.5 as a second possibility to determine $\mathfrak{F}$ in conjunction with H, is actually a line through $(y, z) = (-(r - v), 0)$ perpendicular to G. The latter property can be derived from the dot product of the line normals $\vec{n}_F = (-g, f)$ and $\vec{n}_G = (f, g)$. Figure 5.5 shows the gray, dashed line $V(F')$ which is equal to $V(F)$ except for $g = 0$.

5.2.6. Deformations

A main motivation of this work is the visualization of deformations of blowups, i.e. the case where f and g depend on a certain number of parameters $a_i \in K$. By an interactive visualization, it is possible to study the deformation of the surface when the a_i vary.

In general, the specialization of parameters and the computation of resultants can not be interchanged. To perform an efficient visualization, we need to ensure that we can compute $\mathfrak{F} \in K[x, y, z, a_1, \dots, a_l]$ symbolically and then specialize the parameters in every frame of the deformation. The following describes the behavior of the specialization of the implicit surface.

Lemma 5.2.8. *Let φ be a homomorphism from $K[x, y, z, a_1, \dots, a_l]$ to a subring obtained by a specialization of a subset of the variables $a_1, \dots, a_l$ to elements in K such that $V(\varphi(f), \varphi(g))$ consists of a finite number of points. Then*

$$\varphi(\mathfrak{F}(F, H)) = \pm\mathfrak{F}(\varphi(F), \varphi(H)), \tag{5.38}$$

$$\varphi(\mathfrak{F}(G, H)) = \pm\mathfrak{F}(\varphi(G), \varphi(H)). \tag{5.39}$$

Proof. Under the specialization φ, the degree of v of f, g could change. This is propagated to F and G. To check what happens with $\mathfrak{F}$, we apply the specialization property of resultants (see Lemmata 2.3.32 and 2.3.33): If $\deg_v \varphi(F) = \deg_v F - k$ and $\deg_v \varphi(H) = \deg_v H$, then

$$\varphi(\operatorname{Res}_v(F, H)) = \varphi(\operatorname{lcoeff}_v(H))^k \operatorname{Res}_v(\varphi(F), \varphi(H)). \tag{5.40}$$

The condition $\deg_v \varphi(H) = \deg_v H$ is always satisfied since $-H$ is a monic polynomial with respect to v. This allows to simplify the previous formula to

$$\varphi(\operatorname{Res}_v(F, H)) = \pm \operatorname{Res}_v(\varphi(F), \varphi(H)) \tag{5.41}$$

and therefore

$$z\varphi(\mathfrak{F}(F, H)) = \varphi(z\mathfrak{F}(F, H)) = \pm z\mathfrak{F}(\varphi(F), \varphi(H))). \tag{5.42}$$

The same arguments hold if $\mathfrak{F}$ is computed using G and H. □

When additional parameters are introduced, $\mathfrak{F}$ will usually contain a lot more terms. However, this does not add much overhead to the visualization, since the a_i might be specialized prior to the rendering process. Due to Lemma 5.2.8, $\varphi(\mathfrak{F}(F,H))$ has the same number of terms as $\mathfrak{F}(\varphi(F),\varphi(H))$. Examples of deformations of blowups and their visualization are shown in Figures 5.9 and 5.10.

5.2.7. The degeneration

The assumption of $\dim(V(\varphi(f),\varphi(g)))=0$ in Lemma 5.2.8 is fulfilled for parameters of an open subset of K^l. A degeneration occurs whenever $\dim V(\varphi(f),\varphi(g))$ becomes positive. We will now discuss these degenerations. For simplicity, let us denote the specialized $\varphi(f),\varphi(g)$ again by f,g. Algebraically, the degeneration means that $f=h\cdot\tilde{f}, g=h\cdot\tilde{g}$ for a non-constant polynomial $h\in K[u,v]$ and relatively prime polynomials $\tilde{f},\tilde{g}$. Therefore, $\tilde{X}=V(\tilde{f},\tilde{g})$ consists of a finite number of points in $\mathbb{A}^2_K$.

In the elimination process, we have to compute the resultant $\operatorname{Res}_v(F,H)$ resp. $\operatorname{Res}_v(G,H)$. In this discussion we restrict ourselves to the first resultant. Define $\tilde{F}=\tilde{f}z-((r-v)+y)\tilde{g}$. Then the definition of the resultant as the product of the pairwise differences of roots (see Theorem 2.3.3) provides the equality

$$\operatorname{Res}_v(F,H)=\operatorname{Res}_v(h,H)\cdot\operatorname{Res}_v(\tilde{F},H). \tag{5.43}$$

The surface $\tilde{\mathfrak{F}}$ of the toroidal blowup of $V(\tilde{f},\tilde{g})$ is obtained by dividing the resultant $\operatorname{Res}_v(\tilde{F},H)$ by z. Thus, the surface $\mathfrak{F}$ of the blowup of the degenerated set $X=V(f,g)$ is

$$\mathfrak{F}=\mathfrak{H}\cdot\tilde{\mathfrak{F}}, \tag{5.44}$$

where $\mathfrak{H}=\operatorname{Res}_v(h,H)$. Let $(u_0,v_0)\in\mathbb{D}$ such that $h(u_0,v_0)=0$. Then a point (x_0,y_0,z_0) with $x_0=u_0$ satisfies $H(x_0,y_0,z_0)=0$ if and only if $v=v_0$. That is, H vanishes on the whole circle $y^2+z^2=(r-v_0)^2$ of the plane $x=x_0=u_0$. In other words, $\mathfrak{H}$ is the image of the cylinder over $h=0$ mapped to the torus by the toroidal embedding. Hence, the surface $\mathfrak{F}$ is the union of the toroidal blowup of $V(\tilde{f},\tilde{g})$ and $\mathfrak{H}$. For an example with $V(\tilde{f},\tilde{g})=\emptyset$, see Figure 5.11a.

5.3. Visualization of the implicit form

During the derivation of the results of the previous section, we used the abstract field K. In order to visualize toroidal blowups, we set $K=\mathbb{R}$ again for the remaining part of this chapter. Thus, the toroidal blowups are now defined in three-dimensional affine space. We will use $V(\mathfrak{F})=V_{\mathbb{R}}(\mathfrak{F})$ for the real zero-set of $\mathfrak{F}$. A real point $P\in V(\mathfrak{F})$ of the implicit form of the parametric surface $P(u,v)$ might originate from complex, not necessarily real parameter values $(u,v)\in\mathbb{C}^2$. Fortunately, this is not the case for toroidal blowups.

Proposition 5.3.1. *Let* $P(u,v)\in V_{\mathbb{R}}(\mathfrak{F})$. *Then* $(u,v)\in\mathbb{R}^2$.

Proof. For a given point $P\in V_{\mathbb{R}}(\mathfrak{F})$ of the implicitly given surface, we use Equation (5.3) for the computation of the parameters u and v such that $P=(x(u,v),y(u,v),z(u,v))$. Therefore,

$$\begin{aligned} y(u,v)^2+z(u,v)^2 &= (v-r)^2(\cos^2\alpha+\sin^2\alpha) && (5.45)\\ &= (v-r)^2 && (5.46)\end{aligned}$$

and we have

$$u = x,\ v = r \pm \sqrt{y^2 + z^2}. \tag{5.47}$$

Since $x, y, z, r \in \mathbb{R}$, the parameters u and v have to be real-valued, too. ☐

Given this result, we proceed with the visualization. Implicitly defined surfaces can either be converted into a polygon mesh for rendering or they can be rendered directly using ray tracing techniques. The latter usually provide an excellent quality of the visualization. Although ray tracing is often considered to be computationally intense, recent advances in graphics hardware technology permit the interactive display of algebraic surfaces (provided the degree is not too large) by a simplified method called ray casting.

In the ray casting algorithm, the geometry visible to the viewer is computed by intersecting rays with the objects in the scene. Basically, at least one ray from the virtual eye point through each pixel of the image plane is considered. In order to find the intersection points of an algebraic surface $\mathfrak{F}(x, y, z) = 0$ and the ray $r(t) = (x(t), y(t), z(t))$, the univariate polynomial equation $\mathfrak{F}(x(t), y(t), z(t)) = 0$ has to be solved using e.g. numerical real root finders. Then, the set of intersection points is clipped to a predefined bounding geometry, which is a torus in our case. More sophisticated clipping techniques might be applied (see Section 5.3.1). The remaining points are textured and illuminated using some light reflection model and the computed color is assigned to the corresponding pixel. For further details, see standard literature on computer graphics like [Hil01].

The above algorithm has been adjusted to fit the requirements for an implementation on graphics processors. This has been demonstrated in [Stu07; Stu09; SS11; RS08; Kno+09] and others. Therefore, we will only discuss the new contributions and omit the details that have been covered in the existing literature.

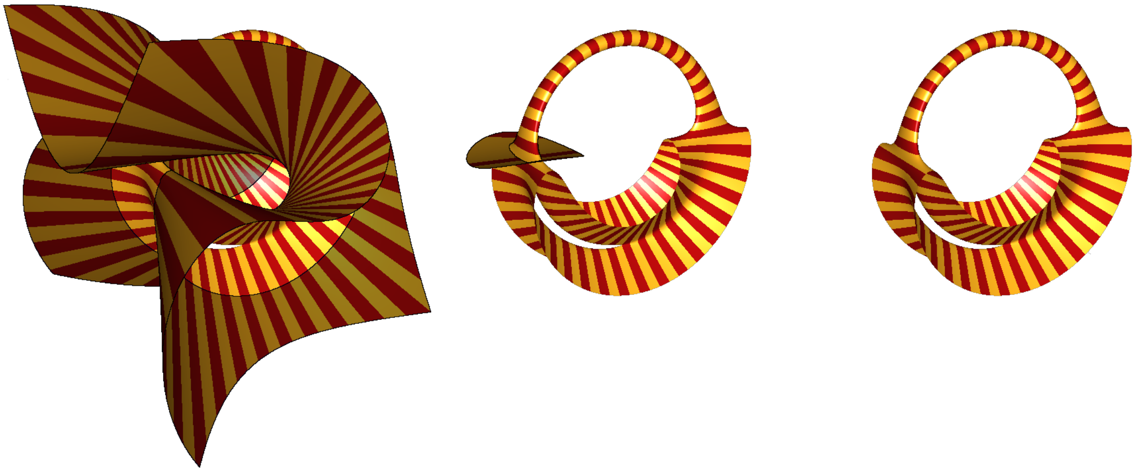
5.3.1. Clipping $V(\mathfrak{F})$ to $\mathbb{T}_{f,g}$

In general, the surface $V(\mathfrak{F})$ is unbounded and contains much more geometry than we actually want to visualize. Because of $\mathbb{T}_{f,g} \subseteq V(\mathfrak{F})$, we have to eliminate $V(\mathfrak{F}) \setminus \mathbb{T}_{f,g}$ for the visualization. The parametric form $P(u, v)$ allows – in principle – the computation of most of the points of $\mathbb{T}_{f,g}$ by the restriction of (u, v) to $\mathbb{D}$. In the implicit form, this correspondence is lost and has to be reconstructed. To be more precise, for a given point $P \in V(\mathfrak{F})$, we have to decide whether there exists an $(u, v) \in \mathbb{D} \setminus V(f, g)$, such that $P = (x(u, v), y(u, v), z(u, v))$, or whether P is located on the exceptional fiber.

We have two simple possibilities to cut off the surface geometry that is outside the $\mathbb{D}$-torus. The first one is to clip each ray against the $\mathbb{D}$-torus, which results in zero, one or two search intervals for the numerical real root finder. The second possibility is to compute all intersections of the ray and the blowup. Then, one checks for each intersection point if it is inside the $\mathbb{D}$-torus. The latter test can be performed in an easy way by checking whether one of the solutions of Equation (5.47) is contained in $\mathbb{D}$. Figure 5.6 illustrates which geometry is cut off.

By definition, all points $P(u, v)$ with $(u, v) \in \mathbb{D} \setminus V(f, g)$ are contained within the $\mathbb{D}$-torus. However, the converse is not true. Thus, the visualization of the implicit form of the blowup inside the $\mathbb{D}$-torus often shows points $P(u, v)$ with $(u, v) \notin \mathbb{D}$. Figure 5.6b illustrates this problem for the case of a simple surface.

For a point inside the $\mathbb{D}$-torus, we always have a solution $(u_0, v_0) \in \mathbb{D}$ and a solution $(u_1, v_1) \notin \mathbb{D}$ (remember that $r > \rho$). The three points P, $P_0 = P(u_0, v_0)$ and $P_1 = P(u_1, v_1)$ are



(a) No clipping.

(b) Clipping against the $\mathbb{D}$-torus.

(c) Clipping to the set $\mathbb{T}_{f,g}$.

Figure 5.6.: Figure (a) shows the surface of $\mathfrak{F}$ without clipping against the $\mathbb{D}$-torus. The surface is unbounded and only clipped to a box for the illustration. The surface parts with dark painting are outside the $\mathbb{D}$-torus. Figure (b) shows the inside part. The dark painted part of figure (b) results from the second solution in Equation (5.47). It does not belong to the toroidal blowup, since $(u_1, v_1) \notin \mathbb{D}$. Figure (c) shows the final result.

located on a circle with radius $|r - v|$ in the plane $x = u$. A small angle between the position vectors of P and P_0 suggests $P = P_0 \in \mathbb{T}_{f,g}$.

As we have seen in Section 5.1.3, the evaluation of P_0 is likely to be numerically very unstable due to the common zeros of f and g in $\mathbb{D}$. Moreover, $P(u_0, v_0)$ is not even defined for $(u_0, v_0) \in V(f, g)$. The error that is introduced in the visualization by using the above criterion to clip the surface is shown in Figure 5.7.

To overcome this difficulty, we investigate the solution (u_1, v_1) instead of (u_0, v_0). We are interested in visualizing the blowup surface for some points in $\mathbb{D}$ and given f and g such that all their common zeros are contained in $\mathbb{D}$. Due to $(u_1, v_1) \notin \mathbb{D}$, the evaluation of $P(u_1, v_1)$ is in most cases well-conditioned. Note that the denominator $f(u_1, v_1)^2 + g(u_1, v_1)^2$ is always non-zero and bounded. Now, we draw P provided

$$\measuredangle(\vec{P}, \vec{P}_1) > \varepsilon_1. \tag{5.48}$$

The motivation for this test is as follows: If $(u_1, v_1) \neq (u_0, v_0)$, the point $P(u_1, v_1)$ is not in $\mathbb{T}_{f,g}$. If the angle between $P(u_1, v_1)$ and P is non-zero or (in the case of round-off error) larger than a predefined ε_1, then P is considered to be in $\mathbb{T}_{f,g}$ and will be visualized.

In the implementation of this test, the angle is not computed directly. Instead, its sine and cosine are used, which are easy to determine by appropriate projections. After some simplifications and with $(f_i, g_i) = (f(u_i, v_i), g(u_i, v_i)), i = 0, 1$, we get

$$\sin \measuredangle(\vec{P}, \vec{P}_1) = \frac{\vec{P}^{\perp} \cdot \vec{P}_1}{||\vec{P}^{\perp}|| \cdot ||\vec{P}_1||} = \operatorname{sgn}(r - v_1) \frac{(-P_z, P_y) \cdot (f_1^2 - g_1^2, 2f_1 g_1)}{|r - v_0|(f_1^2 + g_1^2)}, \tag{5.49}$$

$$\cos \measuredangle(\vec{P}, \vec{P}_1) = \frac{\vec{P} \cdot \vec{P}_1}{||\vec{P}|| \cdot ||\vec{P}_1||} = \operatorname{sgn}(r - v_1) \frac{(P_y, P_z) \cdot (f_1^2 - g_1^2, 2f_1 g_1)}{|r - v_0|(f_1^2 + g_1^2)}. \tag{5.50}$$

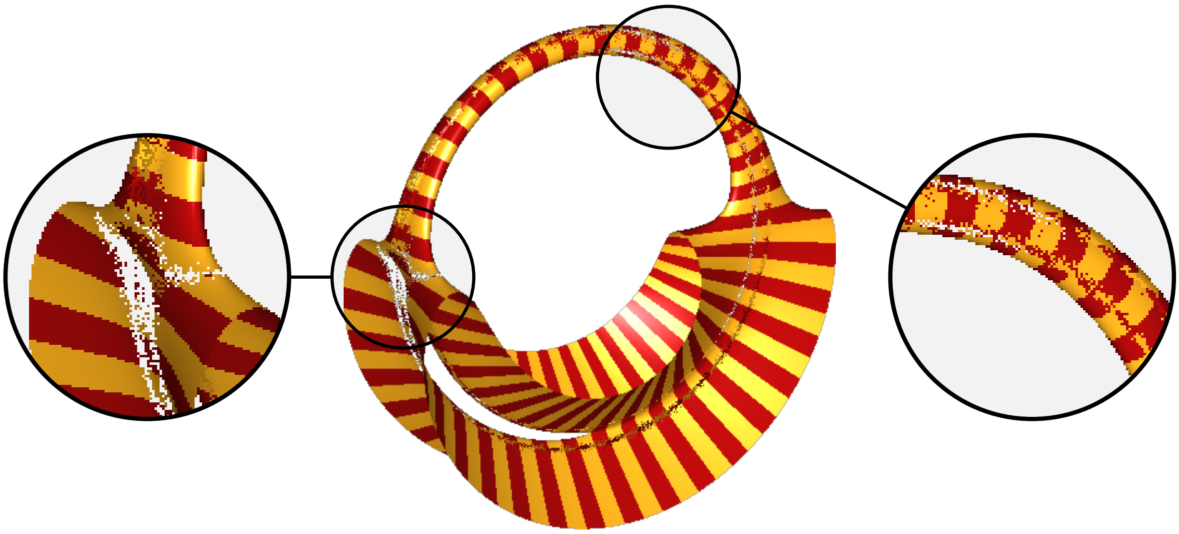


Figure 5.7.: Rendering artifacts caused by the naive idea of clipping $V(\mathfrak{F})$ to $\mathbb{T}_{f,g}$: $P \in V(\mathfrak{F})$ either corresponds to $P_0 = P(u_0, v_0)$ for $(u_0, v_0) \in \mathbb{D}$ or to $P_1 = P(u_1, v_1)$ for $(u_1, v_1) \notin \mathbb{D}$. P should be displayed if $P = P_0$, i.e. if it corresponds to a point above the disc, and discarded otherwise. The evaluation of $P_0 = P(u_0, v_0)$ using the parametric form is numerically very unstable. Therefore, many points close to the exceptional circles are accidentally clipped. This does not happen if Equation (5.48) is used for the clipping instead (see e.g. Figure 5.6c), which is based on the evaluation of $P_1 = P(u_1, v_1)$.

For small angles, a linear function is a good approximation of the sine. The cosine is only needed to decide whether $\measuredangle(\vec{P}, \vec{P}_1) \approx 0$ or $\measuredangle(\vec{P}, \vec{P}_1) \approx \pm\pi$. Therefore, Equation (5.48) transforms into the condition

$$|\sin \measuredangle(\vec{P}, \vec{P}_1)| > \varepsilon_1 \wedge \cos \measuredangle(\vec{P}, \vec{P}_1) \geq 0 \tag{C1}$$

for the acceptance of P as a valid solution.

This test works well as long as P_0 and P_1 are far apart. It will fail e.g. at self intersections of the surface, so that more geometry is clipped than actually necessary. This results in small gaps in the visualization. In order to improve the image quality, we resolve this by accepting a point P that does not pass the test (C1), but satisfies $\measuredangle(\vec{P}_0, \vec{P}_1) < \varepsilon_2$. With the above notation we have

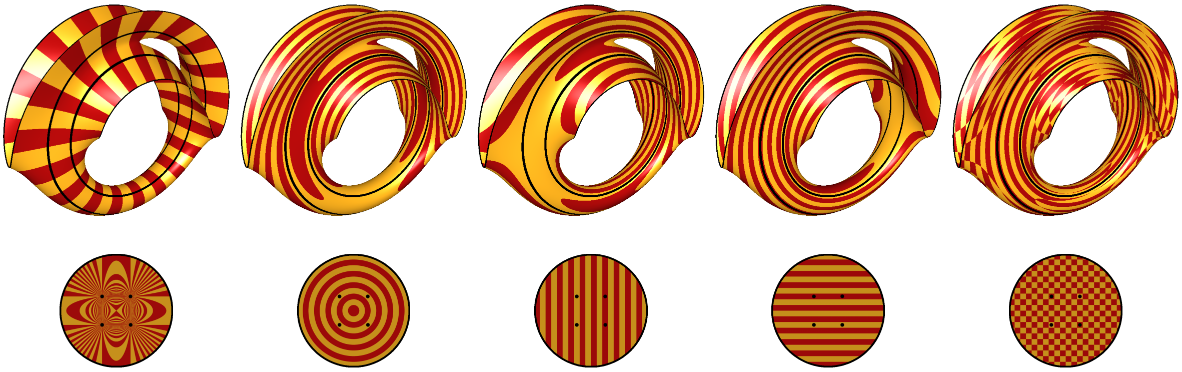
$$\sin \measuredangle(\vec{P}_0, \vec{P}_1) = \frac{\vec{P}_0^{\perp} \cdot \vec{P}_1}{||\vec{P}_0^{\perp}|| \cdot ||\vec{P}_1||} = -\frac{(2f_0g_0, g_0^2 - f_0^2) \cdot (f_1^2 - g_1^2, 2f_1g_1)}{(f_0^2 + g_0^2)(f_1^2 + g_1^2)}, \tag{5.51}$$

$$\cos \measuredangle(\vec{P}_0, \vec{P}_1) = \frac{\vec{P}_0 \cdot \vec{P}_1}{||\vec{P}_0|| \cdot ||\vec{P}_1||} = -\frac{(f_0^2 - g_0^2, 2f_0g_0) \cdot (f_1^2 - g_1^2, 2f_1g_1)}{(f_0^2 + g_0^2)(f_1^2 + g_1^2)}. \tag{5.52}$$

So, a previously discarded point P is visualized, provided

$$|\sin \measuredangle(\vec{P}_0, \vec{P}_1)| < \varepsilon_2 \wedge \cos \measuredangle(\vec{P}_0, \vec{P}_1) \geq 0. \tag{C2}$$

In order to completely close the gap which is caused by the angle ε_1 in (C1) close to a self intersection, the error bound ε_2 is chosen to be slightly larger than ε_1. The results of the combined tests are shown in Figure 5.6c. In this example, the clipping is performed correctly at all pixels of the image.



(a) $m(\text{atan2}(z,y))$ (b) $m(\sqrt{u^2+v^2})$ (c) $m(u)$ (d) $m(v)$ (e) $m(u) \oplus m(v)$

Figure 5.8.: Some texturing patterns of blowup surfaces and the underlying disc $\mathbb{D}$ using Equation (5.53). The function $m(\cdot)$ computes a Boolean value that is used to select either the red or the yellow material. In (a), the texturing of a point is derived from its position in the $\mathbb{D}$-torus. In the other examples, the surface is textured by some texturing of the disc $\mathbb{D}$ according to the (u,v) coordinates. The black color illustrates $V(f,g) = V(u^2-1, v^2-1) = \{(\pm 1, \pm 1)\}$ and the boundary of the disc $\mathbb{D}$ and how both are projected onto the toroidal blowup. That is, the black colored circles in the toroidal blowups visualize the exceptional fiber. In (e), the $\oplus$ operator denotes the exclusive disjunction.

Remark. Note that only the sign of the cosine is used in both tests. Therefore, the divisions in Equations (5.50) and (5.52) are unnecessary and might be skipped. In addition, it is important to mention that the test (C2) needs an expression similar to those for the computation of $P(u_0, v_0)$, which one should avoid due to the numerical instabilities. Nevertheless, in this test, the instability is only significant for the small number of points that are close to the exceptional fiber and additionally close to a self intersection. Notable clipping errors are rare in practice.

5.3.2. Texturing the surface

The texture of the surface should be chosen to enable a viewer to obtain additional information about the blowup or to understand its structure more easily. Two types of coloring schemes seem to be useful: Either the texture of a point is chosen with respect to its (x,y,z)-coordinates or with respect to its (u,v)-coordinates. For each point $P \in \mathbb{T}_{f,g}$, the material is computed using

$$m(d) = \begin{cases} true & (s \cdot d \bmod 2) > 1, \\ false & \text{otherwise}, \end{cases} \tag{5.53}$$

where d is a measure based on either the (x,y,z)- or (u,v)-coordinates of P, s is some user defined factor to scale the pattern, mod is the floating point modulus and the Boolean values are interpreted as two different materials. A few examples of texturing schemes utilizing $m(\cdot)$ are illustrated in Figure 5.8 for the blowup of $X = \{(\pm 1, \pm 1)\}$ as the zero set $V(f,g)$ with $f = u^2 - 1, g = v^2 - 1$.

In Figure 5.8a, the parameter d of a point P is based on the angle between $\vec{P}$ and the y axis. Mapping the resulting coloring scheme to the disc $\mathbb{D}$ produces the contour lines of $z = f(u,v)/g(u,v)$, i.e. of the non-toroidal blowup of the disc. The other texturing schemes in

Figure 5.8 compute d using the (u, v) coordinates of P. Thus, actually the disc $\mathbb{D}$ is textured. Mapping this texture onto the surface shows how different subdivision schemes of the parameter domain (u, v) would affect a triangulation of the surface that is obtained from the parametric form. The problems of a uniform subdivision of the (u, v)-domain (mentioned in Section 5.1.3) become clear in Figure 5.8e since the checkerboard pattern is heavily stretched in the toroidal blowup.

5.3.3. Highlighting the exceptional fibers

While all points $\mathbb{D} \setminus V(f, g)$ are mapped to a single point in the $\mathbb{D}$-torus, the points $X = V(f, g)$ appear as circles in the toroidal blowup. In order to visualize these circles on the surface, it is not sufficient to test $|f(u, v)| \leq \varepsilon$ and $|g(u, v)| \leq \varepsilon$. This leads to circles of non-constant width in the visualization.

A more appropriate choice is to use the (u, v) coordinates of the exceptional points directly. For many interesting examples (including those shown in this chapter), the set of exceptional points $X = V(f, g)$ is known a priori or can be computed easily. Thus, we can either render a torus of small width over each exceptional point or draw points from $\mathbb{T}_{f,g}$ with similar (u, v) coordinates in a different color. This first approach is used in Figure 5.9b. Examples of the latter technique are shown in Figures 5.8, 5.10 and 5.11.

5.4. Implementation and results

In the previous sections, an efficient way to find the implicit equation of the parametric form of a toroidal blowup has been described. We discussed a ray tracing technique for its visualization as well as several clipping methods in order to get the correct visualization of the toroidal blowup over a disc $\mathbb{D}$ as suggested by Brodmann in [Bro95]. In addition, we included the visualization of the exceptional fiber. We also investigated approaches on how to avoid artifacts that might occur in particular visualizations. In this section, an implementation of a renderer is discussed. This is done by means of several examples illustrating the difficulties that might occur.

5.4.1. RealSurf as a tool for visualizing toroidal blowups

The program REALSURF (see [Stu07; Stu09; SS11]) was designed for the interactive visualization of implicit surfaces in real time. It supports additional parameters for an interactive deformation of the given surface. Using this tool, the ray tracing and clipping routines are processed directly on the GPU of the graphics card. For the experiments, an NVIDIA GEFORCE GTX 460 has been used. With an additional script that implements the newly introduced clipping and texturing and the highlighting of the exceptional fiber, REALSURF is used for all ray casting based renderings of toroidal blowups in this chapter. The visualization is performed in real-time for all of them.

It is worth to mention that the current prototypical implementation of the rendering algorithm is efficient for surfaces that have a compact representation with few terms. Our equations for the toroidal blowups are in monomial form and, hence, contain a lot of terms. First experiments suggest that for the examples considered here, the speed of the visualization might be increased by a factor of 5 to 10 by taking advantage of appropriate data structures for polynomials with dense coefficient arrays.

5.4.2. Discussion of examples

Let us look at some practical results. In Figures 5.9 to 5.11, the texturing is performed according to Figure 5.8a. Therefore, the coloring of the disc $\mathbb{D}$ corresponds to contour lines of $z = f/g$. The sets $V(f)$ and $V(g)$ are illustrated in $\mathbb{D}$ using white color, while $X = V(f, g)$ is marked using black dots. Refresh rates for the interactive rendering at a display size of 512×512 are listed in the captions of the figures.

Deformation of points

In Figure 5.9, the common zero-set of $f = au + v^2 - 2a^2, g = (u - 2a)(u - a)$ is deformed by the parameter $a \in \mathbb{R}$. We get two simple points $(a, \pm a)$ and a double point $(2a, 0)$ on a parabola for $a \neq 0$. The construction is symmetric for $+a$ and $-a$. The three points collapse into a fourfold point for $a = 0$ in the origin. This surface is known as Whitney's double umbrella (topologically equivalent to Plücker's conoid).

Deformation of defining equations

In Figure 5.10, the deformed polynomials $f = u^2 + av^2 - a - 1$ and $g = au^2 + v^2 - a - 1$ always intersect in the same set of points $\{\pm 1, \pm 1\}$ provided that $a \in \mathbb{R} \setminus \{\pm 1\}$. For $a < 0$, $V(f)$ and $V(g)$ are hyperbolas, while they are ellipses for $a > 0$. Both yield two parallel lines for $a = 0$. Note that Section 5.4.2 (i.e. $a = 0$) shows the computer generated image of the toroidal blowup of 4 points of the plane included as a hand drawing in [Bro95].

A degeneration

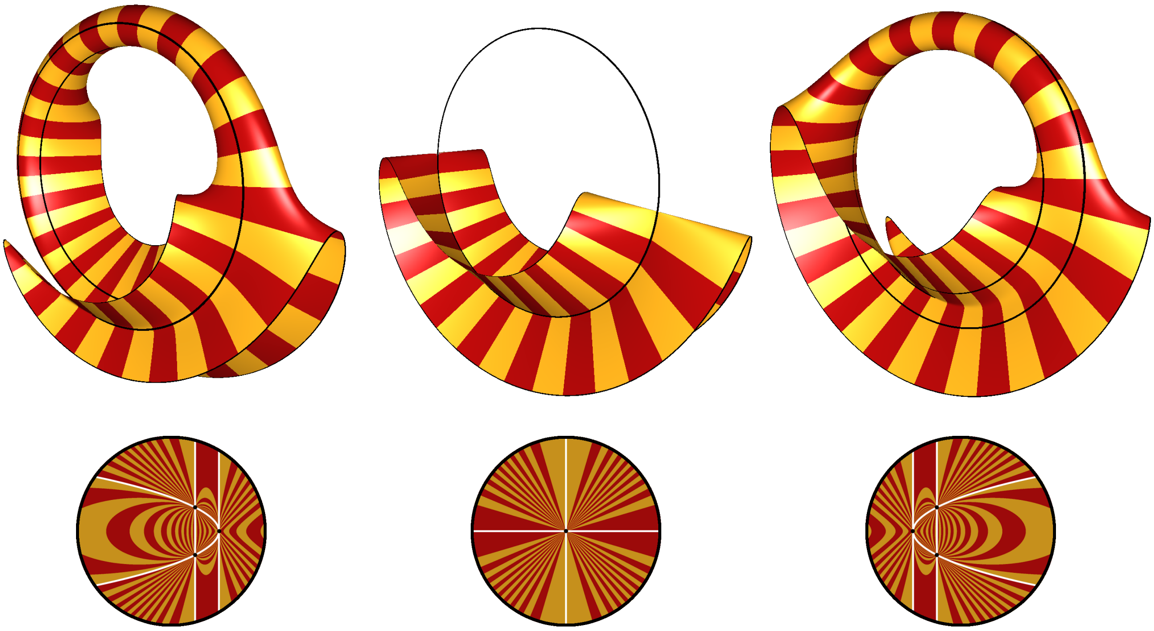
Using the values $a = \pm 1$ in the previous example gives the degenerate case $\dim(V(f, g)) \neq 0$ mentioned in Section 5.2.7. Figure 5.11a shows the rendering for $a = 1$ such that $f(u, v) = g(u, v) = u^2 + v^2 - 2$. Due to Section 5.2.7, the degeneration yields the plane $y = 0$ and the torus with tube radius $\sqrt{2}$ and central radius 8. For $a = -1$, the set $V(f, g)$ consists of the union of the two lines $u + v$ and $u - v$ intersecting in the origin.

Double line and elliptic curve

Figure 5.11b shows a toroidal blowup based on the intersection of a double line $g = v^2$ and an elliptic curve $f = v^2 - u(u - a)(u - b)$. The configuration allows deformations by two parameters. In general, $X = V(f, g)$ consists of three double points. In the case of $a = b \neq 0$, the curve f degenerates to a nodal cubic and X consists of a double and a fourfold point. For $a = b = 0$, it becomes a cuspidal cubic and X consists of a sixfold point at the origin. It is remarkable to see the detailed display of the singularities of the toroidal blowup.

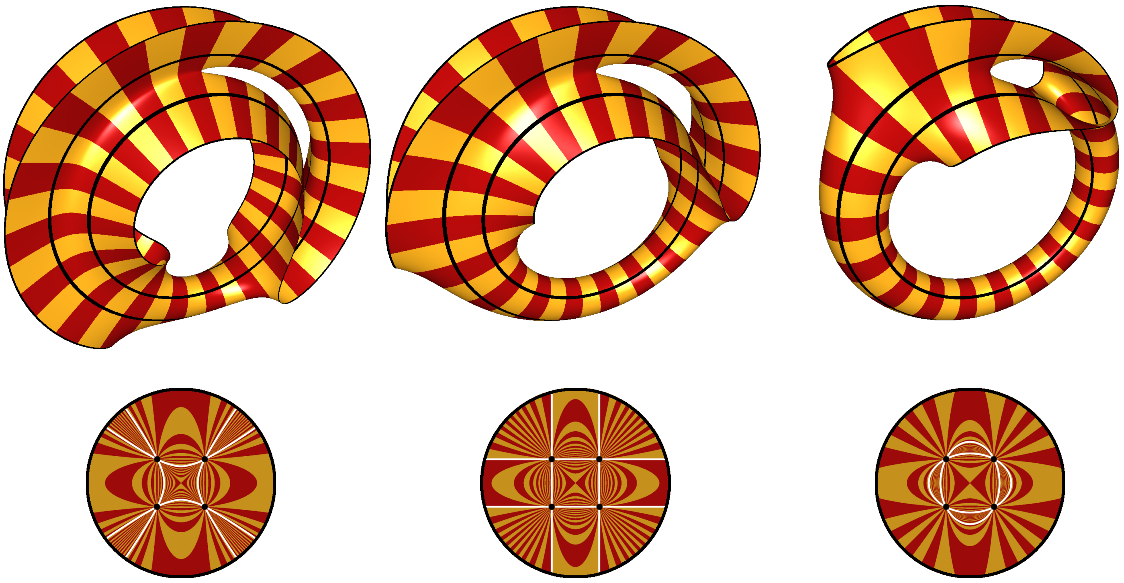
The algebraic wizard hat

The amusing surface in Figure 5.11c, which was given the name "The Algebraic Wizard Hat", was found by coincidence when experimenting with the toroidal blowup based on the Chmutov Curve $f = T_4(u) + T_4(v)$ with the Chebyshev polynomial of the first kind $T_4(x) = 8x^4 - 8x^2 + 1$ of degree 4 and a circle $g = u^2 + v^2 - a^2$. The circle is chosen with a large radius a not intersecting the Chmutov Curve. Hence, the surface contains no exceptional fiber and does not span across the whole torus. Further amazing surfaces might be created by this approach.



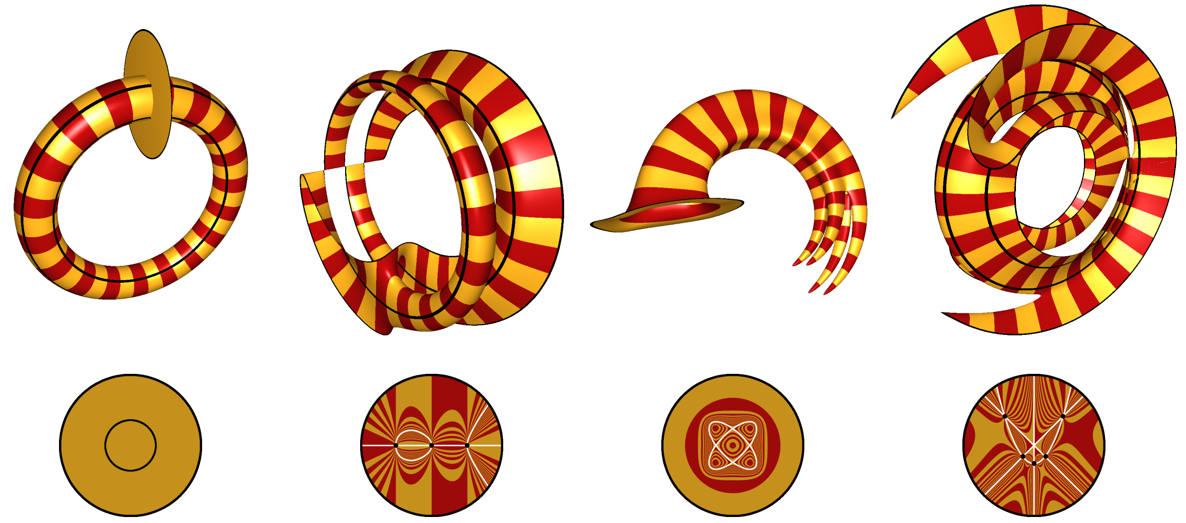
(a) $a = -\frac{1}{2}$ (b) $a = 0$ (c) $a = \frac{1}{2}$

Figure 5.9.: Visualization of the toroidal blowup of three dynamic points with $f(u, v) = au + v^2 - 2a^2$, $g(u, v) = (u - 2a)(u - a)$, $r = 4$ and $\rho = 2$. For $a = 0$, the double point and the two simple points collapse into a fourfold point. The surface $\mathfrak{F}$ is of total degree 5 and is rendered with approximately 125 frames per second.



(a) $a = -\frac{1}{2}$ (b) $a = 0$ (c) $a = \frac{1}{2}$

Figure 5.10.: Visualization of the toroidal blowup of four static points with $f(u, v) = u^2 + av^2 - a - 1$, $g(u, v) = au^2 + v^2 - a - 1$, $r = 8$ and $\rho = 4$. The equation of the surface is of total degree 7 and is rendered with approximately 80 frames per second.



(a) A degeneration of the example in Figure 5.10 with $a = 1$. $\deg(\mathfrak{F}) = 7$; 80 fps.

(b) Elliptic curve intersecting a double line. $\deg(\mathfrak{F}) = 7$; 55 fps.

(c) “The Algebraic Wizard Hat ”. $\deg(\mathfrak{F}) = 9$; 12 fps.

(d) Blowup of five points in the plane. $\deg(\mathfrak{F}) = 7$; 36 fps.

Figure 5.11.: Some more examples of toroidal blowups.

Blowup in five points

The surface in Figure 5.11d shows the toroidal blowup of the plane in 5 points $X = V(f, g)$ with $f = u^2 - v - 1, g = u(u - v)(u + v)$. The clipping still works well for this complex example, but the numerical real root finder in the underlying ray-casting algorithm fails for a few pixels close to the self intersection of $V(\mathfrak{F})$ on the right side of the picture.

5.5. Conclusion and perspectives

The combination of the low-degree implicit form of the toroidal blowup and its interactive visualization using GPU programming yields a significant improvement compared to previous approaches. We are not only able to produce computer generated images of the hand drawn pictures of Brodmann (see [Bro95, Figures 10 and 11]), but also to interactively visualize toroidal blowups of the disc $\mathbb{D}$ given by arbitrary coprime polynomials $f, g \in \mathbb{R}[u, v]$. The method allows interactive deformations of a set of given points as well as the deformation of the defining equations $f, g \in \mathbb{R}[u, v]$ for a fixed set X of finite points such that $X = V(f, g)$. Furthermore, texturing patterns are easily applied on a per-pixel basis. This gives a better insight into the relation between the disc $\mathbb{D}$ and the toroidal blowup $\mathbb{T}_{f,g}$.

For further investigations, it would be interesting to visualize the toroidal blowup of a given plane curve C whose singularities are contained in $V(f, g)$ (see e.g. $C_{\mathbb{T}_{f,g}}$ in Figure 5.1). In order to give a better understanding of the geometry of the blowup, f and g might be highlighted on $\mathbb{T}_{f,g}$. Both problems do not seem to be easily solvable if the curves are required to be drawn with constant width on the toroidal blowup.

6. Parallel computation of resultants on graphics processing units

6.1. Motivation

As we have seen in previous chapters, resultants are one of the fundamental symbolic operations on polynomials that help to understand their common zero set. Unfortunately, their computation is a time consuming task, especially in the multivariate case. In order to reduce the running time in practice, one can try to improve existing sequential algorithms or try to parallelize them. Here, we will investigate a parallel algorithm for the bivariate resultant that is well suited for an implementation on CUDA-capable graphics processing units (GPUs). It shows a remarkable speedup compared to the sequential version provided by the computer algebra system Wolfram Mathematica 6. The main contributions in this chapter are:

- Complete implementation of a modular resultant algorithm over $\mathbb{Z}[x, y]$.
- Efficient way to deal with so called "unlucky" homomorphisms.
- Optimization of modular multiplication of 32 bit integers for GPUs.
- Different parallelization schemes for global and shared GPU memory.
- Large speedup compared to an implementation of the sequential algorithm.

Most of these results have been published in [SS12].

6.2. Related work

This work is based on the modular resultant algorithm presented in [Col71]. There, the multivariate input polynomials over $\mathbb{Z}$ are mapped onto a certain number of univariate polynomials over prime fields. Resultants are calculated for the univariate case and then combined into the final resultant over the integers. The algorithm has been adapted for distributed systems [Bub+95] and for shared memory machines [HL94]. Most recently, there has been some effort to implement algorithms for computing resultants and subresultants on GPUs. One example for the resultant is [Eme10a; Eme10b]. It mainly differs from [Col71] in the way univariate resultants are computed. Instead of using repeated polynomial division with remainder, they build upon a linear algebra approach that exploits the special structure of the Sylvester matrix. Their implementation has been used in [BES11] in order to speed up bivariate polynomial system solving. In [MP11], subresultants are computed on GPUs but they aim for solving modular systems of polynomials so that some parts of the combine phase are unnecessary and, therefore, unimplemented. Note that these recent approaches have been published at about the same time or even later than [SS12], which has been presented in the year 2010.

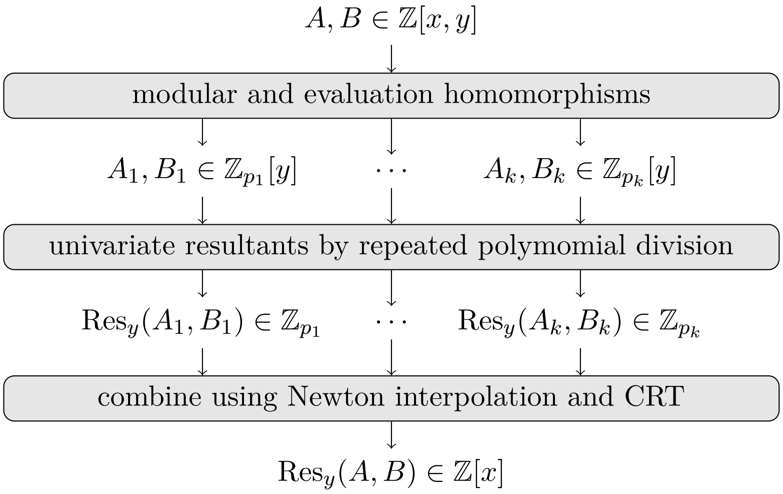

$A, B \in \mathbb{Z}[x, y]$
modular and evaluation homomorphisms
$A_1, B_1 \in \mathbb{Z}_{p_1}[y]$
$\cdots$
$A_k, B_k \in \mathbb{Z}_{p_k}[y]$
univariate resultants by repeated polymomial division
$\mathrm{Res}_y(A_1, B_1) \in \mathbb{Z}_{p_1}$
$\cdots$
$\mathrm{Res}_y(A_k, B_k) \in \mathbb{Z}_{p_k}$
combine using Newton interpolation and CRT
$\mathrm{Res}_y(A, B) \in \mathbb{Z}[x]$

Figure 6.1.: Outline of the parallel algorithm for computing bivariate polynomial resultants.

In addition to the scientific papers related to parallelized (sub-)resultant computations, there is a vast amount of literature covering parallel processing and programming on GPUs in general. Its review is out of the scope of this work. However, [CCPG12] covers most of the technical terms required in this chapter.

6.3. Algorithm outline

In this work, a parallel algorithm for computing the bivariate polynomial resultant $\mathrm{Res}_y(A, B)$ for $A, B \in \mathbb{Z}[x, y]$ is implemented on a CUDA-capable GPU. The following sections are structured according to the divide-conquer-combine strategy of the algorithm: In Section 6.4, it is explained how to apply parallel modular reduction and evaluation homomorphisms on the input polynomials. It is shown that the notion of unlucky homomorphisms is unnecessary in the context of polynomial resultants. This also simplifies subsequent stages of the algorithm. In addition, a fast implementation of GPU-based modular arithmetic is provided. The parallel computation of univariate resultants using either global or shared memory of the GPU will be described in Section 6.5. The univariate resultants are determined by repeatedly applying polynomial division with remainder. Finally, an approach for combining the intermediate results into a final resultant polynomial in $\mathbb{Z}[x]$ is presented in Section 6.6. This is based on Newton interpolation and the Chinese remainder theorem (CRT). Figure 6.1 summarizes the algorithm graphically. The described GPU parallelization is compared with the sequential approach in Section 6.7.

6.4. Divide phase: applying homomorphisms

According to Collins [Col71], we can use homomorphisms to split the resultant calculation into several modular tasks. A homomorphism $\varphi : R \to R'$ of commutative rings R and R' induces a homomorphism of $R[x]$ into $R'[x]$. For a polynomial $A = \sum_i^m a_i x^i \in R[x]$, it is defined by $\varphi(A(x)) = \varphi(\sum_i^m a_i x^i) = \sum_i^m \varphi(a_i) x^i$. The following statement is a special case of Lemma 2.3.33 covering only the resultant.

Corollary 6.4.1. *Let* $A, B \in R[x]$. *If* $\deg(\varphi(A)) = \deg(A)$ *and* $\deg(\varphi(B)) = \deg(B) - k$, $0 \leq k \leq \deg(B)$, *then* $\varphi(\operatorname{Res}(A, B)) = \varphi(\operatorname{lcoeff}(A))^k \operatorname{Res}(\varphi(A), \varphi(B))$.

A similar statement holds for $\deg(\varphi(A)) = \deg(A) - k$ and $\deg(\varphi(B)) = \deg(B)$. If both degrees decrease, the homomorphism is often considered to be *unlucky* as in [HL94], because the commonly used repeated division algorithm for computing the univariate resultant (see Section 6.5) can not be applied. This is a drawback for a parallel algorithm, because unlucky homomorphisms have to be detected and discarded. In the task-parallel algorithm presented in [HL94], unlucky homomorphisms are reported back to the main task. This could not trivially be done on the GPUs available at time of writing. To sort out the bad cases, some kind of stream compaction would be necessary (e.g. by applying prefix sum techniques). Here, we solve this problem using the following special case of Corollary 2.3.34.

Corollary 6.4.2. *If* $\varphi(\operatorname{lcoeff}(A)) = \varphi(\operatorname{lcoeff}(B)) = 0$ *for* $A, B \in R[x]$, *then* $\varphi(\operatorname{Res}(A, B)) = 0$.

According to Corollary 6.4.2, the previously unlucky homomorphisms are actually lucky ones, because the value of the homomorphic image of the resultant is immediately known without further calculations. It seems that the authors of previous algorithms were not aware of this very useful fact, which led to unnecessary complicated implementations in the past.

6.4.1. Reduction modulo a prime number

In the first step of the algorithm, we reduce the integer coefficients of A and B modulo pairwise different prime numbers p_i to coefficients in the prime field $\mathbb{Z}_{p_i}$. This completely eliminates the occurrence of expression swell in subsequent calculations. Several prime numbers are needed in order to reconstruct the integer coefficients of the resultant with the help of the Chinese remainder theorem (see Section 6.6.2). The number of primes needed is bounded by the size of the coefficients of the resultant. The following refinement of Lemma 2.3.29 is stated in [Win96, p. 97]. It takes the possibly different bitsizes of the coefficients of A and B into account.

Lemma 6.4.3. *Let* $A(x, y) = \sum_{i=0}^{m} a_i(x) y^i, B(x, y) = \sum_{i=0}^{n} b_i(x) y^i$ *be polynomials in* $\mathbb{Z}[x, y]$ *and let* $a = \max_{i=0}^{m} ||a_i(x)||_1$, $b = \max_{i=0}^{n} ||b_i(x)||_1$. *If* r *is an integer coefficient of* $\operatorname{Res}_y(A, B)$, *then* $|r| \leq (m + n)! a^n b^m$.

Hence, we select k primes $p_0, \ldots, p_{k-1}$ from a list of precomputed prime numbers such that $P = \prod_{i=0}^{k-1} p_i > |r|$. Due to the ring isomorphism $\mathbb{Z}_{p_0} \times \ldots \times \mathbb{Z}_{p_{k-1}} \cong \mathbb{Z}_P$, we are able to reconstruct the resultant coefficients from their modular images later on. If the largest primes numbers are selected first, we need fewer of them to satisfy $P > |r|$. In the implementation, the integer coefficients are represented in a number system with radix 2^{32}. Reduction modulo a prime number p is easily implemented by viewing the integer as a polynomial over $\mathbb{Z}_{2^{32}}$ and evaluating this polynomial in $\mathbb{Z}_p$ at $2^{32} \bmod p$ using Horner's method. According to Corollary 6.4.2, all prime numbers are valid for the reduction. Therefore, we can reduce all coefficients of A and B modulo all prime numbers in parallel without having any dependencies.

6.4.1.1. Arithmetic in prime fields

In the current implementation, each prime number p satisfies $2^{31} < p < 2^{32}$. Handling addition and subtraction in $\mathbb{Z}_p$ on 32 bit integers is trivial. But the result of a 32 bit multiplication $a \cdot b = hi \cdot 2^{32} + lo$ with $0 \leq hi, lo < 2^{32}$ has a size of 64 bit in general. Decomposing $a \cdot b = q \cdot p + r$, where $r = a \cdot b \bmod p$, by using 64 bit integer division to find q is very slow on current graphics processors. Instead, we use a lower bound $\underline{q}$ on q to eliminate the hi part of $a \cdot b$ and reduce the

Algorithm 6.1: Modular multiplication of 32 bit numbers $a, b \in \mathbb{Z}_p$ using only 32 bit integer addition and multiplication.

```
Data: $a, b \in \mathbb{Z}_p$ with $2^{31} < p < 2^{32}$; $u = \lfloor (2^{32}/p - 1) \cdot 2^{32} \rfloor$; all variables are 32 bit
      integers
Result: $r = a \cdot b \bmod p$
begin
    $lo \leftarrow (a \cdot b)_{lo}$                                      ▷ least significant 32 bit of $a \cdot b$
    $hi \leftarrow (a \cdot b)_{hi}$                                      ▷ most significant 32 bit of $a \cdot b$
                                                              ▷ (__umulhi in CUDA C device code)
    $\underline{q} \leftarrow (hi \cdot u)_{hi} + hi$                     ▷ lower bound on $hi \operatorname{div} p$ with $\underline{q} \geq hi \operatorname{div} p - 1$
    $\underline{lo} \leftarrow (\underline{q} \cdot p)_{lo}$; $\underline{hi} \leftarrow (\underline{q} \cdot p)_{hi}$   ▷ parts of $\underline{q} \cdot p$ used to eliminate $hi$
    $hi \leftarrow hi - \underline{hi}$                                   ▷ try to eliminate $hi$
    if $\underline{lo} \neq 0$ then
        $hi \leftarrow hi - 1$                    ▷ subtract carry arising from lower part of $\underline{q} \cdot p$
    if $hi = 1$ then $\underline{lo} \leftarrow \underline{lo} + p$                        ▷ $\underline{q}$ was 1 to small
    $r \leftarrow lo - \underline{lo}$                                    ▷ 32 bit residue
    if $r < lo$ then $r \leftarrow r - p$                                 ▷ cope with overflow
    if $r \geq p$ then $r \leftarrow r - p$                               ▷ reduce modulo $p$
```

residual 32 bit part modulo p afterwards. The exact value of $\underline{q}$ would be $\lfloor (hi \cdot 2^{32})/p \rfloor$, but this would also involve another division. The main ingredient of the fast modular multiplication is the precomputation of $2^{32}/p$. Due to the size of p, it holds that $1 < 2^{32}/p < 2$. Hence, the 0th binary digit of $2^{32}/p$ is implicitly known. The binary digits $-1, \ldots, -32$ are stored as a 32 bit integer u. Now, $1 + u \cdot 2^{-32}$ is a good lower bound for $2^{32}/p$ with a maximum error smaller than 2^{-32}. The product $\underline{q}' = \lfloor hi \cdot (1 + u \cdot 2^{-32}) \rfloor$ is now equal to $\underline{q}$ or $\underline{q} - 1$, depending on the round-off error. The true value of $\underline{q}$ is obtained by checking if $hi \cdot 2^{32} - \underline{q}' \cdot p > 2^{32}$. Because $(hi \cdot 2^{32} - \underline{q} \cdot p) + lo < 3p$, we have to subtract p at most two times to find the final remainder $r = a \cdot b \bmod p$. See Algorithm 6.1 for the pseudocode and Figure 6.2 for a runtime comparison of the above method and the modular multiplication using a 64 bit modulo operation. Finally, modular inversion, the last operation in $\mathbb{Z}_p$, is implemented using the extended Euclidean algorithm.

6.4.2. Evaluating polynomials

Now, let $A, B \in \mathbb{Z}_p[x, y]$. When calculating the resultant with respect to y, the evaluation homomorphism (see [Col71]) simply maps $\mathbb{Z}_p[x, y]$ to $\mathbb{Z}_p[y]$ by evaluating $A(x, y)$ and $B(x, y)$ at $\deg_x(\operatorname{Res}_y(A, B)) + 1$ pairwise different positions $x = x_i \in \mathbb{Z}_p$. It follows from Corollary 6.4.2 that the choice of the x_i is arbitrary as long as $\deg_x(\operatorname{Res}_y(A, B)) + 1 < p$. Due to our choice of $2^{31} < p < 2^{32}$ this limitation is irrelevant in practice. Therefore, we choose $x_i = i$, which also simplifies the reconstruction phase (see Section 6.6.1). The required number of interpolation nodes is bounded by $\deg_x(\operatorname{Res}_y(A, B)) \leq \deg_y(A) \deg_x(B) + \deg_y(B) \deg_x(A)$ (see Lemma 2.3.28 and [Win96, p. 97]).

The evaluation of the polynomials is again based on Horner's method. Due to the independence of the evaluation nodes and the coefficients of A and B, parallelization is easily achieved.

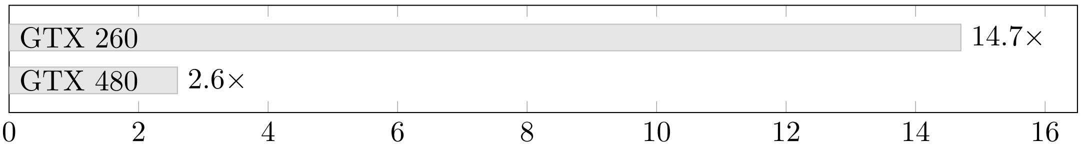


Figure 6.2.: Speedup of the fast 32 bit modular multiplication compared to modular multiplication using 64 bit integer division on NVIDIA GeForce GTX 260 and 480 GPUs.

6.5. Conquer Phase: Calculating resultants of univariate polynomials

We first state some properties of the resultant taken from [GCL92, p. 408, p. 411], which we will use in the parallel algorithms. The corollary can also be derived easily from the properties of signed subresultants. See Section 2.3.

Corollary 6.5.1. *Let* $A = \sum_{i=0}^{m} a_i y^i$ *and* $B = \sum_{i=0}^{n} b_i y^i$ *be univariate polynomials in* $\mathbb{Z}_p[y]$ *of nonzero degree,* $c \in \mathbb{Z}_p$ *a nonzero constant and* $A = Q \cdot B + R$ *a decomposition of* A *by* B *into quotient* Q *and remainder* $R = \sum_{i=0}^{l} b_i y^i$*. Then*

$$\operatorname{Res}(c, B) = c^n, \quad \operatorname{Res}(A, B) = (-1)^{mn} \operatorname{Res}(B, A), \quad \operatorname{Res}(B, A) = b_n^{m-l} \operatorname{Res}(B, R), \tag{6.1}$$

$$\operatorname{Res}(cA, B) = c^n \operatorname{Res}(A, B), \quad \operatorname{Res}(y^k A, B) = b_0^k \operatorname{Res}(A, B),\ k \geq 0. \tag{6.2}$$

The first line of equations in Corollary 6.5.1 allows us to reduce the degrees of the involved polynomials by successive polynomial division until the remainder is constant. The repeated calculation of modular inversion during the polynomial division can be avoided by using polynomial pseudodivision instead. It computes the decomposition $b_n^{m-n+1} A = QB + R$ (see Lemma 2.3.16 or [Knu97b, p. 425ff], for example). Since we are working in the coefficient ring $\mathbb{Z}_p$, the pseudodivision does not cause the severe expression swell it is so well known for. Now, we have

$$\operatorname{Res}(B, A) = b_n^{-n(m-n+1)} \operatorname{Res}(B, b_n^{m-n+1} A) = b_n^{(m-l)-n(m-n+1)} \operatorname{Res}(B, R). \tag{6.3}$$

If A and B are non-constant polynomials, then $(m - l) - n(m - n + 1) \leq -l \leq 0$ and we can calculate $b_n^{n(m-n+1)-(m-l)}$ using the square-and-multiply technique. Suppose that the product of all these factors of the resultant (arising from the repeated pseudodivision) has been computed, then a single modular inverse yields the final result.

6.5.1. Parallelization on global memory

Due to the homomorphic mapping, the first idea to parallelize is to process all independent homomorphic images in parallel. Because it is currently not possible to store a sufficiently large number of images of even moderate degree within the shared memory of the GPU, the first version of the algorithm operates exclusively on global GPU memory. One thread calculates one resultant. If all the homomorphic images of A and B are stored within a 2D-array, each polynomial occupying a single column, then coalesced memory access can be assured for the first polynomial division.

After the division, the degree of the remainder will almost always be $\deg(B) - 1$, but it can be lower for a few or even for all threads. This can force different threads to perform uncoalesced and therefore slow memory accesses during the next iteration. Although this case is rare due to the specialization property of the resultant, the uncoalesced access patterns are kept until the end of the resultant computation. To solve this issue, we apply the last property of Corollary 6.5.1 from right to left and calculate $b_0^{-k} \operatorname{Res}(B, y^k R)$ with $k = \deg(B) - 1 - \deg(R)$ instead of $\operatorname{Res}(B, R)$ in the next iteration. Thus, all threads carry on with a remainder of the same degree. Unfortunately, this construction is not applicable if $b_0 = 0$. In this case, we use that $\operatorname{Res}(B, R) = \operatorname{Res}(B - R, R)$. This follows easily by comparing the Sylvester matrices $\operatorname{Syl}(B, R)$ and $\operatorname{Syl}(B - R, R)$. Since $\deg(R) < \deg(B)$, the second one can be constructed from the first by using only row operations that do not change the determinat. If r_0 is also zero, then $\operatorname{Res}(B, R)$ is zero anyway. Otherwise, $\operatorname{coeff}_0(B + R) = r_0 \neq 0$ and we apply the above degree elevation of R to $\operatorname{Res}(B - R, R)$.

The number of accesses to global GPU memory can be reduced further. After the initial polynomial division, we have $\deg(A) \geq \deg(B) + 1$. The next division gives a remainder of degree $\leq \deg(A) - 2$. Thus, we have to calculate $cA - q_1 yB - q_0 B$ with $q_0, q_1 \in \mathbb{Z}_p$ in order to obtain the second remainder. Both subtractions are done within a single loop by reusing already loaded coefficients of B. Intermediate results are stored locally, thus saving another global memory store and load. This rather technical construction can significantly improve performance on GPU architectures without a memory cache.

6.5.2. Parallelization on shared memory

We now restrict the size of A and B to fit into the shared memory of a block of threads on the GPU. Without loss of generality, we assume $\deg(A) \geq \deg(B)$ and use a thread block with $\deg(B)$ threads to calculate a single resultant. The algorithm is almost identical to the previous one except that additional parallelization is now achieved in the inner loop of the polynomial pseudodivision. Each thread is attached to a single coefficient during the computation. Furthermore, we delay the exponentiation of the b_n factors (see Equation (6.3)) until all of them are known. Then, the exponentiation of all these factors is done in parallel and their product is calculated using parallel reduction. Note that accesses to shared memory are always coalesced. The special treatment used for global memory is never applied. The pseudocode is shown in Algorithm 6.2.

Although the thread utilization is reduced after each iteration due to the decreasing degree of B, this algorithm is quite fast on GPUs that do not have a cache for global memory. A runtime comparison of both algorithms is shown in Table 6.1.

6.6. Combine phase: reconstructing the integer resultant

We now reconstruct the final integer resultant from the homomorphic images of the resultant through polynomial interpolation and by applying the Chinese remainder theorem.

6.6.1. Polynomial interpolation

The interpolation is performed on the points (i, r_i), where r_i is the resultant at $x = i$. We apply the classical Newton interpolation, which represents a polynomial $P(x)$ in the form

Algorithm 6.2: Parallel univariate resultant computation on shared memory.

```
Data: $A, B \in \mathbb{Z}_p[y]$ with $\deg(A) \geq \deg(B)$; $t$ is the thread index
Result: $\mathrm{Res}(A, B)$
begin
    $k \leftarrow 0$                                          ▷ used to index factors of resultant
    if $t = 0$ then $s \leftarrow 1$                                ▷ sign of the resultant
    while $\deg(B) > 0$ do
        $R \leftarrow A$
        for $i \leftarrow \deg(A) - \deg(B)$ to $0$ do              ▷ parallel pseudodivision
            if $t \leq \deg(B)$ then
                $r_{t+i} \leftarrow b_n r_{t+i} - r_{n+i} b_t$
        ▷ now $R = \mathrm{rem}(b_n^{m-n+1} A, B)$
        if $t = 0$ then                           ▷ collect factors for modular inversion
            $m \rightarrow \deg(A)$; $n \rightarrow \deg(B)$; $l \rightarrow \deg(R)$
            $\beta_k \leftarrow \mathrm{lcoeff}(B)$                                  ▷ base of factor
            $e_k \leftarrow n(m - n + 1) - (m - l)$                          ▷ exponent of factor
            $s \leftarrow s \cdot (-1)^{\deg(A)\deg(B)}$                     ▷ update sign of resultant
        $k \leftarrow k + 1$
        $(A, B) \leftarrow (B, R)$
    if $t = 0$ then $\beta_k \leftarrow \mathrm{lcoeff}(B)$; $e_k \leftarrow \deg A$         ▷ base case with $\deg(B) = 0$
    if $t \leq k$ then $\beta_t \leftarrow \beta_t^{e_t}$                  ▷ $k$ exponentiations in $\mathcal{O}(\log e_t)$ steps
    reduce $\beta_0 = \prod_{i=0}^{k-1} \beta_i$ in parallel         ▷ parallel reduction in $\mathcal{O}(\log k)$ steps
    if $t = 0$ then
        $\mathrm{Res}(A, B) \leftarrow \beta_0^{-1} \beta_k s$   ▷ single modular inverse; base case factor; sign
```

$P(x) = \sum_{k=0}^{n} f_k N_k(x)$ with Newton basis polynomials $N_k(x)$. The coefficients f_k are computed efficiently by the scheme of divided differences.

6.6.1.1. Newton basis polynomials

Due to Corollary 6.4.2, we chose the interpolation nodes $0, 1, 2, \ldots, n$. For these nodes, the Newton basis polynomials have the structure

$$N_k(x) = \prod_{l=0}^{k-1} (x - l) = \sum_{l=0}^{k} \begin{bmatrix} k \\ l \end{bmatrix} (-1)^{k-l} x^l, \tag{6.4}$$

where $\begin{bmatrix} k \\ l \end{bmatrix}$ are the Stirling numbers of the first kind. They are recursively given by

$$\begin{bmatrix} k \\ l \end{bmatrix} = \begin{bmatrix} k-1 \\ l-1 \end{bmatrix} + (k-1) \begin{bmatrix} k-1 \\ l \end{bmatrix} \tag{6.5}$$

with base cases $\begin{bmatrix} k \\ 0 \end{bmatrix} = \begin{bmatrix} k \\ k \end{bmatrix} = 1$ and $\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ l \end{bmatrix} = 0$ (see [GKP90, p. 243ff]). In the implementation, $k - 1$ threads[*] calculate the coefficients of $N_k(x)$ in parallel by applying Equation (6.5) on

---

[*] Although the polynomial has $k + 1$ coefficients, the zeroth and leading coefficient are always equal to one. No extra threads are required here.

the coefficients of $N_{k-1}(x)$, which are stored in shared memory. Once $N_k(x)$ is known, it is multiplied by f_k and added to the intermediate value of $P(x)$. Hence, only $N_k(x)$ and $N_{k-1}(x)$ are stored and $N_{k-1}(x)$ is overwritten by $N_{k+1}(x)$ in the next iteration. It is also possible to precompute all of the $N_0(x), \ldots, N_n(x)$ for all prime numbers, but one has to keep the quadratic space requirements in mind.

6.6.1.2. Divided differences

The definition of the divided differences supplied by [GCL92, p. 188] is suitable for our application. With the special interpolation nodes $0, 1, 2, \ldots, n$ it simplifies to

$$f_k = (((\cdots((r_k - f_0)k^{-1}) - f_1)(k-1)^{-1} - \cdots f_{k-2})2^{-1} - f_{k-1})1^{-1} \bmod p_i \qquad (6.6)$$

We start by computing the modular inverses $1^{-1}, 2^{-1}, \ldots, n^{-1} \bmod p_i$ in parallel by n threads. Then, thread k calculates f_k by applying Equation (6.6) step by step. The f_k which is subtracted by each thread in iteration $k+1$ is available since iteration k and distributed among the threads via shared memory at almost no cost.

6.6.1.3. Overlapping computations

A closer look at the previous two algorithms reveals another possibility for optimization. In iteration k, we need k threads to compute the Newton basis polynomials and $n-k$ threads to work on the remaining divided differences. Thus, we can overlap both computations in a single kernel, keeping all $n+1$ threads occupied during the whole interpolation process. Even the sum $P(x) = \sum_{k=0}^{n} v_k N_k(x)$ is computed along with the Newton polynomials. For iteration k of the sum, only $v_k \in \mathbb{Z}_{p_i}$ and $N_k(x) \in \mathbb{Z}_{p_i}[x]$ are needed. Thus, v_{k-1} and $N_{k-1}(x)$ are overwritten, as soon as they are no longer needed. Additionally, we store the growing number of coefficients for the $N_k(x)$ and the shrinking number of modular inverses needed by the v_k within the same array. The intermediate coefficients of the interpolation polynomial $P(x)$ are kept locally by each thread. This results in a memory efficient algorithm using only $n+2$ 32 bit shared memory cells: $n+1$ cells for storing the modular inverses $1^{-1}, \ldots, n^{-1} \bmod p_i$ respectively the coefficients of the $N_k(x)$ and one cell for distributing v_k among the threads.

6.6.2. Lifting from prime fields to integers

The last task is to combine the corresponding coefficients of the interpolation polynomials from different prime fields into an integer number. By the Chinese remainder theorem, a solution for the simultaneous congruences

$$\begin{aligned} u &\equiv u_0 \mod p_0 \\ u &\equiv u_1 \mod p_1 \\ &\vdots \\ u &\equiv u_k \mod p_k \end{aligned} \qquad (6.7)$$

of an integer u and its modular images u_i always exists. We follow Garner's algorithm, which is presented well in the book of Knuth [Knu97b, p. 284ff]. The integer u is written in mixed radix form

$$u = v_0 + p_0(v_1 + p_1(v_2 + \cdots + p_{n-1}v_n)\cdots) \qquad (6.8)$$

with mixed radix digits $v_i \in \mathbb{Z}_{p_i}$ defined as

$$v_i = (\cdots((u_i - v_0)p_0^{-1} - v_1)p_1^{-1} - \cdots - v_{i-1})p_{i-1}^{-1} \bmod p_i. \tag{6.9}$$

The similarity between Equations (6.6) and (6.9) is obvious. Therefore, we use the same scheme for the parallel computation of the mixed radix digits v_i.

6.6.2.1. Converting from mixed radix to fixed radix notation

The conversion process in Equation (6.8) involves alternate multiplication of a prime number and addition of a mixed radix digit, both 32 bit integers, on the intermediate value of u, which is a multi-precision integer stored in memory with respect to the basis 2^{32}. During the conversion, each thread stores a 32 bit digit. The multiplication with the prime number is done in parallel for each intermediate digit of u. For a digit at position t, this results in a 64 bit number whose upper 32 bit have to be propagated to the thread that takes care of digit $t+1$. Note that the addition of the mixed radix digit is easily integrated as the carry at the least significant 32 bit digit stored by thread 0.

In the implementation, we use the warp vote functions[†] available on the GPU to determine quickly if there is still a carry to propagate from thread t to thread $t+1$. Although the carry might ripple from the least significant digit to the most significant one, this rarely occurs in practice. The above approach has been used for reasons of simplicity in this first GPU based implementation.

6.7. Experimental results

Two types of benchmarks are provided in this section. Both are based on a NVIDIA GeForce GTX 260 (compute capability 1.3) and 480 GPUs (compute capability 2.0) and a 2.8 GHz Intel Xeon Prestonia CPU.

The computation times for all parts of the parallel resultant algorithm for 65535 instances of the respective task are listed in Table 6.1. The time needed for one instance is too short for accurate measurements. Modular reduction and evaluation of polynomials are very fast operations. Their complexity is linear in the input size and each task is processed by a single thread without any communication. The complexity of the other parts of the algorithm is quadratic in the input size. On the GTX 260 GPU, the univariate resultant on global memory is relatively slow compared to the parallel variant, that operates on shared memory. Due to the on-chip cache for global memory, the GTX 480 GPU acts very well on the resultant on global memory. This version is sometimes even faster than the resultant on shared memory since it allows full thread utilization without any synchronization. Polynomial interpolation and the lifting process from prime fields to integer coefficients via the Chinese remainder theorem are very similar in their nature. Unfortunately, the lifting to integers involves mixed precision arithmetic and the computation of many modular inverses. The former also leads to a high communication overhead within a thread block, which is reflected in the benchmark results.

In Table 6.2, the computing times for several resultants of polynomials of various degrees and coefficient sizes are listed. The results are compared against the sequential modular resultant

---

[†] On a CUDA capable GPU, the threads are executed in parallel in small groups of size w, where w is a hardware dependent constant. These groups of w threads are called *warps*. A *warp vote* can be interpreted as a reduce operation covering all w threads in a warp. This allows to compute Boolean functions with w arguments (such as the logical disjunction of the carry flags) in constant time.

		input size								
		2	4	8	16	32	64	128	256	512
GTX 260	modular reduction	0.1	0.1	0.2	0.4	0.8	1.5	3.1	6.2	12.3
	evaluation	0.1	0.1	0.2	0.4	0.8	1.6	3.2	6.3	12.5
	resultant global memory	0.6	4.8	20.3	74.9	280.9	1127.6	4409.5	17423.2	94637.4
	resultant shared memory	8.8	32.7	43.0	62.9	121.5	280.6	868.8	3251.0	13230.8
	interpolation	2.5	5.7	10.8	21.4	52.4	150.0	497.1	1790.6	7040.5
	CRT	13.1	35.6	83.2	183.0	530.2	1550.1	4258.2	11790.4	39005.0
GTX 480	modular reduction	0.1	0.1	0.1	0.1	0.1	0.3	0.5	1.1	2.2
	evaluation	0.1	0.1	0.1	0.1	0.2	0.4	0.9	1.8	3.6
	resultant global memory	0.1	0.3	1.2	3.6	12.0	46.1	202.1	761.5	5194.8
	resultant shared memory	4.1	13.3	17.7	26.1	42.9	84.8	249.3	870.0	3254.6
	interpolation	1.4	2.5	4.5	8.4	16.1	35.0	110.5	398.2	1576.6
	CRT	7.0	17.8	40.5	89.1	197.0	542.9	1400.9	3531.9	9062.6

Table 6.1.: Computing times for the different parts of the parallel resultant algorithm for two different NVIDIA GeForce GPUs in milliseconds. The timings reflect the runtime for 65535 instances of the respective task. The input size determines: size of the number to reduce in limbs* (modular reduction); number of coefficients of the input polynomials (polynomial evaluation, univariate resultant on global memory and shared memory, interpolation); prime numbers (CRT).

algorithm implemented in MATHEMATICA 6. We obtain substantial speedups with the parallel approach. From the last column we see, how much time is spent on which part of the parallel modular resultant algorithm. The application of the homomorphisms is almost negligible. Due to the large number of homomorphic images, most time is spent on the univariate resultant. The next part, the Newton interpolation, is quite fast. It also benefits from our choice of interpolation nodes. The final phase, the lifting to integer coefficients, currently involves mixed precision arithmetic and a lot of communication, preventing the algorithm from being as efficient as the interpolation.

6.8. Conclusion and future work

In this chapter, a parallel bivariate polynomial resultant algorithm over the integers, which relies on homomorphisms to achieve parallelization, has been implemented on a CUDA-capable graphics processor. It has been shown that the notion of unlucky homomorphisms is unnecessary in the context of resultants. This greatly simplifies the implementation of the algorithm, since the choice of prime numbers and evalution points is almost arbitrary. All stages of the algorithm exhibit a large amount of parallelism. The presented approach achieves high speedups on the GPU. The parts of the parallel implementation can also be used separately, e.g. to compute resultants over $\mathbb{Z}_p[x, y]$, $\mathbb{Z}[y]$ and $\mathbb{Z}_p[y]$.

Further investigations are necessary to extend the algorithm to other germane symbolic operations. In order to compute signed subresultants, it would be necessary to base the poly-

---

[*] The binary digits of a multi-precision integer are stored in an array whose elements have the size of a machine word which is 32 bit in the above implementation. These 32 bit digits are also called *limbs*.

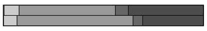
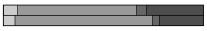
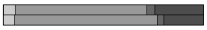
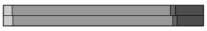
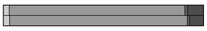
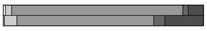
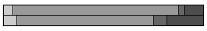
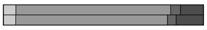
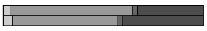
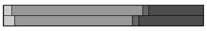
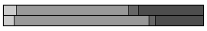
group id	deg(Res)	#moduli	#limbs*	$\deg_y(A)$	$\deg_x(A)$	$\deg_y(B)$	$\deg_x(B)$	MATHE-MATICA	GTX 260	speedup	GTX 480	speedup	time distri-bution
1	510	509	10	26	10	25	10	232.92 s	0.881 s	**264**×	0.258 s	**904**×	
	510	511	6	43	6	42	6	259.47 s	1.107 s	**234**×	0.352 s	**737**×	
	510	502	5	52	5	50	5	278.19 s	1.347 s	**207**×	0.388 s	**717**×	
	510	511	3	85	3	85	3	371.36 s	2.250 s	**165**×	0.588 s	**632**×	
	510	507	4	128	2	127	2	467.83 s	4.195 s	**112**×	1.134 s	**413**×	
2	200	46	1	20	5	20	5	3.95 s	0.038 s	**104**×	0.005 s	**725**×	
	300	69	1	30	5	30	5	10.38 s	0.099 s	**105**×	0.017 s	**613**×	
	400	93	1	40	5	40	5	26.98 s	0.147 s	**184**×	0.041 s	**657**×	
3	200	511	13	20	5	20	5	27.95 s	0.389 s	**72**×	0.790 s	**355**×	
	300	510	9	30	5	30	5	79.02 s	0.685 s	**115**×	0.145 s	**547**×	
	400	510	7	40	5	40	5	148.83 s	0.757 s	**197**×	0.253 s	**589**×	

Table 6.2.: Benchmarks for resultant computations on random polynomials of various degrees. The computation time of the CUDA-based resultant algorithm is compared to the time MATHEMATICA 6 takes for the sequential modular resultant. The last column graphically shows the percentage of the computation time consumed by each task (□ modular reduction, □ evaluation, ▨ resultant on shared memory, ■ interpolation, ■ CRT) for both GPUs (top: GTX 260, bottom: GTX 480). In the first test group, the degree of the resultant and the size of the coefficients in the resultant are held approximately constant. The second group shows several examples, where the size of the coefficients in the input is smaller than 2^{32}. For the third test group, the size of the coefficients in the resultant is approximately constant again.

nomial remainder sequence on the signed subresultant structure theorem (see Theorem 2.3.12). Although this modification will not have major influence on the divide and the conquer phases, the combine phase has to be run through for each of the signed subresultants in the sequence. Hence, it is likely that the lifting from prime fields to integers will be the bottleneck of this approach as Tables 6.1 and 6.2 indicates. However, determining only the degrees of the signed subresultants is much simpler. This only requires a simple reduce operation for each subresultant which computes the maximal degree of all of its homomorphic images.

It is also possible to modify the algorithm to compute polynomial GCDs instead of the resultant if we use the signed subresultant PRS: By applying a reduce operation to all homomorphic images of polynomial i of the signed subresultant sequence we can identify polynomials in the sequence that are identically zero. Afterwards, we can identify the GCD among the subresultants by Theorem 2.3.19. The speed of the GCD computation should be comparable to that of the resultant.

For sparse problems, the chosen parallelization scheme still uses a large number of homomorphic images since it is guided by the worst case bounds on the size of the resultant. This suggests to use the presented implementation primarily to compute resultants of dense polynomials.

Appendix A.

Description and rasterizations of challenging real algebraic plane curves

The following appendix briefly reviews and discusses the different types of challenging curves that were used extensively in Section 3.9 for benchmarking the algorithm proposed in Chapter 3. See [Lab10b] for the construction of these curves, the derivation of their properties and for the description of yet undefined mathematical operators and notions. All images shown in this appendix have been rasterized using the algorithm presented above.

A.1. Solitary points

A *solitary point* p of a curve $\mathcal{C}$ is an isolated singularity with an open neighborhood $U \subset \mathbb{R}$ that does not contain any other points of $\mathcal{C}$. These points form zero-dimensional real components of the curve. Therefore, curve visualization algorithms that rely on sampling techniques miss solitary points with a probability of one in general. Since solitary points are singular and thus critical, the algorithm presented in Chapter 3 rasterizes them correctly.

Challenge 1 (curves with many solitary points). *Visualize the real zero set of*

$$KO_d(x,y) = \prod_{k=0}^{d-1}\prod_{l=0}^{d-1}\left(e^{\frac{2k\pi i}{d}}\sqrt[d]{x} + e^{\frac{2l\pi i}{d}}\sqrt[d]{y} + 1\right) \tag{A.1}$$

for $d \in \mathbb{N}$, $d \geq 3$.

Note that $KO_d(x,y)$ is a real polynomial of degree d. The KO_d curves are so called *rational Harnack curves*. They consist of a smooth real halfbranch and $\frac{1}{2}(d-1)(d-2)$ solitary points. This is the maximum possible number for $d > 4$. See Figure A.1 for examples.

Challenge 2 (higher solitary points). *Visualize the real zero set of*

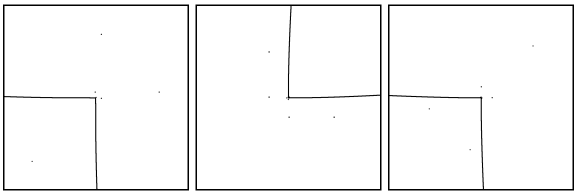
$$SP_{k,l}(x,y) = x^{2k} + y^{2l} \tag{A.2}$$

for $k, l \in \mathbb{N}$ and

$$f^2_{k,l,+}(x,y) = (y - x^k)^l + y^{kl} \tag{A.3}$$

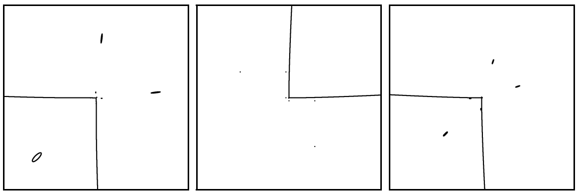
for $k \in \mathbb{N}$ and $l \in 2\mathbb{N}$.

The curves $SP_{k,l}$ and $f^2_{k,l,+}$ have only a single real solution which is the origin, i.e. a solitary point. In contrast to the KO_d curves, which have many simple solitary points, i.e. they are ordinary double points, the single solitary point of $SP_{k,l}$ and $f^2_{k,l,+}$ has a higher multiplicity. At the origin, we have $\operatorname{zerotang}_{SP_{k,l}} = \deg(SP_{k,l})$ and $\operatorname{zerotang}_{f^2_{k,l,+}} = lk^2$, which is quadratic in $\deg(f^2_{k,l,+})$ for a fixed value of l. We do not provide a separate figure since it would only show a dot at the origin.



(a) KO_5 (b) KO_6 (c) KO_7

Figure A.1.: Rasterizations of some of the KO_d curves of Challenge 1. Some of the solitary points are so close to the origin that they are not visible in the above rasterizations.



(a) $\widetilde{KO}_{5,10^{-1}}$ (b) $\widetilde{KO}_{8,10^{-5}}$ (c) $\widetilde{KO}_{9,10^{-1}}$

Figure A.2.: Rasterizations of some of the $\widetilde{KO}_{d,\varepsilon}$ curves of Challenge 3. If ε is choosen small enough, the small ovals look like solitary points. See e.g. $\widetilde{KO}_{8,10^{-5}}$ in (b). Obviously, zooming onto one of the ovals would reveal the actual topology.

A.2. Smooth curves

A curve with many real components is challenging for algorithms that analyze the curve prior to rendering since every (closed) component has at least one critical point. These are often used to subdivide the curve into a set of monotone segments. Furthermore, if the components are small compared to the size of one pixel, this challenge is similar to Challenge 1. Nevertheless, choosing a higher image resolution can enable sampling algorithms to achieve a correct rasterization of small components. This is usually not the case for solitary points.

Challenge 3 (many small ovals). *Visualize the real zero set of*

$$\widetilde{KO}_{d,\varepsilon}(x,y) = KO_d(x,y) + \varepsilon x \frac{\partial KO_d}{\partial x}(x,y) + \varepsilon y \frac{\partial KO_d}{\partial y}(x,y) \tag{A.4}$$

for $d \in \mathbb{N}$, $d \geq 3$, *and* $\varepsilon = 10^{-i}$ *with* $i \in \mathbb{N}$.

These curves have the same number of real components as the KO_d curves but each solitary point is deformed into a small oval if ε is small enough (otherwise some ovals might join). See Figure A.2 for examples.

Challenge 4 (nested ovals). *Visualize the real zero set of*

$$Nest^2_{d,\varepsilon,k}(x,y) = (x+y)\varepsilon^{k+\lfloor d/2 \rfloor} + \prod_{j=1}^{\lfloor d/2 \rfloor} (x^2 + y^2 - \varepsilon^j) \tag{A.5}$$

for $d \in \mathbb{N}$, $d \geq 2$, $\varepsilon = 10^{-i}$ *with* $i \in \mathbb{N}$ *and* $k \in \mathbb{N}_0$.

These curves are challenging because the size of the nested ovals shrinks exponentially with the nesting level. Furthermore, the largest oval has a radius of approximately ε. Note that some of the $\lfloor d/2 \rfloor$ ovals might join depending on the parameters ε and k. See Figure A.3.

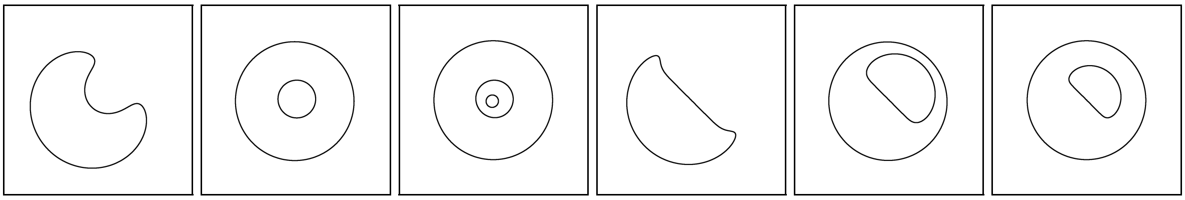
Challenge 5 (small non-real ovals). *Visualize the real zero set of*

$$SP_{k,l,\varepsilon}(x,y) = x^{2k} + y^{2l} + \varepsilon \tag{A.6}$$

for $k, l \in \mathbb{N}$ *and*

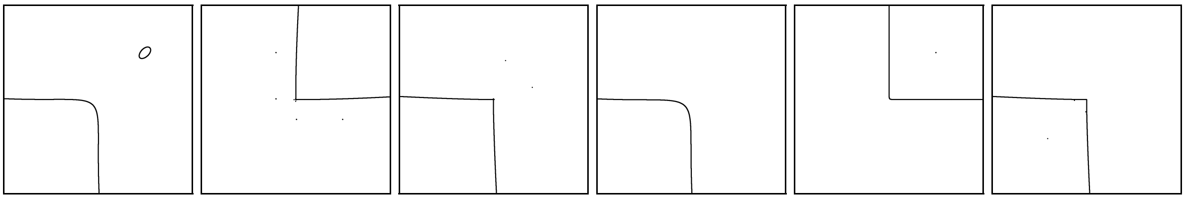
$$f^2_{k,l,+,\varepsilon}(x,y) = (y - x^k)^l + y^{kl} + \varepsilon \tag{A.7}$$

for $k \in \mathbb{N}$, $l \in 2\mathbb{N}$ *and* $\varepsilon = 10^{-i}$ *with* $i \in \mathbb{N}$.



(a) $Nest^2_{4,10^{-1},0}$ (b) $Nest^2_{4,10^{-1},1}$ (c) $Nest^2_{6,10^{-1},2}$ (d) $Nest^2_{12,10^{-1},0}$ (e) $Nest^2_{12,10^{-1},1}$ (f) $Nest^2_{12,10^{-1},2}$

Figure A.3.: Rasterizations of some of the $Nest^2_{d,\varepsilon,k}$ curves of Challenge 4. Depending on ε and k, some of the $\lfloor d/2 \rfloor$ ovals might join. See e.g. $Nest^2_{12,10^{-1},\{0,1,2\}}$ in (d)–(f).



(a) $KO^-_{3,10^{-1}}$ (b) $KO^-_{6,10^{-2}}$ (c) $KO^-_{9,10^{-8}}$ (d) $KO^+_{3,10^{-1}}$ (e) $KO^+_{6,10^{-2}}$ (f) $KO^+_{9,10^{-8}}$

Figure A.4.: Rasterizations of some of the $KO^-_{d,\varepsilon}$ and $KO^+_{d,\varepsilon}$ curves of Challenge 6. The sign in $KO^\pm_{d,\varepsilon}$ determines which solitary points of KO_d become real and which become non-real ovals.

The curves $SP_{k,l,\varepsilon}$ and $f^2_{k,l,+,\varepsilon}$ have no real solution, but their graphs are very close to zero in the neighborhood of the origin due to the high zero tangency of $SP_{k,l}$ and $f^2_{k,l,+}$ at $(0,0)$. Many numerical algorithms treat small values as zero and will therefore assume that the curves in this challenge have real solutions. Exact results may require time consuming computations. Consider for example the fiber $y = 0$. The polynomial $F(x,0)$ has two conjugate complex roots very close to the origin. A real root isolation algorithm based on Descartes (see Section 2.4.2) would have to subdivide the real line until the two complex roots are separated in order to determine that there is no real solution. In contrast, the real root counting method using (principal) signed subresultant sequences would immediately reveal the absence of real roots of $F(x,0)$. Since the curves in this challenge do not have real solutions, no figure is provided.

Challenge 6 (many small real and non-real ovals). *Visualize the real zero set of*

$$KO^-_{d,\varepsilon}(x,y) = KO_d(x,y) - \varepsilon \tag{A.8}$$

and

$$KO^+_{d,\varepsilon}(x,y) = KO_d(x,y) + \varepsilon \tag{A.9}$$

for $d \in \mathbb{N}$, $d \geq 3$, *and* $\varepsilon = 10^{-i}$ *with* $i \in \mathbb{N}$.

In this challenge, all singularities of the KO_d curves on one side of their one-dimensional real component are deformed into small real ovals while the singularities on the other side become small non-real ovals. These curves are not difficult with respect to the type of singularity that is deformed into an oval, but with respect to the number of ovals since they are derived from the KO_d curves, which only have ordinary double points. See Figure A.4 for examples of the $KO^-_{d,\varepsilon}$ and $KO^+_{d,\varepsilon}$ curves.

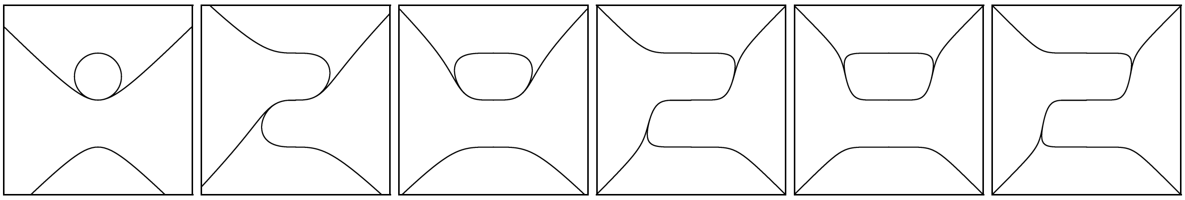

(a) $f^2_{2,2,-}$ (b) $f^2_{3,2,-}$ (c) $f^2_{4,2,-}$ (d) $f^2_{7,2,-}$ (e) $f^2_{8,2,-}$ (f) $f^2_{9,2,-}$

Figure A.5.: Rasterizations of some of the $f^2_{k,l,-}$ curves of Challenge 7. The tangency of the halfbranches at the origin can be raised to an arbitrary high level by adjusting k.

A.3. High tangencies of halfbranches

The following curves show halfbranches that have a high tangency at a common singular point.

Challenge 7 (high tangencies at isolated singularities). *Visualize the real zero set of*

$$f^2_{k,l,-}(x,y) = (y - x^k)^l - y^{kl} \tag{A.10}$$

for $k \in \mathbb{N}$, $k \geq 2$ and $l = 2$.

The curves of Challenge 7 have four real halfbranches connected by an isolated singularity at the origin. There, the halfbranches are hard to distinguish since $\operatorname{tang} f^2_{k,l,-}(0,0) = \frac{2k^2}{2} = \frac{2d^2}{8}$, i.e. $\operatorname{tang} f^2_{k,l,-}(0,0)$ grows quadratically in the degree of the curve. This is a problem for many numerical algorithms since densely packed segments are very similar to a multiple component that has been deformed slightly yielding two simple components. Even small rounding errors can push parts of such curves into the complex domain. See Figure A.5 for correctly visualized examples.

Challenge 8 (high tangencies at non-isolated singularities). *Visualize the real zero set of*

$$ni_m(x,y) = x^m \tag{A.11}$$

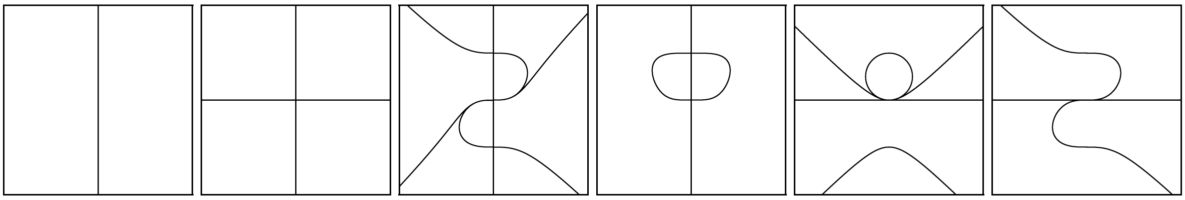
$$ni_{m,k}(x,y) = x^m y^k \tag{A.12}$$

$$nix_{m,n,k,l}(x,y) = x^m (f^2_{k,l,-}(x,y))^n \tag{A.13}$$

$$niy_{m,n,k,l}(x,y) = y^m (f^2_{k,l,-}(x,y))^n \tag{A.14}$$

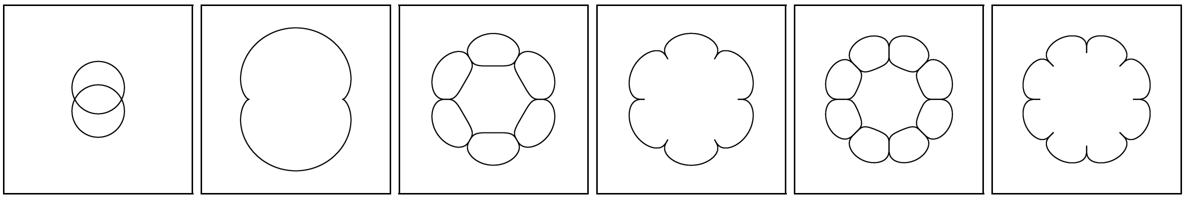
for $k, l, m, n \in \mathbb{N}$, $k, l, m, n \geq 2$.

A non-isolated singularity of a plane curve is also a multiple component of the curve. If an algorithm does not compute the squarefree part of the defining polynomial, the problems mentioned in Challenge 7 will not only occur locally but globally. Although quite challenging for numerical algorithms, the curves ni_m and $ni_{m,k}$ are simple ones for most algorithms with symbolic precomputations since multiple lines are detected and removed easily. Establishing the squarefreeness of $nix_{m,n,k,l}$ and $niy_{m,n,k,l}$ is a harder problem since in contrast to a vertical or horizontal line $f^2_{k,l,-}(x,y)$ is not a factor of $\operatorname{cont}_x(f^2_{k,l,-})$ resp. $\operatorname{cont}_y(f^2_{k,l,-})$. Once the squarefree part is computed, the rasterization of ni_m and $ni_{m,k}$ is trivial and the rasterization of $nix_{m,n,k,l}$ and $niy_{m,n,k,l}$ is equal to the rasterization of $f^2_{k,l,-}$ assuming that vertical and horizontal lines are rendered separately. See Figure A.6 for examples.



(a) ni_m (b) $ni_{m,k}$ (c) $nix_{m,n,3,2}$ (d) $nix_{m,n,4,3}$ (e) $niy_{m,n,2,4}$ (f) $niy_{m,n,3,3}$

Figure A.6.: Rasterizations of the ni_m, $ni_{m,k}$ curves and of some of the $nix_{m,n,k,l}$ and $niy_{m,n,k,l}$ curves of Challenge 8. Note that the geometry of $nix_{m,n,k,l}$ and $niy_{m,n,k,l}$ is independent of m and n since these parameters define the multiplicity of the respective component. The parameter l is not fixed in this challenge. Therefore, the variety of curves occurring in this challenge is larger than in Challenge 7. See e.g. $nix_{m,n,4,3} = x^m(f^2_{4,3,-})^n$ in (d) and $niy_{m,n,3,3} = y^m(f^2_{3,3,-})^n$ in (f).



(a) $C_{2,2}$ (b) $C_{2,3}$ (c) $C_{2,6}$ (d) $C_{2,7}$ (e) $C_{2,8}$ (f) $C_{2,9}$

Figure A.7.: Rasterizations of some of the $C_{k,l}$ curves of Challenge 9.

A.4. Many isolated singularities

A non-isolated singularity, i.e. a multiple component, can be factored out such that the curve contains only a finite number of isolated singularities. It remains to find the isolated singularities which are a subset of the critical points. Therefore, a large number of isolated singularities can be challenging for algorithms, that need to determine singular or critical points.

Challenge 9 (many singularities with high tangencies). *Visualize the real zero set of*

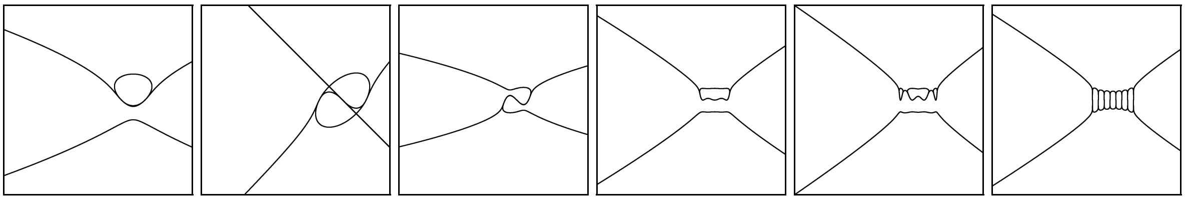
$$C_{k,l}(x,y) = (dfold_{\lfloor l/k \rfloor}(x,y))^k - (x^2+y^2-1)^l \tag{A.15}$$

for $k=2$, $l \in \mathbb{N}$, $l > k$, where

$$dfold_d(x,y) = \prod_{j=1}^{d} \left(x \sin \frac{2\pi j}{d} + y \cos \frac{2\pi j}{d} \right) \tag{A.16}$$

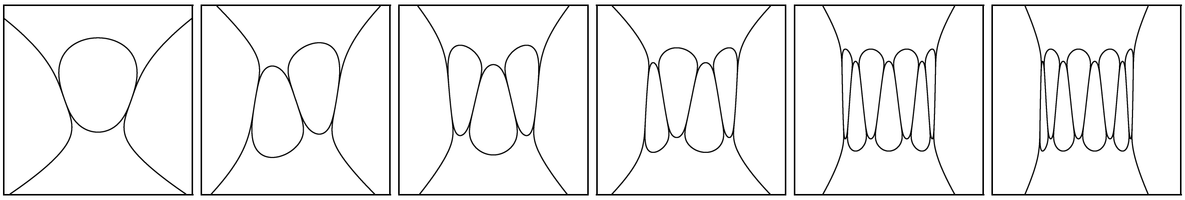
are d straight lines through the origin (see Challenge 13).

The $C_{k,l}(x,y)$ curves have $2\lfloor l/k \rfloor$ isolated singularities at the intersections of a circle and the $\lfloor l/k \rfloor$ straight lines through the origin, i.e. the number of singularities grows linear with $\deg C_{k,l} = 2l$. Additionally, the halfbranches at each singularity have the same tangent direction. See Figure A.7 for examples.



(a) $F^2_{3,2,2,-}$ (b) $F^2_{3,2,3,-}$ (c) $F^2_{5,2,3,-}$ (d) $F^2_{7,2,3,-}$ (e) $F^2_{9,2,4,-}$ (f) $F^2_{9,2,8,-}$

Figure A.8.: Rasterizations of some of the $F^2_{k,l,m,-}$ curves of Challenge 10 which have singularities at integer coordinates.



(a) $FT^2_{3,2,2,-}$ (b) $FT^2_{4,2,3,-}$ (c) $FT^2_{5,2,4,-}$ (d) $FT^2_{7,2,5,-}$ (e) $FT^2_{9,2,8,-}$ (f) $FT^2_{9,2,9,-}$

Figure A.9.: Rasterizations of some of the $FT^2_{k,l,m,-}$ curves of Challenge 11. The coordinates of the singularities are non-rational except for the origin.

Challenge 10 (many singularities with high tangencies at integer coordinates). *Visualize the real zero set of*

$$F^2_{k,l,m,-}(x,y) = \left(y - \left(\prod_{i=1}^{m}(x-i)\right)^{\lfloor k/m \rfloor}\right)^{l} - y^{kl} \tag{A.17}$$

for $l = 2$ and $k, m \in \mathbb{N}$, $2 \leq m \leq k$.

This challenge is very similar to the previous one except that all singularities are located at integer coordinates on the line $y = 0$. See Figure A.8 for examples.

Challenge 11 (many singularities with high tangency at non-rational coordinates). *Visualize the real zero set of*

$$FT^2_{k,l,m,-}(x,y) = (y - T_m(x)^{\lfloor k/m \rfloor})^{l} - y^{kl} \tag{A.18}$$

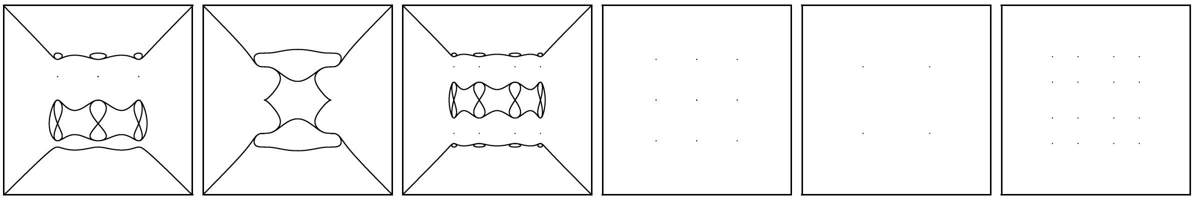
for $l = 2$, $k, m \in \mathbb{N}$, $2 \leq m \leq k$ and

$$T_0(x) = 1 \tag{A.19}$$

$$T_1(x) = x \tag{A.20}$$

$$T_m(x) = 2xT_{m-1}(x) - T_{m-2}(x) \quad \text{for } m \geq 2. \tag{A.21}$$

Geometrically, Challenge 11 seems to be similar to Challenge 10. But the singularities are located at the roots of the Chebyshev polynomials $T_m(x)$, which are all non-rational if $x = 0$ is excluded. It is much more difficult to work with coordinates from an algebraic extension of $\mathbb{Q}$ than with rational coordinates. Depending on the algorithm used for the computation of the (raster) position of the singularities, there can be significant differences in the running times for Challenges 10 and 11. See Figure A.5 for examples of the $FT^2_{k,l,m,-}$ curves.



(a) $FTT^2_{2,2,3,-}$ (b) $FTT^2_{3,2,2,-}$ (c) $FTT^2_{2,2,4,-}$ (d) $FTT^2_{2,2,3,+}$ (e) $FTT^2_{3,2,2,+}$ (f) $FTT^2_{2,2,4,+}$

Figure A.10.: Rasterizations of some of the $FTT^2_{k,l,m,-}$ and $FTT^2_{k,l,m,+}$ curves of Challenge 12.

Challenge 12 (many higher singularities with high tangency at non-rational coordinates). *Visualize the real zero set of*

$$FTT^2_{k,l,m,-}(x,y) = (T_m(y) - T_m(x)^k)^2 - T_m(y)^{2k} \tag{A.22}$$

and

$$FTT^2_{k,l,m,+}(x,y) = (T_m(y) - T_m(x)^k)^2 + T_m(y)^{2k} \tag{A.23}$$

for $k, m \in \mathbb{N}$, $k, m \geq 2$.

In this challenge, the number of singularities at non-rational coordinates is m^2 instead of m as in the previous challenge. This is caused by the substitution of the variable y by $T_m(y)$. Additionally, the singularities are of an higher order. $FTT^2_{k,l,m,-}$ and $FTT^2_{k,l,m,+}$ have the same set of singularities, but for $FTT^2_{k,l,m,+}$ all of them are solitary points, i.e. the $FTT^2_{k,l,m,-}$ curves have no one-dimensional real part.

A.5. Singularities with more than four halfbranches

Until now, most of the curves have only four halfbranches at an isolated singularity. In [Lab10b], the author uses the informal term *complicated singularity* in order to describe singularities which have a larger number of halfbranches. See Figure A.10 for examples.

Challenge 13 (maximum number of halfbranches). *Visualize the real zero set of*

$$dfold_d(x,y) = \prod_{j=1}^{d} \left(x \sin \frac{2\pi j}{d} + y \cos \frac{2\pi j}{d} \right) \tag{A.24}$$

and

$$dfold^{fl}{}_d(x,y) = dfold_d(x,y) - (x^2+y^2)^{\lfloor d/2 \rfloor + 1} \tag{A.25}$$

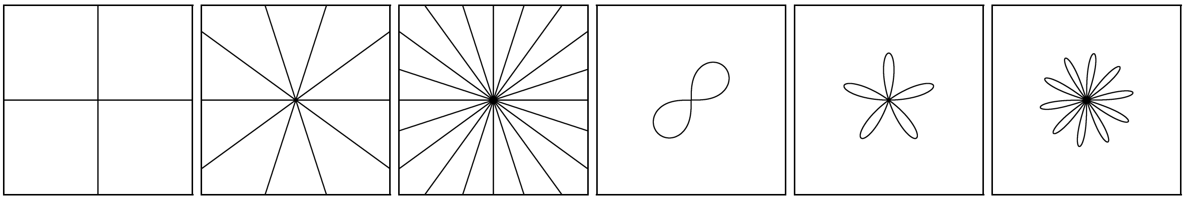
for $d \in \mathbb{N}$, $d \geq 2$.

The $dfold_d$ curves are the d straight lines through the origin which have a d-gon symmetry. At the origin, $dfold_d$ has the maximum number of halfbranches, which is $2d$. Since it is easy to factorize curves that only consist of straight lines, the curves $dfold^{fl}{}_d$ are given. They have almost the same structure at the origin but are irreducible. See Figure A.11 for examples.

Challenge 14 (large number of halfbranches and high tangency). *Visualize the real zero set of*

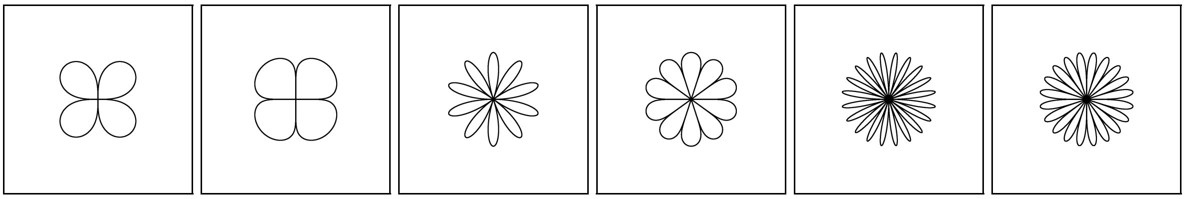
$$dfold_{k,l}(x,y) = (dfold_k(x,y))^2 - (x^2+y^2)^{k+l} \tag{A.26}$$

for $k, l \in \mathbb{N}$, $k \geq 2$.



(a) $dfold_2$ (b) $dfold_5$ (c) $dfold_{12}$ (d) ${dfold^{fl}}_2$ (e) ${dfold^{fl}}_5$ (f) ${dfold^{fl}}_{12}$

Figure A.11.: Rasterizations of some of the $dfold_d$ and ${dfold^{fl}}_d$ curves of Challenge 13. The tangent directions of the halfbranches of $dfold_d$ and ${dfold^{fl}}_d$ are the same at the origin.



(a) $dfold_{2,2}$ (b) $dfold_{2,6}$ (c) $dfold_{5,1}$ (d) $dfold_{5,6}$ (e) $dfold_{10,1}$ (f) $dfold_{10,3}$

Figure A.12.: Rasterizations of some of the $dfold_{k,l}$ curves of Challenge 14. Two adjacent halfbranches have the same tangent direction at the origin. Their tangency is adjustable by the parameter l.

The curves in this challenge look very similar to the ${dfold^{fl}}_d$. They have $4k$ real halfbranches connected to the origin, i.e. the number of halfbranches is not maximal. In contrast to the previous challenge, two adjacent halfbranches have the same tangent direction. The tangency can be raised to an arbitrary high level by adjusting the parameter l. See Figure A.12 for examples.

Challenge 15 (normal forms of unimodal singularities). *Visualize the real zero set of*

$$J^{\pm}_{10+k} = x^3 \pm x^2y^2 + ay^6 + k \tag{A.27}$$

$$X^{\pm\pm}_{9+k} = \pm x^4 + x^2y^2 \pm ay^{4+k} \tag{A.28}$$

$$Y^{\pm\pm}_{r,s} = \pm x^2y^2 \pm x^r + ay^s \tag{A.29}$$

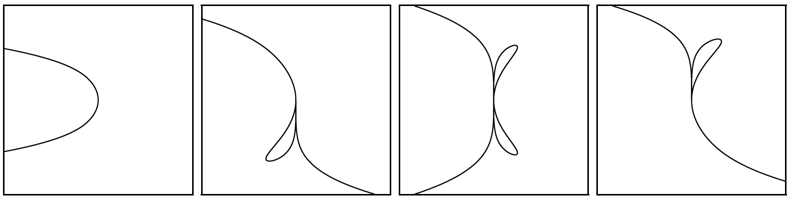
$$\tilde{Y}^{\pm\pm}_{r} = \pm(x^2 \pm y^2)^2 + ax^r \tag{A.30}$$

for $k, r, s \in \mathbb{N}$, $r, s > 4$, $a \in \{\frac{5}{7}, \frac{9}{7}, \frac{13}{7}\}$.

The singularities of type A, D and E are sometimes referred to as *simple singularities*. They have appeared in many of the curves presented in previous challenges. There is also a classification of non-simple singularities. The polynomials given in Challenge 15 are the normal forms of the so-called *unimodal singularities*. See Figures A.13 to A.16 for renderings of all possible shapes.

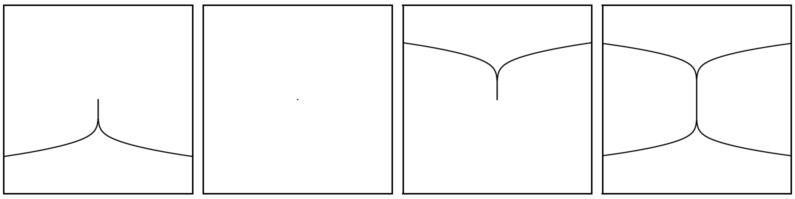
A.6. Discriminants

Discriminants are an essential tool to study the occurrence of multiple roots of polynomials or polynomial systems. In [Lab10b], only one example is given.

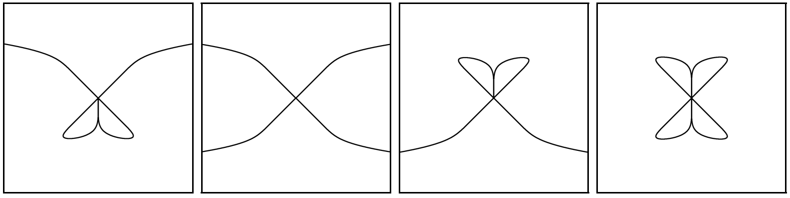


(a) J^{+}_{10+4} (b) J^{+}_{10+5} (c) J^{-}_{10+4} (d) J^{-}_{10+5}

Figure A.13.: Rasterizations of some of the $J^{\pm}_{10+k}$ curves of Challenge 15 for $a = \frac{9}{7}$.



(a) X^{++}_{9+11} (b) X^{++}_{9+12} (c) X^{+-}_{9+11} (d) X^{+-}_{9+12}



(e) X^{-+}_{9+11} (f) X^{-+}_{9+12} (g) X^{--}_{9+11} (h) X^{--}_{9+12}

Figure A.14.: Rasterizations of some of the $X^{\pm\pm}_{9+k}$ curves of Challenge 15 for $a = \frac{9}{7}$.

Challenge 16 (discriminants). *Visualize the real zero set of the discriminant* $D \in \mathbb{Q}[a, b]$ *of*

$$x^6 + ay^3 - y = 0 \tag{A.31}$$

$$y^6 + bx^3 - x = 0. \tag{A.32}$$

In [Dic+07, Example 2.9 and Proof of Theorem 1.1] this discriminant is computed using

$$h_1 = x^6 + ay^3 - y, \tag{A.33}$$

$$h_2 = y^6 + bx^3 - x, \tag{A.34}$$

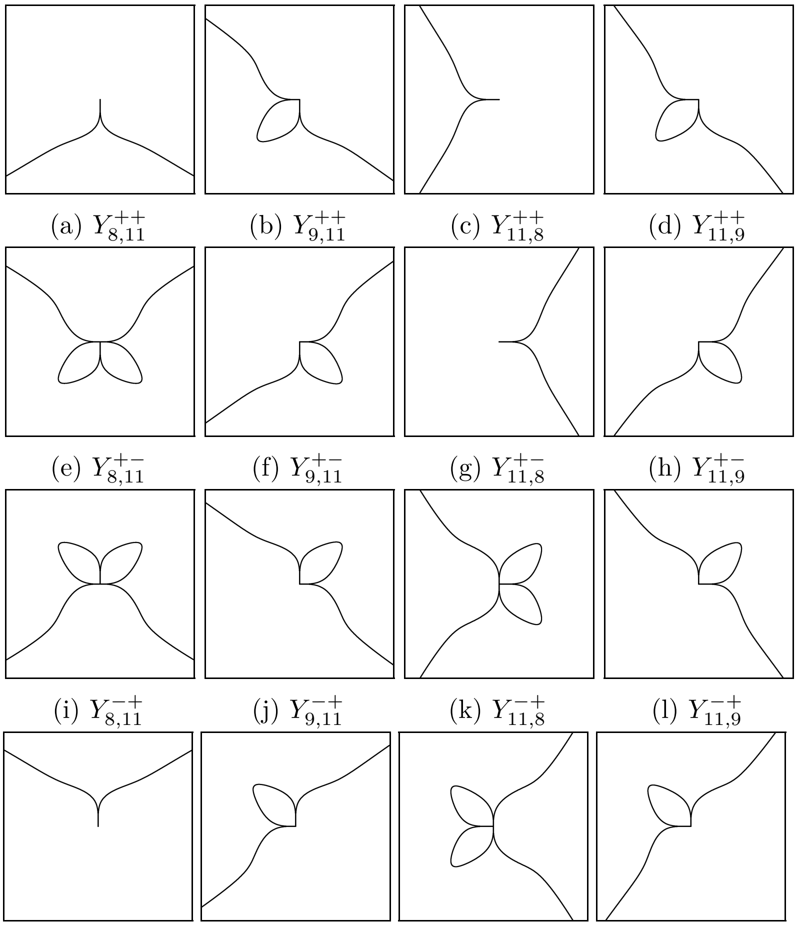
$$p = \mathrm{Res}_y(h_1, \det J_{h_1,h_2}(a, b)) \tag{A.35}$$

$$q = \mathrm{Res}_y(h_2, \det J_{h_1,h_2}(a, b)) \tag{A.36}$$

$$\tilde{R} = \mathrm{Res}_x(p, q), \tag{A.37}$$

where $J_{h_1,h_2}(a, b)$ is the Jacobi matrix of h_1 and h_2 with respect to a and b. The discriminant D is a factor of the polynomial $\tilde{R}$. Due to the degree bound $\deg D \leq 236$, it is easy to extract D from a factorization of $\tilde{R}$ since all but one factor have a higher degree.

The main challenge in visualizing D is its high degree, which is 90. Furthermore, D has several singularities close to the origin. Using the algorithm of [EKW07], the isolation of the critical points of D did not succeed after 8 days. Therefore, no image is provided for this challenge. See [Dic+07] for a visualization.



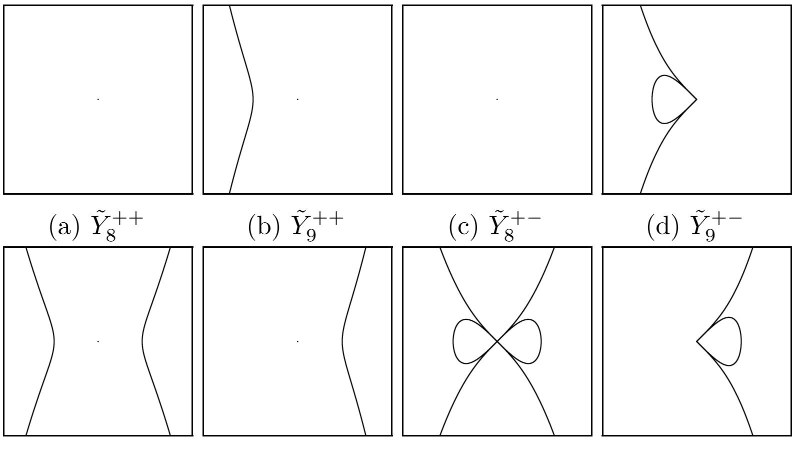
(a) $Y^{++}_{8,11}$ (b) $Y^{++}_{9,11}$ (c) $Y^{++}_{11,8}$ (d) $Y^{++}_{11,9}$

(e) $Y^{+-}_{8,11}$ (f) $Y^{+-}_{9,11}$ (g) $Y^{+-}_{11,8}$ (h) $Y^{+-}_{11,9}$

(i) $Y^{-+}_{8,11}$ (j) $Y^{-+}_{9,11}$ (k) $Y^{-+}_{11,8}$ (l) $Y^{-+}_{11,9}$

(m) $Y^{--}_{8,11}$ (n) $Y^{--}_{9,11}$ (o) $Y^{--}_{11,8}$ (p) $Y^{--}_{11,9}$

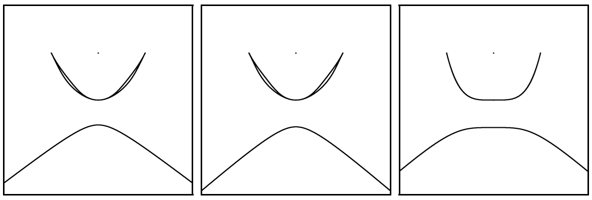
Figure A.15.: Rasterizations of some of the $Y^{\pm\pm}_{r,s}$ curves of Challenge 15 for $a = \frac{9}{7}$.



(a) $\tilde{Y}^{++}_{8}$ (b) $\tilde{Y}^{++}_{9}$ (c) $\tilde{Y}^{+-}_{8}$ (d) $\tilde{Y}^{+-}_{9}$

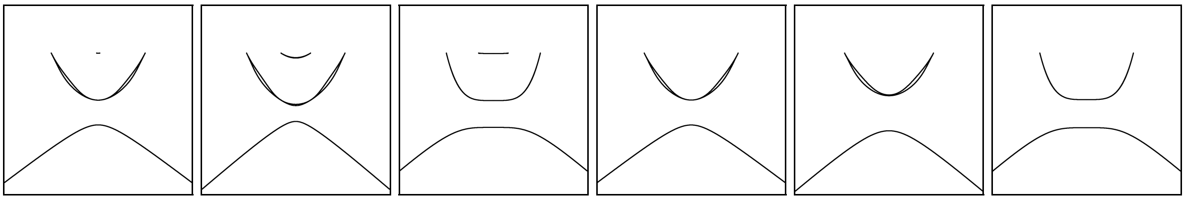
(e) $\tilde{Y}^{-+}_{8}$ (f) $\tilde{Y}^{-+}_{9}$ (g) $\tilde{Y}^{--}_{8}$ (h) $\tilde{Y}^{--}_{9}$

Figure A.16.: Rasterizations of some of the $\tilde{Y}^{\pm\pm}_{r}$ of Challenge 15 curves for $a = \frac{9}{7}$.



(a) $SA_{2,2}$ (b) $SA_{2,4}$ (c) $SA_{4,2}$

Figure A.17.: Rasterizations of some of the $SA_{k,l}$ curves of Challenge 17.



(a) $SA_{2,2,+10^{-3}}$ (b) $SA_{2,4,+10^{-1}}$ (c) $SA_{4,2,+10^{-2}}$ (d) $SA_{2,2,-10^{-3}}$ (e) $SA_{2,4,-10^{-1}}$ (f) $SA_{4,2,-10^{-2}}$

Figure A.18.: Rasterizations of some of the $SA_{k,l,\pm\varepsilon}$ curves of Challenge 18.

A.7. Curves with several difficulties

Some of the previous curves already combined several aspects that are challenging for visualization algorithms. This section expands the list of curves where several difficulties show up at once.

Challenge 17 (solitary point with high zero-tangency and halfbranches with high tangency). *Visualize the real zero set of*

$$SA_{k,l}(x,y) = (y-1-x^k)^l(y-k)^l + (y-1)^{kl+1}y^{kl} \tag{A.38}$$

for $k,l \in 2\mathbb{N}$.

The $SA_{k,l}(x,y)$ curves have a solitary point in $(0,1)$ and some other singularities with high tangencies. See Figure A.17 for examples.

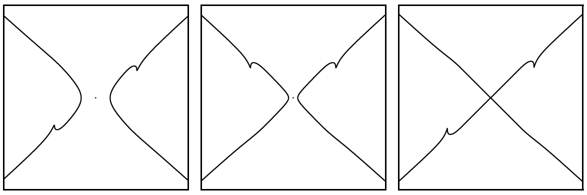
Challenge 18 (halfbranches with high tangency and many critical points and singularities at non-real coordinates). *Visualize the real zero set of*

$$SA_{k,l,\pm\varepsilon}(x,y) = (y-1-x^k \pm \varepsilon)^l(y-k\pm\varepsilon)^l + (y-1)^{kl+1}y^{kl} \tag{A.39}$$

for $k,l \in 2\mathbb{N}$ *and* $\varepsilon = 10^{-i}$, $i \in \mathbb{N}$.

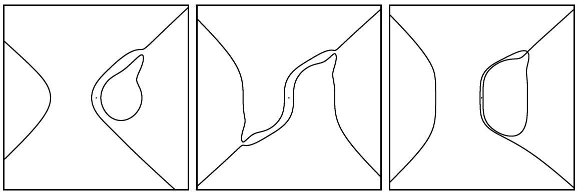
The introduction of the parameter ε deforms the solitary point of $SA_{k,l}$ into a small oval which is real for the $+\varepsilon$ case and non-real for the $-\varepsilon$ case. Furthermore, these curves have many critical points and singularities at imaginary coordinates.

Note that the visualizations of $SA_{2,4,\pm 0.001}$ and $SA_{4,2,\pm 0.001}$ shown in [Lab10b] are wrong. The reason for the wrong visualization might be that Labs mainly used the program SURF [End+10] to draw the curves, which completely misses the solitary points, their deformations and the halfbranches with high tangency. The algorithms studied in the present work yield correct results (see Figure A.18).



(a) $SCA_{5,2,10^{-1}}$ (b) $SCA_{6,2,10^{-2}}$ (c) $SCA_{7,2,10^{-7}}$

Figure A.19.: Rasterizations of some of the $SCA_{k,l,\varepsilon}$ curves of Challenge 19. In (c) the parameter ε is chosen so small that the one dimensional real components seem to join with the solitary point. Rasterizing the curve using an appropriate resolution would separate them again.



(a) $SAA_{2,2,10^{-2}}$ (b) $SAA_{3,2,10^{-2}}$ (c) $SAA_{4,2,10^{-3}}$

Figure A.20.: Rasterizations of some of the $SAA_{k,l,\varepsilon}$ curves of Challenge 20.

Challenge 19 (one dimensional real component close to a solitary point 1). *Visualize the real zero set of*

$$SCA_{k,l,\varepsilon}(x,y) = ((y-x^k)^l + y^{kl})(y^2 - x^2 + \varepsilon) + y^{kl+2} \tag{A.40}$$

for $l = 2$, $k \in \mathbb{N}$, $k \geq 2$ *and* $\varepsilon = 10^{-i}$, $i \in \mathbb{N}$.

The distance of the one-dimensional real component to the solitary point can be controlled using the ε parameter. See Figure A.19 for examples.

Challenge 20 (one dimensional real component close to a solitary point 2). *Visualize the real zero set of*

$$SAA_{k,l,\varepsilon}(x,y) = ((y-x^k)^l + y^{kl})((x-y^k)^l - \varepsilon) + (xy)^{kl} \tag{A.41}$$

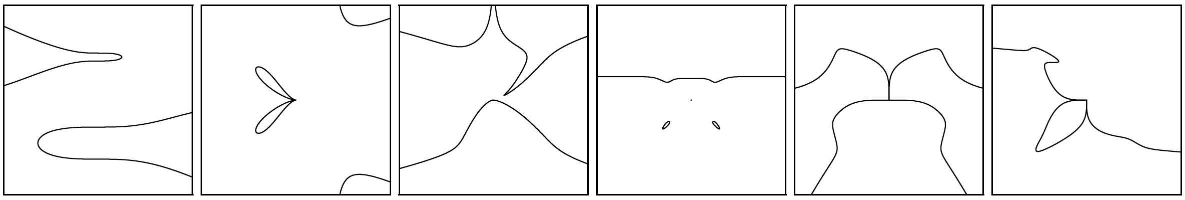
for $l = 2$, $k \in \mathbb{N}$, $k \geq 2$ *and* $\varepsilon = 10^{-i}$, $i \in \mathbb{N}$.

See Figure A.5 for examples of the $SAA_{k,l,\varepsilon}$ curves.

A.8. Random polynomials

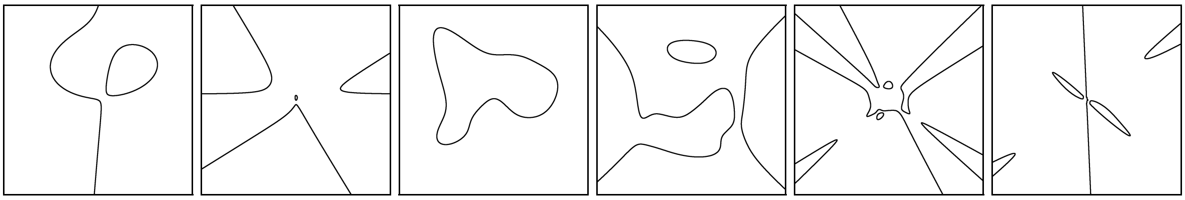
The following two challenges are based on random polynomials. They are not included in [Lab10b] but random polynomials are often used in benchmarks. The polynomials in Challenge 21 are sparse ones while the polynomials in Challenge 22 have dense coefficient arrays. In both cases, five polynomials have been computed and tested for each total degree. The function randpoly(d,t) generates a bivariate polynomial $P \in \mathbb{Z}[x,y]$ of degree at most d and at most t terms in the following way:

- Monomials are chosen uniformly random from the set of all possible monomials of degree up to d (inclusive). This means that it is more likely that a monomial of degree d appears than a monomial of degree $d-1$ because the former class is bigger.



(a) $SpRa_{4,3}$ (b) $SpRa_{7,1}$ (c) $SpRa_{16,2}$ (d) $SpRa_{17,5}$ (e) $SpRa_{23,1}$ (f) $SpRa_{27,3}$

Figure A.21.: Rasterizations of some of the $SpRa_{d,n}$ curves of Challenge 21.



(a) $DeRa_{3,2}$ (b) $DeRa_{5,5}$ (c) $DeRa_{8,1}$ (d) $DeRa_{16,1}$ (e) $DeRa_{19,1}$ (f) $DeRa_{19,3}$

Figure A.22.: Rasterizations of some of the $DeRa_{d,n}$ curves of Challenge 22.

- Exactly t distinct monomials are chosen this way and each one gets a random coefficient $i \in \mathbb{Z}$ assigned with probability

$$\Pr(i) = \begin{cases} 1/5 & \text{for } i = 0, \\ 2/(5|n|(|n|+1)) & \text{otherwise.} \end{cases} \tag{A.42}$$

The function randpoly(d, t) is implemented in the computer algebra system SAGE 5 as `PolynomialRing(ZZ,['x','y']).random_element(d,t)` (see the manual [SAGE11]) . This function has been called repeatedly until $\deg(\text{randpoly}(d, t)) = d$.

Challenge 21 (sparse random polynomials). *Visualize the real zero set of*

$$SpRa_{d,n}(x, y) = \text{randpoly}(d, 5) \tag{A.43}$$

for $d \in \mathbb{N}$, $d \geq 3$ and $n \in \{1, \dots, 5\}$ so that $SpRa_{d,i} \neq SpRa_{d,j}$ for $i \neq j$.

Visualization of some of the $SpRa_{d,n}$ curves are shown in Figure A.21.

Challenge 22 (dense random polynomials). *Visualize the real zero set of*

$$DeRa_{d,n}(x, y) = \text{randpoly}\left(d, \frac{(d+2)(d+1)}{2}\right) \tag{A.44}$$

for $d \in \mathbb{N}$, $d \geq 3$ and $n \in \{1, \dots, 5\}$ so that $DeRa_{d,i} \neq SpRa_{d,j}$ for $i \neq j$.

See Figure A.22 for visualization of some of the $DeRa_{d,n}$ curves.

Appendix B.

Absolute running times of plane curve renderings

On the next pages, we provide the plots of the absolute time needed to rasterize the challenging curves summarized in Appendix A. They have been used as a basis for comparing the three different algorithms in Section 3.9 from a practical point of view. The timings have been determined using a PC running UBUNTU LINUX 12.04 on a 2.4 GHz Intel Core 2 Q6600 Quad CPU having 4 GB of RAM.

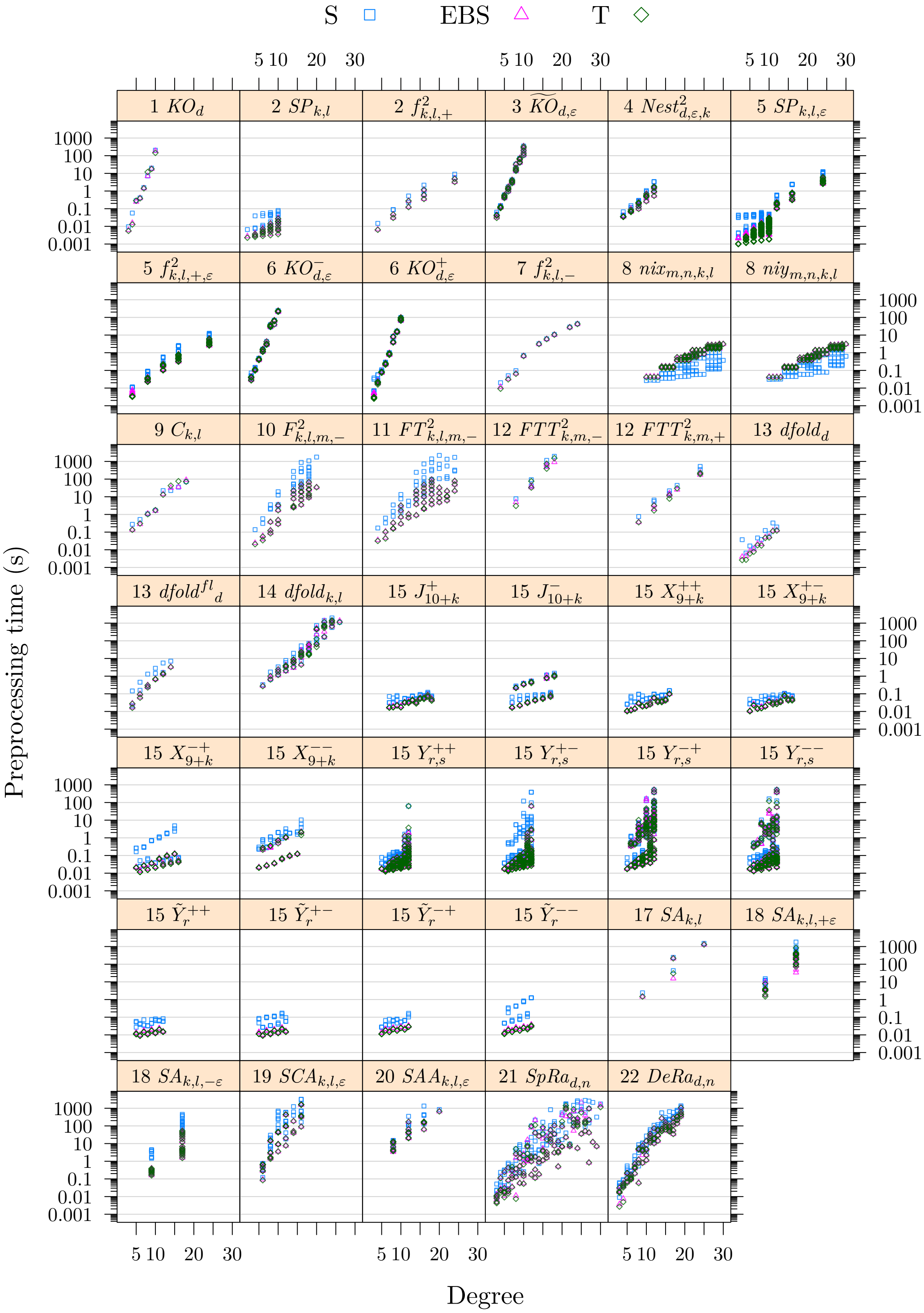


Figure B.1.: Scatter plot of the preprocessing time.

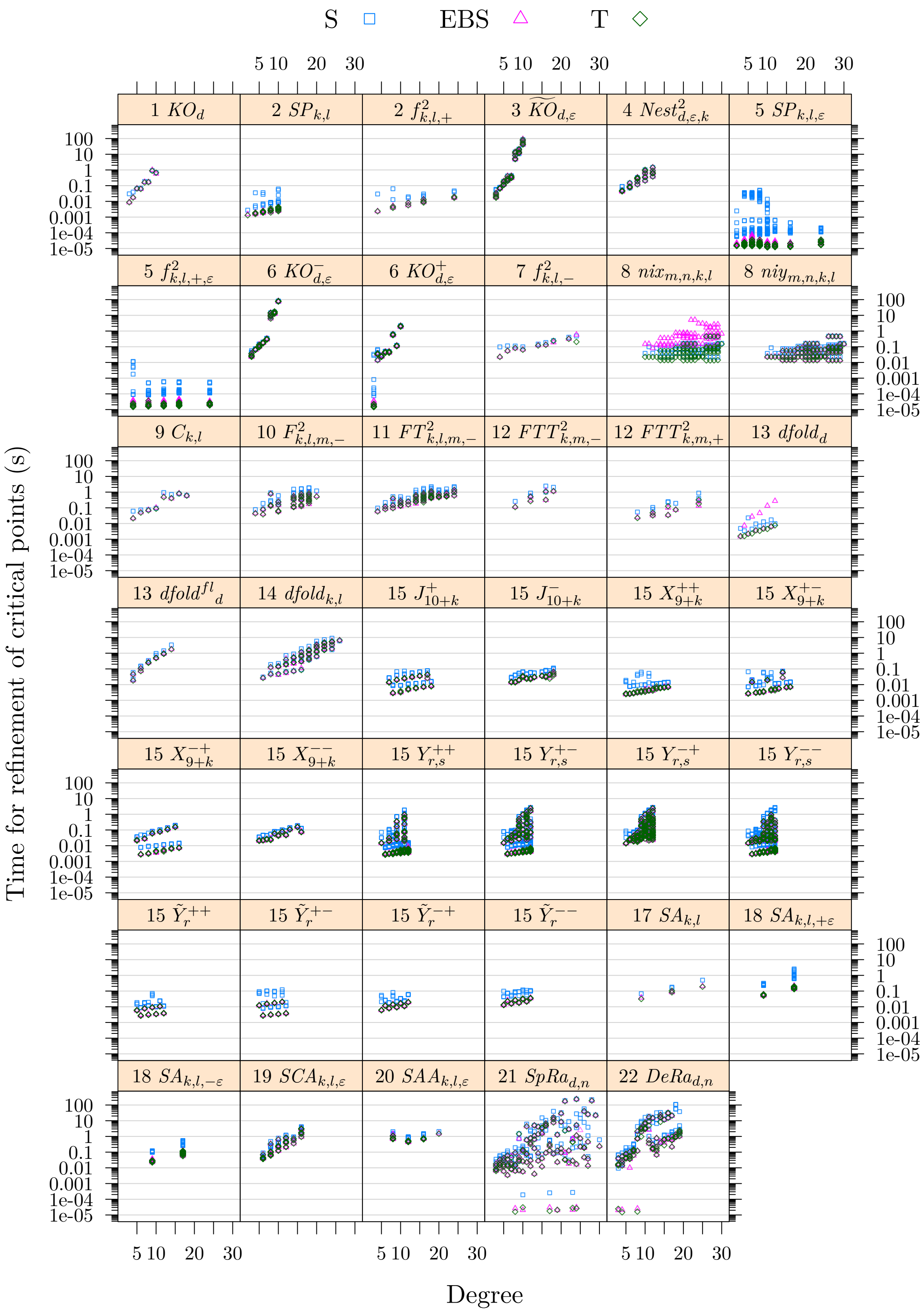


Figure B.2.: Scatter plot of the time for viewport adjustment and critical point refinement.

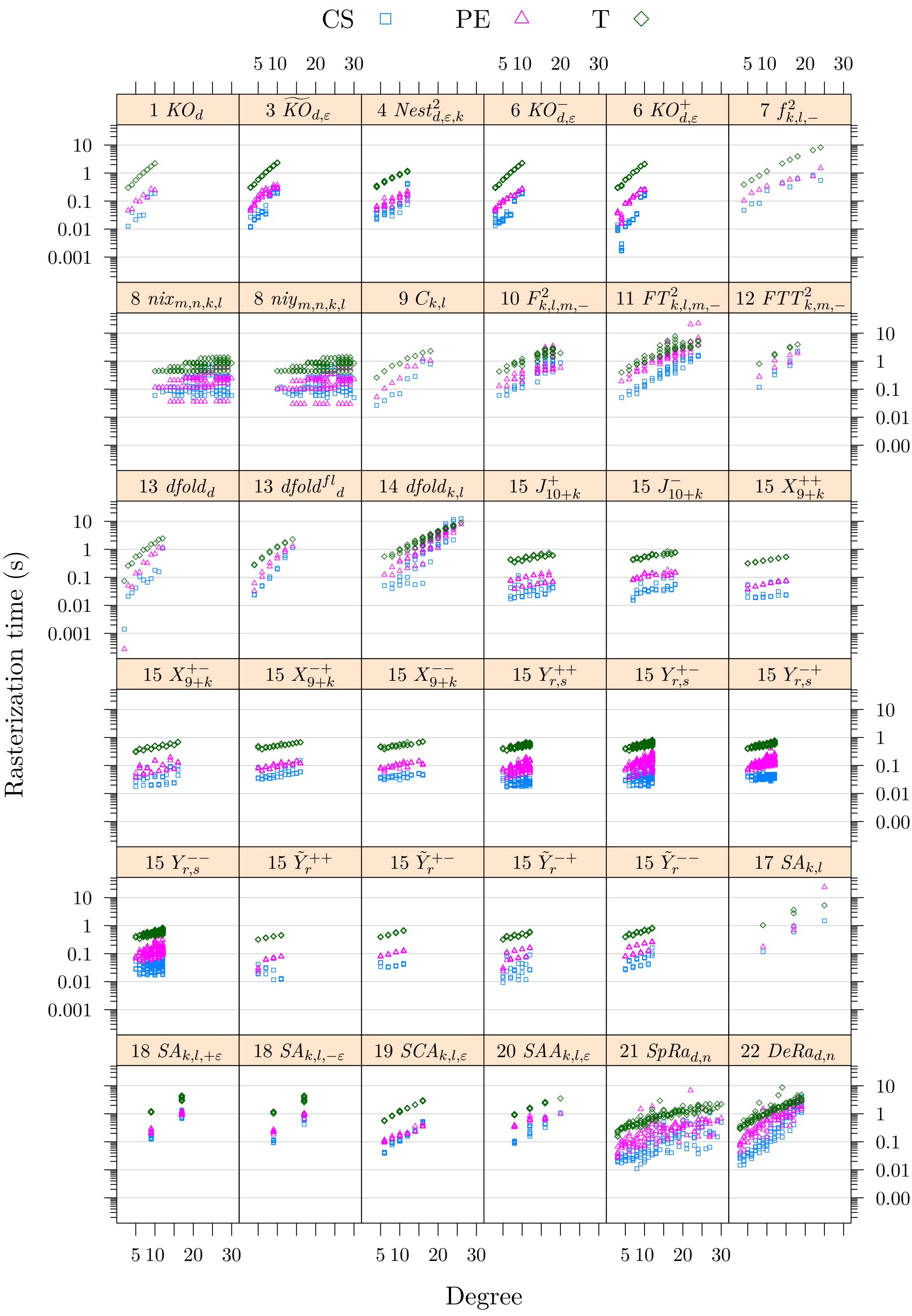


Figure B.3.: Scatter plot of the time for rasterizing on a 512 × 512 grid.

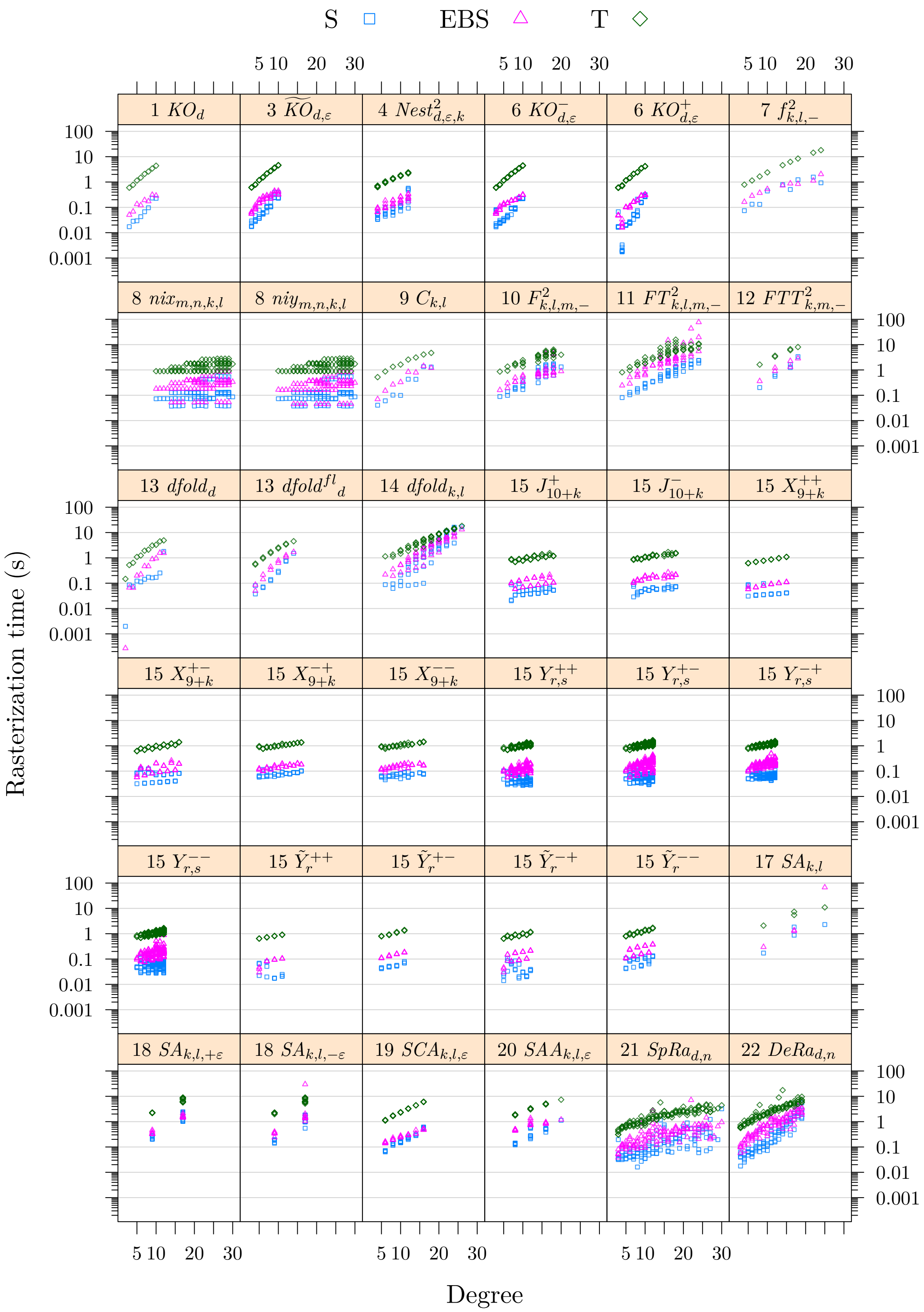


Figure B.4.: Scatter plot of the time for rasterizing on a 1024 × 1024 grid.

Appendix C.

Defining polynomials of example curves

C.1. The real algebraic plane curve "Bundle"

$B(x, y) = 0$ is a curve of degree 26 taken from [EBS09; Eme12], where is it denoted as "Bundle (curvature of Erdős lemniscate)". It has been used in Example 3.6.1 to illustrate the effect of the numerical filtering techniques on the real root counting of polynomials.

$$\begin{aligned}B(x,y) = {} & x^{26} + 29x^{24}y^2 + 254x^{22}y^4 + 1166x^{20}y^6 + 3355x^{18}y^8 \\ & + 6567x^{16}y^{10} + 9108x^{14}y^{12} + 9108x^{12}y^{14} + 6567x^{10}y^{16} \\ & + 3355x^8y^{18} + 1166x^6y^{20} + 254x^4y^{22} + 29x^2y^{24} + y^{26} \\ & + 14x^{20}y + 60x^{18}y^3 - 26x^{16}y^5 - 688x^{14}y^7 - 1988x^{12}y^9 \\ & - 2968x^{10}y^{11} - 2660x^8y^{13} - 1456x^6y^{15} - 458x^4y^{17} \\ & - 68x^2y^{19} - 2y^{21} + 9x^{16} - 136x^{14}y^2 - 116x^{12}y^4 + 232x^{10}y^6 \\ & + 70x^8y^8 - 120x^6y^{10} + 172x^4y^{12} + 152x^2y^{14} - 7y^{16} + 12x^{10}y \\ & - 340x^8y^3 + 952x^6y^5 - 616x^4y^7 + 124x^2y^9 - 4y^{11}\end{aligned} \tag{C.1}$$

C.2. Real algebraic space curves

Below, we provide the defining polynomials for the space curves used for the benchmarks in Section 4.10.

$$\begin{aligned}A_1 &= (x^2 + y^2 + z^2)^2 + 8xyz - 10(x^2 + y^2 + z^2) + 25 \\ B_1 &= \frac{\partial A_1}{\partial z}\end{aligned}$$

$$\begin{aligned}A_2 &= (x^2 + y^2 - 2)(x^2 + y^2 - 1)z^4 + (x^2 + y^2 - 2)xz^3 \\ &\quad + z^2 + (x^2 + y^2 - 1)z + (x^2 + y^2 - 1) \\ B_2 &= \frac{\partial A_2}{\partial z}\end{aligned}$$

$$\begin{aligned}A_3 &= x^2y^2 + y^2z^2 + z^2x^2 - xyz \\ B_3 &= \frac{\partial A_3}{\partial z}\end{aligned}$$

$$\begin{aligned} A_4 &= y^2 - z^2 - 2x + 1 \\ B_4 &= x^2 + z^2 - 1 \end{aligned}$$

$$\begin{aligned} A_5 &= -x^2 + z^2 \\ B_5 &= x^4 - x^2 + y^2 \end{aligned}$$

$$\begin{aligned} A_6 &= x^4 + y^4 + z^4 - 1 \\ B_6 &= x^3y + y^3z + xz^3 \end{aligned}$$

$$\begin{aligned} A_7 &= 10(2z^2 + x^2 + y^2 - 1)^3 - z^2y^3 - 10x^2y^3 \\ B_7 &= \frac{\partial A_7}{\partial z} \end{aligned}$$

$$\begin{aligned} A_8 = {} & 64x^2y^6 + 2304y^8 + 65536x^2y^4z^2 + 4096y^6z^2 - 131072x^3y^2z^3 - 131072xy^4z^3 \\ & + 65536x^4z^4 + 131072x^2y^2z^4 + 65536y^4z^4 - 64x^4y^3 + 128x^2y^5 + 10752y^7 \\ & + 768x^3y^3z - 20864xy^5z - 57344x^2y^3z^2 + 2560y^5z^2 + 65536x^3yz^3 \\ & + 172032xy^3z^3 - 131072x^2yz^4 - 163840y^3z^4 + 98x^6 + 236x^4y^2 - 3590x^2y^4 \\ & - 912y^6 + 272x^5z + 19392x^3y^2z - 77488xy^4z - 20512x^4z^2 - 75584x^2y^2z^2 \\ & - 48352y^4z^2 + 118272x^3z^3 + 397312xy^2z^3 - 291840x^2z^4 - 276480y^2z^4 \\ & + 98304xz^5 + 65536z^6 - 1464x^4y - 10608x^2y^3 - 40344y^5 - 1688x^3yz \\ & + 7816xy^3z + 172096x^2yz^2 - 55328y^3z^2 - 92032xyz^3 + 481280yz^4 + 3033x^4 \\ & + 12216x^2y^2 + 10803y^4 - 10338x^3z + 136902xy^2z + 106392x^2z^2 + 9900y^2z^2 \\ & - 274368xz^3 + 457728z^4 + 1044x^2y + 57096y^3 + 3960xyz - 32940yz^2 - 5724x^2 \\ & - 16686y^2 - 27162xz - 16902z^2 - 22032y + 9963 \\ B_8 = {} & x^3 - 3xy^2 + 4x^2z + 4y^2z - 9z \end{aligned}$$

$$\begin{aligned} A_9 = {} & -x^{16} - 8x^{14}y^2 - 28x^{12}y^4 - 56x^{10}y^6 - 70x^8y^8 - 56x^6y^{10} \\ & - 28x^4y^{12} - 8x^2y^{14} - y^{16} + 16x^6y^2z^8 - 32x^4y^4z^8 + 16x^2y^6z^8 \\ B_9 = {} & x^2 + y^2 + z^2 - 1 \end{aligned}$$

$$\begin{aligned} A_{10} = {} & 128x^8 + 128y^8 + 128z^8 - 256x^6 - 256y^6 - 256z^6 + 160x^4 + 160y^4 \\ & + 160z^4 - 32x^2 - 32y^2 - 32z^2 + 4 \\ B_{10} = {} & \frac{\partial A_{10}}{\partial z} \end{aligned}$$

Appendix D.

Notations and abbreviations

The following symbols, functions, operators and abbreviations appear frequently in this work. Many operators, like, e.g., lcoeff$(\cdot)$, cont$(\cdot)$ and pp$(\cdot)$, can be applied to polynomials from a multivariate polynomial ring $R[x_1, \ldots, x_k]$ by assuming x_k to be the outermost variable in the recursive view of $R[x_1, \ldots, x_k]$, i.e. $R[x_1, \ldots, x_k] = R[x_1, \ldots, x_{k-1}][x_k]$. The outermost variable may also be specified as a subscript, e.g. $\mathrm{lcoeff}_{x_i}(A)$ refers to $A \in R[x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_k][x_i]$. In the following, R denotes a commutative ring, D a unique factorization domain (UFD) and K a field.

$\mathbb{N}$	The set of natural numbers.
$\mathbb{Z}$	The set of integers.
$\mathbb{Q}$	The set of rational numbers.
$\mathbb{R}$	The set of real numbers.
$\mathbb{C}$	The set of complex numbers.
$\overline{K}$	Algebraic closure of K.
$\mathbb{P}(K), \mathbb{P}_K$	Projective closure of K.
$\mathbb{A}^n_K$	n-dimensional affine space over K.
$\mathbb{P}^n_K$	n-dimensional projective space over K.
$R[x_1, \ldots, x_k]$	Polynomial ring in the variables $x_1, \ldots, x_k$ over R.
$R[x_1, \ldots, x_k][x]$	Polynomial ring in x over $R[x_1, \ldots, x_k]$, i.e. the recursive view of $R[x_1, \ldots, x_k, x]$.
$I(V)$	Ideal of all polynomials vanishing on V.
$\langle A_1, \ldots, A_k \rangle$	Ideal generated by the polynomials $A_1, \ldots, A_k$.
$V_K(I_1, \ldots, I_k)$	Common vanishing set of the ideals $I_1, \ldots, I_k \subseteq R[x_1, \ldots, x_n]$ on K^n for $R \subseteq K$. K is omitted if it is clear from the context.
$\deg(A)$	Total degree of the polynomial $A \in R[x_1, \ldots, x_k]$.
$\deg_X(A)$	Total degree of the polynomial $A \in R[x_1, \ldots, x_k]$ w.r.t the variables given in $X \subseteq \{x_1, \ldots, x_k\}$.
$\mathrm{coeff}_i(A)$	Coefficient a_i of the polynomial $A = \sum_{i=0}^n a_i x^i \in R[x]$.
$\mathrm{lcoeff}(A)$	Leading coefficient a_n of the polynomial $A = \sum_{i=0}^n a_i x^i \in R[x]$.
$\mathrm{quot}(A, B)$	Quotient of the division of $A, B \in K[x]$.
$\mathrm{rem}(A, B)$	Remainder of the division of $A, B \in K[x]$.
$\mathrm{pquot}(A, B)$	Pseudoquotient of the division of $A, B \in R[x]$.
$\mathrm{prem}(A, B)$	Pseudoremainder of the division of $A, B \in R[x]$.
$\gcd(A_1, \ldots, A_k)$	Greatest common divisor (GCD) of $A_1, \ldots, A_k \in D$.

$\mathrm{cont}(A)$	Content of the polynomial $A = \sum_{i=0}^{n} a_i x^i \in D[x]$, i.e. the GCD of the coefficients $a_0, \ldots, a_n \in D$.
$\mathrm{pp}(A)$	Primitive part of $A \in D[x]$, i.e. $A / \mathrm{cont}(A)$.
$\mathrm{Syl}(A, B)$	Sylvester matrix of A and B.
$\mathrm{Res}(A, B)$	Resultant of $A, B \in R[x]$.
$\mathrm{SRes}_j(A, B)$	Signed subresultant j of $A, B \in R[x]$.
$\mathrm{SResU}_j(A, B)$	Cofactor of $\mathrm{SRes}_j(A, B)$ w.r.t. to A.
$\mathrm{SResV}_j(A, B)$	Cofactor of $\mathrm{SRes}_j(A, B)$ w.r.t. to B.
$\mathrm{SResQ}_j(A, B)$	Signed subresultant quotient j of $A, B \in R[X]$.
$\mathrm{sres}_j(A, B)$	Principal signed subresultant coefficient j of $A, B \in R[x]$, i.e. $\mathrm{sres}_j(A, B) = \mathrm{coeff}_j(\mathrm{SRes}_j(A, B))$.
$\overline{\mathrm{sres}}_j(A, B)$	Leading signed subresultant coefficient j of $A, B \in R[x]$, i.e. $\overline{\mathrm{sres}}_j(A, B) = \mathrm{lcoeff}(\mathrm{SRes}_j(A, B))$.
$\#\mathrm{rr}(A, I)$	Number of distinct real roots of $A \in \mathbb{R}[x]$ in $I \subseteq \mathbb{R}$.
$\#\mathrm{rrm}(A, I)$	Number of real roots of $A \in \mathbb{R}[x]$ in $I \subseteq \mathbb{R}$ (counted with multiplicity).
$\mathrm{Var}(\mathcal{S})$	Number of sign variations in the sequence $\mathcal{S}$.
$\mathrm{MVar}(\mathcal{S}, a)$	Modified number of sign variations of the sequence $\mathcal{S}$ at $a \in \mathbb{R}$.
$\mathrm{rank}(M)$	Rank of the matrix M.
$\mathrm{bit}(a)$	Bitsize of the integral or rational number a.
$\mathcal{M}(\tau)$	Cost of multiplying two integral numbers of bitsize τ.
$\mathcal{O}, \Omega, \Theta$	Landau symbols describing the limiting behavior of functions.
$\tilde{\mathcal{O}}$	Soft-$\mathcal{O}$, i.e. $f(n) \in \tilde{\mathcal{O}}(g(n))$ is the shorthand for $f(n) \in \mathcal{O}(g(n) \log^k g(n))$, $k \in \mathbb{N}$.
CPU	Central processing unit.
CRT	Chinese remainder theorem.
GCD	Greatest common divisor.
GPU	Graphics processing unit.
PQS	Polynomial quotient sequence.
PRS	Polynomial remainder sequence.
UFD	Unique factorization domain.

Bibliography

[Akr93] A. G. Akritas. "Sylvesters forgotten form of the resultant". In: *Fibonacci Quart* 31 (1993), pp. 325–332.

[AM07] L. Alberti and B. Mourrain. "Visualisation of Implicit Algebraic Curves". In: *Pacific Conference on Computer Graphics and Applications* (2007), pp. 303–312. ISSN: 15504085. DOI: `10.1109/PG.2007.32`.

[AR05] J. G. Alcázar and J. Rafael Sendra. "Computation of the topology of real algebraic space curves". In: *Journal of Symbolic Computation* 39.6 (2005), pp. 719–744. ISSN: 07477171. DOI: `10.1016/j.jsc.2005.01.006`.

[Arn83] D. S. Arnon. "Topologically reliable display of algebraic curves". In: *ACM SIGGRAPH Computer Graphics* 17 (3 1983), pp. 219–227. ISSN: 00978930. DOI: `10.1145/964967.801152`.

[Baj+88] C. L. Bajaj, C. M. Hoffmann, R. E. Lynch, and J. E. H. Hopcroft. "Tracing surface intersections". In: *Computer Aided Geometric Design* 5.4 (1988), pp. 285–307. ISSN: 01678396. DOI: `10.1016/0167-8396(88)90010-6`.

[BES11] E. Berberich, P. Emeliyanenko, and M. Sagraloff. "An Elimination Method for Solving Bivariate Polynomial Systems: Eliminating the Usual Drawbacks". In: *ALENEX'11*. 2011, pp. 35–47.

[BH98] W. Bruns and J. Herzog. *Cohen-Macaulay rings. Rev. ed.* Cambridge Studies in Advanced Mathematics. 39. Cambridge: Cambridge University Press. xiv, 1998.

[BPR03] S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry (Algorithms and Computation in Mathematics)*. Secaucus, NJ, USA: Springer-Verlag, 2003. ISBN: 3540009736.

[Bre65] J. Bresenham. "Algorithm for Computer Control of a Digital Plotter". In: *IBM Systems Journal* 4.1 (1965), pp. 25–30. DOI: `10.1147/sj.41.0025`.

[Bro08] M. Brodmann. "Blowing-up!" In: *The KIAS Newsletter (English version)* 1 (2008), pp. 40–43. URL: `http://www.kias.re.kr/file/NewsletterEnglish.pdf` (visited on 03/22/2013).

[Bro95] M. Brodmann. "Computer-pictures of blowing-ups. (Computerbilder von Aufblasungen.)" German. In: *Elemente der Mathematik* 50.4 (1995), pp. 149–163.

[Bub+95] T. Bubeck, M. Hiller, W. Küchlin, and W. Rosenstiel. "Distributed Symbolic Computation with DTS". In: *Proceedings of Parallel Algorithms for Irregularly Structured Problems, LNCS 980*. Springer-Verlag, 1995, pp. 231–248.

[Bur+08] M. Burr, S. W. Choi, B. Galehouse, and C. K. Yap. "Complete subdivision algorithms, II: isotopic meshing of singular algebraic curves". In: *Proceedings of the 21st International Symposium on Symbolic and Algebraic Computation.* (Linz/Hagenberg, Austria). ISSAC '08. New York, NY, USA: ACM, 2008, pp. 87–94. ISBN: 9781595939043. DOI: `10.1145/1390768.1390783`.

[BZ10] R. Brent and P. Zimmermann. *Modern Computer Arithmetic.* New York, NY, USA: Cambridge University Press, 2010. ISBN: 9780521194693.

[CA76] G. E. Collins and A. G. Akritas. "Polynomial real root isolation using Descarte's rule of signs". In: *Proceedings of the 3rd ACM Symposium on Symbolic and Algebraic computation.* SYMSAC '76. New York, NY, USA: ACM, 1976, pp. 272–275. DOI: `10.1145/800205.806346`.

[CCPG12] *NVIDIA CUDA C Programming Guide.* NVIDIA Corporation. 2012. URL: `http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html` (visited on 03/22/2013).

[CGAL12] CGAL, *Computational Geometry Algorithms Library.* 2012. URL: `http://www.cgal.org` (visited on 03/22/2013).

[CGL09] J.-S. Cheng, X.-S. Gao, and J. Li. "Root isolation for bivariate polynomial systems with local generic position method". In: *Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation.* (Seoul, Republic of Korea). ISSAC '09. New York, NY, USA: ACM, 2009, pp. 103–110. ISBN: 9781605586090. DOI: `10.1145/1576702.1576719`.

[Cha88] R. E. Chandler. "A Tracking Algorithm for Implicitly Defined Curves". In: *IEEE Computer Graphics and Applications* 8 (2 1988), pp. 83–89. ISSN: 02721716. DOI: `10.1109/38.506`.

[Che01] C. C.-A. Cheng. "A chain rule for subresultants". In: *Journal of Pure and Applied Algebra* 157.1 (2001), pp. 33–39. ISSN: 0022-4049. DOI: `10.1016/S0022-4049(00)00018-9`.

[CLO07] D. Cox, J. Little, and D. O'Shea. *Ideals, varieties, and algorithms. An introduction to computational algebraic geometry and commutative algebra.* 3rd ed. Springer-Verlag, 2007. ISBN: 9780387356501.

[Col71] G. E. Collins. "The calculation of multivariate polynomial resultants". In: *SYMSAC '71: Proceedings of the 2nd ACM Symposium on Symbolic and Algebraic Manipulation.* (Los Angeles, California, United States). New York, NY, USA: ACM, 1971, pp. 212–222. DOI: `10.1145/800204.806289`.

[Coo66] S. A. Cook. "On the minimum computation time of functions". PhD thesis. Cambrige, Massachusetts, USA: Hardvard University, 1966.

[DET09] D. I. Diochnos, I. Z. Emiris, and E. P. Tsigaridas. "On the asymptotic and practical complexity of solving bivariate systems over the reals". In: *Journal of Symbolic Computation* 44.7 (2009). International Symposium on Symbolic and Algebraic Computation, pp. 818–835. ISSN: 07477171. DOI: `10.1016/j.jsc.2008.04.009`.

[Dic+07] A. Dickenstein, J. Rojas, K. Rusek, and J. Shih. "Extremal real algebraic geometry and $\mathcal{A}$-discriminants." In: *Moscow Mathematical Journal* 7.3 (2007), pp. 425–452.

[DMR08] D. N. Diatta, B. Mourrain, and O. Ruatta. "On the computation of the topology of a non-reduced implicit space curve". In: *Proceedings of the 21st International Symposium on Symbolic and Algebraic Computation.* (Linz/Hagenberg, Austria). ISSAC '08. New York, NY, USA: ACM, 2008, pp. 47–54. ISBN: 9781595939043. DOI: `10.1145/1390768.1390778`.

[Dod66] C. L. Dodgson. “Condensation of Determinants, Being a New and Brief Method for Computing their Arithmetical Values”. In: *Proceedings of the Royal Society of London* 15 (1866), pp. 150–155. ISSN: 03701662. URL: `http://www.jstor.org/stable/112607` (visited on 03/22/2013).

[Dok+05] T. Dokken, B. Jüttler, H. Shou, R. Martin, G. Wang, A. Bowyer, and I. Voiculescu. “A Recursive Taylor Method for Algebraic Curves and Surfaces”. In: *Computational Methods for Algebraic Spline Surfaces.* Springer-Verlag, 2005, pp. 135–154. ISBN: 9783540271574. DOI: `10.1007/3-540-27157-0_10`.

[EBS09] P. Emeliyanenko, E. Berberich, and M. Sagraloff. “Visualizing Arcs of Implicit Algebraic Curves, Exactly and Fast”. In: *Advances in Visual Computing.* Vol. 5875. Lecture Notes in Computer Science. Springer-Verlag, 2009, pp. 608–619. DOI: `10.1007/978-3-642-10331-5_57`.

[Eig08] A. Eigenwillig. “Real root isolation for exact and approximate polynomials using Descartes’ rule of signs”. PhD thesis. Saarbrücken, Germany: Unveristät des Saarlandes, 2008. URL: `http://scidok.sulb.uni-saarland.de/volltexte/2010/3244` (visited on 03/22/2013).

[EK08] A. Eigenwillig and M. Kerber. “Exact and efficient 2D-arrangements of arbitrary algebraic curves”. In: *Proceedings of the 19th annual ACM-SIAM symposium on Discrete algorithms.* (San Francisco, California). SODA ’08. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2008, pp. 122–131. URL: `http://dl.acm.org/citation.cfm?id=1347082.1347096` (visited on 03/22/2013).

[EKW07] A. Eigenwillig, M. Kerber, and N. Wolpert. “Fast and exact geometric analysis of real algebraic plane curves”. In: *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation.* (Waterloo, Ontario, Canada). ISSAC ’07. New York, NY, USA: ACM, 2007, pp. 151–158. ISBN: 9781595937438. DOI: `10.1145/1277548.1277570`.

[El 08] M. El Kahoui. “Topology of real algebraic space curves”. In: *Journal of Symbolic Computation* 43.4 (2008), pp. 235–258. ISSN: 07477171. DOI: `10.1016/j.jsc.2007.10.008`.

[Eme10a] P. Emeliyanenko. “Modular resultant algorithm for graphics processors”. In: *Proceedings of the 10th international conference on Algorithms and Architectures for Parallel Processing - Volume Part I.* (Busan, Korea). ICA3PP’10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 427–440. ISBN: 9783642131189. DOI: `10.1007/978-3-642-13119-6_37`.

[Eme10b] P. Emeliyanenko. “A complete modular resultant algorithm targeted for realization on graphics hardware”. In: *Proceedings of the 4th International Workshop on Parallel and Symbolic Computation.* (Grenoble, France). PASCO ’10. New York, NY, USA: ACM, 2010, pp. 35–43. ISBN: 9781450300674. DOI: `10.1145/1837210.1837219`.

[Eme12] P. Emeliyanenko. *EXACUS Webdemo.* 2012. URL: `http://exacus.mpi-inf.mpg.de/cgi-bin/xalci.cgi` (visited on 03/22/2013).

[EMT08] I. Emiris, B. Mourrain, and E. Tsigaridas. “Real Algebraic Numbers: Complexity Analysis and Experimentation”. In: *Reliable Implementations of Real Number Algorithms: Theory and Practice.* Ed. by P. Hertling, C. Hoffmann, W. Luther, and N. Revol. Vol. 5045. LNCS. Springer-Verlag, 2008, pp. 57–82.

[End+10] S. Endraß, H. Huelf, R. Oertel, K. Schneider, R. Schmitt, J. Beigel, E. Selder, K. Gharslyan, S. Haasmann, J. Heil, I. Krmcar, B. Li, M. Scherer, S. Sperner, and S. Yaykan. *surf – visualization of real algebraic geometry.* 2010. URL: `http://surf.sourceforge.net` (visited on 03/22/2013).

[ES12] P. Emeliyanenko and M. Sagraloff. “On the complexity of solving a bivariate polynomial system”. In: *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation.* (Grenoble, France). ISSAC ’12. New York, NY, USA: ACM, 2012, pp. 154–161. ISBN: 9781450312691. DOI: `10.1145/2442829.2442854`.

[ESY06] A. Eigenwillig, V. Sharma, and C. K. Yap. “Almost tight recursion tree bounds for the Descartes method”. In: *Proceedings of the 2006 International Symposium on Symbolic and Algebraic Computation.* (Genoa, Italy). ISSAC ’06. New York, NY, USA: ACM, 2006, pp. 71–78. ISBN: 1595932763. DOI: `10.1145/1145768.1145786`.

[Fis94] G. Fischer. *Ebene Algebraische Kurven.* Vieweg Studium: Aufbaukurs Mathematik. Braunschweig, Wiesbaden: Vieweg, 1994. ISBN: 3528072679.

[FS96] L. H. D. Figueiredo and J. Stolfi. “Adaptive Enumeration of Implicit Surfaces with Affine Arithmetic”. In: *Computer Graphics Forum* 15 (5 1996), pp. 287–296.

[Fuj16] M. Fujiwara. “Über die obere Schranke des absoluten Betrages der Wurzeln einer algebraischen Gleichung”. German. In: *Tohoku Mathematical Journal* 10 (1916), pp. 167–171.

[Für07] M. Fürer. “Faster integer multiplication”. In: *STOC.* 2007, pp. 57–66. DOI: `10.1145/1250790.1250800`.

[FYK97] C. Falai, F. Yuyu, and J. Kozak. “Tracing a planar algebraic curve”. In: *Applied Mathematics - A Journal of Chinese Universities* 12 (1 1997), pp. 15–24. ISSN: 10051031. DOI: `10.1007/s11766-997-0002-2`.

[Gat+05] G. Gatellier, A. Labrouzy, B. Mourrain, and J. Técourt. “Computing the Topology of Three-Dimensional Algebraic Curves”. In: *Computational Methods for Algebraic Spline Surfaces.* Springer-Verlag, 2005, pp. 27–43. ISBN: 9783540232742. DOI: `10.1007/3-540-27157-0_3`.

[GCL92] K. O. Geddes, S. R. Czapor, and G. Labahn. *Algorithms for computer algebra.* Dordrecht: Kluwer Academic Publishers Group, 1992.

[GG97] J. von zur Gathen and J. Gerhard. “Fast algorithms for Taylor shifts and certain difference equations”. In: *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation.* (Kihei, Maui, Hawaii, United States). ISSAC ’97. New York, NY, USA: ACM, 1997, pp. 40–47. ISBN: 0897918754. DOI: `10.1145/258726.258745`.

[GKP90] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1990. ISBN: 0201142368.

[GN02] L. Gonzalez-Vega and I. Necula. "Efficient topology determination of implicitly defined algebraic plane curves". In: *Computer Aided Geometric Design* 19 (9 2002), pp. 719–743. ISSN: 01678396. DOI: `10.1016/S0167-8396(02)00167-X`.

[Gon+90] L. González-Vega, H. Lombardi, T. Recio, and M.-F. Roy. "Spécialisation de la suite de Sturm et sous-résultants". French. In: *Informatique théorique et applications* 24 (1990), pp. 561–588.

[Gon+94] L. González-Vega, H. Lombardi, T. Recio, and M.-F. Roy. "Spécialisation de la suite de Sturm". French. In: *Informatique théorique et applications* 28.1 (1994), pp. 1–24.

[GP02] G.-M. Greuel and G. Pfister. *A singular introduction to commutative algebra.* Berlin, New York: Springer-Verlag, 2002. ISBN: 3540428976.

[GRL98] L. González-Vega, T. Recio, and H. Lombardi. "Sturm-Habicht sequence, determinants and real roots of univariate polynomials". In: *Quantifier Elimination and Cylindrical Algebraic Decomposition.* Ed. by B. F. Caviness and J. R. Johnson. Springer-Verlag, 1998, pp. 300–316.

[GW89] T. Garrity and J. Warren. "On computing the intersection of a pair of algebraic surfaces". In: *Computer Aided Geometric Design* 6.2 (1989), pp. 137–153. ISSN: 0167-8396. DOI: `10.1016/0167-8396(89)90017-4`.

[Hab48] W. Habicht. "Eine Verallgemeinerung des Sturmschen Wurzelzählverfahrens". German. In: *Commentarii Mathematici Helvetici* 21.1 (1948), pp. 99–116. ISSN: 00102571. DOI: `10.1007/BF02568028`.

[Har83] J. Harris. *Algebraic Geometry.* 3rd ed. Springer-Verlag, 1983.

[Hie11] P. Hiesinger. "Adaptives Raycasting von algebraischen Flächen". German. Diploma thesis. Halle (Saale), Germany: Martin Luther University Halle–Wittenberg, Institute of Computer Science, 2011.

[Hil01] J. Hill F. S. *Computer Graphics Using OpenGL.* 2nd ed. Prentice Hall, Inc., 2001. ISBN: 0023548568.

[Hir64] H. Hironaka. "Resolution of Singularities of an Algebraic Variety Over a Field of Characteristic Zero: I & II". In: *The Annals of Mathematics.* Second Series 79 (1964), pp. 109–203, 205–326. ISSN: 0003486X. URL: `http://www.jstor.org/stable/1970486` (visited on 03/22/2013).

[HL94] H. Hong and H. W. Loidl. "Parallel computation of modular multivariate polynomial resultants on a shared memory machine". In: *Parallel Processing: CONPAR 94 – VAPP VI.* Vol. 854. Lecture Notes in Computer Science. Springer-Verlag, 1994, pp. 325–336. ISBN: 9783540584308. DOI: `10.1007/3-540-58430-7_29`.

[Hob90] J. D. Hobby. "Rasterization of nonparametric curves". In: *ACM Transactions on Graphics* 9 (3 1990), pp. 262–277. ISSN: 07300301. DOI: `10.1145/78964.78966`.

[Hon97] H. Hong. "Subresultants Under Composition". In: *Journal of Symbolic Computation* 23.4 (1997), pp. 355–365. ISSN: 0747-7171. DOI: `10.1006/jsco.1996.0093`.

[JKR05] J. R. Johnson, W. Krandick, and A. D. Ruslanov. "Architecture-aware classical Taylor shift by 1". In: *Proceedings of the 2005 International Symposium on Symbolic and Algebraic Computation.* (Beijing, China). ISSAC '05. New York, NY, USA: ACM, 2005, pp. 200–207. ISBN: 1595930957. DOI: `10.1145/1073884.1073913`.

[Ker06] M. Kerber. "Analysis of real algebraic plane curves". Diploma thesis. Saarbrücken, Germany: Unveristät des Saarlandes, 2006. URL: `http://www.mpi-inf.mpg.de/~mkerber/mkerber_diplom.pdf` (visited on 03/22/2013).

[Kno+09] A. Knoll, Y. Hijazi, A. Kensler, M. Schott, C. Hansen, and H. Hagen. "Fast Ray Tracing of Arbitrary Implicit Surfaces with Interval and Affine Arithmetic". In: *Computer Graphics Forum* 28.1 (2009), pp. 26–40. ISSN: 14678659. DOI: `10.1111/j.1467-8659.2008.01189.x`.

[Knu97a] D. E. Knuth. *The Art of Computer Programming: Fundamental Algorithms*. 3rd ed. Vol. 1. Addison-Wesley Professional, 1997. ISBN: 0201896834.

[Knu97b] D. E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*. 3rd ed. Vol. 2. Addison-Wesley Professional, 1997. ISBN: 0201896842.

[KP02] S. G. Krantz and H. R. Parks. *The implicit function theorem. History, theory and applications*. Boston, USA: Birkhäuser, 2002. ISBN: 0817642854.

[Kul08] U. Kulisch. *Computer arithmetic and validity. Theory, implementation, and applications*. De Gruyter studies in mathematics. Berlin, Germany: Walter De Gruyter, 2008. ISBN: 9783110203189.

[Lab10a] O. Labs. *A List of Challenges for Real Algebraic Plane Curve Visualization Software*. 2010. URL: `http://www.oliverlabs.net/data/listChallengesVisAlgVar.txt` (visited on 03/22/2013).

[Lab10b] O. Labs. "A List of Challenges for Real Algebraic Plane Curve Visualization Software". In: *Nonlinear Computational Geometry*. Ed. by I. Z. Emiris, F. Sottile, and T. Theobald. Vol. 151. The IMA Volumes in Mathematics and its Applications. Springer-Verlag, 2010, pp. 137–164. ISBN: 9781441909985. DOI: `10.1007/978-1-4419-0999-2_6`.

[Lip10] S. Lipkowski. "Visualisierung algebraischer Kurven mit OpenCL". German. Bachelor's thesis. Halle (Saale), Germany: Martin Luther University Halle–Wittenberg, Institute of Computer Science, 2010.

[LOF01] H. Lopes, J. B. Oliveira, and L. H. d. Figueiredo. "Robust Adaptive Approximation of Implicit Curves". In: *Proceedings of the 14th Brazilian Symposium on Computer Graphics and Image Processing*. SIBGRAPI '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 10–17. ISBN: 0769513301. URL: `http://portal.acm.org/citation.cfm?id=646015.677786` (visited on 03/22/2013).

[LR01] T. Lickteig and M.-F. Roy. "Sylvester-Habicht Sequences and Fast Cauchy Index Computation". In: *Journal of Symbolic Computation* 31.3 (2001), pp. 315–341. ISSN: 07477171. DOI: `10.1006/jsco.2000.0427`.

[Mar+02] R. Martin, H. Shou, I. Voiculescu, A. Bowyer, and G. Wang. "Comparison of interval methods for plotting algebraic curves". In: *Computer Aided Geometric Design* 19.7 (2002), pp. 553–587. ISSN: 01678396. DOI: `10.1016/S0167-8396(02)00146-2`.

[MC92] D. Manocha and J. F. Canny. "Implicit Representation of Rational Parametric Surfaces". In: *Journal of Symbolic Computation* 13 (1992), pp. 485–510.

[MG04] J. F. Morgado and A. J. Gomes. “A Derivative-Free Tracking Algorithm for Implicit Curves with Singularities”. In: *Computational Science - ICCS 2004*. Ed. by M. Bubak, G. D. v. Albada, P. M. A. Sloot, and J. J. Dongarra. Vol. 3039. Lecture Notes in Computer Science. Springer-Verlag, 2004, pp. 221–228. DOI: `10.1007/978-3-540-25944-2_28`.

[Mis93] B. Mishra. *Algorithmic Algebra*. Secaucus, NJ, USA: Springer-Verlag, 1993. ISBN: 3540940901.

[Moo79] R. E. Moore. *Methods and Applications of Interval Analysis*. SIAM Studies in Applied Mathematics, 2. Society for Industrial and Applied Mathematics, 1979. ISBN: 9780898711615.

[Mou+06] B. Mourrain, S. Pion, S. Schmitt, J.-P. Técourt, E. P. Tsigaridas, and N. Wolpert. “Algebraic issues in Computational Geometry”. In: *Effective Computational Geometry for Curves and Surfaces*. Ed. by J.-D. Boissonnat and M. Teillaud. Mathematics and Visualization. Springer-Verlag, 2006. Chap. 3, pp. 117–155.

[MP11] M. M. Maza and W. Pan. “Solving bivariate polynomial systems on a GPU”. In: *ACM Commun. Comput. Algebra* 45.1/2 (2011), pp. 127–128. ISSN: 19322240. DOI: `10.1145/2016567.2016589`.

[MS11] K. Mehlhorn and M. Sagraloff. “A deterministic algorithm for isolating real roots of a real polynomial”. In: *Journal of Symbolic Computation* 46.1 (2011), pp. 70–90. ISSN: 07477171. DOI: `10.1016/j.jsc.2010.09.004`.

[MS99] M. Mignotte and D. Ştefănescu. *Polynomials: an algorithmic approach*. Springer series in discrete mathematics and theoretical computer science. Springer-Verlag, 1999. ISBN: 9789814021517.

[MY95] T. Möller and R. Yagel. *Efficient Rasterization of Implicit Functions*. Tech. rep. Department of Computer and Information Science, Ohio State University Columbus, 1995. URL: `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.30.1319` (visited on 03/22/2013).

[Obr25] N. Obreshkov. “Über die Wurzeln von algebraischen Gleichungen”. German. In: *Jahresbericht der Deutschen Mathematiker-Vereinigung* 33 (1925), pp. 52–64.

[Obr63] N. Obreshkov. *Verteilung und Berechnung der Nullstellen reeller Polynome*. Hochschulbücher für Mathematik. Deutscher Verlag der Wissenschaften, 1963.

[OF00] J. de Oliveira and L. de Figueiredo. “Robust Approximation of Offsets and Bisectors of Plane Curves”. In: *SIBGRAPI Conference on Graphics, Patterns and Images* (2000), p. 139. ISSN: 15301834. DOI: `10.1109/SIBGRA.2000.883906`.

[Ost50] A. M. Ostrowski. “Note on Vincent’s Theorem”. In: *The Annals of Mathematics*. Second Series 52.3 (1950), pp. 702–707. ISSN: 0003486X. DOI: `10.2307/1969443`.

[Pan94] V. Y. Pan. “Simple Multivariate Polynomial Multiplication”. In: *Journal of Symbolic Computation* 18.3 (1994), pp. 183–186. ISSN: 0747-7171. DOI: `10.1006/jsco.1994.1042`.

[Pra11] A. Prager. “Visualisierung von Mehrpunkt-Aufblasungen affiner Varietäten”. German. Diploma thesis. Halle (Saale), Germany: Martin Luther University Halle–Wittenberg, Institute of Computer Science, 2011.

[Pra85] V. Pratt. “Techniques for conic splines”. In: *SIGGRAPH Computer Graphics* 19.3 (1985), pp. 151–160. ISSN: 00978930. DOI: `10.1145/325165.325225`.

[PV04] S. Plantinga and G. Vegter. “Isotopic approximation of implicit curves and surfaces”. In: *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing.* (Nice, France). SGP ’04. New York, NY, USA: ACM, 2004, pp. 245–254. ISBN: 3905673134. DOI: `10.1145/1057432.1057465`.

[Rei97] D. Reischert. “Asymptotically fast computation of subresultants”. In: *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation.* (Kihei, Maui, Hawaii, United States). ISSAC ’97. New York, NY, USA: ACM, 1997, pp. 233–240. ISBN: 0897918754. DOI: `10.1145/258726.258792`.

[RR05] H. Ratschek and J. G. Rokne. “Scci-hybrid Methods for 2d Curve Tracing”. In: *Int. J. Image Graphics* 5.3 (2005), pp. 447–480. DOI: `10.1142/S0219467805001859`.

[RS08] M. Reimers and J. Seland. “Ray Casting Algebraic Surfaces using the Frustum Form”. In: *Computer Graphics Forum* 27.2 (2008), pp. 361–370.

[Rud98] W. Rudin. *Analysis.* Oldenbourg Verlag, 1998. ISBN: 3486241192.

[SA84] T. W. Sederberg and D. C. Anderson. “Implicit Representation of Parametric Curves and Surfaces”. In: *Computer Vision, Graphics and Image Processing* 28 (1984), pp. 72–84.

[Sag12] M. Sagraloff. “When Newton meets Descartes: a simple and fast algorithm to isolate the real roots of a polynomial”. In: *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation.* (Grenoble, France). ISSAC ’12. New York, NY, USA: ACM, 2012, pp. 297–304. ISBN: 9781450312691. DOI: `10.1145/2442829.2442872`.

[SAGE11] W. A. Stein et al. *Sage Mathematics Software (Version 4.7.2).* The Sage Development Team. 2011. URL: `http://www.sagemath.org` (visited on 03/22/2013).

[Sch82] A. Schönhage. “Asymptotically fast algorithms for the numerical muitiplication and division of polynomials with complex coefficients”. In: *Computer Algebra.* Ed. by J. Calmet. Vol. 144. Lecture Notes in Computer Science. Springer-Verlag, 1982, pp. 3–15. ISBN: 9783540116073. DOI: `10.1007/3-540-11607-9_1`.

[Slu70] A. Sluis. “Upperbounds for roots of polynomials”. In: *Numerische Mathematik* 15.3 (1970), pp. 250–262. ISSN: 0029599X. DOI: `10.1007/BF02168974`.

[SS10] C. Stussak and P. Schenzel. “RealSurf – A Tool for the Interactive Visualization of Mathematical Models”. In: *Arts and Technology.* Ed. by F. Huang and R.-C. Wang. Vol. 30. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer-Verlag, 2010, pp. 173–180. ISBN: 9783642115769. DOI: `10.1007/978-3-642-11577-6_22`.

[SS11] C. Stussak and P. Schenzel. “Interactive Visualisation of Algebraic Surfaces as a Tool for Shape Creation”. In: *International Journal of Arts and Technology* 4.2 (2011), pp. 216–228. DOI: `10.1504/IJART.2011.039846`.

[SS12] C. Stussak and P. Schenzel. “Parallel Computation of Bivariate Polynomial Resultants on Graphics Processing Units”. In: *Applied Parallel and Scientific Computing. PARA 2010: State-of-the-art in Scientific and Parallel Computing.* Ed. by K. Jónasson. Vol. 7134. Lecture Notes in Computer Science. Springer-Verlag, 2012, pp. 78–87. ISBN: 9783642281440. DOI: 10.1007/978-3-642-28145-7_8.

[SS13] P. Schenzel and C. Stussak. “Interactive Visualizations of Blowups of the Plane”. In: *IEEE Transactions on Visualization and Computer Graphics* 19.6 (2013), pp. 978–990. ISSN: 10772626. DOI: 10.1109/TVCG.2012.161.

[SS71] A. Schönhage and V. Strassen. “Schnelle Multiplikation großer Zahlen”. German. In: *Computing* 7 (3 1971), pp. 281–292. ISSN: 0010485X. DOI: 10.1007/BF02242355.

[Stu07] C. Stussak. “Echtzeit-Raytracing algebraischer Flächen auf der Graphics Processing Unit”. German. Diploma thesis. Halle (Saale), Germany: Martin Luther University Halle–Wittenberg, Institute of Computer Science, July 2007.

[Stu09] C. Stussak. “RealSurf – A GPU-based Realtime Ray Caster for Algebraic Surfaces”. Poster and extended abstract. In: *Proceedings of the 25th Spring Conference on Computer Graphics.* (Budmerice, Slovakia). Ed. by M. Samuelčík. Comenius University, Bratislava, Slovakia, 2009, pp. 59–61. ISBN: 9788022426445. URL: http://realsurf.informatik.uni-halle.de (visited on 04/08/2013).

[Stu11] C. Stussak. “On Exact Rasterization of Real Algebraic Plane Curves”. In: *Proceedings of the 27th Spring Conference on Computer Graphics.* (Viničné, Slovakia). Ed. by R. D. Tomoyuki Nishita. Comenius University, Bratislava, Slovakia, Apr. 2011, pp. 165–168. ISBN: 9788022330183.

[Stu29] J. C. F. Sturm. “Mémoire sur la résolution des équations numériques”. French. In: *Bulletin des Sciences de Férussac* (11 1829), 419–425.

[SW05] R. Seidel and N. Wolpert. “On the exact computation of the topology of real algebraic curves”. In: *Proceedings of the 21st Annual Symposium on Computational Geometry.* (Pisa, Italy). SCG ’05. New York, NY, USA: ACM, 2005, pp. 107–115. ISBN: 1581139918. DOI: 10.1145/1064092.1064111.

[SWP07] J. Sendra, F. Winkler, and S. Pérez-Daz. *Rational Algebraic Curves.* Algorithms and Computation in Mathematics. Springer-Verlag, 2007. ISBN: 9783540737254.

[Tau93] G. Taubin. “An accurate algorithm for rasterizing algebraic curves”. In: *Proceedings on the 2nd ACM Symposium on Solid Modeling and Applications.* (Montreal, Quebec, Canada). SMA ’93. New York, NY, USA: ACM, 1993, pp. 221–230. ISBN: 0897915844. DOI: 10.1145/164360.164427.

[Tau94] G. Taubin. “Rasterizing Algebraic Curves and Surfaces”. In: *IEEE Computer Graphics and Applications* 14 (1994), pp. 14–23. ISSN: 02721716. DOI: 10.1109/38.267467.

[TE06] E. Tsigaridas and I. Emiris. “Univariate Polynomial Real Root Isolation: Continued Fractions Revisited”. In: *Algorithms ESA 2006.* Ed. by Y. Azar and T. Erlebach. Vol. 4168. Lecture Notes in Computer Science. Springer-Verlag, 2006, pp. 817–828. ISBN: 978-3-540-38875-3. DOI: 10.1007/11841036_72.

[Vin36] A. J. H. Vincent. “Sur la Résolution des Équations Numeriques”. French. In: *Journal de Mathématiques Pures et Appliquées* 1 (1836), pp. 341–372.

[Win96] F. Winkler. *Polynomial Algorithms in Computer Algebra.* Secaucus, NJ, USA: Springer-Verlag, 1996. ISBN: 3211827595.

[Yap00] C.-K. Yap. *Fundamental problems of algorithmic algebra.* Oxford University Press, 2000, pp. I–XV, 1–511. ISBN: 9780195125160.

[YD95] C. K. Yap and T. Dubé. “The exact computation paradigm”. In: *Computing in Euclidean Geometry* 4 (1995), pp. 452–492.

List of figures

1.1. Graphical summary of the main contributions 3

2.1. Exemplary Habicht diagram 16
2.2. Obreshkov's circle theorems 29

3.1. Silhouette curve of a Cayley surface 36
3.2. Correct and wrong rasterizations of the curve $y^2 + x^2(x + 1) = 0$ 36
3.3. Outline of the exact rasterization algorithm for real algebraic plane curves . . . 39
3.4. Possible configuration of pixels (up to symmetry) assigned to a critical point . 42
3.5. Decomposition of the viewport into monotone, non-critical boxes 43
3.6. Example of tracing two curve segments in a monotone, non-critical box 45
3.7. The first column in a non-critical box 46
3.8. Special treatment of the first column in a non-critical box 47
3.9. A critical row with critical points and row boundary intersections 48
3.10. Treatment of non-critical pixels in a critical row 49
3.11. Some curve rasterizations produced by the new algorithm 59
3.12. Density plot of the speedup with respect to the preprocessing 60
3.13. Violin plot of the speedup with respect to the preprocessing 61
3.14. Density plot for the share of the subresultants on the precomputation time . . 62
3.15. Violin plot for the share of the subresultants on the precomputation time . . . 63
3.16. Number of unsuccessful curve rasterizations for each challenge 64
3.17. Density plot of the speedup with respect to the adjustment of the viewport . . 65
3.18. Violin plot of the speedup with respect to the adjustment of the viewport . . . 66
3.19. Density plot of the speedup with respect to the rasterization 67
3.20. Violin plot of the speedup with respect to the rasterization 68

4.1. Outline of the rendering algorithm for real algebraic space curves 73
4.2. Surfaces and intersection curves as defined in Example 4.4.3 75
4.3. Decomposed projection and associated lifting surfaces for Example 4.4.3 76
4.4. Surfaces and intersection curves of Example 4.5.1 77
4.5. Transformed surfaces and intersection curves of Example 4.5.1 for $\lambda = -1$ . . . 79
4.6. Further decomposition of the projection of the space curves of Example 4.5.1 . 80
4.7. Lifting surfaces of Example 4.5.1 in different coordinate systems 81
4.8. Rational parametrization of the space curve of Example 4.6.3 83
4.9. Turning multiple roots over the projection of Example 4.6.6 into simple ones . 85
4.10. Syzygy based reduction applied to Example 4.4.3 86
4.11. Surfaces and space curves containing straight lines 88
4.12. Visualization of the space curves used for benchmark purposes 91
4.13. Surface visualizations created with the program implemented in [Hie11] 95

5.1. Embedding of C in the disc $\mathbb{D}$, the blowup $\mathbb{B}_{f,g}$ and the toroidal blowup $\mathbb{T}_{f,g}$. 98
5.2. Mapping a point of the projective line to a circle 99
5.3. Parameter form of the toroidal blowup with $(f, g) = (u, v)$ (Möbius strip) . . . 101
5.4. Parameter form of the toroidal blowup with $(f, g) = (v^2 - u(u-1)(u+1), v^2)$. 101
5.5. Mapping a point of the projective line to a circle (extended) 107
5.6. Clipping the implicit form of the toroidal blowup 111
5.7. Rendering artifacts caused by the naive clipping idea 112
5.8. Texturing patterns for the toroidal blowup 113
5.9. Deformation of the toroidal blowup of three dynamic points 116
5.10. Deformation of the toroidal blowup of four static points 116
5.11. Some more examples of toroidal blowups 117

6.1. Outline of the parallel resultant algorithm 120
6.2. Speedup of the 32 bit modular multiplication 123

A.1. Rasterizations of some of the KO_d curves of Challenge 1 132
A.2. Rasterizations of some of the $\widetilde{KO}_{d,\varepsilon}$ curves of Challenge 3 132
A.3. Rasterizations of some of the $Nest^2_{d,\varepsilon,k}$ curves of Challenge 4 133
A.4. Rasterizations of some of the $KO^{\mp}_{d,\varepsilon}$ curves of Challenge 6 133
A.5. Rasterizations of some of the $f^2_{k,l,-}$ curves of Challenge 7 134
A.6. Rasterizations of some of the $ni_{m\{,k\}}$ and $ni\{x, y\}_{m,n,k,l}$ curves of Challenge 8 . 135
A.7. Rasterizations of some of the $C_{k,l}$ curves of Challenge 9 135
A.8. Rasterizations of some of the $F^2_{k,l,m,-}$ curves of Challenge 10 136
A.9. Rasterizations of some of the $FT^2_{k,l,m,-}$ curves of Challenge 11 136
A.10. Rasterizations of some of the $FTT^2_{k,l,m,\mp}$ curves of Challenge 12 137
A.11. Rasterizations of some of the $dfold_d$ and ${dfold^{fl}}_d$ curves of Challenge 13 138
A.12. Rasterizations of some of the $dfold_{k,l}$ curves of Challenge 14 138
A.13. Rasterizations of some of the $J^{\pm}_{10+k}$ curves of Challenge 15 for $a = \frac{9}{7}$ 139
A.14. Rasterizations of some of the $X^{\pm\pm}_{9+k}$ curves of Challenge 15 for $a = \frac{9}{7}$ 139
A.15. Rasterizations of some of the $Y^{\pm\pm}_{r,s}$ curves of Challenge 15 for $a = \frac{9}{7}$. 140
A.16. Rasterizations of some of the $\tilde{Y}^{\pm\pm}_r$ of Challenge 15 curves for $a = \frac{9}{7}$ 140
A.17. Rasterizations of some of the $SA_{k,l}$ curves of Challenge 17 141
A.18. Rasterizations of some of the $SA_{k,l,\pm\varepsilon}$ curves of Challenge 18 141
A.19. Rasterizations of some of the $SCA_{k,l,\varepsilon}$ curves of Challenge 19 142
A.20. Rasterizations of some of the $SAA_{k,l,\varepsilon}$ curves of Challenge 20 142
A.21. Rasterizations of some of the $SpRa_{d,n}$ curves of Challenge 21 143
A.22. Rasterizations of some of the $DeRa_{d,n}$ curves of Challenge 22 143

B.1. Scatter plot of the preprocessing time 146
B.2. Scatter plot of the time for viewport adjustment and critical point refinement . 147
B.3. Scatter plot of the time for rasterizing on a 512 × 512 grid 148
B.4. Scatter plot of the time for rasterizing on a 1024 × 1024 grid 149

List of tables

2.1. Signs occurring in Equation (2.62) 27
2.3. Summary of algorithm complexities 33

3.1. Resolution of uncertain sign patterns in a signed subresultant sequence 51
3.2. Resolution of uncertain signs during real root counting of the “Bundle” curve . 52
3.3. Quartile and median of the speedups with respect to the rasterization 66

4.1. Computing times for visualizing the space curves shown in Figure 4.12 93

6.1. Computing times for the different parts of the parallel resultant algorithm . . . 128
6.2. Benchmarks for GPU based resultant computations 129

List of algorithms

3.1. Trivial exact plane curve rasterization 37
3.1. Improved trivial exact plane curve rasterization 37

6.1. Fast multiplication $a \cdot b \bmod p$ of 32 bit integers a, b, p 122
6.2. Parallel univariate resultant computation on shared memory 125

Curriculum vitae

Christian Stussak

Personal details

Born April 17th, 1982 in Torgau (Germany)

German citizenship

Education

10/2008–	Ph. D student in Computer Science Martin-Luther University Halle-Wittenberg (Germany)
11/2007	Diplom (Master's degree) in Computer Science Martin-Luther University Halle-Wittenberg (Germany) Thesis: Echtzeit-Raytracing algebraischer Flächen auf der Graphics Processing Unit (Real time raytracing of algebraic surfaces on the graphics processing unit)
10/2001–11/2007	Studies in Computer Science Martin-Luther University Halle-Wittenberg (Germany) Major subjects: Computer Graphics, Software Engineering Elective subject: Design Informatics (Burg Giebichenstein University of Art and Design Halle)
07/2000	Abitur at Gymansium Jessen (Germany)

Professional positions

05/2012–	Developer of scientific visualization software Mathematisches Forschungsinstitut Oberwolfach (Germany) Project: IMAGINARY — Open mathematics
05/2008–04/2012	Research associate Computer Graphics group at Institute of Computer Science Martin-Luther University Halle-Wittenberg (Germany)

12/2007–04/2008	Developer of scientific visualization software Mathematisches Forschungsinstitut Oberwolfach (Germany) Project: IMAGINARY — Through the eyes of mathematics
03/2003–07/2003	Research assistant Design Informatics group Burg Giebichenstein University of Art and Design Halle (Germany)

Halle (Saale), Monday 25th
November, 2013

. . . .
Christian Stussak

Acknowledgments

I want to thank all those who supported and motivated me throughout these last years.

Special thanks go to Peter Schenzel who piqued my interest in the visualization of algebraic curves and surfaces already during my diploma studies and thereby paved the way to more in-depth research in his working group. His knowledge of algebra, algebraic geometry and computer algebra always enriched our discussions and had a lasting impact on my scientific work.

Furthermore, I want to thank the other members of the AG Computergrafik: Hendrik Bugdoll for his support with some run time measurements and Andreas Bienert for proofreading this thesis for scientific errors.

I owe a debt of gratitude to my wife, Heike Riegler, for her seemingly endless capability to motivate me once and again during the long time this work has taken to finish. Also, I am thankful for her countless advice on improving the text at hand.

Last but not least, I want to thank the team of the IMAGINARY project, who enabled me to continue my research and finish the work on this thesis after my employment at the University Halle-Wittenberg ended.

Danksagung

Mein Dank gilt all denen, die mich in den letzten Jahren begleitet, gefördert, unterstützt und aufgemuntert haben.

Ganz besonders danke ich Peter Schenzel, der mein Interesse für die Visualisierung algebraischer Kurven und Flächen bereits während meines Studiums weckte und damit den Weg für die Vertiefung des Themas in seiner Arbeitsgruppe ebnete. Seine Kenntnisse der Algebra, algebraischen Geometrie und Computeralgebra haben stets unsere Diskussionen bereichert und ich denke, ich konnte einiges davon mitnehmen.

Auch möchte ich den Mitgliedern der AG Computergrafik danken: Hendrik Bugdoll für die Unterstützung bei einigen Laufzeitmessung sowie Andreas Bienert für das fachliche Korrekturlesen dieser Arbeit.

Meiner Frau Heike Riegler bin ich vor allem Dank schuldig für ihr schier endloses Vermögen mich in der langen Zeit bis zur Fertigstellung der Dissertation immer wieder aufs Neue zu motivieren. Auch ihre unzähligen Vorschläge zur Verbesserung des Textes waren mir eine unschätzbare Hilfe.

Zu guter Letzt danke ich dem Team des IMAGINARY-Projekts, das es mir nach meiner Tätigkeit an der Universität Halle-Wittenberg ermöglicht hat, meine Arbeit an der Dissertation fortzuführen und zu beenden.

Declaration under Oath
Eidesstattliche Erklärung

I declare under penalty of perjury that this thesis is my own work entirely and has been written without any help from other people. I used only the sources mentioned and included all the citations correctly both in word or content.

Ich erkläre an Eides statt, dass ich die Arbeit selbstständig und ohne fremde Hilfe verfasst, keine anderen als die von mir angegebenen Quellen und Hilfsmittel benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Halle (Saale), Monday 25th
November, 2013

....
Christian Stussak