Approximate methods for change of representation and their applications in CAGD

Oliver J. D. Barrowclough

2012

Preface

The work that comprises this thesis has been performed as part of the EU-project Shapes, Geometry and Algebra (SAGA) under grant agreement n° PITN-GA-2008-214584. The SAGA network consists of 10 partners including industrial companies, research institutes and universities, based throughout Europe. The broad aims of the project are to exploit mathematical results from fields such as Algebraic Geometry, Numerical Analysis and Computer Algebra, in the applied field of Computer Aided Design and Manufacturing (CAD/CAM). The majority of the work in this thesis is related to Work Package I in the SAGA project, whose aim is to investigate methods for change of representation.

Acknowledgements

I would first like to express my thanks to the main source of funding for this project, which was through my early-stage researcher (ESR) fellowship in the SAGA project. I would also like to thank the Research Council of Norway for the extra funding they provided (IS-TOPP, Project number 201280). I want to express my gratitude to SINTEF for providing a stimulating workplace for me for the past three years. I would also like to thank the other institutions I have been involved with including Center of Mathematics for Applications (CMA) and the Department of Informatics at University of Oslo, and the Institute of Applied Geometry at Johannes Kepler University, Linz.

I would like to thank my main supervisor Dr. Tor Dokken whose previous work inspired the thesis and who provided continual support and insight during the project. He should also receive great credit for the effort he put in to coordinate the SAGA project. Thanks also go to Professor Bert Jüttler who guided me well during my time in Linz, and with whom I collaborated on a paper along with Dr. Tino Schulz.

At SINTEF, I am grateful to all those in the Department of Applied Mathematics. Particular thanks go to Johan Seland for his help in installing his software for algebraic surface visualisation, which has been useful in my research. Also to those in the Geometry group including Jan Thomassen and Vibeke Skytt whose help has been valuable throughout. During my time at SINTEF I have also had the pleasure of working alongside several colleagues from the SAGA project including Jayasimha Bagalkote, Thien Nguyen, Peter Nørtoft, Dang Manh Nguyen, Tatjana Kalinka and Heidi Dahl, who have all contributed to a brighter workplace.

There are many other people I would like to thank for making the whole experience more enjoyable. At CMA, everyone who attended the Geometry seminar, where I learnt a lot about many different topics. Thanks to everyone who was involved in the SAGA project; the SAGA meetings have always been fantastic experiences both professionally and socially. A big thanks

also to everyone in Linz who made me feel so welcome for the three months I spent there.

Last but not least, I would like to thank my family for their support and encouragement, and for putting up with me living abroad for so long. And my greatest thanks go to Elisabeth for her constant support and for being there for me throughout.

Abstract: In modern computer aided design (CAD) systems, there are two main representations of curves and surfaces: parametric and implicit representations. The two representations complement each other in the sense that their properties can be used to solve different geometric problems. Parametric curves are well suited to point generation, whereas implicit representations efficiently determine whether or not a point lies on a curve or surface. The availability of both representations is of great advantage in answering geometric queries, such as intersection and trimming problems. Methods for *change of representation* are thus of great practical importance. In this thesis we present several methods for *implicitization*; the process of changing from the parametric to the implicit representation. The emphasis is on methods which are numerically stable and computationally efficient on modern hardware. We present a variety of approaches to *approximate implicitization* using linear algebra. In particular, we explore the implicitization of both rational parametric representations and envelopes - a type of curve or surface which often has no simple parametric representation. We also introduce a new method for the implicit representation of rational cubic Bézier curves, which can be implemented using explicit formulas.

Contents

Part I

Introduction

Chapter 1

Background

Mathematical modelling of geometric objects plays a central role in the development of many products in modern industry. Applications include automotive and aeronautical engineering, architecture, computer graphics, animation and robotics. To allow for the vast range of required shapes, modern computer aided design (CAD) systems are based on representations which encompass a very general set of freeform curves and surfaces. Freeform representations allow designers to model curves and surfaces of almost any shape, whilst also giving exact representations of common geometric primitives, such as lines, ellipses, planes, ellipsoids and cylinders.

There are several ways that curves and surfaces can be represented mathematically, each of which has properties which may be better suited to certain applications than others. By far the most prevalent representation in CAD is the parametric form. Parametric curves and surfaces allow for easy point generation, which can be used for rendering. Moreover, the popular Bézier and non-uniform rational B-spline (NURBS) parametric forms also give natural geometric control of the shape using a control polygon. However, it is often necessary to perform various geometric operations between the shapes, such as intersections, unions, differences and offsets. For such applications, the implicit form is often more suitable. The implicit form is also best for deciding whether a given point lies inside, outside or exactly on a given curve or surface. For these reasons, methods for *change of representation* are of great practical importance.

From a human perspective, identifying the point where a curve crosses itself (such as the curve pictured in Figure 1.3), is a trivial problem. However, from a mathematical perspective, which is necessary for computational implementation, the problem resorts to polynomial rootfinding. This highlights a connection with the classical subject of algebraic geometry. Algebraic techniques tailored for use in CAD systems emerged about 30 years ago, and have gradually gained traction in the research community. One of the main obstacles to their practical use has been the high polynomial degrees that occur when exact operations are performed. High polynomial degrees often lead to slow algorithms, and problems with numerical stability. To overcome this, *approximate* algebraic methods have been proposed.

Generally, methods for change of representation are computationally expensive. However, the ever increasing computational power in commodity computers opens new doors for their use in industrial systems. In previous years, increased central processing unit (CPU) speeds have removed the barrier for the implementation of some computationally intensive methods. More recently, the rise of the highly parallel graphics processing unit (GPU) in commodity computers has led to an increase in research of high quality visualization methods for implicit surfaces.

Such progressions suggest that future CAD systems have the potential to realize the benefits of using both parametric and implicit representations.

In the majority of this thesis we investigate approximate methods for implicitization - the change of representation from the parametric to the implicit form. We also look at efficient methods for exact implicitization of low degree curves. The work is motivated both as a continuation of recent research into approximate algebraic geometry, and a desire to construct methods better suited for numerical computation on modern architectures.

The thesis consists of two parts. In Part I we provide an introduction to methods for implicitization and their applications. In Chapter 1, we begin with a background to the subject, providing some historical context and continuing through to modern trends. We provide a summary of the scientific work which constitutes the main part of the research in this thesis in Chapter 2. Since the scientific papers are mainly written from a theoretical perspective, we conclude Part I with a chapter highlighting applications of the methods in CAD, computer graphics and robotics. In Part II we present four scientific papers written as a part of the EU project ShApes Geometry and Algebra (SAGA) and the Research Council of Norway project under the same name as this thesis. The papers consist of two journal articles (one published [10], and one under review [8]), one conference proceedings article [9] and one book chapter [11]. We include a single bibliography for the first three chapters at the end of Part I, and separate bibliographies for each of the papers in Part II.

1 Freeform curve and surface representation

A large proportion of the curves and surfaces used in CAD have very simple forms that also exhibit simple mathematical representations. In 2D, these include lines, ellipses, parabolas and hyperbolas, all of which come under the common term of conic sections. In 3D, such surfaces include planes, spheres, cylinders and other shapes known as quadrics. In both the 2D and 3D cases, these shapes can be represented both parametrically and implicitly. However, more advanced models, such as those seen in automobile design, aircraft wings, ship hulls and wind turbines require more general *freeform representations*. The development of systems for the design of freeform curves and surfaces has a long history. Before computer aided design had been popularized, early methods for curve representation in design applications utilized conic sections [33, 45]. Motivated by the increased availability of affordable computing power, more advanced techniques for curve and surface representation appeared around the 1960s. Computer aided geometric design (CAGD) was the term given to the research of such techniques. In this section we describe two of the most popular methods for freeform curve and surface representation in CAGD.

1.1 Parametric representations - Bézier and Splines

Since the conception of CAGD in the 1960s and 1970s, the parametric representation of curves and surfaces has been the staple of CAD systems. De Casteljau and Bézier independently developed methods for the representation of a particular type of polynomial parametric curve, known today as Bézier curves. Bézier curves, of a given degree n, can be defined in terms of Bernstein polynomials $(B_i(t))_{i=0}^n$ and a set of control points $(\mathbf{c}_i)_{i=0}^n$. The univariate Bernstein

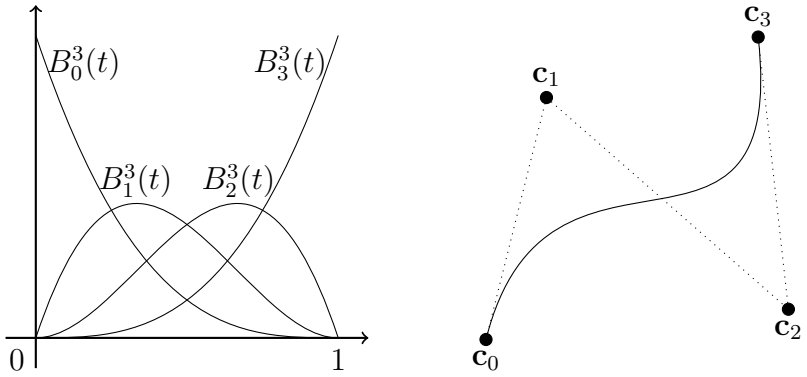


Figure 1.1: Cubic Bernstein basis functions and a polynomial cubic Bézier curve

polynomials of degree n have the following definition:

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}, \quad t \in \mathbb{R}.$$

The Bézier curve, normally defined for $t \in [0,1]$, is represented by

$$\mathbf{p}(t) = \sum_{i=0}^{n} \mathbf{c}_i B_i^n(t).$$

Figure 1.1 shows an example of the Bernstein basis functions for the cubic case ($n = 3$), and a cubic Bézier curve. An important generalization of the polynomial[1] Bézier curve is the rational Bézier curve. These have a similar definition, but weights are defined that adjust the influence each control point has on the curve. The rational curve is given by,

$$\mathbf{r}(t) = \frac{\sum_{i=0}^{n} w_i \mathbf{c}_i B_i^n(t)}{\sum_{i=0}^{n} w_i B_i^n(t)}.$$

Rational curves are necessary for the exact parametric representation of certain shapes, such as circles in the plane and spheres and cylinders in 3D.

The wide adoption of Bézier curves is mainly due to their geometric properties - the curve naturally approximates the piecewise linear interpolant of the control points, known as the control polygon. Moreover, Bernstein polynomials have been shown to have superior numerical properties, and there exist stable recursive algorithms for their evaluation, such as the de Casteljau algorithm. Many of these properties result from the polynomials forming a non-negative partition of unity; that is

$$\sum_{i=0}^{n} B_i^n(t) = 1 \text{ and } B_i(t) \geq 0, \quad \text{for all } t \in [0,1].$$

However, when using Bézier curves, if more control points are required, the degree must be raised. For higher degrees the control polygon exhibits less influence on the curve, and the

[1] Polynomial Bézier curves are also known as non-rational or integral Bézier curves.

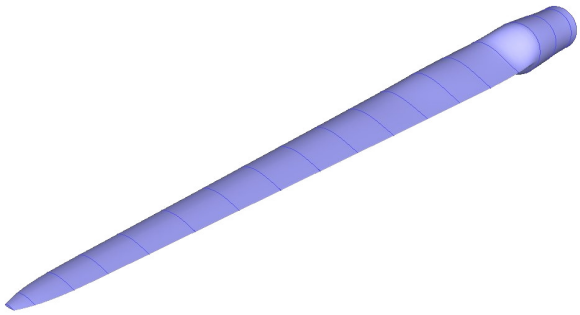


Figure 1.2: Bicubic tensor-product spline surface of a propeller blade

algorithms become more computationally intensive. Although there are several applications of higher degree curves, curves of degree three or less are most prevalent.

To provide more design flexibility without increasing the degree, spline curves have been introduced. These curves can be thought of as piecewise Bézier curves, where the continuity between any two pieces is automatically ensured. The definition is given in terms of the B-spline basis, a generalization of the Bernstein basis, which shares many properties such as partition of unity and the existence of a recursive algorithm for their evaluation, known as the Cox-de Boor algorithm. In the most widely used form, splines appear as non-uniform rational B-splines (NURBS). Like for Bézier curves, the extension from splines to NURBS provides the possibility to model circles and other geometric objects exactly. NURBS curves can always be split into a series of rational Bézier segments by a process known as knot insertion. For this reason, the majority of this thesis will deal with only the case of rational Bézier curves and surfaces.

The development of parametric surfaces has followed essentially the same path as for curves, with two main generalizations - triangular and tensor-product surfaces. The polynomial bases for these surfaces are given respectively by,

$$B^n_{i,j,k}(s,t,u) = \binom{n}{i,j,k} s^i t^j u^k, \quad i+j+k=n,\ s+t+u=1,$$

where s, t and u denote barycentric coordinates, and

$$B^{n_1,n_2}_{i,j}(s,t) = \binom{n_1}{i}\binom{n_2}{j} B^{n_1}_i(s) B^{n_2}_j(t), \quad 0 \le i \le n_1, 0 \le j \le n_2,\ (s,t) \in \Omega$$

where $B^{n_1}_i(s)$, and $B^{n_2}_j(t)$ denote univariate Bernstein basis functions and $\Omega = [s_0, s_1] \times [t_0, t_1]$. The surfaces are defined by a linear combination of these basis functions with control points in $\mathbb{R}^3$. Tensor-product surfaces have received the most attention since their definition is much simpler than the triangular case, and properties can often be determined by considering the two univariate cases separately. Figure 1.2 shows an example of how tensor-product NURBS surfaces can be used in the design of industrial parts.

Planar curves and surfaces in 3D are examples of *hypersurfaces* in dimensions 2 and 3 respectively. In a general dimension n, a hypersurface is defined as a manifold of dimension

$n-1$. Although for CAGD applications we are normally interested only in curves and surfaces, applications in robotics, for example, often require higher dimensional consideration.

The parametric form is best suited for point generation. In particular, using rational parametrizations, one can easily generate points along a curve or surface using only a few arithmetic operations. Other parametrizations, such as those using trigonometric functions, are possible; however, rational parametrizations have found most use since they can represent a large class of shapes and can be evaluated accurately and efficiently.

1.2 Implicit representations

Implicit curves and surfaces (in $\mathbb{R}^2$ or $\mathbb{R}^3$ respectively), are represented as the set of points that satisfy some equation

$$q(\mathbf{x}) = 0,$$

for a scalar valued function q, and variables $\mathbf{x} = (x_1, \ldots, x_d)$, where $d = 2, 3$. When the function q is a polynomial, the curve or surface thus defined is termed an *algebraic* curve or surface. In this thesis we restrict our attention to algebraic curves and surfaces, although those defined by other functions (e.g., radial basis functions) have also found a number of applications [52]. It is also possible to define space curves implicitly, by the intersection of two implicit surfaces in $\mathbb{R}^3$, however this requires two polynomials.

A polynomial of total degree n in two variables x and y can be given in power or monomial form as

$$q(x, y) = \sum_{0 \le i+j \le n} b_{i,j} x^i y^j.$$

We can also use the more general form

$$q(x, y) = \sum_{0 \le i+j \le n} c_{i,j} B_{i,j}(x, y),$$

where the set $\{B_{i,j}(x, y) : 0 \le i + j \le n\}$ is a basis for polynomials of total degree n. The implicit curve is then given by

$$I = \{(x, y) \in \mathbb{R}^2 : q(x, y) = 0\}.$$

In modern CAGD the Bernstein basis is often chosen for implicit representations, whereas more theoretical work in algebraic geometry often uses the monomial basis. The use of the Bernstein basis for implicit representations was apparently first suggested by Sederberg [61, 62], and advocated by Dokken for finding numerically stable implicit approximations [26, 27]. It has also been used by Floater for an explicit method for rational cubic curve implicitization [37].

Classical results from algebraic geometry were generally considered in projective space over the complex numbers, as opposed to affine space over the real numbers, which is the standard in CAGD. As a result, theorems such as Bézouts theorem, give upper bounds on the number of intersections of real curves in affine space [28].

For surfaces we need to define polynomials in three variables. The monomial basis has a simple trivariate extension. The Bernstein basis can also be extended in a natural way to either tetrahedral or tensor-product domains.

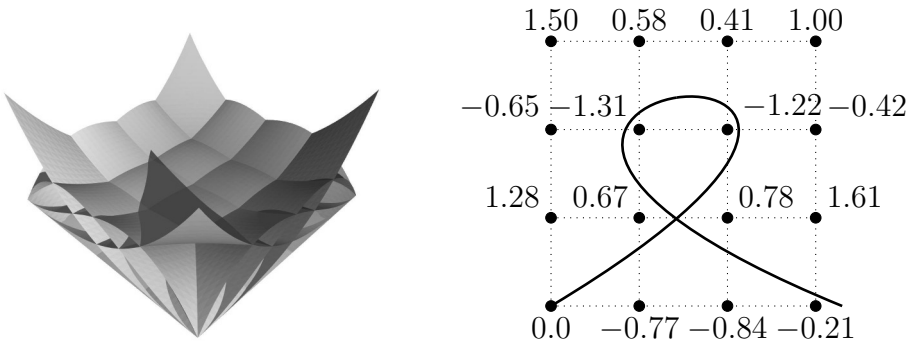


Figure 1.3: Bicubic tensor-product Bernstein basis functions and an implicit curve defined by a bicubic tensor-product Bernstein polynomial with the stated coefficients.

The set of implicit algebraic curves and surfaces encompasses a much wider range of curves and surfaces than can be defined parametrically. For example, in an algebraic surface of total degree six, there are 83 degrees of freedom, giving a huge amount of flexibility to model a large class of surfaces, including freeform surfaces. In Figure 1.4 it can be seen that implicit surfaces even of low degree can represent relatively complex geometries and topologies. Moreover, the implicit representation is closed under several geometric operations such as intersection, union, difference and offset. If parametric representations are used, approximation of such operations is necessary, whereas implicit representations have the potential of giving exact results [40, 63].

Despite the useful properties it exhibits, the implicit representation has found limited use in CAD systems. There are several practical reasons for this. A major drawback of implicit curves and surfaces is that it is difficult to directly control the geometry by manipulating the coefficients of the defining polynomial. Although using a Bernstein representation gives some geometric control of the implicit form, the mechanism for design is still a great deal less intuitive than the parametric control polygon. Moreover, the paradigm of design using control polygons is ubiquitous in modern CAD systems. Thus to be of practical use, implicit representations should be viewed as a complement to the parametric form, rather than an alternative. Other problems with the implicit form include the high polynomial degrees often present when using exact implicit representations of parametric forms, particularly when it comes to surfaces. This leads to problems with computational efficiency and numerical stability.

These challenges mean that algorithms dealing with implicit representations still generate a great deal of research interest. For example, visualization of implicit surfaces has been the subject of several recent research papers [46, 47, 67]. Using such techniques, algebraic surfaces of moderately high degree can be visualized in real-time on today's graphics hardware. These techniques were not possible on previous generations of commodity computers which mainly utilized serial CPU computations, and rely heavily on the possibility to compute in parallel.

The extension to piecewise implicit representations is less natural than the spline representation for parametric curves, not least in defining regions for where the representation should be defined. However, some results in that direction, such as A-splines, are available [5, 7, 44, 61]. We will only consider implicit representations using a single polynomial.

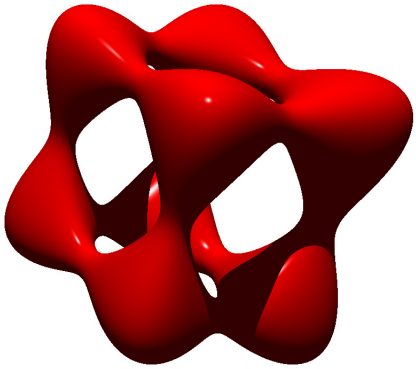


Figure 1.4: An implicitly defined surface of total degree four, visualized using the POV-Rayray tracer [53].

1.3 Change of representation

It is well known that any rational parametric curve or surface can be written in implicit form, the process of which is known as *implicitization*. The conversion in the opposite direction, known as *parametrization*, is not always possible, since the implicit representation encompasses a larger class of shapes. Methods for parametrization of algebraic curves and surfaces have generated some research interest. In [1, 2, 3] methods for computing rational parametrizations of curves and surfaces are given, whenever the curve or surface is rational (that is, has genus zero). A method for approximating any algebraic surface by a rational parametrization is discussed in [77]. As mentioned in the previous sections, parametric forms have become the prevalent representation at the design stage of an application. It is therefore the problem of implicitization that has received most attention from the research community. The main focus of this thesis is on methods for implicitization. In the following section we give an overview of existing methods for implicitization.

Another important problem in CAGD is that of inversion. Inversion formulas are used to determine the parameter value of a particular point which is known to lie on the curve or surface. These are of particular use in intersection algorithms, for finding the parameter values corresponding to intersection in affine space. Inversion formulas are undefined at singularities and are not useful for points which are not known to lie on the curve or surface.

1.4 Alternative curve and surface representations

Although implicit and parametric representations define the majority of geometric objects in CAGD, there also exist alternative representations. For example, in computer animation, subdivision surfaces are the representation of choice. In a similar way to Bézier curves, a control polygon is defined which gives geometric control of the curve or surface to the designer. Previously subdivision surfaces were not regarded as compatible with NURBS representations. However, recent work on defining such compatibilities was presented in the thesis of Cashman [17]. Since the main applications of implicitization are within CAGD and robotics, we do not

consider subdivision curves and surfaces here.

As well as rational parametric forms, there exist geometric objects which may not be given explicitly in either parametric or implicit form. Examples of these include offsets, envelopes and point clouds. Although some rational curves and surfaces have rational offsets, this is not true in general. However, square-root parametrizations of offsets can always be defined. Envelopes, which define a superset of offsets, usually have no simple parametric representation. A planar envelope curve is defined as a curve which touches all members of a given family of curves in the plane. They are useful, for example, in defining boundaries. In Paper III of this thesis we discuss a method for the implicitization of envelope curves. Implicitization of point clouds is also possible using the methods in Paper II. In the case when the points are generated from a rational parametrization, we give results on the quality of the approximation depending on the point distribution in the parameter domain.

In the following section we give an overview of several classical and modern approaches to the problem of implicitization.

2 Methods for implicitization of rational curves and surfaces

Research into implicitization has an interesting history that goes back at least as far as the early part of the nineteenth century. In this section we look at how several approaches have been developed from the point of view of both algebraic geometry and CAGD. We will discuss both methods that require symbolic computation as well as those more suited to numerical implementations. A more in depth review of exact implicitization methods in symbolic computation can be found in [43]. For the sake of simplicity, in the majority of this section we will discuss implicitization of curves. However, most methods are directly extensible to surfaces, although some of the resultant based methods do become more complicated.

2.1 Elimination theory

One approach to implicitization is to think of it as the elimination of the parametric variables. For example, consider the parametric curve given in a Cartesian coordinate system by

$$\mathbf{p}(t) = (x(t), y(t))^T = (t, t^2 - t + 1)^T, \quad t \in \mathbb{R}.$$

Since we have $x(t) = t$, the parametric curve traces out the graph of $y(t)$. Simply by substituting $t = x$ in the equation for $y(t)$ we get $y = x^2 - x + 1$. Thus, the polynomial $q(x, y)$, whose zero set corresponds to the same curve is given by $y - x^2 + x - 1$. This is a simple example of how the parametric variable can be eliminated in order to obtain the implicit equation.

For higher degree curves and surfaces, the elimination of variables becomes more difficult. Using a similar technique of solving for one variable, then substituting into the other, the methods quickly become complicated. This is because polynomial solving is non-linear by nature for degrees higher than one. However, it turns out that the problem can be solved using only linear techniques. Elimination theory provides a systematic method for the treatment of elimination of variables in very general settings.

Algebraic geometry and resultants

One approach to solving the elimination problem is that of resultants. A resultant can be thought of as a tool to test whether a set of polynomials share any common roots. Resultants first appeared in algebraic geometry in the mid nineteenth century [18, 72]. Original methods for computing the resultant led to the definitions of the Sylvester matrix and the more compact Bézout matrix. The following is a brief discussion of resultants for use in rational curve implicitization. For a more extensive review, we refer the reader to [23, 39, 63].

Consider univariate polynomials $f(t) = f_0 + \cdots + f_m t^m$ and $g(t) = g_0 + \cdots + g_n t^n$, with $f_m \neq 0$ and $g_n \neq 0$. The resultant of these can be given as the determinant of the Sylvester matrix, which is defined as follows.

Definition 1. *The Sylvester matrix is given by*

$$Syl(f,g) = \begin{pmatrix} f_m & f_{m-1} & \cdots & f_0 & & & \\ & f_m & f_{m-1} & \cdots & f_0 & & \\ & & \ddots & & & \ddots & \\ & & & f_m & f_{m-1} & \cdots & f_0 \\ g_n & g_{n-1} & \cdots & g_0 & & & \\ & g_n & g_{n-1} & \cdots & g_0 & & \\ & & \ddots & & & \ddots & \\ & & & g_n & g_{n-1} & \cdots & g_0 \end{pmatrix}$$

where all omitted entries are zero.

It can be shown that the two polynomials share a common factor if and only if the determinant of the Sylvester matrix is zero [23]. An alternative method for computing resultants is via the determinant of the Bézout matrix.

Definition 2. *Let* $p = \max(m,n)$. *The Bézout matrix* $\mathrm{Bez}(f,g)$ *is given by* $(b_{ij})_{i=0,j=0}^{p,p}$ *where*

$$b_{ij} = \sum_{k=0}^{\min(i,p+1-j)} f_{j+k-1}g_{i-k} - f_{i-k}g_{j+k-1}$$

and $f_i = 0$ *if* $m < i$ *and* $g_j = 0$ *if* $n < j$.

The Sylvester matrix has a very simple definition in terms of the coefficients of the two polynomials, but is in general of dimension $(m+n) \times (m+n)$. Although the Bézout matrix is slightly more complicated in definition, the matrix has dimension $\max(m,n) \times \max(m,n)$. Thus, for polynomials with equal degree, the Bézout matrix contains one quarter of the elements of the Sylvester matrix (although the Sylvester matrix exhibits some degree of sparsity). Since the bottleneck in resultant computation is in computing the determinant, the smaller Bézout matrix is often preferred in practice.

To put elimination using resultants in the perspective of implicitization, consider a rational parametric planar curve given by

$$\mathbf{p}(t) = \left(\frac{f(t)}{h(t)}, \frac{g(t)}{h(t)}\right)^T, \tag{1.1}$$

where $f(t) = f_0 + \cdots + f_n t^n$, $g(t) = g_0 + \cdots + g_n t^n$ and $h(t) = h_0 + \cdots + h_n t^n$ are univariate polynomials of degree n, and GCD$(f, g, h) = 1$. The implicit equation of the curve is simply given as the resultant of the polynomials $xh(t) - f(t)$ and $yh(t) - g(t)$ [21]. In this case, the elements of the resultant matrices will have entries in the variables x and y. That is, the determinant of the matrix will then define a polynomial q, in x and y which is zero whenever the equations $xh(t) - f(t) = 0$ and $yh(t) - g(t) = 0$ are satisfied (i.e., an implicit equation containing the parametric curve).

For the implicitization of surfaces in $\mathbb{R}^3$, methods for the elimination of two variables from a set of three polynomials are necessary. For this purpose the Macaulay and Dixon resultants, which were developed in the early part of the twentieth century, have been used [24, 48, 51].

An alternative tool for elimination of variables is that of Gröbner bases. Gröbner bases were developed by Buchberger in 1965 [12], and can be used to solve a variety of problems, including implicitization and inversion. Generally, methods using Gröbner bases are slow in comparison to resultant based methods, especially for space curves and surfaces. We will therefore not discuss them further here, but refer the reader to [38] for details.

Reintroduction of elimination theory in CAGD

Many of the classical results on elimination theory had been lost to academic community until the thesis of Sederberg was published in 1983 [60]. Indeed, it appeared that the CAGD community was previously unaware that closed form solutions to the implicitization problem existed. Motivated by applications in CAGD, Sederberg reintroduced the community to the concepts of elimination using resultants [39, 63].

Despite the growing interest in classical methods of algebraic geometry, several obstacles hindered the practical use of the algorithms. Some of the problems, highlighted in the papers [26, 39, 64] among others, include the following:

- **Additional solutions.** It is often the case that the implicit polynomial defined by resultant computation contains factors that are not part of the implicit equation of the curve. Thus, polynomial factorization is required to find the irreducible polynomial of interest. Factorization of polynomials is an undesirable operation in CAGD, especially when using floating points coefficients. This is because small perturbations in the coefficients of a reducible polynomial, which occur when approximating with floating point numbers, can render it irreducible.

- **Slow algorithms.** Computation of the determinant of a matrix with symbolic entries entails $O(n!)$ operations, where n is the dimension of the matrix. Thus, the methods are highly dependent on the size of the matrix, and quickly become slow for high degrees. Often the evaluation times will be unreasonable for applications, and, moreover, symbolic computation is not always easily available outside computer algebra systems.

- **Numerical stability.** For reasons of performance, it is often desirable to use floating point numerics. In CAGD, rational parametric curves and surfaces are mostly given in Bernstein form, whereas the traditional techniques for implicitization typically use the monomial basis. Unfortunately, the basis transformations between Bernstein and monomial bases are ill conditioned, and a loss of accuracy is to be expected [35].

- **High polynomial degrees.** As described previously, the exact algebraic degrees of surfaces can be undesirably high. Sometimes (e.g., if symmetries occur), the algebraic degree will reduce, but extra factors in the implicit equation will occur. High degrees also lead to issues with performance, and numerical stability.

- **Base points.** A base point is defined as a point where $(f(t), g(t), h(t)) = (0, 0, 0)$. In the case that base points occur, traditional methods for to resultant computation will fail (the implicit polynomial will be identically zero). The existence of base points, in fact, simplifies the implicit polynomial in the sense that it reduces the algebraic degree. It should be noted that base points are a common occurrence even in freeform surfaces [64].

- **Unwanted branches and self-intersections.** Whereas parametric curves and surfaces are defined in a specific parameter domain, implicit curves require a 2D region of interest to be defined. It is quite possible, and often the case, that extra branches or self-intersections occur that are not present in the parametric definition. This is, however, a feature of exact implicitization rather than a problem with any specific method. Further techniques such as approximation are required in order to remove them.

Some of these challenges were resolved in the subsequent literature, while some remain today. We will now describe existing approaches which attempt to avoid some of these problems.

Because of the complexity of computing determinants, finding more compact forms of the resultant matrices has been a large area of research. Methods where the dimension of the matrix is reduced and the degree of the entries is increased, were introduced in [64]. These techniques, known as moving curves and surfaces, also deal with the cases when base points occur, and in fact reduce in complexity in these cases. Further work in this direction has led to the definition of μ-bases [19, 20, 69]. We now briefly summarize the concept of μ-bases, which is given in more detail in [19]. A moving line $L(x, y; t) = 0$ is a one-parameter family of lines given by

$$L(x, y; t) = A(t)x + B(t)y + C(t),$$

for univariate polynomials A, B and C. Consider again the parametric curve (1.1). Any such curve can be shown to have a pair of moving lines $p(x, y; t)$ and $q(x, y; t)$ with the following properties:

1. p and q follow the curve (i.e., the point $\mathbf{p}(t_0)$ lies on both $p(x, y, t_0)$ and $q(x, y, t_0)$ for any t_0),

2. any other moving line r, which follows the curve can be expressed as a combination of $r = h_1(t)p + h_2(t)q$, for univariate polynomials h_1, h_2,

3. p and q have the lowest degrees of any pair of moving lines with properties 1 and 2.

Such a pair of moving lines is known as a μ-basis. A resultant of a μ-basis can be defined which gives the implicit equation of the curve. The reason for interest in μ-basis implicitization is that the resultant matrix is much smaller. In general, the matrix dimensions will be $(n-\mu)\times(n-\mu)$, where $0 \leq \mu \leq \lfloor n/2 \rfloor$. Potentially, this can result in matrices with 16 times fewer entries than

the Sylvester matrix. Moreover, efficient algorithms for μ-basis computation exist in the case of curves. Unfortunately, efficient methods for μ-basis computation of general freeform surfaces are still unavailable. There is a great deal of current interest in this research topic. μ-bases can also be used for solving other geometric problems such as locating singularities.

Another method to avoid the costly determinant computation is to keep the implicit representation in matrix form [16, 15]. In matrix based representations, a single matrix is defined (with entries in the implicit variables) whose rank drops exactly when evaluated on the curve or surface. For many problems in CAGD, such as surface intersection, these matrix representations can be used directly, without the need for symbolic expansion [13].

2.2 Numerical methods for implicitization

As the prospect of implicitization in CAGD became more realistic, additional methods began appearing in the literature. Closer cooperation between academia and industry was also formed [28]. Due to the majority industrial systems using floating point arithmetic, a major focus was on numerical methods for implicitization. The techniques that we discuss in this section have relations to both approximation and interpolation theory, as well as the classical theory from algebraic geometry.

Approximate Implicitization

In 1997 a new method for *approximate implicitization* was introduced in the doctoral thesis of Dokken [26]. Utilizing concepts that were widely used in CAGD, such as Bézier and Bernstein representations, the algorithms were well suited to numerical implementation. Moreover, guarantees for the numerical stability were proven. Dokken's method allows the degree of the implicit polynomial to be chosen, and the algorithm proceeds to find an algebraic approximation. Whilst being designed to provide approximations, the method also has the potential to provide exact implicitizations if the chosen degree is high enough and exact precision arithmetic is used. Moreover, the method exhibits very high convergence rates, justifying the suitability for approximate approaches to implicitization.

Methods for approximate implicitization look to find a low degree implicit representation which minimize the *algebraic error* to the parametric curve or surface. That is, to find a polynomial q, which minimizes the value of $|q \circ \mathbf{p}(t)|$ for t in some domain Ω (e.g., $\Omega = [0, 1]$ for Bézier curves). Minimizing the error in affine space is a computationally intractable problem, but the algebraic error provides a good approximation to the geometric error away from singularities [27]. The original approach defined an upper bound on the absolute algebraic error over the entire domain thus giving an approximation to a uniform minimization problem. An alternative approach, known as weak approximate implicitization, which minimized the least squares error was introduced later [22, 29]. Both approaches use the singular value decomposition (SVD) for the numerical approximation. The methods are very general, in the sense that they can implicitize or approximate any rational-parametric hypersurface, whether or not base points occur. The degree can be chosen freely, and constraints can be added to the approximation. It should be noted that finding the optimal choice of degree is a non-trivial problem. Since extensive introductions to approximate implicitization are provided in Papers I and II [9, 10] of

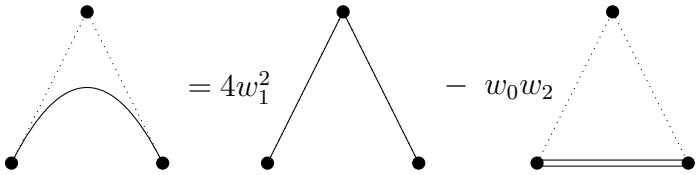


Figure 1.5: Implicit representation of quadratic Bézier curves $q(x, y) := 4w_1^2 L_{01}(x, y)L_{12}(x, y) - w_0 w_2 L_{02}(x, y)^2 = 0$.

this thesis, we omit further details here.

Alternative approaches to approximate implicitization have also been introduced. Wurm and Jüttler introduced a method in [42, 76], which enabled approximation of scattered data by implicit surfaces. Shen et. al. introduced the concept of approximate μ-basis, which led to a different approach to approximate implicitization [69]. Their work also has applications to the degree reduction of curves.

Due to many industrial CAGD problems not having exact solutions, approximation was already a well established and accepted paradigm in the community. There exist standards for acceptable tolerances in geometric models, which define when two points can be considered as one [28]. Thus, the potential for the practical realization of approximate algebraic methods in CAGD is great [41]. The previous EU project GAIA (2002-2005), confirmed feasibility of the methods by conducting experiments using industrial data [68, 78].

Extensions to the work on approximate implicitization, particularly Dokken's method, form a large part of this thesis.

Explicit methods for low degree curves

For low degree curves, explicit methods for implicitization are available. By making a judicious choice of basis (and/or coordinate system), simple expressions for the implicit polynomial coefficients can be found. The simplest case is of course that of a line, which has a trivial implicitization. The implicit representation of *any* non-degenerate conic section, defined in Bézier form as

$$\mathbf{p}(t) = \frac{\sum_{i=0}^{2} \mathbf{c}_i w_i B_i^2(t)}{\sum_{i=0}^{2} w_i B_i^2(t)},$$

is given by the following equation [34]:

$$4\tau_0\tau_2 w_1^2 - \tau_1^2 w_0 w_2 = 0,$$

where τ_0, τ_1 and τ_2 denote the barycentric coordinates defined by the triangle with vertices $\mathbf{c}_0, \mathbf{c}_1$ and $\mathbf{c}_2$. A diagrammatic representation of such a method is pictured in Figure 1.5. For a clarification of the meaning of Figure 1.5, we refer the reader to Paper IV.

How best to extend the above method to cubic and higher degree curves is an interesting problem. For rational cubic curves, one approach, described by Floater in [37], is to choose a Bernstein basis over the triangle defined by the points $\mathbf{c}_0$, $\mathbf{c}_3$ and the intersection point $\mathbf{p}$, of the lines $L_{01}(x, y) = 0$ and $L_{23}(x, y) = 0$ (see Figure 1.6). In terms of this basis, the coefficients of the implicit polynomial can be given by explicit formulas. In particular, four of the 10

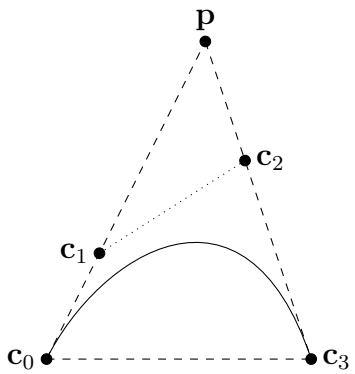


Figure 1.6: Floater's rational cubic implicitization method, $q(\tau_0, \tau_1, \tau_2) := \sum_{i+j+k=n} b_{ijk} B^3_{ijk}(\tau_0, \tau_1, \tau_2) = 0$, where b_{ijk} are defined explicitly. The barycentric coordinate system is defined by the points $\mathbf{c}_0$, $\mathbf{p}$ and $\mathbf{c}_3$ [37].

coefficients are immediately zero; thus, some sparsity in the implicit polynomial is exhibited. Another approach to the implicit representation of cubic curves is presented in Paper IV of this thesis. Floater's method suffers from problems when the tangent lines to the curve at the end points $\mathbf{c}_0$ and $\mathbf{c}_3$ are parallel, due to the point $\mathbf{p}$ being undefined. Also, if either of $\mathbf{c}_1$ or $\mathbf{c}_2$ lie on the line between $\mathbf{c}_0$ and $\mathbf{c}_3$, the coordinate system collapses. The method of Paper IV fails only in the collinear case.

There also exist explicit methods which associate a cubic curve with the projective image of one of three canonical curves [46]. These methods, go back to the nineteenth century [57], but have been utilized recently by Loop and Blinn [46] for graphics applications. They define cubic curves in terms of a single homogeneous implicit equation

$$k^3 - lmn = 0. \tag{1.2}$$

where k, l, m, n are the equations of lines associated with the double point and inflection points. Although for general rational cubic Bézier curves the equation is not trivial to define, there are certain computational advantages in using this method, due to the simplicity of the equation (1.2).

Interpolation and approximation of point data

The theory of interpolation with implicit curves and surfaces, has some interesting and non-trivial features. Using Lagrange interpolation as a curve implicitization method can essentially proceed in one of two ways:

The first method, which we say uses on-curve data, looks for a non-trivial polynomial which vanishes at sufficiently many points along the parametric curve. That is, for a parametric curve $\mathbf{p}$, and sample parameters $(t_i)_{i=0}^N$, we find a polynomial $q : \mathbb{R}^2 \to \mathbb{R}$ such that $q(\mathbf{p}(t_i)) = 0$ for all $i = 0, \ldots, N$. The number of nodes N, required for exact implicitization, is discussed in Paper II.

The second, which we say uses off-curve data, is the method described by Marco and Martinez [50, 51]. In their approach, resultants are computed numerically at sufficiently many

sample points $(x_i, y_j)_{i,j=0,0}^{n_1,n_2}$ in order to generate enough data to solve a multivariate Lagrange interpolation problem. Essentially, the data generated defines the implicit polynomial in a Lagrange basis and the method proceeds to generate the monomial coefficients by using a basis transformation matrix (i.e., a Vandermonde matrix). Computing the determinant of a numerical matrix is much less computationally intensive than that of a symbolic matrix. One reason for this is that for numerical matrices, LU-decomposition is possible, where the matrix is decomposed into the product of a lower and an upper triangular matrix. The determinant is then given by the product of the entries on the diagonals of the matrices. It may be noted that the approach of Marco and Martinez can be used with any type of resultant matrix.

For a given curve, an example of the two methods is shown in Figure 1.7. Using the on-curve method, the data is all known to be zero (since it lies on the curve) but we evaluate the parametric curve at nodes in the parameter domain, in order to find the interpolation nodes in $\mathbb{R}^2$. The implicit coefficients are then given by the solution to a homogeneous linear system. In the off-curve method, the nodes are predetermined (e.g., on a uniform grid), but we have to evaluate the resultant in order to generate data for the interpolation. Both methods are also suitable for the implicitization of general hypersurfaces, although the off-curve method relies on suitable resultants being defined.

We discuss some aspects of on-curve interpolation and approximation in Paper II, and its connection with Dokken's approximate implicitization. In particular, for exact implicitization, care needs to be taken to ensure that the interpolation points lie in general position, so that the unique implicit polynomial can be found. It is well known that any five points in the plane, with no four collinear, determine a conic. This, is due to the fact that there are five degrees of freedom in the implicit representation of a conic (in fact, there are six coefficients, but since implicit representations are unchanged by scalar multiplication, one degree of freedom is removed). The equivalent statement for cubics is complicated by the famous Cayley-Bacharach theorem from algebraic geometry [30].

Theorem 1 (Cayley-Bacharach theorem)**.** *Consider the nine points in which two implicit planar cubic curves, I_1 and I_2, intersect. Then any cubic curve I_3 which passes through any eight of the points, also passes through the ninth.*

When sampling from a rational parametric curve, problems with finding points in general position can be avoided by a small oversampling [10]. This is the reason we use 10 interpolation points, rather than nine, in the example of Figure 1.7.

Both methods can also be used for approximate implicitization. In general, the on-curve method provides better approximations than the off-curve method. In fact, the on-curve method, which performs the approximation in the univariate parameter domain, can be shown to exhibit the same high convergence rates as Dokken's method. For the off-curve method, the approximation would be made over the entire 2D triangular domain of the basis functions. Thus, the error should be expected to be greater in this case. In Paper II, we also show how the Lebesgue constant from approximation theory is important in judging the quality of implicit approximations.

Hermite interpolation using implicit surfaces has also been investigated [4, 6]. These methods attempt to interpolate both position and derivative data, in order to obtain C_1-continuous curves or surfaces. Such requirements can also be met using approximate implicitization, by

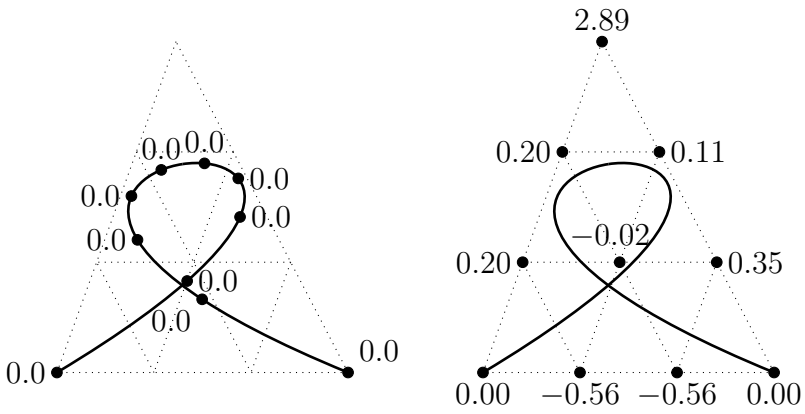


Figure 1.7: Interpolation using parameter-uniform on-curve data (left) and uniform off-curve data (right). The basis for the implicit polynomial is a Bernstein basis over barycentric coordinates of a bounding triangle.

adding constraints to the approximation [26]. The constraints can be added before or after approximation, the former via Lagrangian multipliers, and the latter by combining lower order approximations [29, 73].

2.3 Alternative implicitization methods

Sparse implicitization

The high convergence rates of Dokken's approach to approximate implicitization are due to there being many degrees of freedom in the implicit polynomial. This was highlighted in the proofs of the convergence rates given in [26, 29]. The number of degrees of freedom in a rational parametric planar curve of degree n is much less than the number of degrees of freedom in an implicit polynomial of total degree n (given by $\binom{n+2}{2} - 1$). In general, there will exist a basis for the implicit description where many of the coefficients are zero. If it is known, a priori, which of these coefficients are zero, the complexity of the implicitization problem can be reduced; in some cases substantially [32].

In [32], techniques were introduced which exploited sparsity in the monomial basis representation of the implicit equations. In order to predict the support of the implicit function (i.e., the set of monomials which have non-zero coefficients in the implicit polynomial), the structure of the parametric functions can be analysed. The techniques refer back to the literature on toric elimination theory and utilize the concept of the Newton polygon. Support prediction methods are also well suited for implementation alongside approximate implicitization [10, 31]. Whereas in approximate implicitization we choose to reduce the total degree, techniques using support prediction can remove any other monomials from the basis, even those which are not necessarily of high degree.

However, challenges still remain in integrating the methods for data from industrial systems. One of the main problems with using sparsity in practice is that freeform curves and surfaces do not, in general, exhibit a high degree of sparsity in the monomial basis. Also, since current algorithms only work in the monomial basis, basis transformations are required, which have

potentially ill conditioned behaviour. Using a change of coordinates can introduce a certain level of sparsity, but how best to achieve this appears to be an open problem.

Implicitization of envelope curves

Recently, there has also been an increase in interest of implicitization of envelope curves and surfaces, which have applications in robotics [56, 59]. Consider a rational family of rational curves defined by

$$\mathbf{p}(s,t) = \left(\frac{f(s,t)}{h(s,t)}, \frac{g(s,t)}{h(s,t)} \right), \quad (s,t) \in [s_0, s_1] \times [t_0, t_1].$$

Either s or t can be thought of as a time-like parameter and the remaining parameter (either t or s, respectively), parameterizes the curves. The envelope is a curve which touches all members of the family $\mathbf{p}$. It corresponds to the parameter values (s,t), where the Jacobian becomes singular [59]. A polynomial function $h(s,t)$, called the envelope function, can be defined whose zero set corresponds exactly to such points. In general, the envelope does not have a simple parametrization so methods for implicitization of parametric curves are not relevant. Recently, Schulz and Jüttler showed that approximate implicitization can be adapted to envelope curves [59]. They showed that if q is the exact implicit representation of the envelope, then

$$q(\mathbf{p}(s,t)) = \lambda(s,t) h(s,t)^2$$

for some unknown function λ. Thus, finding polynomials q and λ which approximate this equation gives methods for approximating the envelope.

A distinctive feature of envelope curves, like surfaces, is the high degrees which occur [25]. Envelopes are also of interest in higher dimensions, where the problem of high degrees is exacerbated even further. Approximate implicitization is therefore an important tool in working with such manifolds.

Recent advances

Among the aims of the EU project SAGA, is to conduct further research into methods for change of representation. There are several considerations that should be taken into account in the development of implicitization algorithms for modern CAD systems, and that have guided the work in this thesis. These include:

- The algorithms should be suitable for floating point implementation, in order to obtain satisfactory performance. This requires algorithms that are numerically stable, and avoid reliance on symbolic computation. For ease of implementation, the methods should be based on numerical algorithms which are available on a number of platforms (e.g., fast Fourier transform (FFT), singular value decomposition (SVD)).

- It is generally required that the performance of the algorithms scales up with improvements in modern hardware. Since the modern trend in improved computing performance is through parallelizability (i.e., implementation on GPUs), algorithms that are parallel in nature are desired.

- The methods should be robust, in the sense that they do not fail for certain examples. In particular, all freeform NURBs based curves and surfaces should be implicitizable, whether or not they exhibit base points. In addition, there should be the potential for implicitization hypersurfaces in higher dimensions.
- The methods should have the potential for exact results, if implemented in the correct way. They should also be able to produce approximations for cases where the exact degrees are considered too high.

All methods presented in Part II of this thesis are suitable for numerical implementation and do not rely on symbolic computations. We discuss, where relevant, how parallelism can be exploited and have implemented this in some cases. Prototype implementations have been tested using several programming languages including Python, C++ and CUDA.

Chapter 2

Summary of papers

In this section we give a summary of the main contributions of the papers in Part II. We discuss the concepts within the context of the thesis, and give some supplementary details to aid the intuition behind the approaches. For each paper we also provide some comments on the work with the benefits of hindsight, discussing potential improvements and directions for future research.

Paper I: Approximate Implicitization of Triangular Bézier Surfaces

This paper serves as an introduction to the theory of approximate implicitization, with a new application: triangular Bézier surfaces. We revisit the existing theory of Dokken's original and weak approaches to approximate implicitization [26, 29]. While Dokken's original method is directly extensible to tensor-product domains, the extension to triangular (and more generally simplex) domains requires a fuller formulation and discussion.

Triangular Bézier surfaces have various applications in areas such as computer graphics and CAD, amongst others. Despite tensor-product surfaces being favoured in many systems, rational triangular surfaces have the benefit that they can represent quadrics exactly, and without singular parametrizations. The problem of implicitization of triangular surfaces is thus a natural one. Being based on the properties of Bernstein polynomials, the methods we discuss in this paper are numerically stable. This contrasts with several previous methods which use potentially ill-conditioned basis transformations in order to perform the implicitization in the monomial basis [35].

In general, a rational parametric triangular surface of degree n will have implicit degree n^2 [40]. Thus, even for moderate degree parametric surfaces, the implicit degree increases rapidly. Moreover, the number of degrees of freedom in an exact implicit representation grows on the order of $O(n^6)$, justifying approximation in all but the lowest degree cases.

In the paper, both the original and weak forms of approximate implicitization are described in detail. We highlight symmetries in the algorithm that considerably reduce the number of integrals required from $O(m^6)$ to $O(m^3)$, where m is the implicit degree of the approximation. Due to the arrangement of the algorithm, this is most useful when numerical integration is used. We also present a hybrid method, whereby we simultaneously approximate tensor-product and

triangular patches with the same algebraic surface.

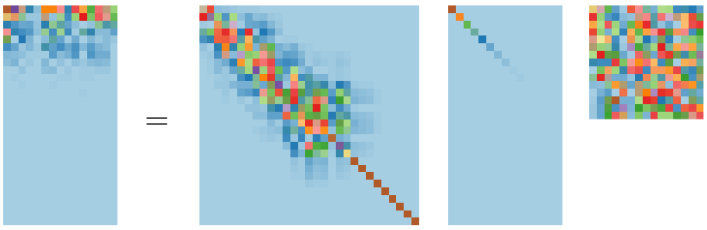
Comments

This paper was written prior to the more general approach of Paper II [10]. Some of the methods in that paper can also be generalized to triangular Bézier surfaces. In particular, the result on Lebesgue constants from [10] is of interest, since good point sets on triangular, and more generally simplex domains, are known [75]. In addition, an orthogonal basis method using Legendre polynomials is well suited for implementation on triangular domains by exploiting Farouki's method for the construction of orthogonal bases using Bernstein polynomials [36]. Such a method may be expected to benefit from both the stability of the Bernstein basis (for high degree implicitization) and the approximation power of the Legendre basis (for lower degree approximation).

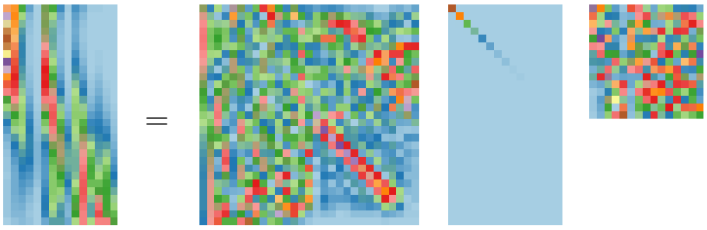
Paper II: Approximate implicitization using linear algebra

In this paper we extend the theory of approximate implicitization, starting from the original method of Dokken [26, 27] and the more general weak approach, which uses integration [22, 29]. The method of approximate implicitization using orthogonal bases is proposed and shown, both theoretically and experimentally, to have better approximation properties than the original method. Whilst giving theoretically the same results as the weak approach, the method is also shown to exhibit much better numerical conditioning. The method therefore provides a good balance between the superior numerical properties of Dokken's method, and the approximation power of the weak approach. We also investigate the use of Lagrange bases in approximate implicitization, using the Lebesgue constant to present new results which verify the quality of approximation for well chosen point distributions. Since sets of points with small Lebesgue constant are known on a wide variety of domains, both in the univariate and multivariate cases, this result has great generality. The paper concludes with a discussion and examples which show the approximation power of the method, even at low degrees.

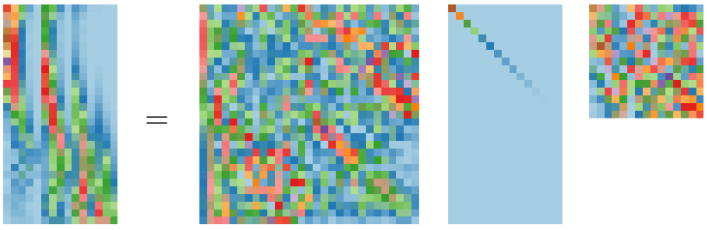
The method of Corless et. al. [22], which is essentially similar to the weak approach of Dokken [27, 29], is the implicitization method of choice in the computer algebra system Maple [49]. Although the method is very general, in the sense that it can implicitize or approximate many different curve and surface parametrizations, it suffers from a lack of numerical stability. We present examples which show that this approach is ill-conditioned, losing several digits of accuracy even for low degree curves, when numerical calculations are used. The Maple documentation proposes increasing the floating point accuracy prior to computation in order to gain sufficient accuracy. While this gives satisfactory results for low degrees, it is computationally more expensive. We propose methods in this paper which preserve the accuracy much better, without resorting to increasing the number of floating point digits used. The methods are therefore faster, and more suitable for implementation in systems that rely on double precision floating point arithmetic. They do, however, rely on the possibility to express multiples of the parameter functions in an orthogonal basis. In the case of rational parametrizations, this is always possible.



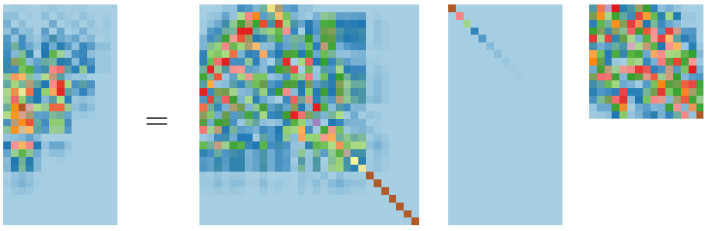
(a) $\mathbf{D}_C = \mathbf{U}_C \mathbf{\Sigma}_C \mathbf{V}_C^T$ for the Chebyshev basis



(b) $\mathbf{D}_L = \mathbf{U}_L \mathbf{\Sigma}_L \mathbf{V}_L^T$ for the Lagrange basis



(c) $\mathbf{D}_B = \mathbf{U}_B \mathbf{\Sigma}_B \mathbf{V}_B^T$ for the Bernstein basis



(d) $\mathbf{D}_M = \mathbf{U}_M \mathbf{\Sigma}_M \mathbf{V}_M^T$ for the monomial basis

Figure 2.1: Implicitization matrices (left) and their singular value decompositions, highlighting the structures for the various polynomial bases. The light blue elements are zero or close to zero.

We make a comparison of four implementations of approximate implicitization of rational curves, which come under the name of different polynomial basis functions. Several of the properties discussed in the paper can be summarized in Figure 2.1. For example, it is clear that using the Chebyshev method, the numbers in the lower rows of the matrix become very small, and thus contribute less to the approximation. Moreover, the approximate diagonality of the left singular matrix shows that the error is approximately equioscillating, since the diagonal elements correspond to Chebyshev basis functions.

When comparing the singular value matrices $\mathbf{\Sigma}_i$ for each basis, it is clear that the Bernstein basis has a more even distribution of singular values. This leads to better numerical stability (c.f., [58]). We see also that the Lagrange and Bernstein methods have very similar numerical structures in the matrices. This is a consequence of them solving approximately the same problem, as proven in Proposition 3, Proposition 5 and Corollary 6 in the paper. As the degree is raised, the similarities between the methods become more clear.

To conclude the paper we present an example of approximate surface implicitization using the Chebyshev basis. We implicitize each of the 32 surface patches of the famous teapot model [64], which exhibit exact algebraic degrees up to 18. It is shown that by choosing approximation degrees roughly one third of the exact degree, excellent approximations can be attained. This is a result of the high convergence rates proven in [26].

Comments

This paper highlighted the fact that the smallest singular values are highly dependent on the choice of basis. Thus, while giving an upper bound on the algebraic error, the inequality

$$|q \circ \mathbf{p}(t)| \leq \max_{t\in\Omega} \|\boldsymbol{\alpha}(t)\|_2 \sigma_{\min}$$

for a given basis $\boldsymbol{\alpha}(t) = (\alpha_i(t))_{i=1}^{L}$, gives little insight into how different bases compare. One attempt to gain such insight may be to use basis transformation matrices. Any polynomial basis $\boldsymbol{\alpha}_1(t)$ can be transformed to another basis of the same degree, via a matrix transformation (i.e., $\boldsymbol{\alpha}_2(t) = \mathbf{T}\boldsymbol{\alpha}_1(t)$, for bases $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2$ and a constant matrix $\mathbf{T}$). For example, if we obtain an approximation q_C via the Chebyshev method, and the transformation matrix between the Chebyshev and Bernstein bases is denoted $\mathbf{T}$, we can say

$$|q_C \circ \mathbf{p}(t)| \leq \tau_{\max}\sigma_{\min}$$

where $\sigma_{\min}$ is the smallest singular value of the Bernstein matrix, and $\tau_{\max}$ is the largest singular values of $\mathbf{T}$. However, because we must use $\tau_{\max}$, this upper bound may not be improved, even if the approximation is improved. Future research, could attempt to understand better the singular value dependence on the bases.

Paper III: Fast approximate implicitization of envelope curves using Chebyshev polynomials

In this paper we combine recent methods presented in [10] and [59] for the efficient approximate implicitization of *envelope curves*. Envelopes of rational families comprise an important class of curves and surfaces, which include offsets, canal surfaces and cyclides [25]. They have applications in robotics (defining boundaries and collision detection), gearing (in order to match pairs of tooth flanks) and CAD (defining offsets) [59]. In this paper we consider the envelope of a rational family of rational planar curves

$$\mathbf{p}(s,t) = (x(s,t)/w(s,t), y(s,t)/w(s,t))^T, \quad (s,t) \in [0,1] \times [0,1].$$

The envelope approximation is based on minimization of a coupled objective function

$$\min_{\|\mathbf{c}\|_2=1} \int_{I\times J} \omega(s,t) \left((q \circ \mathbf{p})(s,t) w(s,t)^d - \lambda(s,t) h(s,t)^2\right)^2 \mathrm{d}(s,t),$$

where q (the implicit polynomial) and λ (the coupling function) are unknown, h is the envelope function and ω the Chebyshev weight function. The envelope function h is defined by the Jacobian determinant of the mapping $\mathbf{p}$. The unknown coefficients of q and λ are contained in the vector $\mathbf{c}$ and are minimized simultaneously. This method is related, yet inherently different to weak approximate implicitization of parametric curves [29].

The original implementation of approximate implicitization of envelope curves provided a proof of concept [59]. However, the implementation involved symbolic expansions, which become very cumbersome for all but the lowest degrees. One intention of this paper was to provide a purely numerical algorithm to give increased performance. In order to achieve this we chose to adapt the Chebyshev method from [10] to envelope computation. This method was chosen for both its superior computational and approximation properties. In addition, the method has better numerical condition for the same reasons as given in [10].

We provide full algorithmic details of the implementation using tensor-product Chebyshev polynomials. The algorithm utilizes the discrete cosine transform (implemented via FFT), which gives a fast method for weighted least squares minimization of the objective function.

Whilst making approximation of envelopes possible, the introduction of the coupling function λ complicates certain theoretical investigations. For example, a major advantage of approximate implicitization of parametric curves has been the high convergence rates, which were proven theoretically by Dokken [26, 27, 29]. However, a theoretical proof of convergence rates for envelope curves, appears to be much more complicated. In order to generate some data on convergence, we studied the convergence rates numerically. The results show that in the simple case of rational envelopes, convergence rates of the same magnitude as Dokken's were attainable. Investigations of higher degree envelope approximations also appeared to give good results, but were not included in the paper due to space limitations.

Comments

The suitability for implementation on heterogeneous architectures is mentioned in the paper. With regard to this, a simple implementation using the GPU programming language CUDA has been attempted. Whilst giving minor improvements to the speed of computation of the matrix, in the case of low degree curves, there is an insufficient level of parallelism to obtain significantly better results than with a CPU. On the other hand, high degree implicitization requires more numerical accuracy than single precision arithmetic, which has been the standard for GPUs until recently.

The approach in this paper is very general and can be applied to tensor-product domains in any dimension. Some applications in robotics call for envelope computation in higher dimensions (i.e., surface envelopes and envelopes of k-parameter families of curves, $k > 1$). Typically, implicitization in high dimensions is a hugely computationally intensive task. However, as the dimension is increased, the level of intrinsic parallelism in the algorithm is also vastly increased. As a direction for future research, it would be interesting to compare how a heterogeneous/GPU implementation of the algorithm for higher dimensional envelope computation compares to a CPU-only implementation. It may be the case that the high level parallelism opens the door to implicitization of hypersurfaces that was not previously realizable.

Another area for future research could be to renew efforts for theoretical work on the convergence rates. The promising results presented in this paper give justification for attempting theoretical proofs, at least in the case of rational envelope curves.

Paper IV: A basis for the implicit representation of rational cubic Bézier curves

As mentioned in the previous chapter, one method to reduce the computational complexity of implicitization is to compute the support of the implicit polynomial; that is, to compute the monomials in the implicit equation which have non-zero coefficients. If many of the monomial coefficients are zero, the support is said to be sparse. Unfortunately, for general freeform curves in floating point arithmetic, the implicit polynomial in monomial form often has full support (i.e., no sparsity). One of the motivations for this paper was to find a basis which in some sense 'follows' the curve, thereby preserving sparsity. Such bases have been defined previously for quadratic and cubic curves [34, 37]. The cubic implicitization method of Floater [37], was described in Chapter 1. In this paper we take a similar approach, but define only four implicit basis functions, which can implicitize all rational cubic Bézier curves except those with collinear control points.

In addition to the apparent sparsity of the method, the coefficients have very simple forms, and can be given by explicit formulas. These formulas are based only on the weights, and relations between the control points of the curve. They are thus independent of the coordinate system and invariant under affine transformations. Moreover, certain factors of the coefficients are shown to be fundamental to the geometry of the curve. These quantities, which we denote Φ_1, Φ_2 and Φ_3, are simple to calculate and give information on the curve simply by observation. Some of the main results include the following:

- The curve degenerates to a conic section if and only if $\Phi_1 = \Phi_2 = \Phi_3 = 0$.
- The curve has an unwanted self-intersection if and only if $\Phi_1\Phi_2 < 0$.
- The parameters of the double point are given by the roots of

$$r(t) = \Phi_1 t^2 + \Phi_3 t(1-t) + \Phi_2(1-t)^2,$$

  which can be written explicitly as

$$\frac{2\Phi_2 - \Phi_3 \pm \sqrt{\Phi_3 - 4\Phi_1\Phi_2}}{2(\Phi_1 + \Phi_2 - \Phi_3)}.$$

- The discriminant $\Delta = \Phi_3^2 - 4\Phi_1\Phi_2$ determines whether the curve has a self-intersection ($\Delta > 0$), a cusp ($\Delta = 0$), or an acnode ($\Delta < 0$).

In addition, we give a barycentric formula for the double point of the curve in terms of its control points. This formula, or slight variations thereof, is valid in all cases, even when the curve has collinear control points.

We also consider some algorithmic aspects of how cubic curves can be rendered via the implicit representation. In particular, we define two lines $\tilde{S}_1$ and $\tilde{S}_2$, which, in the case of unwanted self-intersections, can be used to visually eliminate the singularity. In the case that the coefficients become zero (which indicates conic degeneration) another formula for the implicit representation is presented.

Comments

One of the main disadvantages of the method is that it fails when any three points are collinear. For floating point implementations the representation becomes less numerically stable close to collinear configurations[1]. Defining additional basis functions to overcome this appears to complicate the method somewhat, and the structure of the coefficient formulas appears to be lost. Currently, in collinear cases, we resort to subdivision. Although this sufficiently solves the problem, more direct methods would be of interest.

In addition to stability problems with collinear control points, the method is unstable near conic degenerations. However, the basis functions still provide support for representation of the conic section, under a different set of coefficients. It should be investigated whether a smooth transition between the cubic and conic coefficients is possible.

Inflection points are an important feature of rational cubic curves, which were not considered in the paper. It should be investigated whether formulas for inflection points are possible within the simple notational framework of this paper.

The basis functions described using this method give some geometric intuition into the construction of implicit representations of cubic curves. In a sense, the method can be thought of as using a change of coordinates, but it is not clear exactly how this occurs. It would be interesting to find out whether the method could be expressed compactly in terms of some form

[1]Experimentally it appears that the instabilities do not cause a problem until the control points are collinear within a tolerance on the order of machine epsilon.

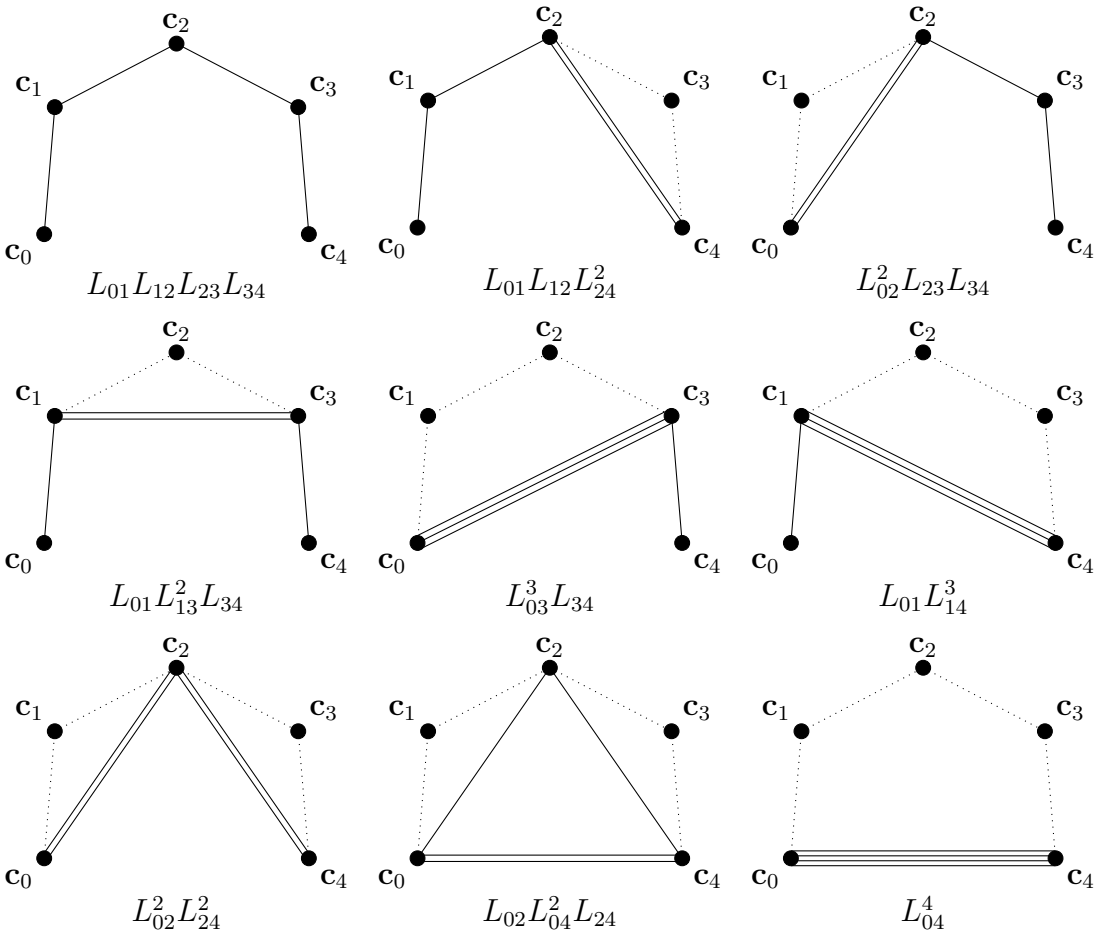


Figure 2.2: A diagrammatic representation of the zero sets of nine basis functions, which appear to support the implicit representation of rational quartic Bézier curves, when no three control points are collinear. The number of lines between any two points $\mathbf{c}_i$ and $\mathbf{c}_j$, reflect the multiplicity with which $L_{ij}(x, y)$ appears in the basis function.

of generalized barycentric coordinates. For example, quadrilateral coordinates, Wachspress coordinates and mean values coordinates can all be used to treat cases with four points.

Although coming from a completely different approach, there appears to be some connection with μ-bases and resultants. In particular, elementary operations on the resultant matrices can also be used to define the function $r(t)$ [13]. Moreover, those methods are general and work for any degree. It would be of great interest to investigate further connections between μ-basis methods and those presented in this paper. Also, extending the method to higher degrees would be of interest. Initial work on this has been attempted, and experimentally, it appears the basis functions in Figure 2.2 support the implicit equations of all rational quartic Bézier curves, with no three control points collinear.

Chapter 3

Applications of methods for change of representation

The intention of this section is to highlight both new and existing applications of the methods described in the scientific papers of Part II of this thesis. We consider three main applications within CAD intersection algorithms, visualization and robotics. We also discuss some open problems that have arisen by using the methods in practice. Since this section utilizes concepts from the scientific papers of this thesis, it may be that the reader wishes to gain familiarity with those concepts before proceeding with this section.

1 Intersection algorithms

Intersection algorithms in CAD were the principal motivation behind the development of approximate implicitization [26]. Whereas transversal intersections can be successfully approximated using subdivision schemes [65], tangential (and near tangential) intersections pose a more difficult problem. For such cases, the use of approximate implicitization can be helpful [27, 28, 70]. Algorithms for surface trimming are also related to intersection algorithms. We describe how the method from Paper IV can be useful in computing surface trimming regions.

1.1 Self-intersections

In this section we discuss self-intersections of curves and surfaces. We are only interested in single rational curves or surfaces, as opposed to piecewise NURBS self-intersections, which are more closely related to the problem of general intersection, covered in Section 1.2.

There exist several methods for computing self-intersections of rational curves and surfaces, including [14, 15, 54, 74]. In this section we mainly focus on the methods in [74], which are well suited to approximate implicitization.

Self-intersection of planar rational curves

For an implicitly defined curve q, we refer to points where

$$q(x, y) = 0 \quad \text{and} \quad \nabla q(x, y) = (0, 0)$$

as singularities of the curve. All self-intersections of algebraic curves are singularities, but the converse is not necessarily true. Singularities can also occur as cusps and isolated points known as acnodes.

The simplest examples of self-intersections are for low degree curves. Clearly, in $\mathbb{R}^2$, a line will never intersect itself. Conic sections will also never exhibit singularities unless they can be decomposed into two lines (i.e., the implicit polynomial is reducible). For rational cubic curves with real coefficients, there always exists a real singularity, in the form of a self-intersection (crunode), cusp or acnode[1] [63]. In Paper IV we present an explicit formula for the location of the singularity of a cubic curve in terms of a barycentric combination of its control points [8]. We also give very simple conditions for when the singularity defines a self-intersection, and in particular, an *unwanted self-intersection*. The parameter values of the singularity are also stated explicitly as the roots of a quadratic polynomial $r(t)$.

For most higher degree curves explicit formulas are not generally available and we often resort to numerical rootfinding. In [73], a method for computing self-intersections using approximate implicitization is presented. For a given parametric curve $\mathbf{p}(t)$, its normal $\mathbf{n}(t)$ and its (approximate) implicit representation q, the roots of the polynomial

$$\nabla q(\mathbf{p}(t)) \cdot \mathbf{n}(t), \tag{3.1}$$

are shown to give candidates for self-intersections. It is normally possible to find exact implicit representations of rational parametric curves, since the degrees involved are not excessively high. However, this polynomial is not of the lowest possible degree, and will contain extra factors. In the case of cubic curves, for example, it has degree six, much higher than the desired degree of two. In the general case of rational curves of degree n, we want to find a polynomial $r_n(t)$ of degree $(n-1)(n-2)$. There exist alternative methods to find the polynomial $r_n(t)$ using resultants, which work for any degree [14, 21]. However, these methods are better suited to symbolic computation. It would be of substantial practical interest to extend the method of Paper IV to higher degrees, as this would provide a method more suitable for numerical implementation, thereby aiding applicability in CAD systems.

Self-intersections of surfaces in $\mathbb{R}^3$

For surfaces, similar methods for computing self-intersections can be used; however, they are generally more complex. The complexity is reflected in the high degrees that appear. The method given in [74] suggests computing the zeros of the bivariate polynomial

$$\nabla q(\mathbf{p}(s,t)) \cdot \mathbf{n}(s,t), \quad \mathbf{n}(s,t) = \partial_s \mathbf{p}(s,t) \times \partial_t \mathbf{p}(s,t), \tag{3.2}$$

in order to find candidates for self-intersections. However, as mentioned previously, exact implicitization is often not computationally feasible for surfaces. Thomassen proposes the use of approximate implicitization for use in computing self-intersections. As shown previously by Dokken [26, 27, 29] and in Papers I-III of this thesis [9, 10, 11], excellent approximations are possible at vastly lower degrees than the exact degree. Thus, in the case of surfaces,

[1] For implicit curves of degree three and higher there also exist non-singular curves, which do not have rational parametrizations (e.g., elliptic curves).

Thomassen's approach to computing self-intersections will give excellent numerical results. In addition, reducing the degree of the implicit polynomial often removes unwanted branches, such as those that can occur in the polynomial (3.2).

1.2 Curve and surface intersection via algebraic substitution

The problem of intersection between two general freeform parametric curves or surfaces is generally much more complex than self-intersections. For example, whereas a planar cubic curve can only intersect itself once, two cubic curves can intersect in up to nine points in the plane. This can be generalized to higher degrees by the classical result of Bézout's theorem [23]:

Theorem 2 (Bézout's theorem)**.** *Let $q_1(x, y) : \mathbb{R}^2 \to \mathbb{R}$ and $q_2(x, y) : \mathbb{R}^2 \to \mathbb{R}$ be irreducible polynomials of total degree m_1 and m_2 respectively, which define two algebraic curves. Then the total number of intersections between the curves (including complex intersections, intersections of higher multiplicity than one, and intersections at infinity) is given by $m_1 m_2$.*

Again, there are many methods for computing general intersections including both algebraic and subdivision approaches [28, 65]. We do not provide a review of the methods here, but refer the reader to the survey articles [28, 70] and references cited therein. We note that while subdivision methods perform well and give stable results for transversal intersections, tangential intersections pose more of a problem. In such cases, use of the implicit or approximate implicit representations can be useful.

General substitution method for intersection

For the sake of generality, in this section we outline the method for surface/surface intersections - the case of curves is similar. This method utilizes both implicit and parametric representations for computing intersections [26, 65]. Suppose we have two rational parametric surfaces $\mathbf{p}_1(s, t) : \Omega_1 \to \mathbb{R}^3$ and $\mathbf{p}_2(u, v) : \Omega_2^2 \to \mathbb{R}^3$ for $\Omega_1, \Omega_2 \subset \mathbb{R}^2$. Suppose we also have respective implicit representations $q_1(x, y, z) : \mathbb{R}^3 \to \mathbb{R}$ and $q_2(x, y, z) : \mathbb{R}^3 \to \mathbb{R}$. In computing the intersection, we are interested in finding the parameter values $(s, t) \in \Omega_1$ and $(u, v) \in \Omega_2$ such that $\mathbf{p}_1(s, t) = \mathbf{p}_2(u, v)$. We are also interested in the set

$$P = \mathbf{x} \in \mathbb{R}^3 : \mathbf{p}_1(s, t) = \mathbf{p}_2(u, v)$$

If we were to deal only with the parametric surfaces, we need to deal with four parameters s, t, u and v. Using both parametric and implicit representations, this can be reduced to two. The (s, t)-preimage of the intersection is defined as

$$S = \{(s, t) \in \Omega_1 : q_2(\mathbf{p}_1(s, t)) = 0\}.$$

A subset of the points $(s, t) \in S$ is normally computed numerically. For each point (s_i, t_i) thus computed, it must be checked that the (u, v)-preimage of the mapped point $\mathbf{p}_1(s_i, t_i)$ is in the domain Ω_2. Sederberg and Parry suggest the use of inversion formulas for this purpose [65]. It

is also possible to use the reverse procedure of computing

$$U = \{(u, v) \in \Omega_2 : q_1(\mathbf{p}_2(s, t)) = 0\}.$$

However, this approach will be slow since each pair of points in the two domains would need to be compared [74].

It is also possible to use approximate implicit representations in place of q_1 and q_2, in a direct analogue of the above procedure. This was proposed in Dokken's thesis [26]. The improved approximations and computational times given by the methods for approximate implicitization in Papers I and II, will provide better results than previous implementations.

One aspect of computing intersections, which is not taken into account when using approximate implicitization, is that of topological consistency. This is concerned with the intersection results having compatible definitions in the different domains. According to [71], topological consistency requires that the representations of the intersection in the two parameter domains and the representation in $\mathbb{R}^3$, should all correspond to the same curve. In general, the parameter domain preimage of surface intersections (which can be either points or curves) will not be rational. Song et. al. propose a linear perturbation method, whereby the parametric surfaces are altered slightly in order to force the intersection curve to be rational [71]. In general, *approximate* implicit methods will not give topologically consistent results. However, this would be an interesting direction for future research.

1.3 Surface trimming

Although tensor-product NURBS surfaces are the predominant surface representation in CAGD, often an additional operation is used to bound the region in which a NURBS surface is defined. This process, known as surface trimming, defines regions of the parameter domain which correspond either to valid or invalid surface points. The curves in the parameter domain are often expressed as densely sampled piecewise linear curves [34]. One reason for using piecewise linear curve data is that the problem of computing whether the point is considered valid or invalid is simplified, by counting ray intersections. However, such methods have limited accuracy.

The data for defining trimming curves may be generated, for example, from the intersection of two surfaces. This may be in the form of points in the parameter domain that can be interpolated or approximated, and may also include derivative data. For example, a common approach is that of cubic Hermite interpolation, where we find a cubic curve $\mathbf{p}$, such that

$$\mathbf{p}(0) = \mathbf{p}_0,\ \mathbf{p}(1) = \mathbf{p}_1,\ \mathbf{p}'(0) = \mathbf{m}_0,\ \text{and}\ \mathbf{p}'(1) = \mathbf{m}_1,$$

for given data $\mathbf{p}_0$, $\mathbf{p}_1$, $\mathbf{m}_0$ and $\mathbf{m}_1$. The Bézier form of cubic curves is particularly useful for this task since interpolation and tangency constraints at the endpoints are given naturally by the control polygon [34]. In fact, the control points of such a polynomial cubic curve can be given explicitly as

$$\mathbf{c}_0 = \mathbf{p}_0,\ \mathbf{c}_1 = \mathbf{p}_0 + \frac{\mathbf{m}_0}{3},\ \mathbf{c}_2 = \mathbf{p}_1 + \frac{\mathbf{m}_1}{3}\ \text{and}\ \mathbf{c}_3 = \mathbf{p}_0.$$

Because the parametric form of cubic curves is not optimal for deciding which points lie inside or outside a curve, it is not a good choice for trimming curves. However, since we are given

the control points and weights of the curve, the method of Paper IV can be used directly (i.e., no computationally expensive implicitization methods are necessary). Hence, the integration with existing cubic Hermite interpolation schemes is immediate, and will result in a piecewise implicit cubic representation. Moreover, it will provide a robust method for deciding whether or not the point lies within the trimming curve.

For implicitly representing piecewise curves in this way, some post-processing will be required in order to define the correct domains for each segment. Typically, restricting each segment to the convex hull of its control polygon will suffice. However, we must also define what happens outside the union of the convex hulls. One approach could be to use a Delaunay triangularization, similar to the method used in [46] for curve rendering.

2 Rendering of curves and surfaces

Traditionally, the parametric form of B-spline and Bézier curves and surfaces has been used for rendering, since it is very easy to evaluate points lying on the curve or surface. Recently, real-time rendering in the implicit form has generated increased attention, particularly using ray casting and ray tracing. Such implementations rely heavily on the efficiency of modern GPU hardware, which can process per-pixel computations using highly parallel architectures. Since the methods compute on a per-pixel basis, a major advantage is that the resolution is independent of the zoom level or the viewpoint. Resolution independent methods have also been applied to curve rendering.

2.1 Implicit rendering of rational cubic curves

For applications which involve rendering regions bounded by curves, such as font shading, the implicit form of rational cubic and conic curves is important [46, 55]. In [46], conic and cubic curves are evaluated implicitly by projecting one of a set of canonical curves onto screen space. This results in highly efficient computations, with very simple fragment shaders. The technique used in [55] is to employ the implicitization method of [37], along with subdivisions to obtain a simple Bézier arch. In this section, we describe how the method of Paper IV can be used for rendering cubic curves.

It is shown in Paper IV that any rational planar cubic Bézier curve with no three control points collinear, can be written implicitly in terms of four basis functions $(K_i(x,y))_{i=0}^3$ by

$$q(x,y) = \sum_{i=0}^{3} \tilde{b}_i K_i(x,y) = 0,$$

where

$$\begin{aligned}
\tilde{b}_0 &= \frac{\lambda_1\lambda_2}{\lambda_0\lambda_3} - \frac{u_1u_2}{u_0u_3}, \\
\tilde{b}_1 &= \frac{\lambda_1^2}{\lambda_0\lambda_2} - \frac{u_1^2}{u_0u_2}, \\
\tilde{b}_2 &= \frac{\lambda_2^2}{\lambda_1\lambda_3} - \frac{u_2^2}{u_1u_3}, \\
\tilde{b}_3 &= \frac{\lambda_0\lambda_3}{\lambda_1\lambda_2} - \frac{u_0u_3}{u_1u_2}.
\end{aligned}$$

Here, λ_i denotes twice the signed area of the convex hull of $\{\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3\} \setminus \mathbf{c}_i$ and $u_i = \binom{3}{i} w_i$. In the paper, a multiple of these coefficients is chosen. However, for practical purposes, these coefficients are more useful. In particular, they are independent of affine transformations of the curve.

By Corollary 11 in Paper IV, we know that all the coefficients vanish when the curve degenerates to a conic. Thus, when using floating point arithmetic, we would like to define a tolerance for when the curve should be treated as a conic. Using the coefficients $(b_i)_{i=0}^3$, we can choose a tolerance $\epsilon > 0$, such that if $|b_i| < \epsilon$ for each $i = 0, 1, 2, 3$, then we use the implicit representation for conics (see Proposition 12). Experimentally, it seems sufficient to take ϵ to be quite close to machine precision.

A common technique for visualization of implicit curves is to shade regions of the plane according to the sign of the implicit polynomial in those regions. For example, we can shade the region $W = \{(x, y) : q(x, y) \leq 0\}$ in white, and $G = \{(x, y) : q(x, y) > 0\} = W^c$ in grey (e.g., Figure 3.1). However, such a method has some disadvantages in practice. Figure 3.1 shows two cubic Bézier curves rendered with such a method. The control points of these curves lie very close to each other, and yet the curve has flipped orientation. The reason for the change of orientation is that the curve has 'passed through' a conic section (both curves are *strictly cubic*, but they lie very close to a conic section[2]). It is important to realise that this is not an artifact of the specific algorithm; it is in the nature of the implicit representation of cubic curves. This is necessarily true since it is possible to transform between the two curves smoothly, without passing through the conic. This example highlights the unstable nature of cubic curves near conic sections. In [46], Loop and Blinn overcome such problems by adding a test to check that the 'inside' of the curve always lies to the right of the curve, in the direction of increasing parameter. We describe here an alternative technique, which also solves other problems with unwanted self-intersections.

In addition to the implicit polynomial q, we have definitions for two lines, $\tilde{S}_1$ and $\tilde{S}_2$ in terms of the quantities $(\lambda_i)_{i=0}^3$ and $(u_i)_{i=0}^3$. For the explicit definition of the lines, we refer the reader to Proposition 6 in Paper IV. The important property of $\tilde{S}_1$ and $\tilde{S}_2$ is that they intersect each other at the double point, and intersect the curve at $\mathbf{c}_0$ and $\mathbf{c}_3$ respectively. As opposed to the 'inside/outside' approach, we propose defining the regions as follows:

$$W = \{(x, y) : \tilde{S}_1(x, y)q(x, y) \leq 0 \text{ OR } \tilde{S}_1(x, y)\tilde{S}_2(x, y) > 0\}, \quad B = W^c.$$

[2]In actuality, the left figure has an acnode at $(1, 100)$, and the right figure has a crunode at $(1, -100)$.

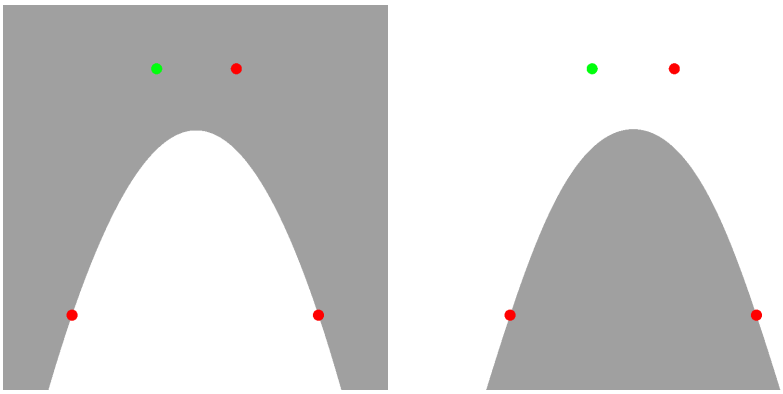


Figure 3.1: An example of two cubic Bézier curves with control points $(0, 0), (b_0, b_1), (2/3, 1)$ and $(1, 0)$ (left: $(b_1, b_2) = (1/3 + 0.01, 1)$, right: $(b_1, b_2) = (1/3 - 0.01, 1)$). The implicit representation flips orientation, despite only a small perturbation of the control points.

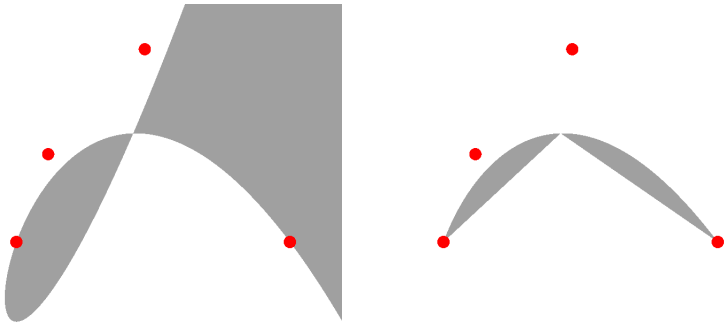


Figure 3.2: An example of rendering the same cubic Bézier curve with the inside/outside method (left) and the proposed method (right). Note that no rootfinding is required. In the right image it may be desirable to colour the triangle bounded by $\mathbf{c}_0$, $\mathbf{c}_3$ and the singularity in black.

Using these definitions gives a smooth representation of the implicit curve, with no unwanted flips near conics. In addition, if an unwanted self-intersection occurs, this method will automatically visually eliminate it (see Figure 3.2). In previous algorithms, this has been achieved by subdividing the curves at the parameter values of the singularity [46, 55].

For standard GPU implementations (e.g. using OpenGL or DirectX), each cubic curve is rendered in two triangular regions whose union make up the convex hull of the control polygon. Thus, subdividing the curve generates more data and requires more triangles for rendering. Since the method above deals with self-intersections without subdivision, it may be expected to perform better than the methods in [46, 55]. However, the fragment shader, which computes whether the points lie inside and outside of the curve, would be more complicated using this method and may hurt the performance. Additionally, in the method of Paper IV, subdivision is necessary when any three control points are collinear.

2.2 Ray tracing surfaces with approximate implicitization

In a similar vein to the rendering of curves, the presence of the GPU has opened new doors to surface rendering in the implicit form. Several topics in the efficient visualization of implicit surfaces appear in the PhD thesis of Seland [67]. However, real-time ray tracing of implicit surfaces is quite severely restricted by the implicit degree and is currently only feasible for degrees approximately ≤ 10. Since a general bicubic parametric patch has implicit degree 18, the potential for exact methods appears to be limited.

The methods for approximate implicitization presented in this thesis have the potential to be coupled with implicit surface ray tracing methods. This would give the benefits of both the geometric control by manipulating the control polygon, and the high quality rendering that ray tracing produces. However, several challenges would need to be resolved.

An inherent problem with the implicit representation is the presence of extraneous branches. These branches define areas of the curve or surface, which are not part of the region of interest. Although approximation has the potential to remove some extraneous branches, they are also a common occurrence in approximate implicitization. One potential method for the removal of branches would be to use inversion formulas. These formulas give the parameter value of a point known to lie on the surface. If the parameter values thus generated lie outside the parameter domain of interest, the point can be discarded. However, inversion formulas are useful only for points that are *known to lie on the surface*. For a given non-singular point $\mathbf{p}_0$ on a surface, it will often be the case that the inversion formula evaluated at the point $\mathbf{p}_0 + \mathbf{e}$ returns values within the parameter domain, for $\epsilon > \|\mathbf{e}\| > 0$, even though the point is not on the surface. Hence, unwanted self-intersections generated from approximations (i.e., points that do not necessarily lie on the exact implicit surface), are unlikely to be identified well by inversion formulas. Although methods for approximate inversion have been discussed, their applicability is unclear [66].

An alternative method for removing branches is to form linear combinations of several good approximations to the surface. When performing the algorithm for approximate implicitization for a given degree m, we obtain a matrix (the right singular matrix of the SVD), that defines a basis for the space of polynomials of total degree m. The space is partitioned in such a way that the approximations (in the form of singular vectors), are ordered in quality by their corresponding singular values. If $\mathbf{v}_{\min}$ and $\mathbf{v}_{\min-1}$ refer to the singular vectors corresponding to the two smallest singular values, we can explore the linear space $\{\tau\mathbf{v}_{\min} + (1-\tau)\mathbf{v}_{\min-1}, \tau \in \mathbb{R}\}$ for an approximation in which extra branches are not present. If there are more good approximations, more dimensions can be added to the linear space. In particular, the orthogonal basis methods introduced in Paper II generally exhibit a larger family of good approximations than the original method [27]. This method for branch removal was employed for the implicitization of the teapot patches in Paper II. However, since criteria for when unwanted branches occur are hard to determine, it is difficult to automate this process. Moreover, though likely, it is not guaranteed that the linear space will contain a solution free from extra branches.

Defining suitable boundaries in which to render is another challenge which is more difficult to overcome for surfaces than curves. For surfaces, a 3D domain must be chosen for rendering. For the teapot patches, it suffices to define the boundaries as the 3D box which is limited by the upper and lower bounds of the control points, in a Cartesian system. However, this is not the

case in general. The four boundary curves of a general bicubic patch can be formed from any cubic space curve. Moreover, these boundary curves do not always define the silhouette of the surface. The method in Paper III for computing envelopes, is related to the problem of finding silhouettes. Utilizing this for the case of rendering may be a direction for future research, although performance would be a major obstacle with current hardware and implementations.

Probably the most severe barrier to using implicit surfaces in practice is performance. There exist competing methods for pixel-accurate rendering of surfaces which use the parametric form, and can render several thousand bicubic surface patches at hundreds of frames per second [79]. While these methods may not produce the same effects as ray traced surfaces, such performance currently eludes implicit surface rendering.

3 Robotics

In Paper III we propose an extension to the method for approximate implicitization of envelope curves first published in [59]. Envelope curves have a variety of applications in robotics, including defining boundaries, collision detection and gearing [56, 59]. Envelope implicitization is also interesting from a theoretical point of view, in giving an explicit definition to the curve.

In [59] a method for piecewise implicit approximation of envelope curves was presented along with several examples. One reason for choosing piecewise approximation was that the method became very computationally expensive for all but the lowest degrees. The new implementation in Paper III allows faster approximations, thus in this section we present examples of higher degree approximations, as opposed to piecewise approximations.

In Figure 3.3 we show the method applied to two different families of curves, both of which are implicitized at degree six. The first, which is a quadratic family of circles of variable radius, has the homogeneous definition

$$\mathbf{p}(s,t) = \sum_{i=0}^{2}\sum_{j=0}^{4}(x_{ij}, y_{ij}, w_{ij})B_i^2(s)B_j^4(t),$$

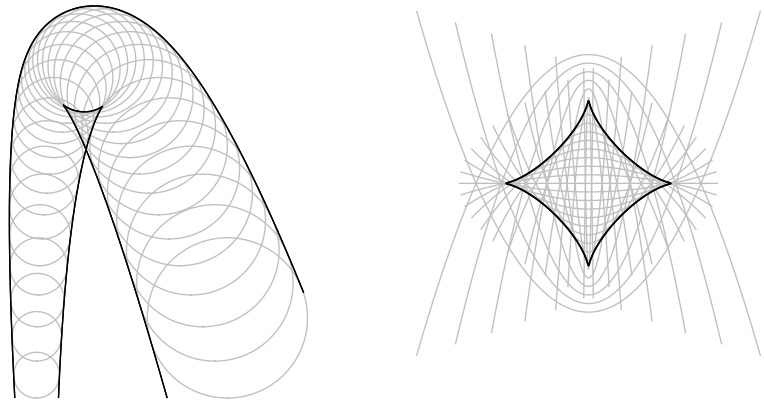
where

$$(x_{ij})_{i,j=(0,0)}^{(2,4)} = \begin{pmatrix} 1/3 & -1/25 & 1/9 & 1/25 & 1/3 \\ 1/3 & -9/100 & 1/9 & 9/100 & 1/3 \\ 2/3 & -7/50 & 2/9 & 7/50 & 2/3 \end{pmatrix},$$

$$(y_{ij})_{i,j=(0,0)}^{(2,4)} = \begin{pmatrix} 0 & 0 & 4/75 & 0 & 0 \\ 1 & 0 & 34/75 & 0 & 1 \\ 0 & 0 & 14/75 & 0 & 0 \end{pmatrix},$$

$$(w_{ij})_{i,j=(0,0)}^{(2,4)} = \begin{pmatrix} 1 & 0 & 1/3 & 0 & 1 \\ 1 & 0 & 1/3 & 0 & 1 \\ 1 & 0 & 1/3 & 0 & 1 \end{pmatrix}.$$

The seemingly higher degree in the t-parameter direction is because the degree has been raised in order to obtain a better parametrization of the circles. The family is really biquadratic in nature. For the implicit degree of the envelope curve we take $m = 6$. However, envelope im-



(a) (b)

Figure 3.3: The envelope curves defined by the families of curves given in Section 3 and computed using the techniques of Paper IV. The implicit representation has degree six in both cases.

plicitization using this method also requires a choice of bidegree for the coupling function λ, which we denote (k_1, k_2). In order to obtain good results for this example, we require $k_1 = 10$ and $k_2 = 16$. This particular choice of bidegree (k_1, k_2), is discussed in more detail in Paper III. The necessity of using such high degrees makes the algorithm rather slow, but the example shows that accurate implicitizations of moderately high degree are possible. Also, it may be noted that the envelope function, $h(s, t)$, has three branches within the region of interest. Thus, attempting to implicitize the entire envelope with a single polynomial could be considered a relatively complicated problem. If a piecewise implementation is used, these branches would normally be considered separately. This example is generated by a medial axis transform of the type which appear commonly in robotics. Of course, in this example, a square root parametrization of the envelope curve is possible, thus less computationally intensive methods could be used by generating point data on the curve.

The second example is a quadratic family of parabolas, defined by

$$(x_{ij})^{(2,4)}_{i,j=(0,0)} = \begin{pmatrix} 1/3 & -1/25 & 1/9 & 1/25 & 1/3 \\ 1/3 & -9/100 & 1/9 & 9/100 & 1/3 \\ 2/3 & -7/50 & 2/9 & 7/50 & 2/3 \end{pmatrix},$$

$$(y_{ij})^{(2,4)}_{i,j=(0,0)} = \begin{pmatrix} 0 & 0 & 4/75 & 0 & 0 \\ 1 & 0 & 34/75 & 0 & 1 \\ 0 & 0 & 14/75 & 0 & 0 \end{pmatrix},$$

with the weight function $w(s, t) = 1$.

Using a single polynomial implicit representation of the envelope rather than a piecewise approximation can be somewhat advantageous. Since we only need a single polynomial for the entire region of interest, there are none of the complications of defining 2D regions for the pieces. In addition, for higher degrees, the approximation can be expected to be somewhat better.

Bibliography

[1] S.S. Abhyankar and C.L. Bajaj. Automatic parameretization of rational curves and surfaces I: conics and conicoids. *Computer-Aided Design*, 19(1):11 – 14, 1987.

[2] S.S. Abhyankar and C.L. Bajaj. Automatic parametrization of rational curves and surfaces II: cubics and cubicoids. *Computer-Aided Design*, 19(9):499–502, November 1987.

[3] S.S. Abhyankar and C.L. Bajaj. Automatic parameterization of rational curves and surfaces III: algebraic plane curves. *Computer Aided Geometric Design*, 1988.

[4] C.L. Bajaj and I. Ihm. Algebraic surface design with Hermite interpolation. *ACM Transactions on Graphics*, 11(1):61–91, 1992.

[5] C.L. Bajaj and I. Ihm. Smoothing polyhedra using implicit algebraic splines. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '92, pages 79–88, New York, NY, USA, 1992. ACM.

[6] C.L. Bajaj, I. Ihm, and J. Warren. Higher-order interpolation and least-squares approximation using implicit algebraic surfaces. *ACM Transactions on Graphics*, 12(4):327–347, October 1993.

[7] C.L. Bajaj and G. Xu. A-splines: local interpolation and approximation using gk-continuous piecewise real algebraic curves. *Computer Aided Geometric Design*, 16(6):557 – 578, 1999.

[8] O.J.D. Barrowclough. A basis for the implicit representation of rational cubic Bézier curves. *Submitted to publisher*, 2013.

[9] O.J.D. Barrowclough and T. Dokken. Approximate implicitization of triangular Bézier surfaces. In *Proceedings of the 26th Spring Conference on Computer Graphics*, SCCG '10, pages 133–140, New York, NY, USA, 2010. ACM.

[10] O.J.D. Barrowclough and T. Dokken. Approximate implicitization using linear algebra. *Journal of Applied Mathematics*, 2012. doi:10.1155/2012/293746.

[11] O.J.D. Barrowclough, B. Jüttler, and T. Schulz. Fast approximate implicitization of envelope curves using Chebyshev polynomials. In Jadran Lenarčič and Manfred Husty, editors, *Latest Advances in Robot Kinematics*, pages 205–212. Springer Netherlands, 2012.

[12] B. Buchberger. *An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal*. PhD thesis, University of Innsbruck, 1965.

[13] L. Busé and T.L. Ba. The surface/surface intersection problem by means of matrix based representations. *Computer Aided Geometric Design*, 8:579–598, 2012.

[14] L. Busé and C. D'Andrea. Singular factors of rational plane curves. *Journal of Algebra*, 357:322–346, 2012.

[15] L. Busé, M. Elkadi, and A. Galligo. Intersection and self-intersection of surfaces by means of Bezoutian matrices. *Computer Aided Geometric Design*, 2:53–68, 2008.

[16] L. Busé and T. Luu Ba. Matrix-based implicit representations of rational algebraic curves and applications. *Computer Aided Geometric Design*, 27(9):681–699, December 2010.

[17] T.J. Cashman. *NURBS-compatible subdivision surfaces*. PhD thesis, University of Cambridge, 2010.

[18] A. Cayley. Note sur la méthode d'élimination de Bézout. *Journal für die reine und angewandte Mathematik*, 53:366–367, 1857.

[19] F. Chen. The μ-basis of a planar rational curve - properties and computation. *Graphical Models*, 64:368–381, February 2003.

[20] F. Chen, D. Cox, and Y. Liu. The μ-basis and implicitization of a rational parametric surface. *Journal of Symbolic Computation*, 39(6):689–706, June 2005.

[21] E.W. Chionh and T.W. Sederberg. On the minors of the implicitization Bézout matrix for a rational plane curve. *Computer Aided Geometric Design*, 18:21–36, 2001.

[22] R.M. Corless, M. Giesbrecht, I. Kotsireas, and S. Watt. Numerical implicitization of parametric hypersurfaces with linear algebra. In *Artificial Intelligence and Symbolic Computation*, pages 174–183. Springer, 2001.

[23] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Undergraduate Texts in Mathematics. Springer, 2010.

[24] A.L. Dixon. The eliminant of three quantics in two independent variables. *Proceedings of the London Mathematical Society*, 2:46–49, 1908.

[25] M. Dohm and S. Zube. The implicit equation of a canal surface. *Journal of Symbolic Computation*, 44(2):111–130, February 2009.

[26] T. Dokken. *Aspects of intersection algorithms and approximation*. PhD thesis, University of Oslo, 1997.

[27] T. Dokken. Approximate implicitization. *Mathematical Methods for Curves and Surfaces: Oslo*, 2000.

[28] T. Dokken. The GAIA project on intersection and implicitization. *Geometric modeling and algebraic geometry*, 2008.

[29] T. Dokken and J.B. Thomassen. Weak approximate implicitization. In *Shape Modeling and Applications, 2006. SMI 2006. IEEE International Conference on*, page 31. IEEE, 2006.

[30] D. Eisenbud, M. Green, and J. Harris. Cayley-Bacharach theorems and conjectures. *Bull. Amer. Math. Soc.*, 33(03):295–325, July 1996.

[31] I.Z. Emiris, T. Kalinka, and C. Konaxis. Implicitization of curves and surfaces using predicted support. *Proceedings of the 2011 International Workshop on Symbolic-Numeric Computation - SNC '11*, 2011:137–146, 2011.

[32] I.Z. Emiris and I.S. Kotsireas. Implicitization exploiting sparseness. In *Geometric And Algorithmic Aspects Of Computer-aided Design And Manufacturing*, volume 67 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 281–298. American Mathematical Society, 2005.

[33] G. Farin. A history of curves and surfaces in CAGD. *Handbook of Computer Aided Geometric Design*, pages 1–23, 2002.

[34] G. Farin. *Curves and surfaces for CAGD: a practical guide*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[35] R.T. Farouki. On the stability of transformations between power and Bernstein polynomial forms. *Computer Aided Geometric Design*, 8(1):29–36, February 1991.

[36] R.T. Farouki, T.N.T. Goodman, and T. Sauer. Construction of orthogonal bases for polynomials in Bernstein form on triangular and simplex domains. *Computer Aided Geometric Design*, 20(4):209 – 230, 2003.

[37] M.S. Floater. Rational cubic implicitization. *Mathematical Methods for Curves and Surfaces*, pages 151–159, 1995.

[38] X.S. Gao and S.C. Chou. Implicitization of rational parametric equations. *Journal of Symbolic Computation*, pages 1–15, 1992.

[39] C.M. Hoffmann. Algebraic and numerical techniques for offsets and blends. *Computations of Curves and Surfaces*, pages 499–528, 1990.

[40] C.M. Hoffmann. Implicit curves and surfaces in CAGD. *Computer Graphics and Applications, IEEE*, 13(1):79–88, 1993.

[41] B. Jüttler, P. Chalmovianský, M. Shalaby, and E. Wurm. Approximate algebraic methods for curves and surfaces and their applications. In *Proceedings of the 21st spring conference on Computer graphics - SCCG '05*, volume 1, page 13, New York, New York, USA, 2005. ACM Press.

[42] B. Jüttler and A. Felis. Least-squares fitting of algebraic spline surfaces. *Advances in Computational Mathematics*, 17(1):135–152, 2002.

[43] I.S. Kotsireas. Panorama of methods for exact implicitization of algebraic curves and surfaces. In F. Chen and D. Wang, editors, *Geometric Computation. In: Lecture Notes Series on Computing*, volume 11, chapter 4, pages 126–155. World Scientific Publishing, 2004.

[44] Q. Li and J. Tian. 2D piecewise algebraic splines for implicit modeling. *ACM Transactions on Graphics*, 28(2):1–19, April 2009.

[45] R.A. Liming and L. Hudson. *Practical analytic geometry with applications to aircraft*. Macmillan, 1944.

[46] C.T. Loop and J. Blinn. Resolution independent curve rendering using programmable graphics hardware. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 1000–1009. ACM, 2005.

[47] C.T. Loop and J. Blinn. Real-time GPU rendering of piecewise algebraic surfaces. *ACM Transactions on Graphics*, 25(3):664, July 2006.

[48] F.S. Macaulay. Some formulae in elimination. *Proceedings of the London Mathematical Society*, 35(1):3–27, 1902.

[49] Maplesoft, a division of Waterloo Maple Inc., Toronto. *Maple User Manual*, 2005-2012.

[50] A. Marco and J.J. Martinez. Using polynomial interpolation for implicitizing algebraic curves. *Computer Aided Geometric Design*, 18(4):309–319, 2001.

[51] A. Marco and J.J. Martinez. Implicitization of rational surfaces by means of polynomial interpolation. *Computer aided geometric design*, 19(5):327–344, May 2002.

[52] B.S. Morse, T.S. Yoo, P. Rheingans, D.T. Chen, and K.R. Subramanian. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05, New York, NY, USA, 2005. ACM.

[53] Persistence of Vision Pty. Ltd. Persistence of vision raytracer (version 3.6), [computer software]. Retrieved from `http://www.povray.org/download/`, 2004.

[54] D. Pekerman, G. Elber, and M.-S. Kim. Self-intersection detection and elimination in freeform curves and surfaces. *Computer-Aided Design*, 40(2):150–159, February 2008.

[55] R. Pfeifle. Rendering cubic curves on a GPU with Floater's implicitization. *Journal of Graphics Tools*, 16(2):105–122, 2012.

[56] M. Rabl, B. Jüttler, and L. Gonzalez-Vega. Exact envelope computation for moving surfaces with quadratic support functions. In Lenarčič and Wenger, editors, *Adv. in Robot Kinematics: Analysis and Design*, pages 283–290. Springer, 2008.

[57] G. Salmon. *A treatise on the higher plane curves*. Dublin, Hodges & Smith, 1852.

[58] J. Schicho and I. Szilágyi. Numerical stability of surface implicitization. *Journal of Symbolic Computation*, 40(6):1291–1301, December 2005.

[59] T. Schulz and B. Jüttler. Envelope computation in the plane by approximate implicitization. *Applicable Algebra in Engineering, Communication and Computing*, 22(4):265–288, July 2011.

[60] T.W. Sederberg. *Implicit and Parametric Curves and Surfaces for Computer Aided Geometric Design*. PhD thesis, Purdue University, 1983.

[61] T.W. Sederberg. Planar piecewise algebraic curves. *Computer Aided Geometric Design*, 1(3):241–255, 1984.

[62] T.W. Sederberg. Piecewise algebraic surface patches. *Computer Aided Geometric Design*, 2(1-3):53–59, 1985.

[63] T.W. Sederberg, D.C. Anderson, and R.N. Goldman. Implicit representation of parametric curves and surfaces. *Computer Vision, Graphics, and Image Processing*, 28(1):72–84, 1984.

[64] T.W. Sederberg and F. Chen. Implicitization using moving curves and surfaces. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, page 308, New York, New York, USA, 1995. ACM.

[65] T.W. Sederberg and S.R. Parry. Comparison of three curve intersection algorithms. *Computer-Aided Design*, 1986.

[66] T.W. Sederberg, J. Zheng, K. Klimaszewski, and T. Dokken. Approximate implicitization using monoid curves and surfaces. *Graphical Models and Image Processing*, 61(4):177–198, July 1999.

[67] J.S. Seland. *Smooth surface visualization using graphics hardware*. PhD thesis, University of Oslo, 2008.

[68] M. Shalaby, J.B. Thomassen, E. Wurm, T. Dokken, and B. Jüttler. Piecewise approximate implicitization: Experiments using industrial data. *Algebraic Geometry and Geometric Modeling*, pages 37–51, 2006.

[69] L. Shen, F. Chen, B. Jüttler, and J. Deng. Approximate μ-Bases of Rational Curves and Surfaces. *Geometric Modeling and Processing-GMP 2006*, pages 175–188, 2006.

[70] V. Skytt. Challenges in surface-surface intersections. In *Computational Methods for Algebraic Spline Surfaces*, pages 11–26. Springer Berlin Heidelberg, 2005.

[71] X. Song, T.W. Sederberg, and J. Zheng. Linear perturbation methods for topologically consistent representations of free-form surface intersections. *Computer Aided Geometric Design*, 2004.

[72] J.J. Sylvester. On a theory of the syzygetic relations of two rational integral functions, comprising an application to the theory of Sturm's functions, and that of the greatest algebraical common measure. *Philosophical Transactions of the Royal Society of London*, 143:407–548, 1853.

[73] J.B. Thomassen. Self-intersection problems and approximate implicitization. *Computational methods for algebraic spline surfaces*, pages 155–170, 2005.

[74] J.B. Thomassen, P.H. Johansen, and T. Dokken. Closest points, moving surfaces, and algebraic geometry. *Mathematical methods for curves and surfaces: Tromsø*, pages 351–362, 2004.

[75] T. Warburton. An explicit construction of interpolation nodes on the simplex. *Journal of Engineering Mathematics*, 56:247–262, 2006. 10.1007/s10665-006-9086-6.

[76] E. Wurm and B. Jüttler. Approximate implicitization via curve fitting. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 240–247. Eurographics Association, 2003.

[77] E. Wurm, B. Jüttler, and M.S. Kim. Approximate rational parameterization of implicitly defined surfaces. *Mathematics of Surfaces XI*, pages 434–447, 2005.

[78] E. Wurm, J.B. Thomassen, B. Jüttler, and T. Dokken. Comparative benchmarking of methods for approximate implicitization. *Geometric Design and Computing, Nashboro Press, Brentwood*, pages 537–548, 2004.

[79] Y.I. Yeo, L. Bin, and J. Peters. Efficient pixel-accurate rendering of curved surfaces. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '12, pages 165–174, New York, NY, USA, 2012. ACM.

Part II

Scientific papers

Paper I: Approximate Implicitization of Triangular Bézier Surfaces

Oliver J.D. Barrowclough and Tor Dokken

In *Proceedings of the 26th Spring Conference on Computer Graphics*, SCCG '10, pages 133-140, New York, NY, USA, 2010. ACM.

Abstract: We discuss how Dokken's methods of approximate implicitization can be applied to triangular Bézier surfaces in both the original and weak forms. The matrices $\mathbf{D}$ and $\mathbf{M}$ that are fundamental to the respective forms of approximate implicitization are shown to be constructed essentially by repeated multiplication of polynomials and by matrix multiplication. A numerical approach to weak approximate implicitization is also considered and we show that symmetries within this algorithm can be exploited to reduce the computation time of $\mathbf{M}$. Explicit examples are presented to compare the methods and to demonstrate properties of the approximations.

1 Introduction

Methods for conversion between the two main representations of curves and surfaces in CAGD, namely the parametric and implicit forms, have been widely investigated within the CAGD community. Of these, the parametric form has established itself as the representation of choice in most CAGD systems due to its intuitive geometric nature [9]. However, the implicit form has distinct advantages over the parametric form in solving certain geometrical problems and thus the possibility to have a dual representation is, in some circumstances, appealing [10]. For example, the implicit representation allows us to immediately determine whether a given point lies on the curve or surface. Although exact formulas can be devised for low degree surfaces, higher order parametric geometries require computationally expensive algorithms such as recursive subdivision. Implicit representations are also useful in intersection problems. Notably, ray tracing of implicitly defined surfaces is much quicker than ray tracing of parametric surfaces. Despite these advantages, exact implicit representations of rational parametric curves and surfaces lead to high polynomial degrees in the implicit equation. In general, implicit equations of high degree are not desirable due to often having extraneous branches and singularities that are not necessarily present in the parametric form. They also exhibit a lack of numerical stability [15].

The procedure of converting from the rational parametric to the implicit form of a curve or surface is called implicitization. Traditional methods using Gröbner bases or resultants, focused solely on exact implicitization. Exact implicit representations use exact arithmetic, whereas in CAD and CAGD, the use of floating point arithmetic is desirable due to performance. Approximate implicitization provides numerically stable methods to approximate a parametric surface using lower degree implicit equations. In [2], a method for approximate implicitization was introduced that allows us to choose the degree of the implicit equation to be defined. The theory behind this approach to approximate implicitization of rational parametric manifolds in $\mathbb{R}^l$ has been thoroughly developed in [2, 3]. A similar approach known as weak approximate implicitization was developed in [5]. The aim of this paper is to present both these methods, in the special case of approximate implicitization of triangular Bézier surfaces. Although other methods of approximate implicitization exist [11, 16, 15, 18], the original and weak methods that we follow provide fast algorithms with a high order of convergence that are well suited to curves and surfaces defined in a partition of unity basis [17].

While approximate implicitization of tensor-product Bézier surfaces is a fairly simple extension of approximate implicitization of 2D rational parametric curves, triangular Bézier surfaces are somewhat more difficult. They are, however, expressed naturally in terms of a bivariate Bernstein basis over a triangular domain, which forms a partition of unity. This allows us to follow the steps of original approach fairly directly.

Unfortunately, implicitization algorithms tend to be computationally expensive and as such are hindered in CAGD applications. This paper will highlight some symmetries in the numerical approach to the algorithm that can be exploited to reduce the computation time, and thus improve the prospect of dual representations in CAGD.

This paper will be organised as follows. Section 2, will briefly introduce the concepts required to define triangular Bézier surfaces, and highlight some properties of Bernstein polynomials that are important for approximate implicitization. Section 3 will present the procedure for approximate implicitization in the context of Bézier triangles, both in the original and weak forms. It will highlight some new observations that significantly reduce the number of computations required in the numerical form of the algorithm. The accuracy and convergence rates of approximate implicitization will also be stated. Section 4 will describe a simple example of approximate implicitization of a Bézier triangle before concluding with some examples that are more relevant in practice.

2 Triangular Bézier Surfaces

Triangular Bézier surfaces, also known as Bézier triangles, were developed by Paul de Casteljau to offer a natural generalization of Bézier curves to surfaces [6]. Although, tensor-product patches may be more intuitive (and are certainly used more widely in CAGD), the triangular patches are in some sense a more fundamental generalization. In this section we recall the notation of Bézier triangles and state some simple results about Bernstein polynomials. For a comprehensive review of these concepts we refer the reader to [6, 7].

2.1 Barycentric Coordinates

In this paper we will make extensive use of barycentric coordinates, both over triangles and tetrahedra. Barycentric coordinates over triangles provide a natural domain in which to define the Bézier triangle, whereas tetrahedral barycentric coordinates will be used to define the implicit surface. We introduce the notation in the general form to capture both these circumstances in a common definition.

Barycentric coordinates allow us to express any point $\mathbf{x} \in \mathbb{R}^l$ as

$$\mathbf{x} = \sum_{i=1}^{l+1} \beta_i \mathbf{a}_i, \quad \sum_{i=1}^{l+1} \beta_i = 1,$$

where $\mathbf{a}_i \in \mathbb{R}^l$ are points defining the vertices of a non-degenerate simplex in $\mathbb{R}^l$.

The conversion between Cartesian coordinates $\mathbf{x} = (x_1, \ldots, x_l)$ and barycentric coordinates $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_{l+1})$ over the simplex with vertices $(\mathbf{a}_1, \ldots, \mathbf{a}_{l+1})$, is given by the following relation:

$$\begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{a}_1 & \ldots & \mathbf{a}_{l+1} \\ 1 & \ldots & 1 \end{pmatrix} \boldsymbol{\beta}. \tag{1}$$

If a point lies within the simplex which defines the barycentric coordinate system, the barycentric coordinates of that point are guaranteed to be non-negative. This leads to good numerical stability if all the points in the algorithm are contained within the relevant simplex. We define the domain Ω to be the triangle formed by a bivariate barycentric coordinate system, and Λ to be the tetrahedron formed by a trivariate barycentric coordinate system. Unless explicitly stated, all subsequent coordinates in this paper are assumed to be barycentric.

2.2 Bernstein Polynomials

The notation used when describing Bernstein polynomials and Bézier triangles is greatly simplified by making use of multi-indices. These provide a natural way to label the basis functions and can be related to regular indices by choosing an ordering. For multi-indices $\mathbf{i} = (i_1, \ldots, i_{l+1})$ and $\mathbf{j} = (j_1, \ldots, j_{l+1})$ we have the following definitions:

- $|\mathbf{i}| = i_1 + \cdots + i_{l+1}$,
- $\mathbf{i} + \mathbf{j} = (i_1 + j_1, \ldots, i_{l+1} + j_{l+1})$,
- for barycentric coordinates $\boldsymbol{\beta}$, define $\boldsymbol{\beta}^{\mathbf{i}} = \beta_1^{i_1} \cdots \beta_{l+1}^{i_{l+1}}$,
- the multinomial coefficients are defined as

$$\binom{n}{\mathbf{i}} = \frac{n!}{i_1! i_2! \cdots i_{l+1}!},$$

- the ordering of choice is the lexicographical ordering, described by $(i_1, \ldots, i_{l+1}) < (j_1, \ldots, j_{l+1})$ if and only if there exists an index k such that $i_k < j_k$ and $i_r = j_r$ for all $r < k$.

We now define the Bernstein basis polynomials of degree n as

$$B_{\mathbf{i}}^n(\boldsymbol{\beta}) = \binom{n}{\mathbf{i}} \boldsymbol{\beta}^{\mathbf{i}}, \quad |\mathbf{i}| = n,$$

where $\boldsymbol{\beta}$ are barycentric coordinates.

In this paper, care must be taken to distinguish between triangular and tetrahedral Bernstein polynomials as the notation differs only by the variable they are defined under. A triangular Bernstein polynomial will be defined in the variable $\mathbf{s} \in \Omega$, whereas a tetrahedral Bernstein polynomial will be defined for $\mathbf{u} \in \Lambda$. We use the variables $\boldsymbol{\beta}$ when describing general barycentric coordinates.

We will now state three important properties of Bernstein polynomials that will be used in the implicitization algorithm:

- The Bernstein basis forms a partition of unity. That is

$$\sum_{|\mathbf{i}|=n} B_{\mathbf{i}}^n(\boldsymbol{\beta}) = 1, \tag{2}$$

  for all barycentric coordinates $\boldsymbol{\beta}$.

- There is a simply derived formula for multiplying Bernstein polynomials of the same form (i.e., triangular or tetrahedral Bernstein polynomials), which is given as follows:

$$B_{\mathbf{i}}^n(\boldsymbol{\beta}) B_{\mathbf{j}}^m(\boldsymbol{\beta}) = \frac{\binom{n}{\mathbf{i}}\binom{m}{\mathbf{j}}}{\binom{n+m}{\mathbf{i}+\mathbf{j}}} B_{\mathbf{i}+\mathbf{j}}^{n+m}(\boldsymbol{\beta}), \tag{3}$$

  with $|\mathbf{i}+\mathbf{j}| = |\mathbf{i}| + |\mathbf{j}| = m + n$.

- The integral over any Bernstein basis function of given degree is constant. In particular, for the Bernstein basis polynomials over a triangle of unit area [8]:

$$\int_\Omega B_{\mathbf{i}}^n(\mathbf{s}) \, \mathrm{d}\mathbf{s} = \frac{1}{(n+1)(n+2)}. \tag{4}$$

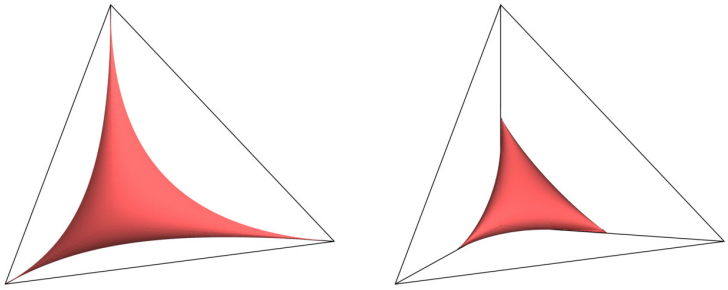
  This implies that the integral of any polynomial $q(\mathbf{s})$ defined in the triangular Bernstein basis is given by:

$$\int_\Omega q(\mathbf{s}) \, \mathrm{d}\mathbf{s} = \frac{1}{(n+1)(n+2)} \sum_{|\mathbf{i}|=n} b_{\mathbf{i}}. \tag{5}$$

2.3 Bézier Triangles

We can now state the definition of a degree n Bézier triangle with control points $(\mathbf{c}_{\mathbf{i}})_{|\mathbf{i}|=n}$ in terms of the triangular Bernstein basis as follows:

$$\mathbf{p}(\mathbf{s}) = \sum_{|\mathbf{i}|=n} \mathbf{c}_{\mathbf{i}} B_{\mathbf{i}}^n(\mathbf{s}). \tag{6}$$



(a) $\mathbf{p}_1(\mathbf{s})$ (b) $\mathbf{p}_2(\mathbf{s})$

Figure 1: Examples of Bézier triangles $\mathbf{p}_1(\mathbf{s})$ defined in Section 2.3 and $\mathbf{p}_2(\mathbf{s})$ defined in Section 4.3. Exact and approximate implicitizations of the latter surface are in Figure 2.

The control points $\mathbf{c_i}$ can be defined in any space but we restrict them to lie in $\mathbb{R}^3$ since we are interested in surfaces. We consider only the points $\mathbf{s}$ in the domain Ω so that the entire Bézier triangle lies within the convex hull of its control points.

Figure 1(a) shows an example of a degenerate quadratic Bézier triangle $\mathbf{p}_1(\mathbf{s})$, with Cartesian control points

$$\begin{gathered}\mathbf{c}_{200} = (1,0,0),\\ \mathbf{c}_{110} = (0,0,0),\ \ \mathbf{c}_{101} = (0,0,0),\\ \mathbf{c}_{020} = (0,1,0),\ \ \mathbf{c}_{011} = (0,0,0),\ \ \mathbf{c}_{002} = (0,0,1).\end{gathered}$$

In Section 6 we will see three alternative quadratic implicit approximations of this surface. Notice that the lexicographical ordering here is given by reading the control points from left to right and top to bottom.

3 Approximate Implicitization

In this section we outline the approach to approximate implicitization presented in [2, 3], in the context of Bézier triangles. Both the original approach and the so-called weak approach will be described, closely following the procedure given in [5]. We will also look at a numerical approach to the algorithm in greater detail.

The exact implicitization of a degree n Bézier triangle may require a degree as high as n^2. It should be noted that if a degree high enough for an exact implicitization is chosen and the algorithm is executed using exact arithmetic, then these methods will be exact. Use of floating point arithmetic will result in small rounding errors.

We begin by stating the formal definition of approximate implicitization:

An algebraic surface defined by the points $\mathbf{u} \in \mathbb{R}^3$ such that $q(\mathbf{u}) = 0$ for some polynomial q, approximates the parametric surface $\mathbf{p}(\mathbf{s})$ within a tolerance of ϵ if there exists a vector-valued function $\mathbf{g}(\mathbf{s})$ of unit length, and an error function $\delta(\mathbf{s})$ such that

$$q(\mathbf{p}(\mathbf{s}) + \delta(\mathbf{s})\mathbf{g}(\mathbf{s})) = 0, \tag{7}$$

and

$$\max_{\mathbf{s}\in\Omega} |\delta(\mathbf{s})| < \epsilon.$$

We do not attempt to find the functions $\mathbf{g}(\mathbf{s})$ and $\delta(\mathbf{s})$ directly. Instead, we aim to find a polynomial q of chosen degree m that minimizes the algebraic distance $|q(\mathbf{p}(\mathbf{s}))|$ between the parametric and implicit surfaces. Certainly, if $q(\mathbf{p}(\mathbf{s})) = 0$, then we have an exact implicitization. In Section 3.5 we will see that this approach is also justified for approximate implicitization.

The method we use to find the polynomial q, both in the original and weak approaches is to find the coefficients $b_{\mathbf{i}}$ of q when expressed in the Bernstein basis of chosen degree m :

$$q(\mathbf{u}) = \sum_{|\mathbf{i}|=m} b_{\mathbf{i}} B^m_{\mathbf{i}}(\mathbf{u}). \tag{8}$$

The difference between original and weak approximate implicitization is the choice of how to minimize the algebraic distance. The original approach attempts to minimize the pointwise error

$$\max_{s\in\Omega} |q(\mathbf{p}(\mathbf{s}))|,$$

whereas the weak approach minimizes by integration:

$$\int_\Omega (q(\mathbf{p}(\mathbf{s})))^2 \, \mathrm{d}\mathbf{s}.$$

3.1 The Original Approach

We follow the same steps as in approximate implicitization of tensor-product Bézier surfaces and Bézier curves, only now using the triangular Bernstein basis functions. As we will see, the details differ somewhat in the triangular case.

Since we have chosen q to be of degree m, and $\mathbf{p}(\mathbf{s})$ is defined to be degree n, the expression $q(\mathbf{p}(\mathbf{s}))$ will be a polynomial of degree mn in $\mathbf{s}$. Such a polynomial can be factorized in the Bernstein basis $(B^{mn}_{\mathbf{j}})_{|\mathbf{j}|=mn}$ with coefficients $d_{\mathbf{i},\mathbf{j}}$. To obtain these coefficients, we form the following composition of the coordinate functions of $\mathbf{p}(\mathbf{s})$ with each Bernstein basis function $B^m_{\mathbf{i}}$:

$$B^m_{\mathbf{i}}(\mathbf{p}(\mathbf{s})) = \sum_{|\mathbf{j}|=mn} d_{\mathbf{i},\mathbf{j}} B^{mn}_{\mathbf{j}}(\mathbf{s}). \tag{9}$$

Note that $d_{\mathbf{i},\mathbf{j}}$ can be calculated explicitly by using (3), the product rule for Bernstein bases. An example of how this is done is presented in Section 4.1.

Now, using (8) and (9) we get

$$\begin{aligned} q(\mathbf{p}(\mathbf{s})) &= \sum_{|\mathbf{i}|=m} b_{\mathbf{i}} B_{\mathbf{i}}^{m}(\mathbf{p}(\mathbf{s})) \\ &= \sum_{|\mathbf{i}|=m} b_{\mathbf{i}} \left(\sum_{|\mathbf{j}|=mn} d_{\mathbf{i},\mathbf{j}} B_{\mathbf{j}}^{mn}(\mathbf{s}) \right) \\ &= \sum_{|\mathbf{j}|=mn} B_{\mathbf{j}}^{mn}(\mathbf{s}) \left(\sum_{|\mathbf{i}|=m} d_{\mathbf{i},\mathbf{j}} b_{\mathbf{i}} \right). \end{aligned} \tag{10}$$

Since the matrix $\mathbf{D}$ defined by the coefficients $(d_{\mathbf{i},\mathbf{j}})_{|\mathbf{i}|=m,|\mathbf{j}|=mn}$ is fundamental to the theory of approximate implicitization, we summarize its construction in the following proposition:

Proposition 1. *The* $\binom{m+3}{3} \times \binom{mn+2}{2}$ *matrix* $\mathbf{D}$ *for approximate implicitization of triangular Bézier surfaces can be constructed by repeated multiplication of the coordinate functions of* $\mathbf{p}(\mathbf{s})$*, according to the equation (9).*

Writing the unknown coefficients $(b_{\mathbf{i}})_{|\mathbf{i}|=m}$ and the basis functions $(B_{\mathbf{j}}^{mn}(\mathbf{s}))_{|\mathbf{j}|=mn}$ in vectors $\mathbf{b}$ and $\mathbf{B}^{mn}(\mathbf{s})$ respectively, we restate (10) as

$$q(\mathbf{p}(\mathbf{s})) = \mathbf{B}^{mn}(\mathbf{s})^{T}\mathbf{D}\mathbf{b}. \tag{11}$$

We may impose, without loss of generality, the normalization condition $\|\mathbf{b}\| = 1$. Since the Bernstein basis forms a partition of unity, using the factorization (11) we get

$$\begin{aligned} \max_{\mathbf{s}\in\Omega} |q(\mathbf{p}(\mathbf{s}))| &= \max_{\mathbf{s}\in\Omega} |\mathbf{B}^{mn}(\mathbf{s})^{T}\mathbf{D}\mathbf{b}| \\ &\leq \max_{\mathbf{s}\in\Omega} \|\mathbf{B}^{mn}(\mathbf{s})\| \|\mathbf{D}\mathbf{b}\| \leq \|\mathbf{D}\mathbf{b}\|. \end{aligned}$$

The approximation may well be good outside the region of interest Ω, but the result used here, that $\|\mathbf{B}^{mn}(\mathbf{s})\| \leq 1$, is specific only to the domain Ω. A standard result from linear algebra tells us that $\min_{\|\mathbf{b}\|=1} \|\mathbf{D}\mathbf{b}\| = \sigma_{\min}$, where $\sigma_{\min}$ is the smallest singular value of $\mathbf{D}$. So, in particular we have

$$\min_{\|\mathbf{b}\|=1} \max_{\mathbf{s}\in\Omega} |q(\mathbf{p}(\mathbf{s}))| \leq \sigma_{\min}. \tag{12}$$

We can thus minimize the left hand side of the inequality by performing a singular value decomposition (SVD) on the matrix $\mathbf{D}$. The vector $\mathbf{b}_{\min}$ corresponding to the smallest singular value $\sigma_{\min}$ of $\mathbf{D}$ would then give the best candidate for the approximation.

3.2 The Weak Approach

Recall that the weak approach attempts to minimize the algebraic distance by minimizing the integral $\int_{\Omega} (q(\mathbf{p}(\mathbf{s})))^2 \, \mathrm{d}\mathbf{s}$. Here we approach this problem using the exact integration formula (4). However, the weak approach also introduces the possibility to perform a numerical integration. In Section 3.3 we will discuss this further.

Using the factorization (11) we can perform the integral as follows:

$$\begin{aligned}\int_\Omega \left(q(\mathbf{p}(\mathbf{s}))\right)^2 \,\mathrm{d}\mathbf{s} &= \int_\Omega \left(\mathbf{B}^{mn}(\mathbf{s})^T\mathbf{D}\mathbf{b}\right)^2 \,\mathrm{d}\mathbf{s} \\ &= \mathbf{b}^T\mathbf{D}^T\left(\int_\Omega \mathbf{B}^{mn}(\mathbf{s})^T\mathbf{B}^{mn}(\mathbf{s})\,\mathrm{d}\mathbf{s}\right)\mathbf{D}\mathbf{b} \\ &= \mathbf{b}^T\mathbf{D}^T\mathbf{A}\mathbf{D}\mathbf{b}, \end{aligned} \tag{13}$$

where $\mathbf{A}$ is the symmetric matrix defined by $(a_{\mathbf{i},\mathbf{j}})_{|\mathbf{i}|=mn,|\mathbf{j}|=mn}$

$$\begin{aligned} a_{\mathbf{i},\mathbf{j}} &= \int_\Omega B^{mn}_{\mathbf{i}}(\mathbf{s})B^{mn}_{\mathbf{j}}(\mathbf{s})\,\mathrm{d}\mathbf{s} \\ &= \frac{\binom{mn}{\mathbf{i}}\binom{mn}{\mathbf{j}}}{\binom{2mn}{\mathbf{i}+\mathbf{j}}}\int_\Omega B^{2mn}_{\mathbf{i}+\mathbf{j}}(\mathbf{s})\,\mathrm{d}\mathbf{s} \\ &= \frac{\binom{mn}{\mathbf{i}}\binom{mn}{\mathbf{j}}}{\binom{2mn}{\mathbf{i}+\mathbf{j}}}\frac{1}{(2mn+1)(2mn+2)}. \end{aligned}$$

We may define the matrix $\mathbf{M}$ by

$$\mathbf{M} = \mathbf{D}^T\mathbf{A}\mathbf{D}. \tag{14}$$

Then, similarly to the original approach, an SVD of $\mathbf{M}$ will give us a candidate for a weak approximate implicitization of $\mathbf{p}(\mathbf{s})$. We again choose the vector corresponding to the smallest singular value for the best candidate. The construction of $\mathbf{M}$ is summarized as follows:

Proposition 2. *The* $\binom{m+3}{3} \times \binom{m+3}{3}$ *matrix* $\mathbf{M}$ *formed in weak approximate implicitization of triangular Bézier surfaces can be built by the matrix multiplication* $\mathbf{D}^T\mathbf{A}\mathbf{D}$, *where the matrix* $\mathbf{A}$ *depends only on* m *and* n.

Since $\mathbf{A}$ is only dependent on the degrees m and n, it could in fact be pre-calculated, meaning the construction of $\mathbf{M}$ is reduced to making two matrix multiplications.

This method may be particularly useful when combining the original and weak approximations in order to remove unwanted branches, as the $\mathbf{D}$ matrix must already be calculated. By combining the best approximations from the original and weak forms, we will obtain another approximation with a high convergence rate. Since both the approximations will be 'good' in the area of interest, but may have different branches, the combination may remove these unwanted branches.

For a detailed discussion of the relationship between the weak and original forms of approximate implicitization, we refer the reader to [5]. Here we simply state the main results:

$$|q(\mathbf{p}(\mathbf{s}))| \leq \frac{1}{\sqrt{\lambda_{\min}}}\|\mathbf{\Sigma}\mathbf{U}\mathbf{D}\mathbf{b}\|,$$

and

$$\sqrt{\int_\Omega \left(q(\mathbf{p}(\mathbf{s}))\right)^2 \,\mathrm{d}\mathbf{s}} \leq \sqrt{\lambda_{\max}}\|\mathbf{D}\mathbf{b}\|,$$

where $\mathbf{\Sigma}$ is a diagonal matrix containing the square roots of the eigenvalues $\lambda_{\min}, \ldots, \lambda_{\max}$ of $\mathbf{A}$, and $\mathbf{A} = \mathbf{U}^T(\mathbf{\Sigma}^2)\mathbf{U}$.

3.3 Numerical Approximation

As the exact integration in weak approximate implicitization can be replaced by a numerical integration, the need for an explicit rational parametric form is removed. Numerical integration only requires that the surface can be evaluated. This allows, for example, procedural surfaces to be approximated. Integration using numerical methods allows for quick building of the $\mathbf{M}$ matrix. In addition, we show that the algorithm exhibits symmetries that further enhance its efficiency. The results of this section can be easily generalized to apply to weak approximate implicitization of rational parametric manifolds in $\mathbb{R}^l$.

In the previous section we constructed $\mathbf{M}$ via matrix multiplications. Perhaps a more natural method to construct $\mathbf{M}$ is to perform the integration using the equation (8). Using this method we obtain an element-wise formula for $\mathbf{M}$, which we can evaluate by making use of (3):

$$m_{\mathbf{i},\mathbf{j}} = \int_\Omega B^m_{\mathbf{i}}(\mathbf{p}(\mathbf{s}))B^m_{\mathbf{j}}(\mathbf{p}(\mathbf{s}))\,\mathrm{d}\mathbf{s} \tag{15}$$

$$= \frac{\binom{m}{\mathbf{i}}\binom{m}{\mathbf{j}}}{\binom{2m}{\mathbf{i}+\mathbf{j}}}\int_\Omega B^{2m}_{\mathbf{i}+\mathbf{j}}(\mathbf{p}(\mathbf{s}))\,\mathrm{d}\mathbf{s}. \tag{16}$$

This method eliminates the need to compute $\mathbf{D}$, but we are now required to evaluate the polynomials $B^{2m}_{\mathbf{i}+\mathbf{j}}(\mathbf{p}(\mathbf{s}))$ in order to use (5) for the integration. This is in comparison to evaluating the expressions $B^m_{\mathbf{i}}(\mathbf{p}(\mathbf{s}))$ required to build $\mathbf{D}$. Due to the lower polynomial degrees involved in the latter, the construction of $\mathbf{M}$ by first computing $\mathbf{D}$ and then applying (14), is preferable for the exact integration. However, equation (16) provides a direct method that would be preferable if using numerical integration, since it avoids the polynomial multiplication.

Inspecting (15) we clearly see that $\mathbf{M}$ is symmetric. However, there exist other symmetries which allow us to avoid repeated calculation of the integrals for each element $m_{\mathbf{i},\mathbf{j}}$. Equation (16) shows that there are in fact only $\binom{2m+3}{3}$ unique integrals required. We can thus pre-calculate these integrals using some chosen numerical integration method:

$$\left(\int_\Omega B^{2m}_{\mathbf{k}}(\mathbf{p}(\mathbf{s}))\right)_{|\mathbf{k}|=2m}. \tag{17}$$

Exploiting these symmetries results in the required number of integrals being proportional to m^3 rather than m^6. This result is summarized in the following proposition:

Proposition 3. *The $\binom{m+3}{3} \times \binom{m+3}{3}$ matrix $\mathbf{M}$ formed in weak approximate implicitization of triangular Bézier surfaces can be built by pre-computing the $\binom{2m+3}{3}$ integrals in (17), and multiplying the relevant integrals with the coefficients $\frac{\binom{m}{\mathbf{i}}\binom{m}{\mathbf{j}}}{\binom{2m}{\mathbf{i}+\mathbf{j}}}$.*

Since the degree of the integrand is $2mn$, it is vital that the numerical integration techniques used, exhibit numerical stability up to high polynomial degrees. For example, approximating a cubic Bézier triangle by a cubic implicit surface, requires the numerical integration of a bivariate polynomial of degree 18.

3.4 Approximating Rational Bézier Triangles

Rational Bézier triangles give extra flexibility in CAGD and are in fact required to be able to represent general quadric surfaces exactly. Although this can be done with rational tensor-product patches, some degeneracy is necessary, and hence singularities are introduced. We will show in this section that the algorithm for approximate implicitization of rational Bézier triangles is only a short extension of the non-rational version. We first introduce the concept of rational Bézier triangles, as described in [7].

A rational Bézier triangle of degree n is defined similarly to the non-rational case as follows:

$$\mathbf{r}(\mathbf{s}) = \sum_{|\mathbf{i}|=n} \mathbf{c}_\mathbf{i} R^n_\mathbf{i}(\mathbf{s}),$$

where

$$R^n_\mathbf{i}(\mathbf{s}) = \frac{w_\mathbf{i} B^n_\mathbf{i}(\mathbf{s})}{\sum_{|\mathbf{i}|=n} w_\mathbf{i} B^n_\mathbf{i}(\mathbf{s})} = \frac{g_\mathbf{i}(\mathbf{s})}{h(\mathbf{s})}.$$

The $w_\mathbf{i}$ denote weights assigned to each control point $\mathbf{c}_\mathbf{i}$. Note that the basis $(R_\mathbf{i}(\mathbf{s}))_{|\mathbf{i}|=n}$ defines a partition of unity, so the original approach to approximate implicitization can be used in a similar way for rational Bézier triangles. In fact, on forming the expression $q(\mathbf{r}(\mathbf{s}))$, we can factor out the denominator, which allows us to consider only the numerator for an exact implicitization [17]. Since the numerator is simply a regular Bézier triangle (albeit with the weights absorbed into the control points), this implies that we can find implicitly defined quadrics from non-rational Bézier triangles. We show this as follows:

$$\begin{aligned} q(\mathbf{r}(\mathbf{s})) &= \sum_{|\mathbf{j}|=m} b_\mathbf{j} B^m_\mathbf{j}(\mathbf{r}(\mathbf{s})) \\ &= \frac{1}{(h(\mathbf{s}))^m} \sum_{|\mathbf{j}|=m} b_\mathbf{j} B^m_\mathbf{j} \left(\sum_{|\mathbf{i}|=n} \mathbf{c}_\mathbf{i} g_\mathbf{i}(\mathbf{s}) \right). \end{aligned} \tag{18}$$

We obtain an exact implicitization if and only if the sum over $|\mathbf{i}| = n$ in (18) is zero; but these are exactly the same conditions for exact implicitization on non-rational Bézier triangles, allowing us to disregard $h(\mathbf{s})$ in the algorithm. That is, we may perform the implicitization on

$$\sum_{|\mathbf{i}|=n} \mathbf{c}_\mathbf{i} g_\mathbf{i}(\mathbf{s}) = \sum_{|\mathbf{i}|=n} \boldsymbol{\gamma}_\mathbf{i} B^n_\mathbf{i}(\mathbf{s}),$$

where $\boldsymbol{\gamma}_\mathbf{i} = w_\mathbf{i} \mathbf{c}_\mathbf{i}$. We may also disregard h for approximate implicitizations, however this will come at some expense to the quality of approximation if the function h has large variations.

3.5 Accuracy in Affine Space and Convergence Rates

The intention of this section is to show why approximate implicitization works, and to state a result about the quality of the approximation. For a more in-depth coverage of these topics see [2].

Recall the definition of approximate implicitization from the beginning of this section. This

definition ensures that the implicit and parametric curves lie close together in affine space. However, by minimizing the algebraic distance, as we did in the algorithm, we cannot necessarily guarantee that the affine error will be small. The affine and algebraic errors are related by the following Taylor expansion of (7):

$$q(\mathbf{p}(\mathbf{s})) + \delta(\mathbf{s})\mathbf{g}(\mathbf{s}) \cdot \nabla q(\mathbf{p}(\mathbf{s})) + \cdots = 0.$$

Suppose we have a polynomial q such that $q(\mathbf{p}(\mathbf{s})) \approx 0$. Then the above equation shows that either $\nabla q(\mathbf{p}(\mathbf{s}))$ or $\delta(\mathbf{s})$ must be small. Certainly, away from singularities, where the gradient $\nabla q(\mathbf{p}(\mathbf{s}))$ does not vanish, $\delta(\mathbf{s})$ will be small, meaning the approximation in affine space is good. This justifies the approach to approximate implicitization outlined above, away from singularities. In the region of singularities, the neighbourhood of the singular point or curve will attract the approximation to the correct shape; however, the singularities themselves may be smoothed out. A clear example of this is the approximation in Figure 2. Here, the approximation is attracted to the non-singular part of the surface and the singular curves are 'smoothed out'. We will consider this example further in Section 4.3.

We can improve the approximation in affine space by performing the approximation over a smaller region of the parametric surface. The convergence rates of approximate implicitization, as the size of the region to be approximated is reduced, have been investigated in [4]. Here we state the result most relevant to this paper; the convergence rate of surfaces in $\mathbb{R}^3$. Given a closed box of diameter h in Ω around a point $\mathbf{s}_0$, we have the convergence rate

$$O\left(h^{\lfloor \frac{1}{6}\sqrt{(9+12m^3+72m^2+132m)}\rfloor - \frac{1}{2}}\right). \tag{19}$$

Here, $\lfloor x \rfloor$ denotes the integer part of x.

4 Examples of Implicitization of Bézier Triangles

In this section we present examples of approximate implicitization of triangular Bézier surfaces. We begin with a simple example that can be calculated by hand, before moving on to more computationally intensive examples. Our first example will find an implicit surface that approximates a single quadratic Bézier triangle.

4.1 A First Example

Recall the definition of $\mathbf{p}_1(\mathbf{s})$, the degenerate quadratic Bézier surface mentioned in Section 2.3 and pictured in Figure 1(a). The control points also form a tetrahedron over which we can define the barycentric coordinate system. Using these barycentric coordinates, the patch is described by,

$$\mathbf{p}_1(\mathbf{s}) = (B^2_{200}(\mathbf{s}), B^2_{020}(\mathbf{s}), B^2_{002}(\mathbf{s}), B^2_{011}(\mathbf{s}) + B^2_{101}(\mathbf{s}) + B^2_{110}(\mathbf{s})). \tag{20}$$

For this example, we choose to approximate $\mathbf{p}_1(\mathbf{s})$ by a quadratic implicit surface, in order to keep the matrix $\mathbf{D}$ to a manageable size. However, an exact implicitization in fact requires an implicit surface of degree four.

A trivariate polynomial of degree two, represented in the tetrahedral Bernstein basis can be written as follows:

$$q(\mathbf{u}) = \sum_{|\mathbf{i}|=2} b_{\mathbf{i}} B^2_{\mathbf{i}}(\mathbf{u}),$$

for barycentric coordinates $\mathbf{u}$.

Now, to construct the 15×10 matrix $\mathbf{D}$, we simply expand the expression (9), for each of the basis functions $(B^2_{\mathbf{i}}(\mathbf{u}))_{|\mathbf{i}|=2}$ and write the resulting coefficients in the columns of $\mathbf{D}$. We use the lexicographical ordering system to relate the entries of the matrix, to the multi-indices.

The first column in the matrix $\mathbf{D}$ contains the coefficients of $B^2_{2000}(\mathbf{p}_1(\mathbf{s}))$ which by the definition (20) and the product rule (3) is equal to $(B^2_{200}(\mathbf{s}))^2 = B^4_{400}(\mathbf{s})$. The first column of $\mathbf{D}$ is thus the vector of coefficients that are all zero except for the first:

$$(1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)^T.$$

Similarly, the second column is calculated by expanding $B^2_{1100}(\mathbf{p}_1(\mathbf{s}))$ giving

$$(0, 0, 0, \frac{1}{3}, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)^T.$$

Continuing in this way we get the matrix,

$$\mathbf{D} = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{2}{3} \\
0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{2}{3} \\
0 & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & \frac{2}{3} \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & \frac{2}{3} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & \frac{2}{3} \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & \frac{2}{3} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0
\end{pmatrix}.$$

The correct and accurate construction of this matrix can be confirmed by checking that the rows sum to 1 (see Theorem 4.3 in [3]). As we have proceeded using exact methods, we expect no errors here.

We now perform an SVD on this matrix, and choose the vector $\mathbf{b}$ corresponding to the smallest singular value $\sigma_{\min}$. The singular values of $\mathbf{D}$ are

$$(1.70471, 1.45296, 1.45296, 1.38925, 1.00000,$$
$$1.00000, 1.00000, 0.33333, 0.33333, 0.22984),$$

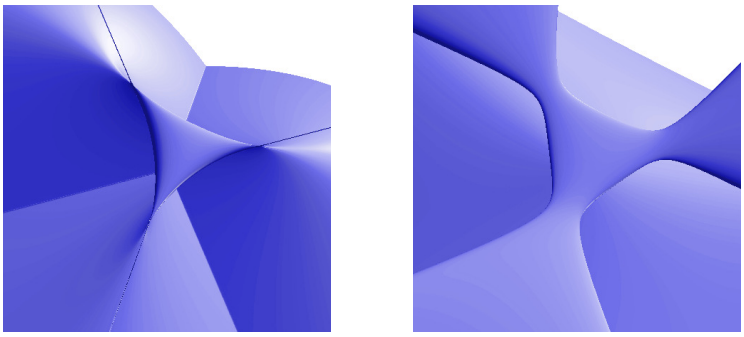


Figure 2: Exact (left) and approximate (right) implicitizations of a quadratic Bézier triangle with singularities $\mathbf{p}_2(\mathbf{s})$ (see Figure 1(b)).

and the normalized vector corresponding to $\sigma_{\min} = 0.22984$ is

$$\begin{aligned}\mathbf{b}_{\text{orig}} = (&0.00000, -0.57062, -0.57062, -0.01616, 0.00000,\\ &-0.57062, -0.01616, 0.00000, -0.01616, 0.14966).\end{aligned}$$

This vector defines a candidate for an approximate implicitization of $\mathbf{p}_1(\mathbf{s})$.

The approach of weak approximate implicitization is equally well suited to this example. As stated previously, property (5) allows us to integrate Bernstein polynomials by summing the coefficients of the Bernstein basis and dividing by a constant factor. For simplicity, we proceed here using the element-wise definition of $\mathbf{M}$, (16):

$$m_{\mathbf{i},\mathbf{j}} \quad = \quad \frac{\binom{2}{\mathbf{i}}\binom{2}{\mathbf{j}}}{\binom{4}{\mathbf{i}+\mathbf{j}}} \int_{\Omega} B^4_{\mathbf{i}+\mathbf{j}}(\mathbf{p}_1(\mathbf{s}))\, \mathrm{d}\mathbf{s}.$$

For example, the first entry for $\mathbf{i} = \mathbf{j} = (2, 0, 0, 0)$, is calculated by making the integration

$$\int_{\Omega} B^4_{4000}(\mathbf{p_1}(\mathbf{s})) = \int_{\Omega} (B_{200})^4 = \int_{\Omega} B^8_{800}.$$

This is a degree eight Bernstein polynomial with first coefficient equal to one and all other coefficients equal to zero. The first value of the matrix is thus $m_{1,1} = 1/90$. The other values of the matrix can be computed similarly to get

$$\mathbf{M} = \begin{pmatrix}
\frac{1}{90} & \frac{1}{1260} & \frac{1}{1260} & \frac{1}{84} & \frac{1}{6300} & \frac{1}{18900} & \frac{23}{18900} & \frac{1}{6300} & \frac{23}{18900} & \frac{4}{675} \\
\frac{1}{1260} & \frac{1}{1575} & \frac{1}{9450} & \frac{23}{9450} & \frac{1}{1260} & \frac{1}{9450} & \frac{23}{9450} & \frac{1}{18900} & \frac{1}{2100} & \frac{31}{9450} \\
\frac{1}{1260} & \frac{1}{9450} & \frac{1}{1575} & \frac{23}{9450} & \frac{1}{18900} & \frac{1}{9450} & \frac{1}{2100} & \frac{1}{1260} & \frac{23}{9450} & \frac{31}{9450} \\
\frac{1}{84} & \frac{23}{9450} & \frac{23}{9450} & \frac{16}{675} & \frac{23}{18900} & \frac{1}{2100} & \frac{31}{4725} & \frac{23}{18900} & \frac{31}{4725} & \frac{67}{3150} \\
\frac{1}{6300} & \frac{1}{1260} & \frac{1}{18900} & \frac{23}{18900} & \frac{1}{90} & \frac{1}{1260} & \frac{1}{84} & \frac{1}{6300} & \frac{23}{18900} & \frac{4}{675} \\
\frac{1}{18900} & \frac{1}{9450} & \frac{1}{9450} & \frac{1}{2100} & \frac{1}{1260} & \frac{1}{1575} & \frac{23}{9450} & \frac{1}{1260} & \frac{23}{9450} & \frac{31}{9450} \\
\frac{23}{18900} & \frac{23}{9450} & \frac{1}{2100} & \frac{31}{4725} & \frac{1}{84} & \frac{23}{9450} & \frac{16}{675} & \frac{23}{18900} & \frac{31}{4725} & \frac{67}{3150} \\
\frac{1}{6300} & \frac{1}{18900} & \frac{1}{1260} & \frac{23}{18900} & \frac{1}{6300} & \frac{1}{1260} & \frac{23}{18900} & \frac{1}{90} & \frac{1}{84} & \frac{4}{675} \\
\frac{23}{18900} & \frac{1}{2100} & \frac{23}{9450} & \frac{31}{4725} & \frac{23}{18900} & \frac{23}{9450} & \frac{31}{4725} & \frac{1}{84} & \frac{16}{675} & \frac{67}{3150} \\
\frac{4}{675} & \frac{31}{9450} & \frac{31}{9450} & \frac{67}{3150} & \frac{4}{675} & \frac{31}{9450} & \frac{67}{3150} & \frac{4}{675} & \frac{67}{3150} & \frac{22}{525}
\end{pmatrix}.$$

The accuracy of the construction of this matrix can be confirmed by checking that the elements

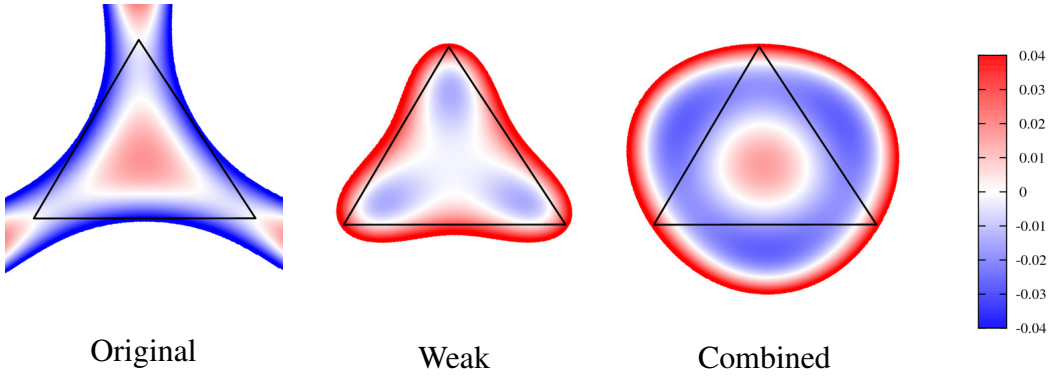


Figure 3: Colour maps showing the algebraic approximation errors $q_{\mathbf{b}}(\mathbf{p}_1(\mathbf{s}))$ over the domain Ω bounded by the triangle. Note that the white parts correspond to intersection curves between the implicit and parametric surfaces.

sum to $\frac{1}{2}$ (see Theorem 2 in [5]).

Again, performing an SVD on this matrix and choosing the vector corresponding to the smallest singular value will define an implicit equation that is a candidate for approximation:

$$\begin{aligned}\mathbf{b}_{\text{weak}} = (&0.03985, 0.56837, 0.56837, -0.09313, 0.03985,\\ &0.56837, -0.09313, 0.03985, -0.09313, -0.00859).\end{aligned}$$

Although this simple example has no extraneous branches, in order to illustrate the possibility of modelling the shape of the approximation, we include a combined approximation. This is obtained by summing the coefficients of the original and weak approximations and renormalizing:

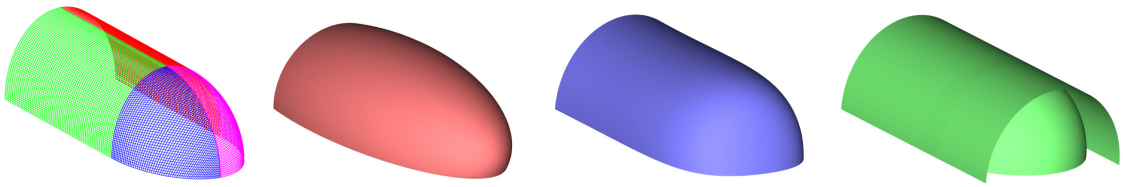
$$\begin{aligned}\mathbf{b}_{\text{comb}} = (&0.11496, -0.00652, -0.00652, -0.31523, 0.11496,\\ &-0.00652, -0.31523, 0.11496, -0.31523, 0.81371).\end{aligned}$$

Figure 3 shows the algebraic distance between the parametric and approximate implicit surfaces. The three approximations exhibit different behaviour with regard to where the surfaces intersect and the positions of the maximum error. This illustrates the possibility of modelling the surfaces to obtain certain characteristics. Alternative approximations could also be formed by taking different combinations of the two surfaces, by combining approximations corresponding to other singular values in the SVD, or by adding constraints to the algebraic equation.

When constructing this example we ensured that the corners of the Bézier triangle were reused as vertices in the tetrahedral barycentric coordinate system, with the remaining fourth vertex positioned symmetrically with respect to these three points. This symmetry is reflected in the intersection curves between the triangular Bézier surfaces and the approximations in Figure 3. In this example, the original approximate implicitization intersects the corners of the triangular Bézier surface; however, the interpolation is special to this case. It is easy to construct examples with the same collocation of surface corners and tetrahedral vertices where the approximate implicit generated by the original approach does not intersect the corners of

the triangular Bézier surface.

4.2 Approximation of Several Patches with One Implicit Surface



(a) parametric form (b) quadratic approximation (c) cubic approximation (d) quartic approximation

Figure 4: Implicit approximations of the surface described in Section 4.2. Note that the quartic approximation, which is an exact implicitization up to rounding error, is defined by the product of two polynomials and hence extra branches are present.

In many circumstances it may be desirable to approximate several surface patches simultaneously, by a single implicit surface. This is possible using either the original or weak methods [2, 5].

Suppose we have several parametric surfaces $\mathbf{p}_1(\mathbf{s}), \ldots, \mathbf{p}_r(\mathbf{s})$. To find an implicit surface that approximates all these surfaces we may proceed as before to build matrices $\mathbf{D}_i$ corresponding to the individual manifolds $\mathbf{p}_i(\mathbf{s})$. However, before performing the SVD, we stack the matrices to define

$$\mathbf{D} = \begin{pmatrix} \mathbf{D}_1 \\ \vdots \\ \mathbf{D}_r \end{pmatrix}.$$

Using the weak form we build matrices $\mathbf{M}_i$ corresponding to the manifolds $\mathbf{p}_i(\mathbf{s})$, but instead of stacking, we sum the matrices to form

$$\mathbf{M} = \sum_{i=1}^{r} \mathbf{M}_i.$$

Performing an SVD on $\mathbf{M}$ then defines the weak approximation.

In fact, we are not restricted to approximating surfaces of the same type. There is also the possibility to simultaneously approximate points, curves and surfaces with different parametric forms. To exemplify this we approximate a surface defined by two rational tensor-product Bézier patches describing a half-cylinder, and two rational Bézier triangles describing a quarter-sphere, as pictured in Figure 4(a). The quadratic, cubic and quartic approximations displayed in Figure 4 demonstrate some interesting properties of implicit representations. The quadratic approximation, which is in fact described by an ellipsoid, is clearly quite different from the parametric surface and, for most purposes, would not be a sufficient approximation. The cubic approximation is visually what we expect to see, and does indeed provide a close approximation. When we increase the degree to four, as expected, we obtain an exact implicit representation, though this is defined by the product of two polynomials which describe the cylindrical and

Degree m	1	2	3	4
$\sigma_{\min}$ of $\mathbf{p}_1(\mathbf{s})$	1.0	0.22984	0.047868	0.0
$\sigma_{\min}$ of $\mathbf{p}_2(\mathbf{s})$	1.0	0.62773	0.31596	0.0

Table 1. Difference in the smallest singular values of $\mathbf{D}$ for a Bézier triangle with singularities $(\mathbf{p}_2(\mathbf{s}))$ and without singularities $(\mathbf{p}_1(\mathbf{s}))$.

spherical parts separately. Consequently, when visualizing the surface we see extra branches that are not present in the parametric representation.

4.3 Approximate Implicitization of Surfaces with Singularities

A simple example of a quadratic Bézier triangle $\mathbf{p}_2(\mathbf{s})$ that exhibits singularities is constructed by taking the three corner control points to be at the Cartesian origin $(0, 0, 0)$, and the three central control points to be at (1,0,0), (0,1,0) and (0,0,1). This is pictured in Figure 1(b). An exact quartic implicitization and an approximate cubic implicitization of $\mathbf{p}_2(\mathbf{s})$ are pictured in Figure 2. We will now compare the approximations of this example with the example from Section 4.1, to see how the singular surface suffers from worse approximations. Table 1 lists the singular values for implicit approximations up to degree four, obtained by the original method. Both of the surfaces require degree four for an exact implicitization. However, the singular values of $\mathbf{p}_1(\mathbf{s})$ are much smaller for the quadratic and cubic approximations, indicating better approximations.

5 Conclusion

This paper described how the original and weak methods of approximate implicitization can be applied to triangular Bézier surfaces. It presented examples which exhibit properties of the various approaches to approximate implicitization. It also highlighted ways in which to improve the efficiency of the algorithm in the numerical case, by exploiting symmetries in the calculations.

Acknowledgements

This work has been supported by the European Community under the Marie Curie Initial Training Network "SAGA - Shapes, Geometry and Algebra" Grant Agreement Number 21458, and by the Research Council of Norway through the IS-TOPP program. We would like to thank Johan Simon Seland at SINTEF for making his real-time algebraic surface visualization system available to us (Figure 2). We have also made use of the software Axel, `http://axel.inria.fr/` (Figures 1 and 4).

Bibliography

[1] Jules Bloomenthal and Brian Wyvill, editors. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[2] T. Dokken. *Aspects of intersection algorithms and approximations*. PhD thesis, University of Oslo., 1997.

[3] T. Dokken. Approximate implicitization. In *Mathematical methods for curves and surfaces*, pages 81–102. Vanderbilt Univ. Press, Nashville, TN, 2001.

[4] T. Dokken and J. B. Thomassen. Overview of approximate implicitization. In *Topics in algebraic geometry and geometric modeling*, volume 334, pages 169–184. Amer. Math. Soc., Providence, RI, 2003.

[5] T. Dokken and J. B. Thomassen. Weak approximate implicitization. In *SMI '06: Proceedings of the IEEE International Conference on Shape Modeling and Applications 2006*, page 31, Washington, DC, USA, 2006. IEEE Computer Society.

[6] G Farin. Triangular Berstein-Bézier patches. *Comput. Aided Geom. Des.*, 3(2):83–127, 1986.

[7] Gerald Farin. *Curves and surfaces for CAGD: a practical guide*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[8] Rida T. Farouki, T. N. T. Goodman, and Thomas Sauer. Construction of orthogonal bases for polynomials in Bernstein form on triangular and simplex domains. *Computer Aided Geometric Design*, 20(4):209 – 230, 2003.

[9] Christoph M. Hoffman. Implicit curves and surfaces in CAGD. *IEEE Comput. Graph. Appl.*, 13(1):79–88, 1993.

[10] Bert Jüttler, Pavel Chalmovianský, Mohamed Shalaby, and Elmar Wurm. Approximate algebraic methods for curves and surfaces and their applications. In *SCCG '05: Proceedings of the 21st spring conference on Computer graphics*, pages 13–18, New York, NY, USA, 2005. ACM.

[11] Vaughan Pratt. Direct least-squares fitting of algebraic surfaces. *SIGGRAPH Comput. Graph.*, 21(4):145–152, 1987.

[12] Martin Reimers and Johan Seland. Ray casting algebraic surfaces using the frustum form. *Comput. Graph. Forum*, 27(2):361–370, 2008.

[13] T. W. Sederberg, D. C. Anderson, and R. N. Goldman. Implicit representation of parametric curves and surfaces. *Computer Vision, Graphics, and Image Processing*, 28(1):72 – 84, 1984.

[14] Thomas W. Sederberg and Falai Chen. Implicitization using moving curves and surfaces. In *SIGGRAPH 95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 301–308, New York, NY, USA, 1995. ACM.

[15] Thomas W. Sederberg, Jianmin Zheng, Kris Klimaszewski, and Tor Dokken. Approximate implicitization using monoid curves and surfaces. *Graphical Models and Image Processing*, 61(4):177 – 198, 1999.

[16] Gabriel Taubin. Estimation of planar curves, surfaces, and nonplanar space curves defined by implicit equations with applications to edge and range image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(11):1115–1138, 1991.

[17] Jan B. Thomassen. Self-intersection problems and approximate implicitization. In *Computational Methods for Algebraic Spline Surfaces*, pages 155–170. Springer Berlin Heidelberg, 2005.

[18] E. Wurm and B. Jüttler. Approximate implicitization via curve fitting. In *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 240–247, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

Paper II: Approximate implicitization using linear algebra

Oliver J.D. Barrowclough and Tor Dokken

In *Journal of Applied Mathematics*, 2012. doi:10.1155/2012/293746

Abstract: In this paper we consider a family of algorithms for approximate implicitization of rational parametric curves and surfaces. The main approximation tool in all of the approaches is the singular value decomposition, and they are therefore well suited to floating point implementation in computer aided geometric design (CAGD) systems. We unify the approaches under the names of commonly known polynomial basis functions, and consider various theoretical and practical aspects of the algorithms. We offer new methods for a least squares approach to approximate implicitization using orthogonal polynomials, which tend to be faster and more numerically stable than some existing algorithms. We propose several simple propositions relating the properties of the polynomial bases to their implicit approximation properties.

1 Introduction

Implicitization algorithms have been studied in both the CAGD and algebraic geometry communities for many years. Traditional approaches to implicitization have focused on exact methods such as Gröbner bases, resultants and moving curves and surfaces, or syzygies [24]. Approximate methods that are particularly well suited to floating point implementation have also emerged in the past 25 years [3, 6, 7, 9, 22]. These methods are closely related to the algorithms we present, however, those that fit most closely into the framework of this paper include [7, 10, 12, 26].

Implicitization is the conversion of parametrically defined curves and surfaces into curves and surfaces defined by the zero set of a single polynomial. Exact implicit representations of rational parametric manifolds often have very high polynomial degrees, which can cause numerical instabilities and slow floating point calculations. In cases where the geometry of the manifold is not sufficiently complicated to justify this high degree, approximation is often desirable. Moreover, for CAGD systems based on floating point arithmetic, exact implicitization methods are often unfeasible due to performance issues. The methods we present, attempt to find 'best fit' implicit curves or surfaces of a given degree m (the definition of 'best fit' varies with regard to the chosen method of approximation). One important property of all the algorithms is that they are guaranteed to give exact implicitizations for sufficiently high implicit

degrees, up to numerical stability. In addition, some of the methods are also suitable for implementation in exact arithmetic, hence constituting alternative methods for exact implicitization.

For simplicity of notation, we proceed for the majority of the paper to describe the implicitization of curves. In Sections 2, 3 and 4 we introduce the notation and review existing methods. In Section 5 we present a new method for approximate implicitization using orthogonal polynomials and prove a theoretical relation to the previous methods. Implementations of the methods using different basis functions will be presented in Section 6 and a qualitative comparison and discussion given in Section 7. Finally, the generalization to both tensor-product and triangular surfaces will be covered in Section 8.

2 Preliminaries

A parametric curve in $\mathbb{R}^2$ is given by $\mathbf{p}(t) = (p_1(t), p_2(t))$ where p_1 and p_2 are functions in t on some parameter domain Ω. Of particular importance both in CAGD and classical algebraic geometry are rational parametric curves (i.e., where p_1 and p_2 are rational functions). In the majority of this paper we will thus restrict our attention to planar rational curves, where the domain of interest is $\Omega = [0, 1]$. In order to use polynomial bases in our construction, we can use the representation of the curves in the projective plane $\mathbb{P}^2$. For a rational parametric curve $\mathbf{p}(t) = (g_1(t)/h(t), g_2(t)/h(t))$ in $\mathbb{R}^2$, where g_1, g_2 and h are polynomials, we thus use the homogeneous description

$$\mathbf{p}(t) = (g_1(t), g_2(t), h(t)).$$

All the methods to be described require a choice of degree m and a choice of basis $(q_k(\mathbf{u}))_{k=1}^M$, for the implicit polynomial. Here M is defined as the number of basis functions in a polynomial of total degree m. Thus, for a general bivariate polynomial, we have $M = \binom{m+2}{2}$. Any polynomial q can be written in terms of such a basis by choosing coefficients $\mathbf{b} = (b_k)_{k=1}^M$:

$$q(\mathbf{u}) = \sum_{k=1}^{M} b_k q_k(\mathbf{u}). \tag{1}$$

The choice of implicit basis is an important factor which has implications for both the stability of the algorithms and the quality of the approximations. However, most of the work in this paper is independent of the choice of implicit basis. In $\mathbb{R}^2$, a good choice is the Bernstein basis in a barycentric coordinate system defined over a triangle containing the parametric curve. For curves in $\mathbb{P}^2$, we use the homogeneous Bernstein basis given by

$$q_{\mathbf{k}}(u, v, w) = \binom{m}{k_1, k_2, k_3} u^{k_1} v^{k_2} w^{k_3}, \quad \text{for } |\mathbf{k}| = k_1 + k_2 + k_3 = m,$$

where u, v and w denote the homogeneous coordinates and $\mathbf{k} = (k_1, k_2, k_3)$ denotes a multi-index. We order the basis by letting q_k correspond to $q_{\mathbf{k}}$ for $k = 1, \ldots, M$, where $k = k(\mathbf{k})$ denotes lexicographical order on the indices k_1, k_2 and k_3. Unless otherwise stated, we will assume that the implicit basis $(q_k(\mathbf{u}))_{k=1}^M$ is the Bernstein basis. In particular, it forms a partition

Algebraic degree m	1	2	3	4	5	6	7	8
Convergence rate k	2	5	9	14	20	27	35	44

Table 1: Convergence rates for approximate implicitization of sufficiently smooth parametric curves in $\mathbb{R}^2$ by algebraic curves of degree m, given by $k = \frac{1}{2}(m+1)(m+2) - 1$.

Algebraic degree m	1	2	3	4	5	6	7	8
Convergence rate k	2	3	5	7	10	12	14	17

Table 2: Convergence rates for approximate implicitization of sufficiently smooth parametric surfaces in $\mathbb{R}^3$ by algebraic surfaces of degree m, given by $k = \lfloor \frac{1}{6}\sqrt{9 + 12m^3 + 72m^2 + 132m} - \frac{1}{2} \rfloor$.

of unity

$$\sum_{k=1}^{M} q_k(\mathbf{u}) \equiv 1.$$

The choice of degree m determines the number of degrees of freedom the implicit curve will have. If the chosen degree m is sufficiently high, all the algorithms will produce exact results, up to numerical stability. If the degree is lower than necessary, approximations are produced.

Since we are searching for implicit representations and want to avoid the trivial solution $q \equiv 0$, we add a normalization constraint to q in the approximation. How best to make this choice has been discussed by several authors (see [22] for an overview). However, as we use the singular value decomposition (SVD) as the means of approximation, our results are given with the quadratic normalization $\|\mathbf{b}\|_2 = 1$.

The techniques in this paper focus on minimization of the objective function $q \circ \mathbf{p}$ over the space of polynomials $\{q : \|\mathbf{b}\|_2 = 1\}$, where q is defined by (1) in a fixed implicit basis. Such a minimization, although not directly minimizing the Hausdorff distance between the implicit and parametric curves, is closely related to the geometric approximation problem. It has been shown that minimization of $q \circ \mathbf{p}$ gives excellent results in geometric space away from singularities [9].

3 Approximate implicitization - the original approach

In 1997 a class of techniques for approximate implicitization of rational parametric curves, surfaces and hypersurfaces was introduced in [9]. The approximation quality of the techniques was substantiated by a general proof that the methods exhibit very high convergence rates, as shown in Tables 1 and 2. Extensions of this original approach, all of which inherit these high convergence rates, form the basis of this paper.

The guiding principle behind these methods is to find a polynomial q which minimizes the maximal algebraic error in a given parameter domain Ω. That is, in the given implicit basis $(q_k)_{k=1}^M$, to find the coefficients $\mathbf{b} = (b_k)_{k=1}^M$ which solve

$$\min_{\|\mathbf{b}\|=1} \max_{t \in \Omega} |q(\mathbf{p}(t))|. \tag{2}$$

The solution to this problem, which we call the minimax (or uniform) approximation, is not easy to find exactly. However, approximations to the minimax solution can be found directly, using linear algebra.

We notice that the expression $q(\mathbf{p}(t))$ is a univariate polynomial of degree mn in t. We can thus approach the problem by first factorizing the error expression in a polynomial basis $\boldsymbol{\alpha}(t) = (\alpha_j(t))_{j=1}^{L}$, where $L = mn + 1$, as follows:

$$q(\mathbf{p}(t)) = \boldsymbol{\alpha}(t)^T \mathbf{D}_\alpha \mathbf{b}, \tag{3}$$

where $\mathbf{D}_\alpha$ is a matrix whose columns are the coefficients of $q_k(\mathbf{p}(t))$ expressed in the $\boldsymbol{\alpha}$-basis and $\mathbf{b}$ is the unknown vector of implicit coefficients. Now, we have

$$\begin{aligned}
\min_{\|\mathbf{b}\|=1} \max_{t\in\Omega} |q(\mathbf{p}(t))| &= \min_{\|\mathbf{b}\|=1} \max_{t\in\Omega} |\boldsymbol{\alpha}(t)^T \mathbf{D}_\alpha \mathbf{b}|, \\
&\leq \max_{t\in\Omega} \|\boldsymbol{\alpha}(t)\|_2 \min_{\|\mathbf{b}\|=1} \|\mathbf{D}_\alpha \mathbf{b}\|_2, \\
&= \max_{t\in\Omega} \|\boldsymbol{\alpha}(t)\|_2 \, \sigma_{\min}, \qquad (4)
\end{aligned}$$

giving an upper bound on the maximal algebraic error, dependent on the choice of basis. Here, we have used the fact that

$$\min_{\|\mathbf{b}\|_2=1} \|\mathbf{D}_\alpha \mathbf{b}\|_2 = \sigma_{\min},$$

where $\sigma_{\min}$ is the smallest singular value of $\mathbf{D}_\alpha$. We thus choose $\mathbf{b} = \mathbf{v}_{\min}$, the right singular vector corresponding to $\sigma_{\min}$, as an approximate solution to the problem. The other singular vectors corresponding to larger singular values also give candidates for approximation that generally decrease in quality as the singular values increase [12]. It is important to note that the value of $\sigma_{\min}$ is dependent on the choice of basis $\boldsymbol{\alpha}$.

In this paper we use the following 'normalization' to compare the approximation qualities of different polynomial bases:

$$\max_{t\in\Omega} \|\boldsymbol{\alpha}(t)\|_2 = 1. \tag{5}$$

It should be noted that bases with different scaling coefficients on the individual basis functions will produce different results. For example, the standard Legendre basis, where each basis function $(P_j)_{j=1}^{L}$ has the normalization $P_j(1) = 1$, will produce quantitatively different results to the Legendre basis normalized with respect to the L_2-inner product. This choice of scaling is somewhat arbitrary, but experience shows that small differences in the scaling result in small differences in the approximation. Thus, for the bases we study, standard choices will be made.

Given a choice of basis functions $\boldsymbol{\alpha} = (\alpha_i)_{i=1}^{L}$, we summarize the general approach of this section in the following algorithm:

Algorithm 1. *Input: a rational parametric curve $\mathbf{p}(t)$ of degree n on the interval $[0, 1]$, and a degree m for the implicit polynomial:*

1. *For each basis function $(q_k)_{k=1}^{M}$, compute the vector $\mathbf{d}_k = (d_{j,k})_{j=1}^{L}$ of coefficients such that $q_k(\mathbf{p}(t)) = \sum_{j=1}^{L} d_{j,k}\alpha_j(t)$,*
2. *Construct a matrix $\mathbf{D}_\alpha = (\mathbf{d}_k)_{k=1}^{M}$ from the column vectors $\mathbf{d}_k$,*

3. *Perform an SVD* $\mathbf{D}_\alpha = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, *and select* $\mathbf{b} = \mathbf{v}_{\min}$, *the right singular vector corresponding to the smallest singular value* $\sigma_{\min}$ *as an approximate solution.*

This algorithm is known as the original method in the $\boldsymbol{\alpha}$-basis; however, for this paper, we will refer to it simply as the $\boldsymbol{\alpha}$-method for an arbitrary basis $\boldsymbol{\alpha}$.

4 Weak approximate implicitization

Two approaches to approximate implicitization by continuous least squares minimization of the objective function were introduced simultaneously in 2001 in [7, 11], and further developed in [12]. These methods perform minimization in the weighted L_2-inner product:

$$\min_{\|\mathbf{b}\|_2=1} \langle q \circ \mathbf{p}\ ,\ q \circ \mathbf{p} \rangle_w = \min_{\|\mathbf{b}\|_2=1} \int_\Omega w(t) q(\mathbf{p}(t))^2 \,\mathrm{d}t, \tag{6}$$

where $w(t)$ is some weight function defined on the domain of approximation Ω.

After choosing a basis for the implicit representation we obtain a linear algebra problem as before:

$$\min_{\|\mathbf{b}\|_2=1} \mathbf{b}^T \mathbf{M}_w \mathbf{b}, \tag{7}$$

where $\mathbf{M}_w = (m_{k,l})_{k=1,l=1}^{M,M}$ is given by

$$m_{k,l} = \langle q_k \circ \mathbf{p}\ ,\ q_l \circ \mathbf{p} \rangle_w. \tag{8}$$

This approach eliminates the need for a choice of basis, but a choice of weight function is necessary. The standard approach in [7, 11] has been to take $w(t) \equiv 1$. Later we will discuss the benefits of choosing the Chebyshev weight function on $[0, 1]$, $w(t) = 1/\sqrt{t(1-t)}$, instead.

This problem, unlike the minimax problem, can be solved directly if the parametric components are integrable. We simply take $\mathbf{b} = \mathbf{v}_{\min}$, the eigenvector corresponding to the smallest eigenvalue of $\mathbf{M}_w$. Since the matrix is symmetric, it has orthonormal eigenvectors. Similarly to the previous method, eigenvectors corresponding to larger eigenvalues give gradually degenerating approximations.

We summarize this algorithm, for a given weight function w, as follows:

Algorithm 2. *Input: a parametric curve* $\mathbf{p}(t)$ *on the interval* $[0, 1]$, *and a degree* m *for the implicit polynomial:*

1. *Construct a matrix* $\mathbf{M}_w$ *by performing the integrals according to*

$$m_{k,l} = \langle q_k \circ \mathbf{p}, q_l \circ \mathbf{p} \rangle_w$$

   for $k, l = 1, \ldots, M$,

2. *Compute the eigendecomposition* $\mathbf{M}_w = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$,

3. *Select* $\mathbf{b} = \mathbf{v}_{\min}$, *the eigenvector corresponding to the smallest eigenvalue* $\lambda_{\min}$.

Algorithms following the procedure above are known under different names in the literature; weak approximate implicitization in [12], numerical implicitization in [7] and the eigenvalue method in [13]. For the rest of this paper we will call it the weak method.

As previously mentioned, the methods of this section are suitable for either exact or approximate implicitization. They can be performed using either symbolic or numerical integration, however the former is generally only required when performing exact implicitizations in exact precision. For applications where floating point precision is sufficient, numerical quadrature rules provide a much faster alternative. The methods also have wide generality since they can be applied to any parametric forms with integrable components, not only rational parametric forms. There are, however, some significant disadvantages in choosing this method in practice. Firstly, due to the high degree of the integrand, the integrals can take a relatively long time to evaluate, even when numerical quadrature rules are used. Secondly, and more importantly, the condition numbers of the matrices $\mathbf{M}_w$ can be significantly larger than the condition numbers of the $\mathbf{D}_\alpha$ matrices from the previous section, leading to issues with numerical stability.

Since $\mathbf{M}_w$ is a symmetric positive (semi) definite matrix, it has a decomposition $\mathbf{M}_w = \mathbf{K}^T\mathbf{K}$, where the singular values of $\mathbf{K}$ are the square roots of the singular values of $\mathbf{M}_w$. This decomposition is not unique; however, in the next section we show that it is possible to construct such a matrix directly, without first computing $\mathbf{M}_w$. The condition number of $\mathbf{K}$ will be the square root of the condition number of $\mathbf{M}_w$, hence we obtain the solution to the least squares problem in a more stable manner. Example 5.1 demonstrates how the lack of stability in the weak method compares to the new method described in the following section.

5 Approximate implicitization using orthonormal bases

To make the connection between the original method and the weak method in the previous section, we consider the factorization (3). We can then express $\mathbf{M}_w$ in terms of $\mathbf{D}_\alpha$ and a new matrix $\mathbf{A}$ [12]. That is, we get

$$\int_\Omega w(t)q(\mathbf{p}(t))^2 \,\mathrm{d}s = \mathbf{b}^T\mathbf{D}_\alpha^T\mathbf{A}\mathbf{D}_\alpha\mathbf{b} \tag{9}$$

where $\mathbf{A} = (a_{i,j})_{i=1,j=1}^{L,L}$ is given by $a_{i,j} = \langle\alpha_i, \alpha_j\rangle_w$. Note that $\mathbf{A}$ is a Gramian matrix in the $\boldsymbol{\alpha}$-basis on the weighted L^2-inner product. This gives us a clue as to how to improve the weak method by the use of orthonormal bases. A polynomial basis $(\alpha_i)_{i=1}^L$ is said to be orthonormal with respect to the weighted L^2-inner product $\langle\cdot,\cdot\rangle_w$, if

$$\langle\alpha_i, \alpha_j\rangle_w = \delta_{ij}, \qquad \text{for all } i, j = 1, \ldots, L,$$

with δ_{ij} denoting the Kronecker delta.

Theorem 1. *Let $\boldsymbol{\alpha}$ be a polynomial basis, orthonormal with respect to the given inner product $\langle\cdot,\cdot\rangle_w$, and $\mathbf{D}_\alpha$ and $\mathbf{M}_w$ be defined by (3) and (8) respectively. Then $\mathbf{M}_w = \mathbf{D}_\alpha^T\mathbf{D}_\alpha$.*

Proof. By (9) and the definition of $\mathbf{M}_w$, we have $\mathbf{M}_w = \mathbf{D}_\alpha^T\mathbf{A}\mathbf{D}_\alpha$ for any basis $\boldsymbol{\alpha}$. But since $\boldsymbol{\alpha}$ is orthonormal with respect to the inner product $\langle\cdot,\cdot\rangle_w$, we have $a_{i,j} = \langle\alpha_i, \alpha_j\rangle_w = \delta_{i,j}$ (i.e., $\mathbf{A} = \mathbf{I}$). □

We notice that the right singular vectors of $\mathbf{D}_\alpha$ are exactly the orthonormal eigenvectors of $\mathbf{D}_\alpha^T\mathbf{D}_\alpha$, and the singular values of $\mathbf{D}_\alpha$ are the non-negative square roots of the eigenvalues of $\mathbf{D}_\alpha^T\mathbf{D}_\alpha$. Thus, with Theorem 1 in mind, we see that if the $\boldsymbol{\alpha}$-basis is orthonormal with respect to w, then the results of the original method and the weak method are the same[1]. That is, $\mathbf{D}_\alpha$ is a candidate for the matrix $\mathbf{K}$ from the previous section. Such a matrix, as defined by (3), can also be given elementwise by

$$d_{j,k} = \int_\Omega w(t)\alpha_j(t)q_k(\mathbf{p}(t))\,\mathrm{d}t. \tag{10}$$

In practice, there often exist algorithms for computing coefficient expansions in orthogonal bases that are more efficient than using the elementwise definition. We will mention later how the Chebyshev and Legendre methods can utilize algorithms based on the fast Fourier transform (FFT) to generate the $\mathbf{D}_\alpha$ matrices.

We should note that the matrix $\mathbf{D}_\alpha$ is of dimension $L \times M$, compared with the $M \times M$ matrix $\mathbf{M}_w$. For $n \geq 3$, we have $L \geq M$, so finding the singular vectors of $\mathbf{D}_\alpha$ may be more computationally expensive than computing the eigenvectors of $\mathbf{M}_w$. However, since the matrix construction is usually the dominant part of the algorithm, these differences do not affect the overall complexity of the algorithms. Moreover, the increase in accuracy more than justifies any small increase in computational complexity in part of the algorithm.

5.1 Example

In order to compare the numerical stability of the two approaches to least squares minimization of the objective function, we turn to a familiar example; exact implicitization of a rational parametric circular arc, which is defined in projective space by

$$\mathbf{p}(t) = \big(p_1(t), p_2(t), h(t)\big) = \left(2t, 1-t^2, 1+t^2\right), \quad t \in [0,1].$$

We perform the implicitization using the homogeneous Bernstein basis functions $q_k(u,v,w) \in \{u^2, 2uv, 2uw, v^2, 2vw, w^2\}$, for $k = 1,\ldots,6$, and degree $m = 2$. If performed using exact arithmetic, the implicitization vector given by both the weak and orthonormal basis methods is

$$\mathbf{b} = (3^{-1/2}, 0, 0, 3^{-1/2}, 0, -3^{-1/2}).$$

However, performing the algorithms in double precision, we obtain the results

$$\begin{aligned}\mathbf{b}_{\text{weak}} = (&0.577350269173099, -1.63\times10^{-11}, 2.19\times10^{-11},\\ &0.577350269178602, 1.87\times10^{-11}, -0.577350269217176)\\ \mathbf{b}_{\text{orth}} = (&0.577350269189627, 5.52\times10^{-16}, -8.47\times10^{-16},\\ &0.577350269189626, -4.16\times10^{-16}, -0.577350269189625)\end{aligned}$$

[1]Note that we do not specify that the orthonormal basis must be a polynomial basis, only a basis for the relevant space of functions. For example, curves defined by trigonometric polynomials can be treated in a similar way, since that basis is orthogonal.

for the weak and the orthogonal basis methods respectively[2]. Examining the respective relative errors (in the infinity norm)

$$\begin{aligned} \|\mathbf{b}_{\text{weak}} - \mathbf{b}\|_\infty / \|\mathbf{b}\|_\infty &= 4.77 \times 10^{-11}, \\ \|\mathbf{b}_{\text{orth}} - \mathbf{b}\|_\infty / \|\mathbf{b}\|_\infty &= 1.73 \times 10^{-15}, \end{aligned}$$

we see that the orthogonal basis method preserves the accuracy much better than the weak method, with the former only preserving approximately 11 digits of accuracy. The orthogonal basis method preserves all but the last the digit of the implicitization, up to double precision. It should be noted that this is an example of implicitization with degree $m = 2$; as the degree is raised, such numerical errors can become more serious. The example in Section 7.2 shows how higher degree implicitizations using the weak method can give unpredictable results.

6 Examples of the original approach with different bases

The previous sections justified why the original approach is preferable to the weak approach in cases where the expression $q(\mathbf{p}(t))$ can be expressed in terms of orthogonal functions. In this section we will look at examples of specific implementations using orthogonal polynomial bases. We will also consider alternative implementations using non-orthogonal bases, which produce approximations to the least squares solution. We state some simple propositions which unify the methods and also consider computational aspects of the algorithms.

6.1 Jacobi polynomial bases

The most commonly used orthogonal polynomial bases in approximation theory are the Legendre basis $(P_j)_{j=1}^L$ and the Chebyshev basis (of the first kind) $(T_j)_{j=1}^L$. These are both special cases of Jacobi polynomials, which are orthogonal with respect to the weight functions defined by

$$w_{\alpha,\beta}(t) = t^\alpha (1-t)^\beta,$$

on the interval $[0, 1]$, and are defined for $\alpha, \beta > -1$. The Legendre and Chebyshev cases are given by $\alpha = \beta = 0$ and $\alpha = \beta = -\frac{1}{2}$ respectively. It is well known that the Chebyshev expansion has an efficient construction via use of the discrete cosine transform (DCT-I) which can be implemented via fast Fourier transform (FFT) [5]. The Legendre expansion can also be constructed efficiently by transforming from the Chebyshev coefficients (in $O(n)$ operations for a degree n polynomial [1]). The implementation of approximate implicitization exploiting the speed of the FFT will be the subject of another paper. Here we will look at the properties of the matrices $\mathbf{D}_P$ for the Jacobi bases, and the properties of the approximations. The following proposition is an analogue of Theorem 4.3 in [10], for Jacobi polynomial bases.

Proposition 2. *Let $\mathbf{D}_P$ be the matrix defined by (3) in a Jacobi polynomial basis $(P_j(t))_{j=1}^L$,*

[2]The orthogonal basis method was implemented with Legendre polynomials, as described in Section 6.1.

for any $\alpha, \beta > -1$, *normalized to* $\|P_j\|_\infty = 1$ *for* $j = 1, \ldots, L$. *Then*

$$\sum_{k=1}^{M} d_{j,k} = \begin{cases} 1, & \text{if } j = 1, \\ 0, & \text{if } j \geq 2. \end{cases}$$

Proof. Since an orthogonal polynomial basis is degree ordered, one of the functions must be identically a non-zero constant, which, by the normalization condition is equal to 1. Consider the vector $\mathbf{b} = (1, \ldots, 1)$, which ensures that $q(\mathbf{p}(t)) = 1$ for all $t \in [0, 1]$, by the partition of unity on the implicit basis $(q_k)_{k=1}^M$. Clearly, only the basis function of degree zero can have a non-zero coefficient. By (3), we have the expansion

$$q(\mathbf{p}(t)) \quad = \quad \sum_{j=1}^{L} P_j(t) \sum_{k=1}^{M} d_{j,k}.$$

But this is equal to 1 if and only if $\sum_{k=1}^M d_{1k} = 1$ and $\sum_{k=1}^M d_{j,k} = 0$ for $j \geq 2$. □

One property of Chebyshev expansions of a continuous function is that the error introduced by truncating the expansion is dominated by first term after the truncation, if the coefficients decay quickly enough [17]. For curves that we wish to approximate (with relatively simple forms), the coefficients do tend to decay quickly, so the coefficients in the lower rows of the matrix tend to be dominated by those above them.

The Chebyshev basis is well known for giving good approximations to minimax problems in approximation theory (see [17] for an overview). This also seems to be the case for approximate implicitization, with the resulting error normally being close to equioscillating. In fact, experiments show that in almost all test cases, the number of roots in the error function given by the Chebyshev method is greater than or equal to the convergence rate, for the given implicit degree m (see Table 1). Thus the Chebyshev method appears to give a 'near best' approximation in the sense that the error normally oscillates a maximum number of times.

Our experience in the choice between Legendre and Chebyshev polynomials is that the difference in approximation quality is minor. Chebyshev expansions are slightly quicker to compute and require less programming effort than their Legendre counterparts [1]. In addition they tend to eliminate the spike in error at the end of the intervals that appears in the Legendre method. However, both algorithms provide efficient and numerically stable methods for (weighted) least squares approximation over the entire interval Ω.

6.2 Discrete approximate implicitization - the Lagrange basis

One of the simplest and fastest implementations of approximate implicitization is to perform discrete least squares approximation of points sampled on the parametric manifold, similar to the methods in [3, 22]. In our setting, this can be implemented as the original method but with the $\boldsymbol{\alpha}$-basis chosen to be the Lagrange basis at the given nodes. The matrix $\mathbf{D}_{\mathrm{L},L}$ defined by (2) in the Lagrange basis of degree $L - 1$ can be given elementwise by

$$d_{j,k} = q_k(\mathbf{p}(t_j)), \quad j = 1, \ldots, L, \text{ and } k = 1, \ldots, M, \tag{11}$$

where $t_j \in \Omega$ are nodes in the parameter domain.

The result of approximate implicitization in the Lagrange basis depends both on the number of points sampled and the density of the point distribution in the parameter domain. Since Lagrange polynomials are neither orthogonal nor degree ordered, they do not solve a least squares problem of type (14). However, we can form a direct relation between the discrete and continuous least squares problems, as follows:

Proposition 3. *Let $\mathbf{p}(t)$ be a planar parametric curve with bounded, piecewise continuous components on the interval $[0,1]$ and let $\mathbf{D}_{\mathrm{L},L}$ be the matrix defined by (11) at uniform nodes. Then the (k,l) element of ${\mathbf{D}_{\mathrm{L},L}}^T\mathbf{D}_{\mathrm{L},L}$ converges to a constant multiple of the (k,l) element of $\mathbf{M}_1$, defined by (8), as the number of samples L is increased.*

Proof. Since each of the parametric components of the curve is bounded and piecewise continuous on the interval and q_k is a polynomial, we know that $q_k \circ \mathbf{p}$ is bounded and piecewise continuous for each k. Let $h_L := 1/(L-1)$. Then, for uniform samples $(t_j)_{j=1}^L$, where $t_j = h_L(j-1)$ in the parameter domain we see that

$$\begin{aligned}
\lim_{L\to\infty} h_L({\mathbf{D}_{\mathrm{L},L}}^T\mathbf{D}_{\mathrm{L},L})_{k,l} &= \lim_{L\to\infty} \sum_{j=1}^{L} h_L q_k(\mathbf{p}(t_j)) q_l(\mathbf{p}(t_j)), \\
&= \int_\Omega q_k(\mathbf{p}(t)) q_l(\mathbf{p}(t)) \, \mathrm{d}t, \\
&= (\mathbf{M}_1)_{k,l},
\end{aligned}$$

for $k,l = 1,\ldots,M$. □

Sampling more points gives ever closer approximations of the true least squares approximation (for any given L, the constant h_L is absorbed into the singular values of the matrix and does not affect the singular vectors). However, as we have seen, the Legendre method can solve the (unweighted) least squares problem exactly, without excessive sampling and in a way that best preserves the numerical precision. Although the point sampling method does tend to give good approximations when the number of samples L is large enough, it is a relatively small increase in computational complexity and programming effort to use Legendre expansions instead. The main strength of the Lagrange method lies in its simplicity: it is easy to implement, computationally inexpensive and highly parallelizable.

Alternative choices of nodes are also interesting to investigate. Using the inequality (4), we can introduce the bound

$$\begin{aligned}
\min_{\|\mathbf{b}\|_2=1} \max_{t\in\Omega} |q(\mathbf{p}(t))| &\leq \|\boldsymbol{\alpha}(t)\|_2 \, \sigma_{\min}, \\
&\leq \Lambda(\boldsymbol{\alpha}) \, \sigma_{\min},
\end{aligned}$$

where $\Lambda(\boldsymbol{\alpha})$ is the Lebesgue constant from interpolation theory defined by $\Lambda(\boldsymbol{\alpha}) = \max_{t\in\Omega} \|\boldsymbol{\alpha}(t)\|_1$. Thus we may expect to obtain better results from point distributions with smaller Lebesgue constants. In particular, we will see in Section 6.3 that by using the Bernstein basis we achieve a Lebesgue constant of 1. However, it is possible that with smaller Lebesgue constants come

larger singular values. Thus it is important to balance between minimizing the Lebesgue constant and the singular value in order to obtain the best bound for the algebraic error.

A point distribution of particular interest is that of the Chebyshev points. On $[0, 1]$ this is defined as:

$$t_j = \frac{1}{2}\left(1 - \cos\left(\frac{j\pi - \pi}{L - 1}\right)\right), \quad j = 1, \ldots, L. \tag{12}$$

Conversion from the Lagrange basis at Chebyshev points to the Chebyshev basis can be performed by a DCT-I of the Lagrange coefficients [17]. Implementing the algorithm in this way gives a fast procedure for the Chebyshev method of Section 6.1. Proposition 3 can be extended to show that sampling at an increasing number of Chebyshev points causes the solution to converge to that of the weak method with the Chebyshev weight function (see Proposition 4). However, interpolation in these points is known from approximation theory to give very good approximation properties of its own [5]. One reason for this is the small Lebesgue constants associated with Chebyshev points.

Proposition 4. *Let $\mathbf{p}(t)$ be a planar parametric curve with bounded, piecewise continuous components on the interval $[0, 1]$ and let $\mathbf{D}_{\mathrm{L},L}$ be the matrix defined by (11) at Chebyshev points given by (12). Then the (k, l) element of ${\mathbf{D}_{\mathrm{L},L}}^T\mathbf{D}_{\mathrm{L},L}$ converges to a constant multiple of the (k, l) element of $\mathbf{M}_w$, defined by (8) with the weight function $w(t) = 1/\sqrt{t(1-t)}$, as the number of samples L is increased.*

Proof. We use the change of variable $t(\theta) = (1 - \cos(\pi\theta))/2$, and note first that

$$\sqrt{t(\theta)(1 - t(\theta))} = \sin(\pi\theta)/2, \quad \text{and} \quad \frac{\mathrm{d}t}{\mathrm{d}\theta} = \pi\sin(\pi\theta)/2.$$

As in Proposition 3, $q_k \circ \mathbf{p}$ is bounded and piecewise continuous for each k. In order to simplify the notation, let $f_k = q_k \circ \mathbf{p}$ for each k, and let $h_L = 1/(L - 1)$. Then, for uniform samples θ_j in the domain $[0, 1]$ (which correspond to the Chebyshev points $t_j = t(\theta_j)$ in the parameter domain), we see that

$$\begin{aligned}
\lim_{L\to\infty} h_L({\mathbf{D}_{\mathrm{L},L}}^T\mathbf{D}_{\mathrm{L},L})_{k,l} &= \lim_{L\to\infty} \sum_{j=1}^{L} h_L f_k(t(\theta_j)) f_l(t(\theta_j)), \\
&= \int_\Omega f_k(t(\theta)) f_l(t(\theta))\, \mathrm{d}\theta, \\
&= \int_\Omega \frac{f_k(t(\theta)) f_l(t(\theta))\sin(\pi\theta)}{2\sqrt{t(\theta)(1 - t(\theta))}}\, \mathrm{d}\theta, \\
&= \frac{\pi}{2}\int_\Omega \frac{f_k(t) f_l(t)}{\sqrt{t(1-t)}}\, \mathrm{d}t, \\
&= \frac{\pi}{2}(\mathbf{M}_w)_{k,l},
\end{aligned}$$

for $k, l = 1, \ldots, M$. Note that $t(\Omega) = \Omega$, so the integral is taken over the same domain even after change of variables. □

6.3 Bernstein polynomial basis

The approach in [9] was to choose $\boldsymbol{\alpha}$ to be a non-negative partition of unity basis such as the Bernstein basis (i.e., $\sum_i \alpha_i(t) \equiv 1$ and $0 \leq \alpha_i(t) \leq 1$). This ensures that $\|\boldsymbol{\alpha}(t)\|_2 \leq \Lambda(\boldsymbol{\alpha}) = 1$ for all $t \in \Omega$ and so the smallest singular value gives an upper bound for the error

$$\min_{\|\mathbf{b}\|_2=1} \max_{t\in\Omega} |q(\mathbf{p}(t))| \leq \sigma_{\min}.$$

Approximation in the Bernstein basis $(B_j)_{j=1}^L$, has the advantage that it is easily generalizable to both tensor-product and simplex domains in higher dimensions [4]. If the parametric curves are given in spline or Bézier form, it is natural to use the Bernstein coefficients, since there exist numerically stable algorithms for computing the compositions, without resorting to sampling [8]. Despite the slightly less favourable approximation qualities of the Bernstein basis (see Figure 1), this method performs sufficiently well to be integrated into CAGD systems that are based on Bézier or spline curves and surfaces. It also appears to be the most stable method if the degree m is chosen high enough for an exact implicit representation (see Section 7).

The Bernstein method is closely related to both the Lagrange and Legendre methods seen previously. It is in fact easy to see that the $\mathbf{D}_{\mathrm{B},L}$ matrix in the Bernstein basis of degree $L-1$, converges asymptotically to the $\mathbf{D}_{\mathrm{L},L}$ matrix in the Lagrange basis of degree $L-1$ at uniform nodes, as the degree is raised.

Proposition 5. *Let $\mathbf{D}_{\mathrm{B},L}$ be the matrix defined by (2) in the degree $L-1$ Bernstein polynomial basis. Then the (j,k) element of $\mathbf{D}_{\mathrm{B},L}$ converges to the (j,k) element of $\mathbf{D}_{\mathrm{L},L}$, defined by (2) in the Lagrange polynomial basis at uniform nodes.*

Proof. It is well known that the Bernstein coefficients of a polynomial tend to the values of the polynomial as the degree is raised, as follows [16]:

$$\lim_{L\to\infty} (\mathbf{D}_{\mathrm{B},L})_{j,k} = q_k\left(\mathbf{p}\left(t_j\right)\right), \text{ for all } j = 1,\ldots,L,$$

where $t_j = (j-1)/(L-1)$. But the elements on the right-hand side are simply the elements of $\mathbf{D}_{\mathrm{L},L}$ in the Lagrange basis at uniform nodes. □

We can thus deduce the following convergence property of the Bernstein method as an immediate consequence of Propositions 3 and 5:

Corollary 6. *Let $\mathbf{D}_{\mathrm{B},L}$ be the matrix defined by (2) in the Bernstein polynomial basis of degree $L-1$. Then the (k,l) element of ${\mathbf{D}_{\mathrm{B},L}}^T\mathbf{D}_{\mathrm{B},L}$ converges to a constant multiple of the (k,l) element of $\mathbf{M}_1$, defined by (8), as the degree is raised.*

6.4 Exact implicitization using linear algebra

As mentioned previously, when the degree m is chosen high enough to give an exact implicit representation and the algorithms are implemented in exact precision, all the methods can give exact results. The choice of basis in the exact case is irrelevant to the resulting polynomial and only affects only the implementation complexity and computational speed. For example, the elements of the matrix using the Lagrange method can be generated by choosing rational

nodes represented in exact arithmetic. As with the floating point implementation, the matrix can be built very quickly in parallel, but rather than using SVD we can exploit algorithms for finding the kernel of a matrix. A similar method for exact implicitization is described in [26]. However, the matrix there is expanded in the monomial basis, which leads to computationally expensive expansions for high degrees. It is noted that it is possible to reduce the complexity of that method in certain cases by exploiting the special structures of the algorithm and sparsity in the resulting matrix. In general, sparsity is not a feature of the Lagrange method, however, the matrices can be built more efficiently. The Bernstein method can also be implemented in exact arithmetic, however, similarly to the method in [26], it suffers from issues with computationally expensive expansions. Although it is possible to implement the Chebyshev and Legendre methods in exact precision using the elementwise definition (10), the evaluation of such integrals will be slow. The fast algorithms for generating the matrices using FFT are not suitable for an exact implementation.

When using the original method for approximate implicitization, we represent the error function $q \circ \mathbf{p}$ in a basis of degree mn. In the Lagrange basis we thus choose the number of nodes L, to be one more than the degree of the basis $L = mn + 1$. This is shown to be the smallest number of samples one can take in order to guarantee an exact implicitization method in the following proposition:

Proposition 7. *Suppose we are given a non-degenerate rational parametric planar curve $\mathbf{p}(t)$ of degree n (i.e., the degree of the algebraic representation is n). Then the number of unique samples required to guarantee an exact implicitization by the Lagrange method is given by $K = n^2 + 1$.*

Proof. Since the implicitization is exact, we know that there exists a unique polynomial q of degree n with coefficients $\mathbf{b}$ such that $q \circ \mathbf{p} \equiv 0$. By the theory described in Section 3, we can write

$$q(\mathbf{p}(t)) = \sum_{j=1}^{K} \alpha_j(t) \sum_{k=1}^{M} d_{j,k} b_k \equiv 0,$$

where $(\alpha_j)_{j=1}^{K}$ is a basis for polynomials of degree $K-1$, and $K = n^2+1$. Since the polynomials $(\alpha_j)_{j=1}^{K}$ are linearly independent, we have

$$\sum_{k=1}^{M} d_{j,k} b_k = 0,$$

for $j = 1, \ldots, K$, and since $\mathbf{D}_\alpha \neq 0$, the vector $\mathbf{b}$ must lie in the null space of $\mathbf{D}_\alpha$. This shows that any basis of degree $K - 1$ can be used for exact implicitization. In the univariate case, Lagrange polynomials defined by K points form a basis for polynomials of degree up to $K - 1$, if and only if all the K points are unique. Thus, choosing K unique points in the parameter domain is sufficient to guarantee an exact implicitization.

To see that choosing fewer than K points is insufficient, we consider parameter values corresponding to double points on the curve. Let $K_1 = \frac{1}{2}(n + 1)(n + 2) - 1$ denote the minimum number of points in general position on a curve of degree n required to define the curve. Let the total number of possible double points on a rational curve of degree n be given by $K_2 = \frac{1}{2}(n - 2)(n - 1)$ [20]. Then up to K_2 points on the curve can correspond to more than

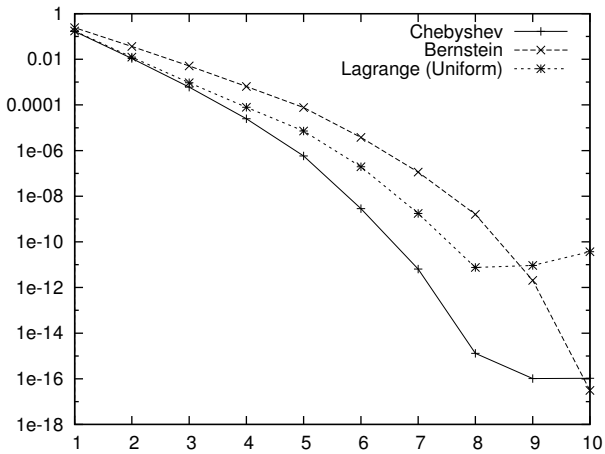


Figure 1: The average uniform algebraic error, $\max|q(\mathbf{p}(t))|$, of the approximate implicitization of 100 Bézier curves of degree 10, with random control points in the triangle defined by $(1,0)$, $(0,0)$ and $(0,1)$, with implicit approximation degrees between 1 and 10.

one parameter value. Thus the minimum number of samples guaranteeing a unique exact implicitization is given by

$$K_1 + K_2 = K.$$

□

When searching for exact implicitizations, we generally want the implicit polynomial of *lowest degree* that contains the parametric curve. Since the normalized coefficient vector $\mathbf{b}$, given by an exact implicitization of lowest degree is unique, the kernel of the matrix would be expected to be 1-dimensional. A kernel of dimension higher than one indicates that the implicit polynomial defined by any vector in the kernel is reducible, and thus the degree m can be reduced.

7 Comparison of the algorithms

So far, we have presented several approaches to exact and approximate implicitization using linear algebra. The approaches exhibit different qualities in terms of approximation, conditioning and computational complexity. The intention of this section is to provide a comparison of the algorithms.

7.1 Algebraic error comparisons

Figure 1 plots the average uniform algebraic error, $\max_{t\in[0,1]}|q(\mathbf{p}(t))|$, of the approximations of 100 Bézier curves of degree 10, for algebraic degrees $m = 1,\ldots,10$. We use a barycentric coordinate system defined on the triangle $(1,0)$, $(0,0)$ and $(0,1)$ for the implicit representation, and choose random control points within this region to define the Bézier curves. We compare the performance of the Lagrange basis (at uniform nodes)[3], the Bernstein basis and the Chebyshev basis. As the results in the Chebyshev and Legendre bases are very similar, we include only the

[3] Any reference to the Lagrange basis throughout this section will be assumed to be at uniformly spaced nodes.

Chebyshev basis. The monomial basis, transformed to the interval $[0, 1]$, was also considered but not included due to its vastly inferior approximation qualities. All the algorithms were performed in double precision.

For each degree up to the exact degree, $m = 10$, the Chebyshev basis gave the best uniform minimization of the objective function $q \circ \mathbf{p}$. The maximum error for degree m in the Chebyshev basis was, in general, approximately equivalent to the maximal error in the Lagrange basis of degree $m + 1$ and the Bernstein basis of degree $m + 2$. For the Chebyshev basis, the maximal errors level out to roughly double-precision accuracy at degree eight, whereas for the Bernstein basis, the required degree for machine precision was 10. Although the Lagrange basis performs better than the Bernstein basis for low degrees, the higher degree approximations are distorted to the extent that the exact implicitization, at degree 10, loses several orders of magnitude in accuracy. This appears to be due to the large deviation in the extrema of Lagrange basis functions at uniform nodes, which are associated with large Lebesgue constants. An example of such an error distribution is pictured in Figure 4c. The spike in error can be reduced by sampling more points, as the error converges to the weak approximation (c.f., Proposition 3), or by choosing point distributions with smaller Lebesgue coefficients.

It should be noted that for the Bernstein and Lagrange methods, the maximum of the algebraic error normally occurs at the end points of the interval, and is normally much higher than the average error across the interval (see Figure 4). Moreover, the error away from the ends of the interval can sometimes be smaller in these bases than in the Chebyshev basis. Hence, they generally perform better than Figure 1 may suggest, in relation to the Chebyshev basis. The Chebyshev basis, however, tends to make the error roughly equioscillating throughout the interval. In addition, topological constraints imposed by approximating with lower degrees than necessary mean that even when the algebraic error is small, the geometric error can be somewhat different, especially for curves with many singularities.

7.2 A visual comparison of the methods

As discussed previously, minimizing the algebraic error does not necessarily minimize the geometric distance between the implicit and parametric curves. In order to visually compare how the methods perform in terms of geometric approximations, Figure 3 plots the implicit approximations of the parametric curve pictured in Figure 2. The curve was chosen as an example that represents the general approximation properties of each of the bases. However, it should be noted that different examples can give different results.

We see that for the quartic approximation, the Lagrange and Chebyshev methods are already performing fairly well with only some detail lost close to the double point singularities. Despite exhibiting several intersections with the parametric curve, the Bernstein method gives little reproduction of the shape. The monomial approximation bears almost no resemblance to the original curve. For the quintic approximation, the Chebyshev and Lagrange bases again perform very similarly, giving excellent approximations that replicate the singularities well. These approximations would be sufficient for many applications. The Bernstein method performs similarly to the Chebyshev and Lagrange approximations of degree four, with only some loss of detail at each of the double points. Again the monomial basis gives almost no replication of the curve. It is also interesting to note the presence of extraneous branches visible in the Bernstein,

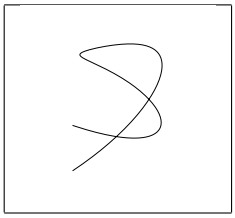


Figure 2: A parametric Bézier curve of degree seven, with control points $(\frac{1}{5}, \frac{1}{10})$, $(\frac{1}{2}, \frac{3}{10})$, $(\frac{1}{2}, \frac{1}{2})$, $(\frac{3}{10}, \frac{1}{2})$, $(0, 0)$, $(0, \frac{4}{5})$, $(\frac{4}{5}, 0)$ and $(\frac{1}{5}, \frac{1}{5})$. Implicit approximations of this curve appear in Figure 3.

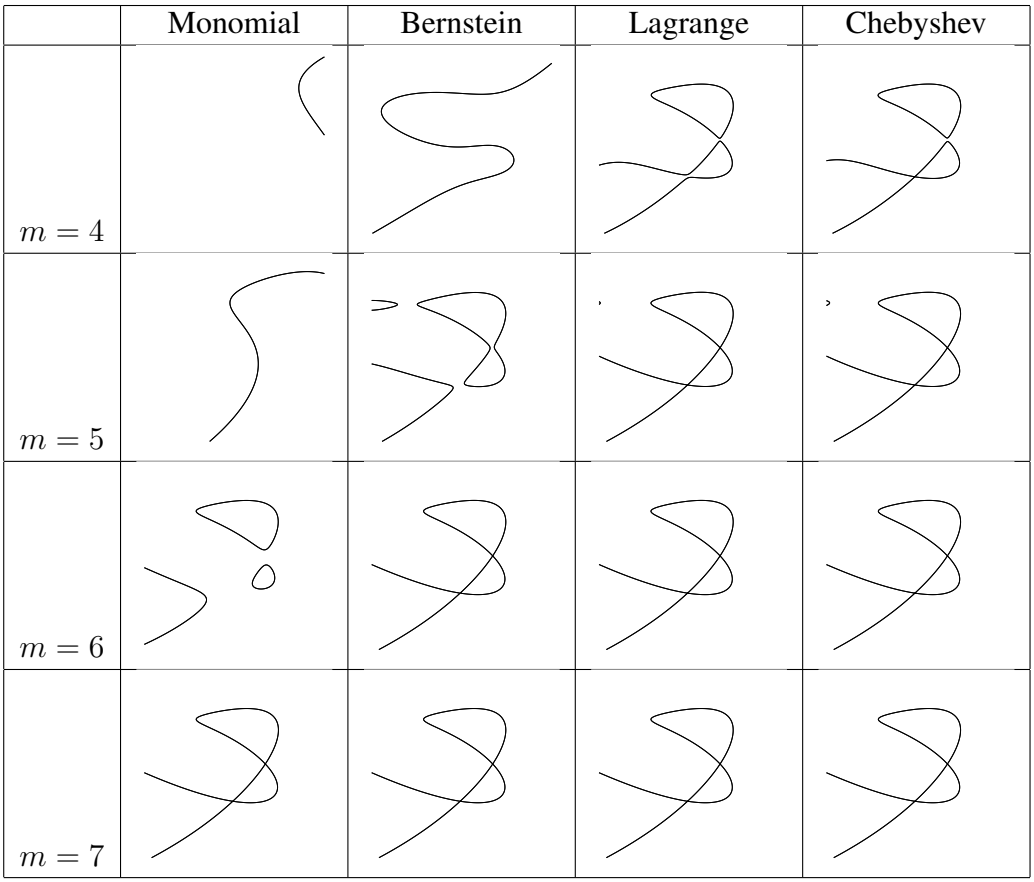


Figure 3: Implicit plots of the approximations of the degree seven Bézier curve pictured in Figure 2, for implicit degrees $m = 4, 5, 6$ and 7, in the monomial, Bernstein, Lagrange and Chebyshev bases.

Lagrange and Chebyshev approximations at degree five. This is a feature which may occur with any of the methods. At degree six the Bernstein, Lagrange and Chebyshev methods all give excellent results over the entire interval. The monomial method is beginning to show good approximation at the centre of the interval, however, this deteriorates towards the ends. At degree seven we expect exact results, up to numerical stability, for all of the algorithms. Visually, the implicitizations in all of the bases agree very closely.

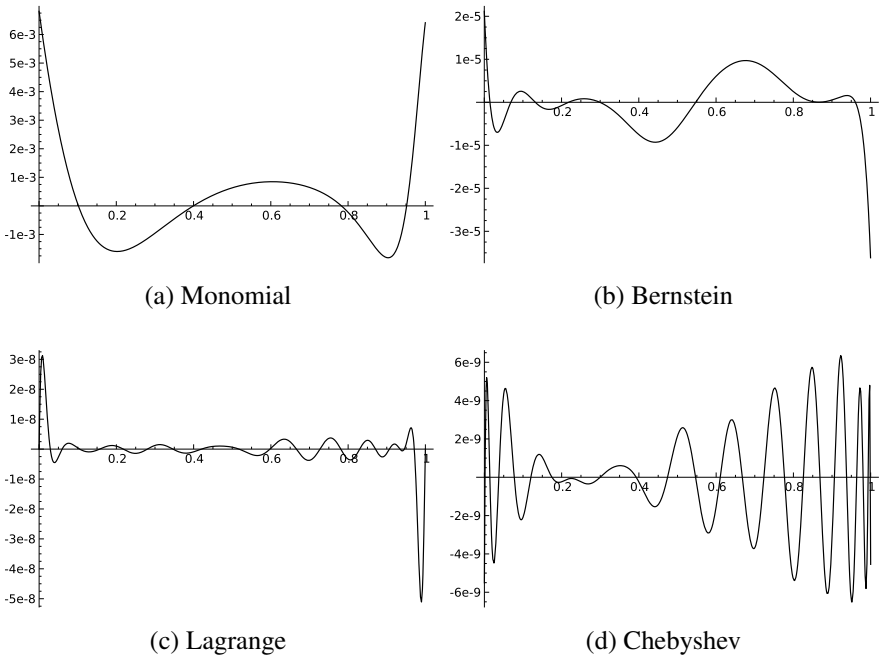
For degree seven, we can also perform the Lagrange method in exact precision as described in Section 6.4. Using this method we obtain a null space of dimension one, which confirms that the correct algebraic degree is in fact seven. We may also use this exact implicitization to compare the relative errors e_α (using the infinity norm) for the implicitizations given by the different bases, as follows:

$$\begin{aligned} e_{\text{mono}} &= 1.17 \times 10^{-8}, \\ e_{\text{bern}} &= 7.46 \times 10^{-11}, \\ e_{\text{lag}} &= 2.01 \times 10^{-4}, \\ e_{\text{cheb}} &= 1.11 \times 10^{-5}. \end{aligned}$$

The numerical stability properties of the Bernstein basis are well documented in mathematical literature (see for example [15]). It appears that these properties are also preserved by the implicitization algorithm presented here, with the Bernstein basis outperforming the other bases to some orders of magnitude. In relation to the numerical stability of the methods, it is interesting to note the distribution of singular values given by the singular value decomposition of the $\mathbf{D}_\alpha$ matrices [23]. For this comparison we normalize the singular values to all lie in $[0, 1]$. The Bernstein basis has one singular value close to zero in double precision; the second smallest being approximately 7.74×10^{-9}. This shows that the solution is quite unique. The case is similar in the monomial basis, however, the second smallest singular value is approximately 6.24×10^{-10}. For the Lagrange and Chebyshev bases the second smallest singular values are 2.54×10^{-15} and 1.17×10^{-14} respectively. Such close proximity between the smallest singular values leads to issues with numerical stability. However, it can also be useful since, as discussed previously, the singular vectors corresponding to the larger singular values are also candidates for approximation. Thus we would expect the second best approximation in the Chebyshev and Lagrange bases to be much better than the second best approximation in the Bernstein basis.

It is also interesting to note that when attempting to use the weak method for approximate implicitization as an exact method here, we obtain a completely different solution, with relative error given approximately by $e_{\text{weak}} = 0.607$. This seems to be due to the fact that the nine smallest eigenvalues (which are equal to the singular values, since $\mathbf{M}$ is symmetric positive semi-definite), are all below machine precision. Thus, the solution given by the weak method could be a combination of any of the nine corresponding eigenvectors. However, despite the large relative error, the weak method still returns a solution which gives a reasonable geometric approximation.

Typical algebraic error distributions obtained from the methods in this section are displayed in Figure 4. A more accurate approximation of the geometric error can be given by dividing the algebraic error by the norm of the gradient of the given implicit representation. However, since the methods we present do not control the gradient, we have not included such plots here.



(a) Monomial (b) Bernstein

(c) Lagrange (d) Chebyshev

Figure 4: Typical algebraic error distributions $q(\mathbf{p}(t))$, for the different bases. These are taken from the quintic implicit approximations of the parametric curve in Figure 2.

7.3 Discussion

In the example of section 7.2, it is striking how well the Lagrange basis performs in relation to the Chebyshev basis. Despite the spike in error near the ends of the interval, the geometric approximation appears to remain relatively good; and in some cases, even better than the Chebyshev. Thus, the comparisons in this section may lead to different conclusions as to which is the best algorithm to choose in general. It is clear that the monomial basis performs relatively badly, but the other bases all tend to perform well. The speed and simplicity of the Lagrange method is appealing, whereas the stability of exact implicitization in the Bernstein basis is desirable for many applications. The fact that the Legendre and Chebyshev methods have the guarantee of solving a least squares problem (see Theorem 1), and that there exist efficient algorithms for computing them, also gives justification for choosing them as general methods. As an overview of this qualitative comparison, we display various aspects of the implementations in Table 3. The table summarizes how the algorithms perform in terms of the stability, generality and what sort of problems they solve (i.e., least squares or uniform approximations).

One undesirable property of approximate implicitization is the possibility of introducing new singularities that are not present in the parametric curve. As the implicit polynomial representation is global, we cannot control what happens outside the interval of approximation. In particular, there could appear self-intersections of the curve within the interval of interest. This is an artifact that can appear using any of the methods described in this paper. However, such problems can be avoided by adding constraints to the algebraic approximation [25], or by using information about the gradient of the implicit curve in the approximation [18]. In general, adding such constraints will reduce the convergence rates of these methods [10].

The computation times for each of the methods varies. In all the current implementations of

	Least squares	Uniform	Stability	Generality
Lagrange	Good	OK	OK	Any
Legendre	Very good	Good	OK	Rational
Chebyshev	Very good	Very Good	OK	Rational
Bernstein	OK	OK	Very Good	Rational
Weak	Very good	OK	Very Bad	Integrable

Table 3: A qualitative comparison of the algorithms. The least squares, and uniform columns refer to how well the algorithms perform in terms of producing such approximations in the algebraic error function.

the methods, the matrix generation is the dominant part of the algorithm, and the SVD is generally fast. When constructing the matrices, the monomial and Bernstein methods suffer from computationally expensive expansions for high degrees, whereas the Chebyshev and Lagrange methods are based on point sampling and FFT, which can be implemented in parallel. Computational features of the methods will be the subject of further research, including exploiting the parallelism of GPUs to enhance the algorithms.

8 Approximate implicitization of surfaces

In this section we will discuss how the methods presented for curves in the preceding sections generalize to surface implicitization. We will also provide a visual example of approximate implicitization of surfaces.

A parametric surface in $\mathbb{R}^3$ is given by $\mathbf{p}(s,t) = (p_1(s,t), p_2(s,t), p_3(s,t))$ where p_1, p_2 and p_3 are functions in parameters $(s,t) \in \Omega \subset \mathbb{R}^2$. Similarly to curves, in $\mathbb{P}^3$, we use the homogeneous form of such a surface to write

$$\mathbf{p}(s,t) = (g_1(s,t), g_2(s,t), g_3(s,t), h(s,t)),$$

for bivariate polynomials g_1, g_2, g_3 and h.

Although we have the option of using tensor-product polynomials for the implicit representation, here we choose polynomials of total degree m. An implicit polynomial q of total degree m can be described in a basis $(q_k(\mathbf{u}))_{k=1}^M$, where $M = \binom{m+3}{3}$, with coefficients $\mathbf{b} = (b_k)_{k=1}^M$. The choice of using the Bernstein basis for the implicit polynomial can be extended by considering a barycentric coordinate system defined over a tetrahedron containing the parametric surface. For surfaces in $\mathbb{P}^3$, the homogeneous Bernstein basis is given by

$$q_{\mathbf{k}}(u,v,w,z) = \binom{m}{\mathbf{k}} u^{k_1} v^{k_2} w^{k_3} z^{k_4}, \qquad \text{for } |\mathbf{k}| = k_1 + k_2 + k_3 + k_4 = m,$$

where u, v, w and z denote the homogeneous coordinates, and $\mathbf{k} = (k_1, k_2, k_3, k_4)$ denotes a multi-index. Again, this basis forms a partition of unity and we order it by letting q_k correspond to $q_{\mathbf{k}}$, for $k = 1, \ldots, M$, where $k = k(\mathbf{k})$ denotes lexicographical ordering.

When applying the original algorithm for approximate implicitization, we observe that the

expression $q \circ \mathbf{p}$ is a bivariate polynomial in s and t. As such, we can write the function in a basis $\boldsymbol{\alpha}(s,t) = (\alpha_j(s,t))_{j=1}^{L}$ as

$$q(\mathbf{p}(s,t)) = \boldsymbol{\alpha}(s,t)^T \mathbf{D}_\alpha \mathbf{b}. \tag{13}$$

The description of $\boldsymbol{\alpha}(s,t)$ and the number of basis functions L is dependent on the type of parametric surface. We thus make distinctions for the two most interesting cases - tensor-product surfaces, and surfaces on triangular domains - in the following subsections.

The weak method presented in Section 4 can also be generalized to surfaces. The problem can be stated as

$$\min_{\|\mathbf{b}\|_2=1} \int_\Omega w(s,t) q(\mathbf{p}(s,t))^2 \, \mathrm{d}s \, \mathrm{d}t, \tag{14}$$

where $w(s,t)$ is some weight function defined on the domain of approximation Ω. In this way, we obtain a linear algebra problem as before, where the solution is given by the eigenvector corresponding to the smallest eigenvalue of a matrix $\mathbf{M}_w = (m_{k,l})_{k=1,l=1}^{M,M}$, defined by

$$m_{k,l} = \int_\Omega w(s,t) q_k(\mathbf{p}(s,t)) q_l(\mathbf{p}(s,t)) \, \mathrm{d}s \, \mathrm{d}t. \tag{15}$$

8.1 Tensor-product parametric surfaces

For rational tensor-product surfaces of bidegree $\mathbf{n} = (n_1, n_2)$, we can write,

$$\mathbf{p}(s,t) = \sum_{i_1=0}^{n_1} \sum_{i_2=0}^{n_2} \mathbf{c}_{i_1,i_2} \phi_{i_1}(s) \psi_{i_2}(t),$$

where $(\phi_{i_1})_{i_1=0}^{n_1}$ and $(\psi_{i_2})_{i_2=0}^{n_2}$ are bases for univariate polynomials of respective degree n_1 and n_2, the domain $\Omega = [a,b] \times [c,d]$ is the Cartesian product of two univariate intervals, and $\mathbf{c}_{i_1,i_2} \in \mathbb{P}^3$ for $i_1 = 0, \ldots, n_1$ and $i_2 = 0, \ldots, n_2$. In this case, equation (13) can be written in a tensor-product basis $\boldsymbol{\alpha}(s,t) = (\alpha_{j_1}(s)\beta_{j_2}(t))_{j_1=1,j_2=1}^{L_1,L_2}$ of bidegree $m\mathbf{n} = (mn_1, mn_2)$ as follows:

$$q(\mathbf{p}(s,t)) = \sum_{k=1}^{M} b_k \sum_{j_1=1}^{L_1} \sum_{j_2=1}^{L_2} \hat{d}_{(j_1,j_2),k} \alpha_{j_1}(s) \beta_{j_2}(t),$$

where $L_1 = mn_1 + 1$ and $L_2 = mn_2 + 1$. We must use an ordering for the indices (j_1, j_2) in order to enter the coefficients $\hat{d}_{(j_1,j_2),k}$ in matrix form. Again we choose the lexicographical ordering $j = j(j_1, j_2) = (j_1 - 1)L_2 + j_2$, so that

$$\mathbf{D}_{\alpha,\beta} = (d_{j,k})_{j=1,k=1}^{L,M}, \tag{16}$$

where $L = L_1 L_2 = m^2 n_1 n_2 + mn_1 + mn_2 + 1$ and $d_{j,k} = \hat{d}_{(j_1,j_2),k}$. The algorithm then proceeds as for curves by using the singular value decomposition and selecting the vector corresponding to the smallest singular value.

The univariate bases $\boldsymbol{\alpha} = (\alpha_{j_1})_{j_1=1}^{L_1}$ and $\boldsymbol{\beta} = (\beta_{j_2})_{j_2=1}^{L_2}$ can be be chosen arbitrarily, as in the case for curves. In this way, Theorem 1, Propositions 3, 4, 5, 7 and Corollary 6 from the

univariate case, all carry over to the tensor-product case with minimal effort[4]. This applies also to higher dimensional tensor-product hypersurfaces. As an example, consider Theorem 1. In the tensor-product case, we have the following, almost verbatim to the univariate case:

Theorem 8. *Let $\boldsymbol{\alpha}$ be a tensor-product polynomial basis, orthonormal with respect to the given inner product $\langle \cdot, \cdot \rangle_w$, and $\mathbf{D}_{\alpha,\beta}$ and $\mathbf{M}_w$ be defined by (16) and (15) respectively. Then $\mathbf{M}_w = \mathbf{D}_{\alpha,\beta}^T \mathbf{D}_{\alpha,\beta}$.*

Proof. Similar to Theorem 1, with the relevant inner product given by

$$\langle f, g \rangle_w = \int_\Omega w(s,t) f(s,t) g(s,t) \, \mathrm{d}s \, \mathrm{d}t,$$

for real bivariate polynomials f, g. □

8.2 Triangular surfaces

For rational surfaces of total degree n on a triangular domain Ω, we can write,

$$\mathbf{p}(s,t) = \sum_{i=1}^{N} \mathbf{c}_i \phi_i(s,t),$$

where $(\phi_i)_{i=1}^N$ is a basis for bivariate polynomials of total degree n with $N = \binom{n+2}{2}$, and $\mathbf{c}_i \in \mathbb{P}^3$ for $i = 1, \ldots, N$. In this case, equation (13) can be written in a bivariate basis $\boldsymbol{\alpha} = (\alpha_j)_{j=1}^L$ of total degree mn as follows:

$$q(\mathbf{p}(s,t)) = \sum_{k=1}^{M} b_k \sum_{j=1}^{L} d_{j,k} \alpha_j(s,t),$$

where, $L = \binom{mn+2}{2}$. Lexicographical ordering on the respective degrees of s and t in the basis $\boldsymbol{\alpha}$ can be used to enter the coefficients in matrix form, $\mathbf{D}_\alpha = (d_{j,k})_{j,k=1,1}^{L,M}$.

Surfaces on triangular domains may be considered a more fundamental generalization than tensor-product surfaces, however, they often exhibit several difficulties not present in the tensor-product case. For example, most practical applications of the weak method of Section 4 use numerical quadrature, a method which is difficult to implement on triangular domains for high degrees. Since the degree of the integrand in the weak implicitization of a surface of degree n has degree $2mn$, high degree quadrature rules are required. For example, exact implicitization of a general quintic parametric surface, with implicit degree $m = n^2 = 25$ would need a quadrature rule accurate to order 250. Although it would be wise to use a lower degree approximation in this case (e.g., $m = 5$), the degree of the integrand would still be too high for many quadrature rules on triangular domains. High degree quadrature rules can be constructed by first transforming the domain into a square, and using two univariate quadrature rules. However, these rules are not symmetric in the triangle and require more samples than is mathematically necessary [19].

[4]For a tensor-product extension of Proposition 7, the samples should be taken on a tensor-product grid and the number of samples is given by $K = (2n_1 n_2^2 + 1)(2n_1^2 n_2 + 1)$.

Certain methods for approximate implicitization are, however, easy to generalize. For example, the Bernstein basis has a natural representation on simplex domains using barycentric coordinates, and thus the use of approximate implicitization on triangular surfaces in this basis is simple [4]. The convergence result of Corollary 6 can be extended to this case, together with well established degree elevation algorithms, in order to obtain better approximation results.

The Lagrange basis method from Section 6.2, which was based on sampling, can also be extended to triangular surfaces. However, when choosing samples, it is essential that the Lagrange polynomials defining the $\boldsymbol{\alpha}$-basis are linearly independent; that is, that they do in fact form a basis. For curves, choosing unique parameters for the sample points was sufficient (see Proposition 7). However, for surfaces we must add the requirement that the samples must not all lie on a curve of degree mn in the parameter domain, where the number of samples is given by $L = \binom{mn+2}{2}$.

Orthogonal polynomials on triangular domains also exist and an extension of Theorem 1 holds in this case. However, fast algorithms for generating the coefficeints appear to be difficult to replicate, thus limiting the applicability of this method in practice. In particular, there appears to be no direct analogue of the FFT method for generating Chebyshev coefficients. We refer the reader to [14] for a discussion of orthogonal polynomials on triangular domains.

8.3 An example of surface implicitization

As an example of approximate implicitization of tensor-product surfaces we will consider the problem of approximating the well known Newells' teapot model. It is stated in [24] that "the 32 bicucic patches defining Newells' teapot provide a surprisingly diverse set of tests for moving surface implicitization". In that paper, properties of the moving surface implicitization algorithm for the different patches are discussed and exact implicit degrees for each patch are stated. We will consider the same 32 patches here, but instead use approximate methods, where the degree of approximation has been chosen manually to give good visual results.

Each of the 32 bicubic parametric surfaces has been approximated using the tensor-product Chebyshev method and the degrees stated in Table 4. The resulting surface has been ray traced in Figure 5 using POV-Ray [21], and clearly gives an excellent implicit approximation to the parametric teapot model. Moreover, the degrees of the surface patches are greatly reduced from the exact degrees, which are also stated in Table 4. The highest degree patch in the approximated model is six, whereas if exact methods are used patches of degree up to 18 are required. Generally, it appears that for visual purposes, the degree can be reduced to roughly one third of the exact degree.

This example shows one potential application of approximate implicitization, however, there are several factors that should be noted. Firstly, a significant amount of user input was required to generate the approximations of the teapot patches. This involved choosing degrees that were suitable for each patch, and also choosing approximations without extra branches in the region of interest. This was done by considering approximations corresponding to other singular values than the smallest. For example, for the upper handle patches we chose the approximation corresponding to the fourth smallest singular value. For each increased singular value, the convergence rate of the method is reduced by one [12]. However, even with the reduced convergence rates, the Chebyshev method continues to give excellent approximations. User input was

Figure 5: Teapot defined by 32 approximately implicitized patches from Section 8.3, with degrees given in Table 4 and ray traced using POV-Ray [21].

also given to define the clipping region for the ray tracing of each patch. In this example, the clipping regions were boxes parallel to the xy, xz and yz-planes. however, in more complicated examples it is not always suitable to use box regions for this purpose.

Another feature of this example is that the continuity between the parametric patches has been approximated very well in the implicit model. This is mainly due to the high convergence rates, which give good approximations over the entire surface region. However, in this case, there is also symmetry in the model meaning the edge curves where the patches meet can be approximated in a symmetric way. To achieve this we have used the monomial basis for the implicit representation since it is symmetric around the z-axis. For more general models, such symmetry would not be possible.

9 Conclusions

We have presented and unified several new and existing methods for approximate implicitization of rational curves using linear algebra. Theoretical connections between the different methods have been made together with qualitative comparisons. Extensions of the methods to both tensor-product and triangular surfaces have been discussed. By considering various issues such as approximation quality and computational complexity, we regard the Chebyshev and Legendre methods as the algorithms of choice for approximation of most rational parametric curves. However, to obtain good numerical stability when using floating point arithmetic for exact implicitization, the Bernstein basis is a more favourable choice. Future research could include how the methods can be improved, for example, by exploiting sparsity as in [13], or by adding continuity constraints to the approximations [2].

	Exact degree	Approximate degree
$4 \times$ rim	9	4
$4 \times$ upper body	9	3
$4 \times$ lower body	9	3
$2 \times$ upper handle	18	4
$2 \times$ lower handle	18	4
$2 \times$ upper spout	18	5
$2 \times$ lower spout	18	6
$4 \times$ upper lid	13	3
$4 \times$ lower lid	9	4
$4 \times$ bottom	15	3

Table 4: Exact implicit degrees of the 32 Newells' teapot patches and the degrees used for approximate implicitization in Section 8.3.

Bibliography

[1] B.K. Alpert and V. Rokhlin. A fast algorithm for the evaluation of Legendre expansions. *SIAM J. Sci. Stat. Comput.*, 12:158–179, January 1991.

[2] C.L. Bajaj and I. Ihm. Algebraic surface design with Hermite interpolation. *ACM Transactions on Graphics*, 11(1):61–91, 1992.

[3] C.L. Bajaj, I. Ihm, and J. Warren. Higher-order interpolation and least-squares approximation using implicit algebraic surfaces. *ACM Trans. Graph.*, 12:327–347, October 1993.

[4] O.J.D. Barrowclough and T. Dokken. Approximate implicitization of triangular Bézier surfaces. In *Proceedings of the 26th Spring Conference on Computer Graphics*, SCCG '10, pages 133–140, New York, NY, USA, 2010. ACM.

[5] Z. Battles and L.N. Trefethen. An extension of MATLAB to continuous functions and operators. *SIAM Journal on Scientific Computing*, 25(5):1743–1770, 2004.

[6] J.H. Chuang and C.M. Hoffmann. On local implicit approximation and its applications. *ACM Trans. Graph.*, 8:298–324, October 1989.

[7] R.M. Corless, M. Giesbrecht, I.S. Kotsireas, and S.M. Watt. Numerical implicitization of parametric hypersurfaces with linear algebra. In *Revised Papers from the International Conference on Artificial Intelligence and Symbolic Computation*, AISC '00, pages 174–183, London, UK, 2001. Springer-Verlag.

[8] T.D. DeRose, R.N. Goldman, H. Hagen, and S. Mann. Functional composition algorithms via blossoming. *ACM Transactions on Graphics (TOG)*, 12(2):113–135, 1993.

[9] T. Dokken. *Aspects of intersection algorithms and approximations*. PhD thesis, University of Oslo, 1997.

[10] T. Dokken. Approximate implicitization. In *Mathematical methods for curves and surfaces*, pages 81–102. Vanderbilt Univ. Press, Nashville, TN, 2001.

[11] T. Dokken, H.K. Kellerman, and C. Tegnander. *An approach to weak approximate implicitization*, pages 103–112. Vanderbilt University, Nashville, TN, USA, 2001.

[12] T. Dokken and J.B. Thomassen. Weak approximate implicitization. In *SMI '06: Proceedings of the IEEE International Conference on Shape Modeling and Applications 2006*, page 31, Washington, DC, USA, 2006. IEEE Computer Society.

[13] I.Z. Emiris and I.S. Kotsireas. Implicitization exploiting sparseness. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 67:281, 2005.

[14] R.T. Farouki. Construction of orthogonal bases for polynomials in Bernstein form on triangular and simplex domains. *Computer Aided Geometric Design*, 20:209–230, April 2003.

[15] R.T. Farouki and T.N.T. Goodman. On the optimal stability of the Bernstein basis. *Mathematics of Computation*, 65(216):1553–1567, October 1996.

[16] M. Floater and T. Lyche. Asymptotic convergence of degree-raising. *Advances in Computational Mathematics*, 12:175–187(13), February 2000.

[17] A. Gil, J. Segura, and N.M. Temme. *Numerical Methods for Special Functions*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1 edition, 2007.

[18] B. Jüttler and A. Felis. Least-squares fitting of algebraic spline surfaces. *Advances in Computational Mathematics*, 17:135–152, 2002. 10.1023/A:1015200504295.

[19] J.N. Lyness and R. Cools. A survey of numerical cubature over triangles. *Proceedings of Symposia in Applied Mathematics*, 48:127–150, 1994.

[20] J.W. Milnor. *Singular points of complex hypersurfaces*. Annals of mathematics studies. Princeton University Press, 1968.

[21] Persistence of Vision Pty. Ltd. Persistence of vision raytracer (version 3.6), [computer software]. Retrieved from `http://www.povray.org/download/`, 2004.

[22] V. Pratt. Direct least-squares fitting of algebraic surfaces. *SIGGRAPH Comput. Graph.*, 21(4):145–152, 1987.

[23] J. Schicho and I. Szilagyi. Numerical stability of surface implicitization. *Journal of Symbolic Computation*, 40(6):1291–1301, December 2005.

[24] T.W. Sederberg and F. Chen. Implicitization using moving curves and surfaces. In *SIGGRAPH 95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 301–308, New York, NY, USA, 1995. ACM.

[25] T.W. Sederberg, J. Zheng, K. Klimaszewski, and T. Dokken. Approximate Implicitization Using Monoid Curves and Surfaces. *Graphical Models and Image Processing*, 61(4):177–198, July 1999.

[26] D. Wang. A simple method for implicitizing rational curves and surfaces. *Journal of Symbolic Computation*, 38(1):899–914, July 2004.

Paper III: Fast approximate implicitization of envelope curves using Chebyshev polynomials

Oliver J.D. Barrowclough, Bert Jüttler and Tino Schulz

In *Latest Advances in Robot Kinematics*, Jadran Lenarčič and Manfred Husty, editors, pages 205-212. Springer Netherlands, 2012.

Abstract: Consider a rational family of planar rational curves in a certain region of interest. We are interested in finding an approximation to the implicit representation of the envelope. Since exact implicitization methods tend to be very costly, we employ an adaptation of approximate implicitization to envelope computation. Moreover, by utilizing an orthogonal basis in the construction process, the computational times can be shortened and the numerical condition improved. We provide an example to illustrate the performance of our approach.

1 Introduction

In geometric applications there are two basic standards for representing curves, namely the *parametric* and the *implicit* descriptions. Both descriptions feature specific advantages and disadvantages that complement each other. For instance, parametric curves allow the simple generation of point samples, while implicit forms support the decision of point location queries. In many applications, such as intersection computations, it is an advantage if both representations are available, and conversion algorithms are therefore of substantial practical interest. The conversion processes are called *parametrization* and *implicitization*, respectively.

A rational curve may always be implicitized, whereas the opposite is not true [10]. Several techniques for *exact* implicitization exist, e.g. Gröbner bases, moving curves/surfaces, or methods based on resultants, (see e.g. [5]). However, due to their computational complexity, their practical use is often restricted to low-degree curves. Moreover, the variety obtained by exact implicitization may contain unexpected branches and self-intersections.

A valid alternative to exact methods is *approximate implicitization*; cf. [3, 4]. Instead of the exact variety, a low degree approximation is used to represent the shape of the geometric object in a certain region of interest. This technique can be implemented using floating point numbers and thus it offers faster computation, while having very high convergence rates. As shown in

[2], the speed-up may be increased even further by using an orthogonal basis in the construction process.

These features make approximate implicitization a promising candidate for an efficient computation of *envelope curves*. Envelopes are used in different contexts in mathematics and applications, ranging from gearing theory and geometrical optics to NC-machining and Computer-Aided Design. In robotics, envelopes are ubiquitous, appearing for instance as singularities or boundaries. The theory of envelopes is covered by the classical literature, and is continuously extended, due to their practical importance [1, 6, 7, 8].

Approximate implicitization has recently been adapted to the computation of envelopes in [9]. As shown there, the idea is feasible and most properties of the original method can be preserved, such as the possibility of obtaining the exact solution. However, the convergence behaviour for higher degrees has not previously been studied and the computations are still fairly expensive, needing integrals of products of high degree polynomials.

The present paper uses the latest results from approximate implicitization to obtain a fast and efficient algorithm for approximating the envelope. This will make the use of implicit methods more attractive and moreover allow us to study the convergence behaviour experimentally. The paper is organized as follows: In Section two we will recall the basics of envelopes of planar curves. After that, the third section shows how approximate implicitization can be used to compute envelopes and derives a fast and efficient algorithm. The performance of our approach is illustrated with an example and discussion in Section four.

2 Envelopes of Rational Families of Curves

Consider the family of rational curves

$$\mathbf{p}(s,t) = \big(x(s,t)/w(s,t), y(s,t)/w(s,t)\big)^T, \quad (s,t) \in I \times J \tag{1}$$

where x, y and w are bivariate polynomials of bidegree (n_1, n_2) with $\gcd(x,y,w) = 1$ and $I, J \subset \mathbb{R}$ are closed intervals. We assume that $w(s,t) \neq 0$ for all $(s,t) \in I \times J$. Either s or t can be thought of as the time-like parameters, and the remaining parameter t or s is then used to parameterize the curves forming the family.

The *envelope* of the mapping $\mathbf{p}$ consists of those points where its Jacobian $\mathbf{J}(s,t)$ becomes singular. We observe that $\det \mathbf{J}(s,t) = h(s,t)/w(s,t)^3$, where

$$h(s,t) = \det \begin{pmatrix} x(s,t) & \partial_s x(s,t) & \partial_t x(s,t) \\ y(s,t) & \partial_s y(s,t) & \partial_t y(s,t) \\ w(s,t) & \partial_s w(s,t) & \partial_t w(s,t) \end{pmatrix}. \tag{2}$$

The function h is called the *envelope function*, since its zero set determines those points in the parameter domain which are mapped to the envelope. Unfortunately, certain parts of the zero set of h may degenerate under the mapping $\mathbf{p}$.

The earlier paper [9] describes how these *improper factors* can be removed from h. This can be done via some gcd computation and gives the *reduced envelope function* $\tilde{h}$. The image of the zero set of $\tilde{h}$ under $\mathbf{p}$ is called *proper part* of the envelope.

Let $q : \mathbb{C}^2 \to \mathbb{C}$ be the polynomial which defines *the implicit equation of the proper part of the envelope* of $\mathbf{p}$. According to Theorem 1 of [9], there exists a real polynomial $\lambda(s,t)$ such that

$$(q \circ \mathbf{p})\,(s,t) w(s,t)^d = \lambda(s,t)\tilde{h}(s,t)^2. \tag{3}$$

Equation (3) is linear with respect to the coefficients of q and λ. Let

$$q(\mathbf{x}) = \mathbf{c}_q^T \beta(\mathbf{x}) \quad \text{and} \quad \lambda(s,t) = \mathbf{c}_\lambda^T \alpha(s,t), \tag{4}$$

where $\beta(\mathbf{x}) = (\beta_k(\mathbf{x}))_{k=1}^M$ and $\alpha(s,t) = (\alpha_i(s)\alpha_j(t))_{(i,j)=(0,0)}^{(k_1,k_2)}$ are bases of polynomials in $\mathbf{x}$ and s,t of total degree d and bidegree (k_1,k_2) respectively, where $M = \binom{d+2}{2}$. The coefficients of q and λ with respect to these bases form a vector $\mathbf{c} = (\mathbf{c}_q^T, \mathbf{c}_\lambda^T)^T$. We formulate the *problem of approximate envelope implicitization:* Find the coefficients $\mathbf{c}$ which solve the weighted least squares minimization problem

$$\min_{\|\mathbf{c}\|_2=1} \int_{I\times J} \omega(s,t) \left((q \circ \mathbf{p})(s,t) w(s,t)^d - \lambda(s,t) h(s,t)^2\right)^2 \mathrm{d}(s,t), \tag{5}$$

for a nonnegative weight function ω, and chosen degrees d, k_1 and k_2.

It is important to mention that we use h instead of the exact $\tilde{h}$, since our algorithm uses floating point computations which do not support exact gcd computations. While an exact solution of this simplified problem might produce additional branches, the effect on our low degree approximation will be negligible.

The result of the minimization (5) depends both on the choice of bases of q and λ and on the weight function ω. The standard choice of a triangular Bernstein basis for q and a tensor-product Bernstein basis for λ has been used for the approximations in this paper and also in [9]. However, as a major difference to the approach in [9] where $\omega \equiv 1$, here we use a tensor product Chebyshev weight function on the domain $I \times J$, for the reasons described in the next section.

3 Fast Approximate Implicitization of Envelope Curves

The direct method for finding an approximate implicitization of envelope curves by evaluating high degree integrals is simple, but computationally costly. In addition, the resulting symmetric positive semi-definite matrix can be rather ill conditioned, leading to inaccurate null space computations when using floating point arithmetic. This is similar to the case of approximate implicitization of parametric curves presented in [2]. In that paper, an approach using orthogonal polynomials is presented which greatly improves both the conditioning and the computation time of the problem. In this section we give the details of how to implement the approach to approximate implicitization of envelope curves using Chebyshev polynomials.

3.1 Approximate Implicitization using Chebyshev Polynomials

As described previously, the method works by minimization of the integral (5). Such a problem is aided by expressing the function in a basis orthonormal with respect to the chosen weight

function ω. The objective function is expressible in any tensor product polynomial basis of bidegree

$$(L_1, L_2) = (\max(dn_1, k_1 + 2\deg_s(h)), \max(dn_2, k_2 + 2\deg_t(h))).$$

Thus, choosing an orthonormal basis (e.g., tensor-product Chebyshev polynomials), $\mathbf{T}(s,t) = (T_i(s)T_j(t))_{i=0,j=0}^{L_1,L_2}$ written in vector form and using (4), we can write

$$(q \circ \mathbf{p})(s,t)w(s,t)^d - \lambda(s,t)h(s,t)^2 = \mathbf{T}(s,t)^T(\mathbf{D}_q\mathbf{c}_q + \mathbf{D}_\lambda\mathbf{c}_\lambda), \tag{6}$$

where the matrices $\mathbf{D}_q$ and $\mathbf{D}_\lambda$ contain coefficients in $\mathbf{T}$. Now, defining a matrix

$$\mathbf{D} = (\mathbf{D}_q, \mathbf{D}_\lambda), \tag{7}$$

we claim that the singular vector corresponding to the smallest singular value of $\mathbf{D}$ solves the minimization problem (5). To see this, we prove the following Theorem:

Theorem 1. *Let the matrix* $\mathbf{D}$ *be defined as in (7). Then we have*

$$\min_{\|\mathbf{c}\|_2=1} \int_{I\times J} \omega((q \circ \mathbf{p})w^d - \lambda h^2)^2 = \min_{\|\mathbf{c}\|_2=1} \|\mathbf{D}\mathbf{c}\|_2^2.$$

Proof. By (6) and (7) we have

$$\int_{I\times J} \omega((q \circ \mathbf{p})w^d - \lambda h^2)^2 = \int_{I\times J} \omega(\mathbf{c}^T\mathbf{D}^T\mathbf{T})(\mathbf{T}^T\mathbf{D}\mathbf{c}) \;=\; \mathbf{c}^T\mathbf{D}^T\left(\int_{I\times J} \omega\mathbf{T}\mathbf{T}^T\right)\mathbf{D}\mathbf{c} = \mathbf{c}^T\mathbf{D}^T\mathbf{D}\mathbf{c} \;=\; \|\mathbf{D}\mathbf{c}\|_2^2. \qquad \square$$

□

Since we have $\min_{\|\mathbf{c}\|_2=1} \|\mathbf{D}\mathbf{c}\|_2 = \sigma_{\min}$, where $\sigma_{\min}$ is the smallest singular value of $\mathbf{D}$, the corresponding right singular vector solves the problem. The problem is, however, better conditioned and can be implemented in a more efficient way than the weak approach [2].

3.2 Implementation of the Chebyshev method

The choice of using Chebyshev polynomials for the orthogonal basis is made mainly for computational reasons; the coefficients can be generated via a fast algorithm. This utilizes an existing method outlined for univariate polynomials in [11], which exploits the discrete orthogonality of Chebyshev polynomials at Chebyshev points.

Here we briefly describe the algorithm for efficient generation of tensor-product Chebyshev coefficients. The univariate Chebyshev points of degree L in the interval $[0,1]$ are given by:

$$t_{j,L} = (1 - \cos(j\pi/L))/2, \quad j = 0, \ldots, L.$$

The Chebyshev coefficients of any tensor product polynomial f of bidegree no higher than (L_1, L_2) can then be generated by the following procedure [11]:

- Construct a matrix $\mathbf{f} = (f(t_{i,L_1}, t_{j,L_2}))_{i=0,j=0}^{L_1,L_2}$ of values of the function f at the tensor-product Chebyshev points,

- Extend $\mathbf{f}$ to its even counterpart $\hat{\mathbf{f}}$:

$$\begin{aligned}
\hat{f}_{i,j} &= f_{i,j}, \;\; i = 0, \ldots, L_1, \; j = 0, \ldots, L_2, \\
\hat{f}_{L_1+i,j} &= f_{L_1-i,j}, \;\; i = 1, \ldots, L_1 - 1, \; j = 0, \ldots, L_2, \\
\hat{f}_{i,L_2+j} &= f_{L_1-i,j}, \;\; i = 0, \ldots, L_1, \; j = 1, \ldots, L_2 - 1, \\
\hat{f}_{L_1+i,L_2+j} &= f_{L_1-i,L_2-j}, \;\; i = 1, \ldots, L_1 - 1, \; j = 1, \ldots, L_2 - 1.
\end{aligned}$$

- perform a bivariate fast Fourier transform (FFT) to get $\tilde{\mathbf{f}} = \mathrm{FFT}(\hat{\mathbf{f}})$,

- extract the first $(L_1 + 1, L_2 + 1)$ coefficients of $\tilde{\mathbf{f}}$ to get $\mathbf{g} = (\tilde{f}_{i,j})_{i,j=0}^{L_1,L_2}$. The matrix $\mathbf{g}$ then contains the tensor product Chebyshev coefficients of f.

The algorithm for approximate implicitization proceeds by applying the above procedure to the functions $\{w^d(\beta_k \circ \mathbf{p})\}_{k=1}^{M}$ and $\{-h^2\alpha_l\}_{l=1}^{L_1L_2}$, and arranging the coefficients in matrices $\mathbf{D}_q$ and $\mathbf{D}_\lambda$ according to the definition (7). The efficiency of the method is due to it being based on point sampling and FFT. Moreover, the sampling can be done entirely in parallel making the method highly suitable for implementation on heterogeneous architectures.

4 Numerical results

In this section we present an example of the method along with both computation times and estimations for the convergence rates. In order to generate reliable data, we have chosen a degree six family of lines which has a rational envelope. We can thereby use a parametrization of the envelope to compute the algebraic error of the approximations. The family of lines is pictured in Figure 1, along with the envelope approximations up to the exact implicitization at degree six. For these approximations we take $k_1 = \max(0, dn_1 - 2\deg_s(h))$ and $k_2 = \max(0, dn_2 - 2\deg_t(h))$, since this is also the minimum needed for the exact solution.

It can be seen that with increased degree the approximations converge quickly. It is possible that with higher degrees, extra branches may appear in the region of interest. For example, the approximation of degree five has an extra branch close to the envelope curve. However, such artifacts could be avoided using a suitable collection of low-degree approximations (see [9] for an adaptive algorithm).

In Table 1 we show the computation times for the above approximations. The algorithm has been implemented in the Python programming language using the NumPy library for the built in FFT and singular value decomposition (SVD) algorithms. The results are computed on a 3.4Ghz Intel Core i7-2600 with 8GB RAM.

Instead of increasing the polynomial degree d, one may also improve the quality of the approximations by subdivision; the envelope is then approximated by a piecewise implicit representation. It is thus of interest to see how the approximation improves as the region $\Omega = I \times J$ is reduced.

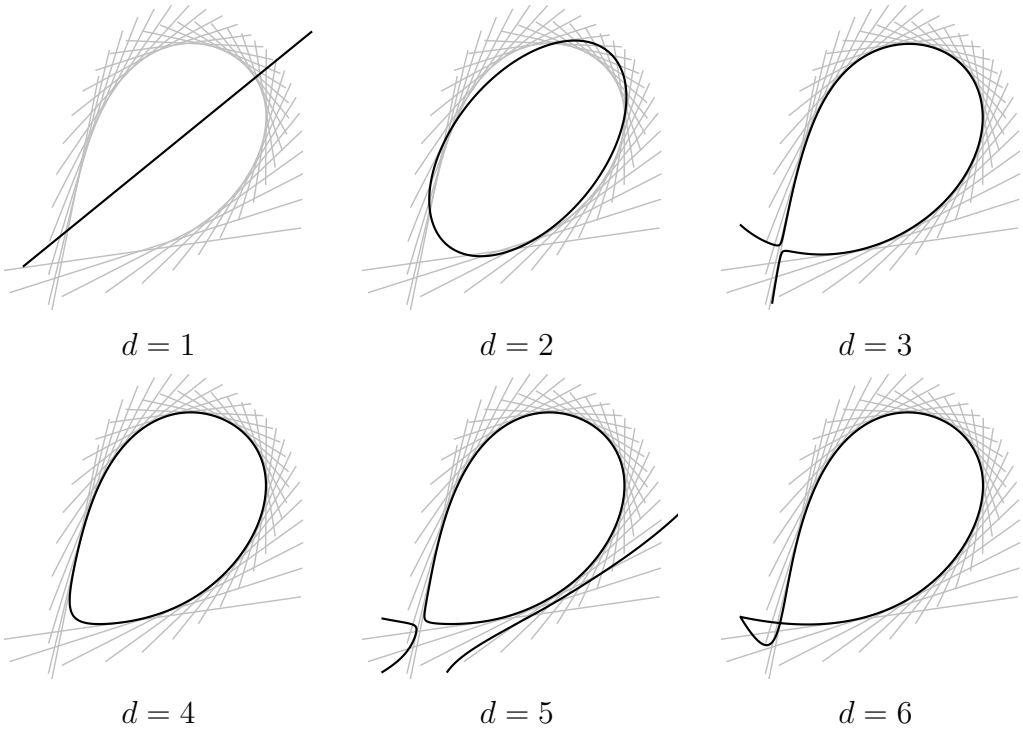


Figure 1: Approximations of the envelope of a family of lines for degrees $d = 1, \ldots, 6$.

Degree d	1	2	3	4	5	6
# coefficients	196	975	2964	7000	14136	25641
Time (s)	0.02	0.04	0.11	0.23	0.45	0.80

Table 1: Computation times and number of matrix coefficients for the examples in Fig. 1.

		Implicit Degree d							
		1		2		3		4	
		$\epsilon_{d,i}$	$r_{d,i}$	$\epsilon_{d,i}$	$r_{d,i}$	$\epsilon_{d,i}$	$r_{d,i}$	$\epsilon_{d,i}$	r_i
Diam.	1	1.69e-1	-	6.23e-3	-	1.16e-4	-	3.96e-6	-
	1/2	1.67e-1	0.02	3.46e-4	4.170	2.62e-6	5.467	2.66e-10	13.86
	1/4	3.04e-2	2.458	1.52e-5	4.511	1.50e-9	10.77	1.34e-14	14.27
	1/8	6.52e-3	2.223	5.02e-7	4.915	2.87e-12	9.028	n/a	n/a
	1/16	1.41e-3	2.213	1.58e-8	4.989	5.63e-15	8.993	n/a	n/a

Table 2: Maximum algebraic error $\epsilon_{d,i}$, of the approximations of the example in Fig.1, together with approximate convergence rates $r_{d,i}$.

Consider a region $\Omega_i = I_i \times J_i$, of diameter 2^{-i} centered on a point (s_0, t_0) in

$$\mathcal{H} = \{(s,t) \in \Omega : h(s,t) = 0\}.$$

For an approximation $q_{d,i}$ of degree d to over the region Ω_i, we define the maximum algebraic error to be

$$\epsilon_{d,i} = \max_{(s,t)\in\mathcal{H}\cap\Omega_i} |q_{d,i}(\mathbf{p}(s,t))|,$$

where the coefficients $\mathbf{c}_{q_{d,i}}$ of $q_{d,i}$ have been renormalized to $\|\mathbf{c}_{q_{d,i}}\| = 1$, in order to give meaningful results. Given two approximations $q_{d,i}$, and $q_{d,i+1}$, on subsequent subdivision regions Ω_i and Ω_{i+1}, we define the convergence rate to be $r_{d,i} = \log_2(\epsilon_{d,i}/\epsilon_{d,i+1})$. Table 2 shows values of $\epsilon_{d,i}$ and $r_{d,i}$ for four successive subdivisions of the example in Figure 1 and degrees d up to four. Values of $\epsilon_{d,i}$ below machine precision have been omitted.

As can be seen from Table 2, the error $\epsilon_{d,i}$ decreases both with increased degree and increased levels of subdivision. The values of $r_{d,i}$, suggest that the convergence rates for $d = 1, 2, 3$ and 4 are approximately two, five, nine and 14 respectively. This corresponds directly to the number of degrees of freedom in approximating with lines, conics, cubics and quartics and is hence as high a convergence as we can expect, supporting our choices for the degrees (k_1, k_2). The results in Table 2 are typical of rational examples we have tested.

It should be noted that in general, envelope curves are not rational. Thus, this example, whilst showing that high convergence rates are attainable, cannot conclude that this is always the case. However, from studying additional examples, our experience shows that convergence behaviour is good in the general setting.

5 Conclusion

We have presented a new implementation of approximate implicitization of envelope curves using Chebyshev polynomials. We have detailed the computation times and convergence behaviour of a specific example, thereby demonstrating the feasibility of our approach. This paper also motivates theoretical work on convergence rates as a direction for future research.

Acknowledgements: The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement n° PITN-GA-2008-214584 (SAGA), and from the Research Council of Norway (IS-TOPP). It was also supported by the Doctoral Program "Computational Mathematics" (W1214) at the Johannes Kepler University of Linz.

Bibliography

[1] K. Abdel-Malek, J. Yang, D. Blackmore, and K. Joy. Swept volumes: foundation, perspectives, and applications. *International Journal of Shape Modeling*, 12(1):87–127, 2006.

[2] O.J.D. Barrowclough and T. Dokken. Approximate implicitization using linear algebra. *Journal of Applied Mathematics*, 2012. doi:10.1155/2012/293746.

[3] T. Dokken. Approximate implicitization. *Mathematical methods for curves and surfaces, Oslo 2000*, pages 81–102, 2001.

[4] T. Dokken and J.B. Thomassen. Weak approximate implicitization. In *IEEE International Conference on Shape Modeling and Applications, 2006. SMI 2006*, pages 204–214, 2006.

[5] C.M. Hoffmann. Implicit curves and surfaces in CAGD. *IEEE Computer Graphics and Applications*, 13(1):79–88, 1993.

[6] Y.J. Kim, G. Varadhan, M.C. Lin, and D. Manocha. Fast swept volume approximation of complex polyhedral models. *Computer-Aided Design*, 36(11):1013–1027, 2004.

[7] M. Peternell, H. Pottmann, T. Steiner, and H. Zhao. Swept volumes. *Computer-Aided Design Appl*, 2:599–608, 2005.

[8] M. Rabl, B. Jüttler, and L. Gonzalez-Vega. Exact envelope computation for moving surfaces with quadratic support functions. In Lenarčič and Wenger, editors, *Adv. in Robot Kinematics: Analysis and Design*, pages 283 – 290. Springer, 2008.

[9] T. Schulz and B. Jüttler. Envelope computation in the plane by approximate implicitization. *Appl. Algebra in Eng., Comm. and Comp.*, 22:265–288, 2011.

[10] J.R. Sendra, F. Winkler, and S. Perez-Diaz. *Rational algebraic curves*. Springer, 2007.

[11] Lloyd N. Trefethen. *Spectral methods in MATLAB*. SIAM, Philadelphia, PA, USA, 2000.

Paper IV: A basis for the implicit representation of rational cubic Bézier curves

Oliver J.D. Barrowclough

Submitted for publication

Abstract: We present an approach to finding the implicit equation of a planar rational parametric cubic curve, by defining a new basis for the representation. The basis, which contains only four cubic bivariate polynomials, is defined in terms of the Bézier control points of the curve. An explicit formula for the coefficients of the implicit curve is given. Moreover, these coefficients lead to simple expressions which describe aspects of the geometric behaviour of the curve. In particular, we present an explicit barycentric formula for the position of the double point, in terms of the Bézier control points of the curve. We also give conditions for when an unwanted singularity occurs in the region of interest. Special cases in which the method fails, such as when three of the control points are collinear, or when two points coincide, will be discussed separately.

1 Introduction

Parametric curves are widely used in CAGD applications, especially in the ubiquitous rational Bézier and B-spline forms, due to their natural geometric properties. It is well known that all planar rational parametric curves can be written in implicit form [11]; that is, as the zero set of a single bivariate polynomial function. The availability of both the implicit and parametric representations, which have properties complementary to each other, is important for various applications in CAGD, such as intersection and surface trimming algorithms. The increase in the presence of GPUs in commodity computers has also led to renewed interest in implicit representations for rendering applications [8, 9]. Since traditionally the design phase happens using the parametric representation, a great deal of research has focussed on *implicitization* - the conversion from the rational parametric, to the implicit form.

Implicitization algorithms, both exact and approximate, have been investigated by many authors [1, 2, 3, 4, 11]. The specific case of implicitization of *planar rational cubic curves* has also received particular attention. In [10], Sederberg et al. present a method which reduces the degrees of freedom in the implicit polynomial from nine to eight by introducing monoid curves.

In [7], Floater makes a choice of basis that allows the implicit representation to be given in terms of only six basis functions - a reduction from the 10 basis functions required to represent all polynomials of total degree three. Floater also gives explicit formulas for the coefficients and conditions for when the curve degenerates to a conic.

Despite being motivated by methods for sparse implicitization [5], the techniques in this paper more closely resemble those of Sederberg et al. [10, 11] and Floater [7]. The method describes how, in most cases, the implicit form of the curve can be defined in terms of only four basis functions. The basis functions are constructed from the control points of a rational cubic curve given in Bernstein-Bézier form. In addition, the coefficients which describe the implicit representation are shown to contain a lot of information about the geometry of the curve. We present an explicit barycentric formula for the position of the double point of the curve in terms of its control points, along with criteria for when the double point is considered 'unwanted'. The important case of degeneration to conic sections is also treated. In the special case when three of the control points are collinear, the method fails; potential remedies for this will also be discussed.

Although much of the paper will utilize a Cartesian system for describing and proving the results, the method is in fact independent of the coordinate system, and is stated in terms of purely geometric quantities. In addition, all of the formulas (except Eq. (16)), including the detection and location of singularities on the curve, can be implemented in exact rational arithmetic; that is, only the operations of addition, subtraction multiplication and division are required. This contrasts with other methods, which often use rootfinding algorithms to find the double point via the parametric representation [7, 8, 9, 14]. This is potentially useful in applications which require exact precision and also aids the speed of floating point implementations.

There are several applications of implicit representations of cubic curves. These include resolution independent curve rendering, as in [8, 9], and intersection algorithms, as in [10, 12, 14]. We also envisage great potential for the use of the techniques in this paper in surface trimming algorithms. A piecewise implicit representation of cubic trimming curves in the parameter domain will give a simple and accurate test for whether a point lies inside or outside the trimming region. The geometric formulas presented in this paper are also interesting from a theoretical perspective.

The paper will proceed as follows. In Section 2 we present the construction of the basis we use for the cubic implicitization, and also present formulas for the coefficients which define the curve. Computation of the double point of the curve will be addressed in Section 3, and Section 4 will cover the case of cubics which degenerate to conic sections. The special case of collinear control points will be discussed in Section 5. Several examples will be presented in Section 6, which highlight the simplicity of the method. We conclude the paper with a discussion of the extension to higher degrees and further work in Section 7. Some extended proofs and basic geometric properties are deferred to the Appendices.

2 A basis for representing rational cubic Bézier curves implicitly

In this section we describe the implicit basis functions and prove some simple facts such as invariance of the coefficients under affine transformations. We begin with some definitions.

2.1 Preliminaries

We assume that the planar rational cubic curve is given in Bernstein-Bézier form with control points $\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3$, and weights w_0, w_1, w_2, w_3. That is, we have

$$\mathbf{p}(t) = \frac{\mathbf{c}_0 w_0 (1-t)^3 + \mathbf{c}_1 3 w_1 (1-t)^2 t + \mathbf{c}_2 3 w_2 (1-t) t^2 + \mathbf{c}_3 w_3 t^3}{w_0 (1-t)^3 + 3 w_1 (1-t)^2 t + 3 w_2 (1-t) t^2 + w_3 t^3}. \tag{1}$$

In the parametric form, Bézier curves are normally rendered within a region of interest corresponding to the parameter values $t \in [0, 1]$. It is also common in the CAGD community to require that the weights are positive. Although such a restriction is easier for us to work with, for the most part, it is not strictly necessary; the method also works with negative weights, zero weights and weights of mixed sign. However, due to the construction which follows, we do require that *no three of the control points are collinear*. It is important to note that this will be an assumption of all the results up to Section 5.

In the following definition, we assume the control points are given in a Cartesian system, $\mathbf{c}_i = (c_{i,0}, c_{i,1})$.

Definition 1. *We define the implicit equation of the line between* $\mathbf{c}_i$ *and* $\mathbf{c}_j$ *to be given by* $L_{ij}(x, y) = 0$, *where*

$$L_{ij}(x, y) = \begin{vmatrix} x & y & 1 \\ c_{i,0} & c_{i,1} & 1 \\ c_{j,0} & c_{j,1} & 1 \end{vmatrix}.$$

We may refer loosely to 'the line L_{ij},' meaning 'the line defined by the equation $L_{ij}(x, y) = 0$'. Note that the norm of the gradient of L_{ij} is equal to the Euclidean distance between the points $\mathbf{c}_i$ and $\mathbf{c}_j$: [1]

$$\|\nabla L_{ij}\|_2 = \|\mathbf{c}_j - \mathbf{c}_i\|_2.$$

Definition 2. *We define a quantity* λ_{ijk} *as follows:*

$$\lambda_{ijk} = \begin{vmatrix} c_{i,0} & c_{i,1} & 1 \\ c_{j,0} & c_{j,1} & 1 \\ c_{k,0} & c_{k,1} & 1 \end{vmatrix}.$$

For compactness of notation we define $\lambda_i = (-1)^{i+1} \lambda_{i+1,i+2,i+3}$ *where the indices* $i+1$, $i+2$ *and* $i+3$ *are taken modulo 4. That is,*

$$\lambda_0 = \lambda_{321}, \quad \lambda_1 = \lambda_{230}, \quad \lambda_2 = \lambda_{103}, \quad \lambda_3 = \lambda_{012}. \tag{2}$$

[1] The linear functions L_{ij}, can in fact be characterised by the three conditions of vanishing at $\mathbf{c}_i$ and $\mathbf{c}_j$, and having constant gradient, proportional to the Euclidean distance between the points. However, for the sake of clarity, we proceed using the definition in the Cartesian system.

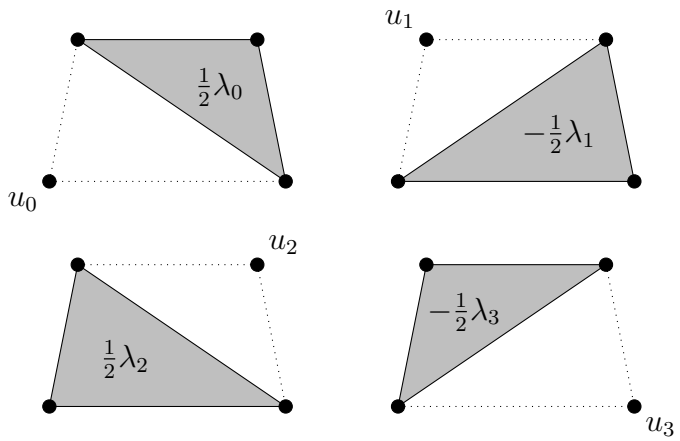


Figure 1: The definition of the coefficients λ_i corresponds to twice the signed areas of the shaded regions. The corresponding weights u_i appear on the opposite vertex.

We also make the definition

$$u_i = \binom{3}{i} w_i,$$

for each of the weights $(w_i)_{i=0}^3$ of the cubic Bézier curve.

Clearly $\lambda_{ijk} = 0$ if any of the i, j, k's are equal. The quantities λ_i represent twice the signed areas of the triangles formed from the control points $(\mathbf{c}_i)_{i=0}^3$, by omitting the point $\mathbf{c}_i$ (i.e., $\mathbf{c}_{i+1}, \mathbf{c}_{i+2}$ and $\mathbf{c}_{i+3}$). The areas, λ_i, are pictured with the corresponding weights, u_i, in Figure 1.

2.2 Implicit basis functions for rational cubic curves

The following definition describes the basis functions we use for the implicit representation:

Definition 3. *We define four basis functions as follows:*

$$\begin{aligned} K_0(x,y) &= L_{01}(x,y)L_{12}(x,y)L_{23}(x,y), \\ K_1(x,y) &= L_{01}(x,y)L_{13}(x,y)^2, \\ K_2(x,y) &= L_{02}(x,y)^2 L_{23}(x,y), \\ K_3(x,y) &= L_{03}(x,y)^3. \end{aligned}$$

A diagrammatic representation of these basis functions is shown in Figure 2. The functions $(K_i)_{i=0}^3$ can be thought of as the implicit representations of the limiting configurations of non-negative weights w_1 an w_2, with $w_0 = w_3 = 1$. For example, K_3 is the implicit polynomial of the rational cubic curve with weights $w_1 = w_2 \to 0$; that is, the line $L_{03}(x, y)$ between $\mathbf{c}_0$ and $\mathbf{c}_3$ with multiplicity three, and parametric representation

$$\frac{\mathbf{c}_0(1-t)^3 + \mathbf{c}_3 t^3}{(1-t)^3 + t^3}.$$

Similarly, K_0 can be thought of as the limiting implicit equation as $w_1 = w_2 \to \infty$, and the other two cases as $1/w_2 = w_1 \to \infty$, and $1/w_1 = w_2 \to \infty$, respectively.

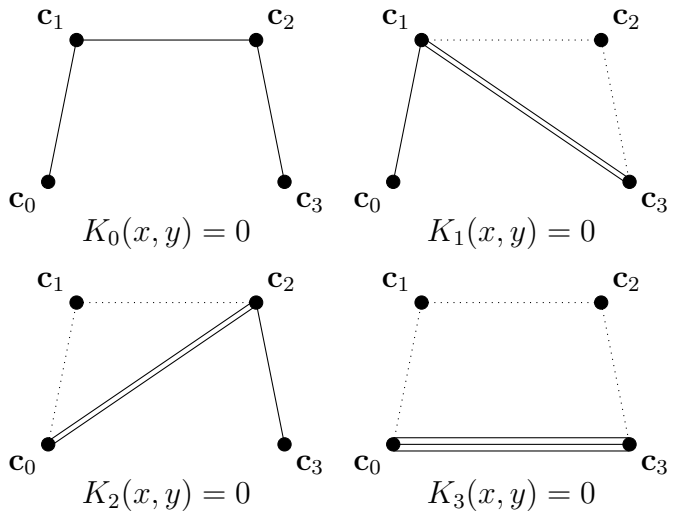


Figure 2: A diagrammatic representation of the zero sets of the basis functions $(K_i)_{i=0}^3$. The number of lines between any two points $\mathbf{c}_i$ and $\mathbf{c}_j$, reflect the multiplicity with which $L_{ij}(x,y)$ appears in the basis function.

It can be shown that the basis functions are linearly dependent if and only if no three control points are collinear. However, we defer this result to Appendix 4.B.

In the following Theorem we establish an explicit formula for the implicit representation of the rational cubic Bézier curve $\mathbf{p}(t)$, in terms of these basis functions.

Theorem 1. *Suppose we are given a non-degenerate cubic Bézier curve, $\mathbf{p}(t)$, such that no three of the control points are collinear. Then the curve has the following equation defining its implicit representation:*

$$q(x,y)=\sum_{i=0}^{3} b_i K_i(x,y)$$

where

$$\begin{aligned} b_0 &= -\lambda_1^2\lambda_2^2 U + u_1^2 u_2^2 \Lambda, \\ b_1 &= \lambda_1^3\lambda_3 U - u_1^3 u_3 \Lambda, \\ b_2 &= \lambda_0\lambda_2^3 U - u_0 u_2^3 \Lambda, \\ b_3 &= \lambda_0^2\lambda_3^2 U - u_0^2 u_3^2 \Lambda. \end{aligned} \tag{3}$$

and $U=\prod_{k=0}^3 u_i$ and $\Lambda=\prod_{k=0}^3 \lambda_i$.

Due to its length, we also defer the proof of this theorem to Appendix 4.B. It is interesting to note that the degrees to which the u_is and λ_is appear in the coefficient formula (3), are closely related to the multiplicities of the basis functions $(K_i)_{i=0}^3$ at the vertices $(\mathbf{c}_k)_{k=0}^3$.

Proposition 2. *The coefficients $(b_i)_{i=0}^3$ defined by (3) are invariant under affine transformations, up to a constant scaling.*

Proof. Formally, suppose we are given a Bézier curve $\mathbf{p}(t)$ with control points $\mathbf{c}_i$ and weights w_i, whose implicit coefficients $(b_i)_{i=0}^3$ are defined by (3). Then, for any affine map Φ, we claim

that the implicit coefficients of the transformed curve $\Phi(\mathbf{p}(t))$, are given by $(\alpha b_i)_{i=0}^3$, for some non-zero constant α. Since affine transformations multiply areas by a non-zero constant, there exists a constant C such that $\tilde{\lambda}_i = C\lambda_i$ where $\tilde{\lambda}_i$ are the areas defined by control points $\tilde{\mathbf{c}}_i$, after the affine transformation (i.e., $\tilde{\mathbf{c}}_i = \Phi\mathbf{c}_i$, for each $i = 0, 1, 2, 3$). Now, since the weights u_i are unchanged, and the λ_is appear homogeneously of degree four in the definition (3), we have $\tilde{b}_i = C^4 b_i$, for each i. □

Since implicit representations are unchanged by non-zero scalar multiplication, we can clearly factor out the constant C^4.

2.3 Evaluating the implicit equation

One potential disadvantage of the method described in this paper, is that when the implicit equation is evaluated, samples are taken at the six different lines in the set

$$\{L_{ij}(x, y) : 0 \leq i < j \leq 3\}.$$

This can, however, be reduced to three evaluations by some simple relations between the lines.

Proposition 3. *Suppose we are given any four points $(\mathbf{c}_i)_{i=0}^3$, with no three collinear. Then, using Definitions 1 and 2, we can write*

$$\lambda_j L_{ij}(x, y) + \lambda_k L_{ik}(x, y) + \lambda_l L_{il}(x, y) \equiv 0, \tag{4}$$

for any choice of $i, j, k, l \in \{0, 1, 2, 3\}$, with all indices distinct.

Proof. Let

$$r(x, y) = \lambda_j L_{ij}(x, y) + \lambda_k L_{ik}(x, y) + \lambda_l L_{il}(x, y).$$

Then

$$r(\mathbf{c}_j) = \lambda_k L_{ik}(\mathbf{c}_j) + \lambda_l L_{il}(\mathbf{c}_j) = \pm(\lambda_k\lambda_l - \lambda_l\lambda_k) = 0,$$

since $L_{ij}(\mathbf{c}_j) = 0$, and $L_{ik}(\mathbf{c}_j)$ and $L_{il}(\mathbf{c}_j)$ must have opposite signs. Similarly, $r(\mathbf{c}_k) = 0$ and $r(\mathbf{c}_l) = 0$. Thus, since r is a linear function which is zero at three non-collinear points, it must be identically zero. □

This Proposition gives us an alternative method to evaluate some of the functions $L_{ij}(x, y)$. We assume that we are given the lines L_{01}, L_{12} and L_{23}, corresponding to the lines in the control polygon. It can easily be shown, using (4), that

$$\begin{aligned}
L_{02}(x, y) &= \frac{\lambda_3 L_{23} - \lambda_1 L_{12}}{\lambda_0}, \\
L_{13}(x, y) &= \frac{\lambda_0 L_{01} - \lambda_2 L_{12}}{\lambda_3}, \\
L_{03}(x, y) &= \frac{\lambda_1\lambda_2 L_{12} - \lambda_0\lambda_1 L_{01} - \lambda_2\lambda_3 L_{23}}{\lambda_0\lambda_3}.
\end{aligned}$$

When using this method as a numerical technique, care should be taken to ensure a sufficient degree of numerical stability. For example, if the denominators become small, it may be better

to choose a different set of three lines to evaluate on, or to compute each line evaluation explicitly. A similar method is used by Sederberg and Parry in [12], in order to simplify the symbolic expansion of the determinant required in their method.

2.4 Properties of the coefficients

In the following definition we give three quantities which can be used to determine several characteristics of the geometry of the curve, such as when the curve degenerates, and in what region the singularity lies.

Definition 4. *We define three quantities ϕ_1, ϕ_2 and ϕ_3 as follows:*

$$\begin{aligned}\phi_1 &= u_0u_2\lambda_1^2 - u_1^2\lambda_0\lambda_2,\\ \phi_2 &= u_1u_3\lambda_2^2 - u_2^2\lambda_1\lambda_3,\\ \phi_3 &= u_1u_2\lambda_0\lambda_3 - u_0u_3\lambda_1\lambda_2.\end{aligned}$$

These quantities are based on the coefficients $(b_i)_{i=0}^3$ and we can write

$$\begin{aligned}b_0 &= \phi_3u_1u_2\lambda_1\lambda_2,\\ b_1 &= \phi_1u_1u_3\lambda_1\lambda_3,\\ b_2 &= \phi_2u_0u_2\lambda_0\lambda_2,\\ b_3 &= \phi_3u_0u_3\lambda_0\lambda_3.\end{aligned} \tag{5}$$

We thus have a compact form of the implicit equation

$$\begin{aligned}q(x,y) = \phi_3(u_1u_2\lambda_1\lambda_2K_0(x,y) + u_0u_3\lambda_0\lambda_3K_3(x,y))\\ + \phi_1u_1u_3\lambda_1\lambda_3K_1(x,y) + \phi_2u_0u_2\lambda_0\lambda_2K_2(x,y).\end{aligned} \tag{6}$$

There is also a relation between the three quantities given as follows:

Proposition 4. *The following two equations hold:*

$$u_3\lambda_2\phi_1 + u_1\lambda_0\phi_2 + u_2\lambda_1\phi_3 = 0,$$

and

$$u_2\lambda_3\phi_1 + u_0\lambda_1\phi_2 + u_1\lambda_2\phi_3 = 0.$$

Proof. The proof is a simple exercise in expanding polynomials after substituting the definitions ϕ_1, ϕ_2 and ϕ_3. We therefore omit the details here. □

3 Double points on cubic curves

Since the curve is rational, there always exists a single double point in the form of a crunode, an acnode or a cusp. It is a surprising fact that the double point of a rational cubic curve with rational coefficients is necessarily rational, despite the fact that the corresponding parameter

values may be irrational or complex. This was apparently first noticed by Sederberg in [10] (Theorem 1):

"Every rational cubic curve defined by polynomials with rational coefficients has a double point whose coordinates are real and rational".

In this section we derive the equations of two lines which intersect at the double point, in terms of the coefficients we have already discussed. We give exact formulas for the location of the double point in terms of barycentric combinations of its control points. We also define a condition which detects when there is an 'unwanted' self-intersection in the region of interest.

We first some give identities for the gradient which will be needed in the proofs in this section. From Definition 1 it is clear that we have

$$\frac{\partial}{\partial x} L_{ij}(x,y) = c_{i1} - c_{j1},$$

and

$$\frac{\partial}{\partial y} L_{ij}(x,y) = c_{j0} - c_{i0}.$$

For compactness of notation we define $\mathbf{c}_{ij}^{\perp} = \begin{pmatrix} c_{i1} - c_{j1} \\ c_{j0} - c_{i0} \end{pmatrix}$. Thus, using the product rule, we can write the gradients of the basis functions as follows:

$$\begin{aligned}
\nabla K_0(x,y) &= \mathbf{c}_{23}^{\perp} L_{01} L_{12} + \mathbf{c}_{12}^{\perp} L_{01} L_{23} + \mathbf{c}_{01}^{\perp} L_{12} L_{23}, \\
\nabla K_1(x,y) &= \mathbf{c}_{13}^{\perp} 2 L_{01} L_{13} + \mathbf{c}_{01}^{\perp} L_{13}^2, \\
\nabla K_2(x,y) &= \mathbf{c}_{02}^{\perp} 2 L_{02} L_{23} + \mathbf{c}_{23}^{\perp} L_{02}^2, \\
\nabla K_3(x,y) &= \mathbf{c}_{03}^{\perp} 3 L_{03}^2.
\end{aligned} \tag{7}$$

In addition, using (6), we can write the gradient function, ∇q, in the compact form

$$\begin{aligned}
\nabla q(x,y) = \phi_3 (u_1 u_2 \lambda_1 \lambda_2 \nabla K_0(x,y) + u_0 u_3 \lambda_0 \lambda_3 \nabla K_3(x,y)) \\
+ \phi_1 u_1 u_3 \lambda_1 \lambda_3 \nabla K_1(x,y) + \phi_2 u_0 u_2 \lambda_0 \lambda_2 \nabla K_2(x,y).
\end{aligned} \tag{8}$$

3.1 Location of the singularity

In this section we locate the position of the singularity in affine space. The following proposition deals with the case where the singularity occurs at one of the endpoints $\mathbf{c}_0$ or $\mathbf{c}_3$.

Proposition 5. *Let the control points and weights of a non-degenerate rational cubic Bézier curve be given, and assume that no three control points are collinear. Then the singularity occurs at the end point* $\mathbf{c}_0$ *if and only if* $\phi_2 = 0$ *or* $u_1 = 0$. *Similarly, the singularity occurs at the end point* $\mathbf{c}_3$ *if and only if* $\phi_1 = 0$ *or* $u_2 = 0$.

Proof. Since the functions $(L_{0i})_{i=1}^3$, evaluated at the endpoint $\mathbf{c}_0$ are all zero, we have $\nabla K_2(\mathbf{c}_0) = \nabla K_3(\mathbf{c}_0) = 0$. Thus

$$\begin{aligned}
\nabla q(\mathbf{c}_0) &= \mathbf{c}_{01}^{\perp} \left(L_{12}(\mathbf{c}_0) L_{23}(\mathbf{c}_0) \left(u_1^2 u_2^2 \Lambda - \lambda_1^2 \lambda_2^2 U \right) + L_{13}^2(\mathbf{c}_0) \left(\lambda_3^3 \lambda_1 U - u_1^3 u_1 \Lambda \right) \right) \\
&= \mathbf{c}_{01}^{\perp} \Lambda u_1^2 \left(u_2^2 \lambda_1 \lambda_3 - u_1 u_3 \lambda_2^2 \right) \\
&= -\mathbf{c}_{01}^{\perp} \Lambda u_1^2 \phi_2.
\end{aligned}$$

Clearly this is zero if and only if $\phi_2 = 0$ or $u_1 = 0$, since $\mathbf{c}_0$ and $\mathbf{c}_1$ are distinct. Similarly, $\nabla q(\mathbf{c}_3) = 0$ if and only if $\phi_1 = 0$ or $u_2 = 0$. □

In non-degenerate cases, except for those with the conditions above, we can derive the equations of two lines, $\tilde{S}_1$ and $\tilde{S}_2$, which both intersect the singularity and either $\mathbf{c}_0$ or $\mathbf{c}_3$.

Proposition 6. *Suppose we have a planar rational cubic Bézier curve with no three control points collinear and non-zero weights. Suppose further that the double point of the curve is finite, and does not lie on either of the endpoints* $\mathbf{c}_0$ *or* $\mathbf{c}_3$*. Define two equations as follows:*

$$\begin{aligned} \tilde{S}_1(x,y) &= L_{02}(x,y)u_2\phi_1 - L_{03}(x,y)u_1\phi_3 = 0, \\ \tilde{S}_2(x,y) &= L_{13}(x,y)u_1\phi_2 - L_{03}(x,y)u_2\phi_3 = 0. \end{aligned} \tag{9}$$

The equations (9) define two lines which intersect the end points of the Bézier curve, $\mathbf{c}_0$ *and* $\mathbf{c}_3$ *respectively. Moreover, the lines intersect each other at the unique double point of the curve. The lines are also determined by the alternative equations*

$$\begin{aligned} \hat{S}_1(x,y) &= L_{01}(x,y)u_2\phi_1 - L_{03}(x,y)u_0\phi_2 = 0, \\ \hat{S}_2(x,y) &= L_{23}(x,y)u_1\phi_2 - L_{03}(x,y)u_3\phi_1 = 0. \end{aligned} \tag{10}$$

Proof. We can immediately see that $\tilde{S}_1(\mathbf{c}_0) = 0$ since, by definition, it is a linear combination of L_{02} and L_{03}. Similarly, $\tilde{S}_2(\mathbf{c}_3) = 0$. It remains to show that both lines intersect at the double point of q.

We now use both equations (9) and (10), which can easily be shown to be equivalent, using Proposition 3. We must prove that both the implicit polynomial q and its gradient ∇q, vanish when these equations are satisfied. From (9) and (10) we infer that

$$L_{01} = \frac{L_{03}u_0\phi_2}{u_2\phi_1},\ L_{23} = \frac{L_{03}u_3\phi_1}{u_1\phi_2},\ L_{02} = \frac{L_{03}u_1\phi_3}{u_2\phi_1},\ L_{13} = \frac{L_{03}u_2\phi_3}{u_1\phi_2}, \tag{11}$$

and

$$L_{12} = \frac{L_{23}\lambda_3 - L_{02}\lambda_0}{\lambda_1} = L_{03}\left(\frac{\phi_1 u_3\lambda_3}{\phi_2 u_1\lambda_1} - \frac{\phi_3 u_1\lambda_0}{\phi_1 u_2\lambda_1}\right). \tag{12}$$

These are well defined since, by Proposition 5, we have $\phi_1 \neq 0$ and $\phi_2 \neq 0$, and the weights are non-zero. Thus, by (6), the formula for q becomes

$$\begin{aligned} q(x,y) &= \phi_3\lambda_0\lambda_3 u_0u_3L_{03}^3 + \phi_3\left(\frac{\phi_1\lambda_3u_3}{\phi_2u_1\lambda_1} - \frac{\phi_3\lambda_0u_1}{\phi_1u_2\lambda_1}\right)\lambda_1\lambda_2u_0u_3L_{03}^3 \\ &\qquad + \frac{\phi_3^2\lambda_1\lambda_3u_0u_2u_3}{\phi_2u_1}L_{03}^3 + \frac{\phi_3^2\lambda_0\lambda_2u_0u_1u_3}{\phi_1u_2}L_{03}^3, \\ &= L_{03}^3\frac{\lambda_3\phi_3u_0u_3}{\phi_2u_1}\left(\phi_2\lambda_0u_1 + \phi_3\lambda_1u_2 + \phi_1\lambda_2u_3\right), \\ &= 0, \end{aligned}$$

by Proposition 4.

In order to prove that $\nabla q = 0$ when $S_1 = S_2 = 0$, we substitute the equations (11) into the gradient identities (7). After expanding the formula (8), we can group together the coefficients

of $\mathbf{c}_{ij}^{\perp}$ to write

$$\nabla q = L_{03}^2 \left(\psi_{01}\mathbf{c}_{01}^{\perp} + \psi_{12}\mathbf{c}_{12}^{\perp} + \psi_{23}\mathbf{c}_{23}^{\perp} + \psi_{02}\mathbf{c}_{02}^{\perp} + \psi_{13}\mathbf{c}_{13}^{\perp} + \psi_{03}\mathbf{c}_{03}^{\perp}\right),$$

where

$$\begin{aligned}
\psi_{01} &= \phi_3 u_0 u_3 \lambda_0 \lambda_1, \\
\psi_{12} &= \phi_3 u_0 u_3 \lambda_1 \lambda_2, \\
\psi_{23} &= \phi_3 u_0 u_3 \lambda_2 \lambda_3, \\
\psi_{02} &= 2\phi_3 u_0 u_3 \lambda_0 \lambda_2, \\
\psi_{13} &= 2\phi_3 u_0 u_3 \lambda_1 \lambda_3, \\
\psi_{03} &= 3\phi_3 u_0 u_3 \lambda_0 \lambda_3.
\end{aligned}$$

The quantities ψ_{ij} can be verified by writing out the coefficients. Thus, we can factor out the common factor $\phi_3 u_0 u_3$, to get

$$\nabla q = L_{03}^2 \phi_3 u_0 u_3 \left(\lambda_0\lambda_1\mathbf{c}_{01}^{\perp} + \lambda_1\lambda_2\mathbf{c}_{12}^{\perp} + \lambda_2\lambda_3\mathbf{c}_{23}^{\perp} + 2\lambda_0\lambda_2\mathbf{c}_{02}^{\perp} + 2\lambda_1\lambda_3\mathbf{c}_{13}^{\perp} + 3\lambda_0\lambda_3\mathbf{c}_{03}^{\perp}\right).$$

Now, we can individually verify each of the following

$$\begin{aligned}
\lambda_0\lambda_1\mathbf{c}_{01}^{\perp} + \lambda_0\lambda_2\mathbf{c}_{02}^{\perp} + \lambda_0\lambda_3\mathbf{c}_{03}^{\perp} &= 0, \\
\lambda_2\lambda_3\mathbf{c}_{23}^{\perp} + \lambda_1\lambda_3\mathbf{c}_{13}^{\perp} + \lambda_0\lambda_3\mathbf{c}_{03}^{\perp} &= 0, \\
\lambda_1\lambda_2\mathbf{c}_{12}^{\perp} + \lambda_0\lambda_2\mathbf{c}_{02}^{\perp} + \lambda_1\lambda_3\mathbf{c}_{13}^{\perp} + \lambda_0\lambda_3\mathbf{c}_{03}^{\perp} &= 0.
\end{aligned}$$

Thus, summing the previous three equations, we must have $\nabla q = 0$, proving the statement. □

The lines defined in Proposition 6 are shown in Figure 3. It can be seen that the segments of the curve limited to the quadrants defined by the lines $\tilde{S}_1$ and $\tilde{S}_2$, are non-singular. This is necessarily true due to rational cubic curves having only one singularity. The following Theorem, which is a consequence of the previous Proposition, gives the location of the singularity in the affine plane.

Theorem 7. *Suppose that the double point* $\mathbf{s}$*, of a non-degenerate cubic curve, is not at infinity. Then the following barycentric combination of the four Béizer control points* $(\mathbf{c}_i)_{i=0}^3$ *determines its location exactly:*

$$\mathbf{s} = \frac{\mathbf{c}_0\lambda_1\phi_1\phi_3u_2^2 + \frac{1}{2}(\mathbf{c}_0\lambda_0 - \mathbf{c}_1\lambda_1 + \mathbf{c}_2\lambda_2 - \mathbf{c}_3\lambda_3)\phi_1\phi_2u_1u_2 - \mathbf{c}_3\lambda_2\phi_2\phi_3u_1^2}{\lambda_1\phi_1\phi_3u_2^2 + \frac{1}{2}(\lambda_0 - \lambda_1 + \lambda_2 - \lambda_3)\phi_1\phi_2u_1u_2 - \lambda_2\phi_2\phi_3u_1^2}$$

The formula can be simplified in a number of different ways, using the identities provided in the previous sections. For example, in the barycentric coordinate system defined by $\mathbf{c}_0$*,* $\mathbf{c}_2$ *and* $\mathbf{c}_3$*, we have*

$$\mathbf{s} = \frac{\mathbf{c}_0\phi_1^2u_2u_3 - \mathbf{c}_2\phi_1\phi_2u_1u_2 + \mathbf{c}_3\phi_2\phi_3u_1^2}{\phi_1^2u_2u_3 - \phi_1\phi_2u_1u_2 + \phi_2\phi_3u_1^2}, \tag{13}$$

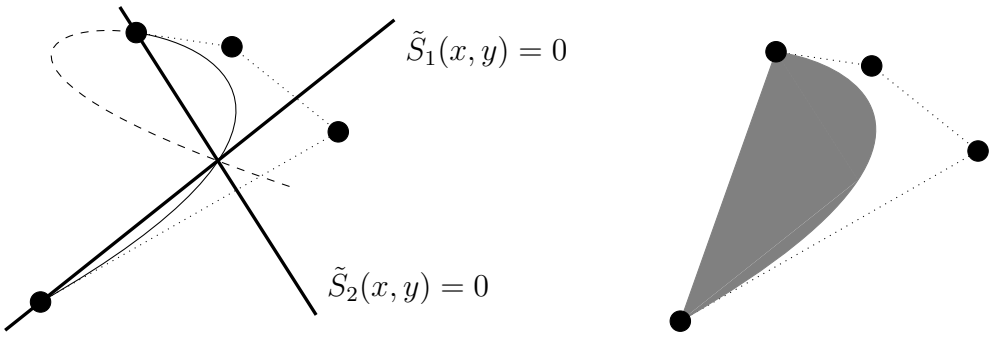


Figure 3: Two lines $\tilde{S}_1$ and $\tilde{S}_2$ (in bold, left), each defined by a linear combination of L_{ij}s, which intersect at the singularity. For an unwanted self-intersection (as pictured), the segments of the curves restricted to the quadrants defined by the two lines can be rendered separately, thus visually eliminating the singularity (right).

or in the barycentric coordinate system defined by $\mathbf{c}_0$, $\mathbf{c}_1$ *and* $\mathbf{c}_3$

$$\mathbf{s} = \frac{\mathbf{c}_0\phi_1\phi_3u_2^2 - \mathbf{c}_1\phi_1\phi_2u_1u_2 + \mathbf{c}_3\phi_2^2u_1u_0}{\phi_1\phi_3u_2^2 - \phi_1\phi_2u_1u_2 + \phi_2^2u_1u_0}. \tag{14}$$

Proof. The formulas can be found by solving the linear equations $\tilde{S}_1(x,y)=0$ and $\tilde{S}_2(x,y)=0$, in a particular coordinate system. It is clear that all the formulas are barycentric combinations of the control points, by observing the coefficients in the numerator and denominator. In order to validate that $\mathbf{s}$ is the intersection of the two lines, we can simply evaluate the functions $\tilde{S}_1$ and $\tilde{S}_2$ at $\mathbf{s}$:

$$\begin{aligned}
\tilde{S}_1(\mathbf{s}) &= \frac{u_1\phi_2}{\phi_1^2u_2u_3 - \phi_1\phi_2u_1u_2 - \phi_2\phi_3u_1^2}\left(-\tilde{S}_1(\mathbf{c}_2)\phi_1u_2 - \tilde{S}_1(\mathbf{c}_3)\phi_3u_1\right)\\
&= \frac{u_1\phi_2}{\phi_1^2u_2u_3 - \phi_1\phi_2u_1u_2 - \phi_2\phi_3u_1^2}\left(-L_{03}(\mathbf{c}_2)\phi_1\phi_3u_1u_2 - L_{02}(\mathbf{c}_3)\phi_1\phi_3u_1u_2\right)\\
&= 0.
\end{aligned}$$

We can similarly show that $\tilde{S}_2$ vanishes at $\mathbf{s}$, and thus, by Proposition 6, $\mathbf{s}$ it the unique double point of the curve. □

Of course, describing the singularity in terms of the control points may not be optimal with respect to numerical stability if the double point lies far from the region of interest, due to the denominator becoming small. However, information about the singularity is normally required only when it interferes with the region of interest of the curve. In these cases, the method performs very well.

3.2 Parametric identities

In complement to the previous section, it is also possible to derive formulas for the parameter values of the singularity in terms of the quantities ϕ_1, ϕ_2 and ϕ_3. For more compactness of notation we first make the following definitions:

Definition 5.

$$\Phi_1 = \phi_1 u_2 u_3, \quad \Phi_2 = \phi_2 u_0 u_1, \quad \Phi_3 = \phi_3 u_1 u_2.$$

Proposition 8. *Suppose we are given a non-degenerate rational cubic parametric curve with no three points collinear. Then the parameter values of the double point are given by the solutions to the quadratic equation* $r(t) = 0$, *where*

$$r(t) = \Phi_1 t^2 + \Phi_3 t(1-t) + \Phi_2 (1-t)^2. \tag{15}$$

Proof. By inserting the parametric form $\mathbf{p}(t)$, into $\tilde{S}_1$ (and factoring out the denominator), we obtain a cubic polynomial whose three roots correspond to the two parameter values t_1 and t_2 of the double point, and the parameter $t = 0$. After factoring out t from the polynomial we obtain $r(t) = \tilde{S}_1(\mathbf{p}(t))/t$, as given above. In the case that $\phi_2 = 0$, $\tilde{S}_1$ degenerates, however, we can simply follow a similar proof, using $\tilde{S}_2$ instead. □

The polynomial $r(t)$ is strictly quadratic except in the case when $\Phi_1 + \Phi_2 - \Phi_3 = 0$. In the non-degenerate case, this condition corresponds to when the singularity occurs at infinity. When $r(t)$ is quadratic, it can be solved explicitly, to give the parameters t_1 and t_2 of the double point as

$$\frac{2\Phi_2 - \Phi_3 \pm \sqrt{\Phi_3 - 4\Phi_1\Phi_2}}{2(\Phi_1 + \Phi_2 - \Phi_3)}. \tag{16}$$

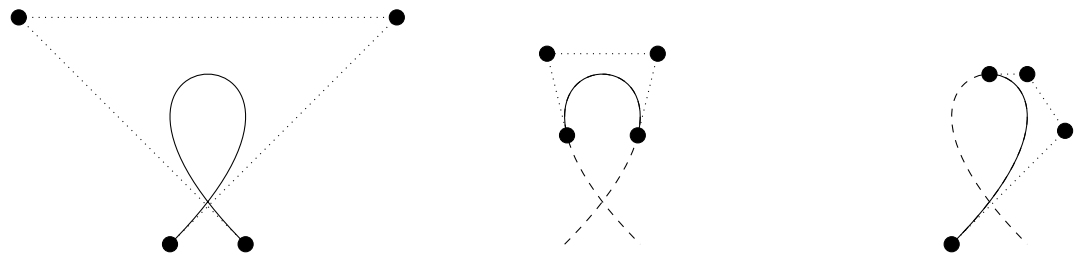
In particular, we have the discriminant $\Delta = \Phi_3^2 - 4\Phi_1\Phi_2$, which defines whether the curve has a self-intersection ($\Delta > 0$), a cusp ($\Delta = 0$), or an acnode ($\Delta < 0$). In the case that the $r(t)$ degenerates, we have a singularity at infinity, with a single inflection point. This appears to be similar to the discriminant described by Stone and DeRose in [13].

The lack of symmetry in the formula above is due to Bernstein polynomials being defined over the interval $[0, 1]$, as opposed to $[-1, 1]$.

3.3 Detecting Unwanted Self-intersections

The parameter values of the double point can occur in several ways. If the two parameters are real and distinct, then a self-intersection occurs; in the case that the two parameters are equal, we have a cusp; and parameter pairs which occur as complex conjugates give rise to isolated singular points known as acnodes.

When the curve is given in Bézier form, the region of interest is the parameter interval $[0, 1]$. If the parameters of the singularity both lie within the interval $[0, 1]$, the self-intersection is normally an intended product of the designer. If the parameters both lie outside the interval $[0, 1]$, this means there will be no singularities in the region of interest; again, this is normally intended by the designer. The case where one parameter lies within the interval and one lies outside is what we term an *unwanted self-intersection* or *unwanted singularity*. Figure 4 shows the three cases. When the parametric representation is used the unwanted case is not normally distinguished, since the curve is only plotted in the region of interest in the parameter domain. However, using implicit representations, it is more difficult to avoid plotting the curve without the unwanted branch and self-intersection, since a 2D region of rendering must be chosen.



(a) Singular segment (b) Non-singular segment (c) Unwanted singularity

Figure 4: Three Bézier representations of the same cubic curve. The solid part of the curve represents the region of interest and dashed part corresponds to parameter values outside $[0, 1]$.

It is possible to detect the presence of unwanted singularities by directly analysing the coefficients Φ_1 and Φ_2. In [7], a method is presented to detect unwanted singularities in special cases, which essentially correspond cubic Bézier curves which form simple arches [9]. The following proposition gives a condition to detect unwanted singularities in *all* non-degenerate cases. We make the condition that a singularity at infinity is not classified as unwanted so that we can assume that $\Phi_1 + \Phi_2 - \Phi_3 \neq 0$.

Proposition 9. *Let the control points and non-zero weights of a non-degenerate rational cubic Bézier curve be given, such that no three control points are collinear. Then there exists an unwanted singularity in the region of interest if and only if* $\Phi_1\Phi_2 < 0$.

Proof. This can be proved by observing that $r(t)$ is a quadratic polynomial in Bernstein form, with coefficients Φ_1, $\Phi_3/2$ and Φ_2, and by the properties of Bernstein polynomials.

($\Longleftarrow$) Assume first that Φ_1 and Φ_2 have opposite signs. Then, by observing the discriminant, we always have two real roots. Since $r(0) = \Phi_2$ and $r(1) = \Phi_1$, we know there is a root in $[0, 1]$ by the intermediate value theorem. We also know that there must be a root outside $[0, 1]$, because, since it is quadratic, $r(t)$ must have the same sign as $r(-t)$ asymptotically. Thus an unwanted self-intersection occurs (c.f. Figure 4(c)).

Assume now that Φ_1 and Φ_2 have the same sign. Since we are only interested in cases of self-intersection, where we have two distinct real roots, we consider the two cases, $\Phi_3 > \sqrt{4\Phi_1\Phi_2}$ and $\Phi_3 < -\sqrt{4\Phi_1\Phi_2}$. In one of the cases, all of the Bernstein coefficients of $r(t)$ have the same sign, so, by the variation diminishing property, both roots must be outside $[0, 1]$. In the other case, Φ_3 has opposite sign to Φ_1 and Φ_2, so the derivatives of r at $t = 0$ and $t = 1$ must have opposite sign. Again, by the intermediate value theorem, $r'(t) = 0$ for some $t \in [0, 1]$, thus both roots of $r(t)$ must be in $[0, 1]$. Thus either we have a singular or non-singular segment (c.f. Figure 4(a) and (b)).

($\Longrightarrow$) Assume first that $r(t)$ has one root in $[0, 1]$ and one outside $[0, 1]$. Then clearly, $r(0) = \Phi_2$ must have opposite sign to $r(1) = \Phi_1$.

Assume that both roots are in $[0, 1]$. Then $r(0) = \Phi_2$ must have the same sign as $r(1) = \Phi_1$. This is similarly the case if both roots are outside $[0, 1]$. □

Of course, in the standard case that all weights are positive, we can use the simplified condition $\phi_1\phi_2 < 0$.

In [8], Loop and Blinn use subdivision at the parameter values of the singularity in order to remove the unwanted branch. Their method involves solving a cubic polynomial for the parameter values of the self-intersection and then running the de Casteljau algorithm for the subdivision. The results of this section suggest an alternative method to remove the singularity. If, according to Proposition 9, we detect an unwanted singularity, we can use the lines defined in Proposition 6 to render non-singular segments of the curve. Such an approach is pictured in Figure 3 (right). This would appear to be advantageous since we avoid the necessity of dealing with two separate curves, and also avoid computing the parameter values of the singularity. Additionally, exact rational arithmetic can be used since we do not need to solve any polynomial equations.

4 Degeneration to conic sections

4.1 Conditions for conic degeneration

Similarly to the method of Floater [7], the coefficients $(b_i)_{i=0}^3$ disappear identically when the curve degenerates to a conic section. The following proposition defines conditions for conic degeneration of the cubic curve.

Theorem 10. *Suppose all the weights $(w_i)_{i=0}^3$ are non-zero and no three of the control points are collinear. Then the cubic curve degenerates to a conic if and only if $\phi_1 = 0$ and $\phi_2 = 0$.*

Proof. Assume we have a quadratic Bézier curve $\mathbf{r}(t)$, with points $\mathbf{a}_0$, $\mathbf{a}_1$ and $\mathbf{a}_2$ and respective weights v_0, v_1 and v_2. By degree elevation we can write the points and weights of the corresponding cubic Bézier curve as [6]:

$$\mathbf{c}_0 = \mathbf{a}_0, \quad \mathbf{c}_1 = \frac{v_0\mathbf{a}_0 + 2v_1\mathbf{a}_1}{v_0 + 2v_1}, \quad \mathbf{c}_2 = \frac{2v_1\mathbf{a}_1 + v_2\mathbf{a}_2}{2v_1 + v_2}, \quad \mathbf{c}_3 = \mathbf{a}_2$$
$$u_0 = v_0, \quad u_1 = v_0 + 2v_1, \quad u_2 = 2v_1 + v_2, \quad u_3 = v_2.$$

If we now compute the quantities λ_i, we see that

$$\lambda_i = \gamma_i \begin{vmatrix} a_{0,0} & a_{0,1} & 1 \\ a_{1,0} & a_{1,1} & 1 \\ a_{2,0} & a_{2,1} & 1 \end{vmatrix},$$

where

$$\gamma_0 = \frac{v_0 v_1}{(v_0 + 2v_1)(2v_1 + v_2)}, \quad \gamma_1 = \frac{v_1}{2v_1 + v_2},$$
$$\gamma_2 = \frac{v_1}{v_0 + 2v_1}, \quad \gamma_3 = \frac{v_1 v_2}{(v_0 + 2v_1)(2v_1 + v_2)}.$$

We can thus scale out the determinant factor, which is constant in each of the λ_i. We can now

easily check that the following equations are satisfied:

$$\begin{aligned} u_0 u_2 \gamma_1^2 &= u_1^2 \gamma_0 \gamma_2, \\ u_1 u_3 \gamma_2^2 &= u_2^2 \gamma_1 \gamma_3. \end{aligned}$$

Thus, by Definition 4, we have $\phi_1 = \phi_2 = 0$.

Now, assume that $\phi_1 = \phi_2 = 0$. By Proposition 5 we know that we have a double point at $\mathbf{c}_0$ if and only if $\phi_2 = 0$ (resp. double point at $\mathbf{c}_3$ iff $\phi_1 = 0$). Since both ϕ_1 and ϕ_2 are zero, and the points $\mathbf{c}_0$ and $\mathbf{c}_3$ are distinct, we have two double points. This can only occur when the curve degenerates to a conic. □

A consequence of the previous propositions, and the linear independency of the basis functions (see Appendix 4.B), is as follows.

Corollary 11. *Suppose that no three control points are collinear and the weights are non-zero. Then the following three statements are equivalent:*

1. *the cubic curve degenerates to a conic,*
2. *the coefficients $(b_i)_{i=0}$ are all zero,*
3. *the implicit polynomial q vanishes identically.*

Proof.

$(1) \implies (2)$: By Theorem 10 we have $\phi_1 = \phi_2 = 0$, and thus $\phi_3 = 0$ by Proposition 4. Then, by definition $b_i = 0$ for $i = 0, 1, 2, 3$.

$(2) \implies (3)$: Trivial.

$(3) \implies (1)$: By the linear independence of the basis functions, the coefficients must all be zero. Then, since the λ_is and u_is are non-zero, we have $\phi_1 = \phi_2 = 0$. The result follows from Theorem 10.

□

The results of this section are similar to those of Wang and Wang [15]. In particular Theorem 2 of [15], defines the same conditions for conic degeneration, but for only positive weights.

4.2 An implicit equation for conics given in cubic Bézier form

In the case of conic degeneration we can use formulas for implicitization from the rational quadratic parametric form of the conic section. However, a more direct approach is to use a formula for the implicit polynomial in terms of the lines and weights defined for the cubic Bézier curve. In addition, the following formula allows for cases which cannot be degree reduced using the same end points (e.g., when the end point tangents are parallel).

Proposition 12. *If the curve degenerates to a conic, its implicit equation is given by*

$$u_0 u_3 L_{03}(x, y)^2 - u_1 u_2 L_{01}(x, y) L_{23}(x, y) = 0.$$

The proof of this Proposition is deferred to Appendix 4.B.

5 Collinear points, zero weights and numerical stability

5.1 Collinear and coincident points

As emphasised earlier, the method fails when any three of the four control points are collinear, including the case when two of the points are coincident. In the collinear case, the basis functions $(K_i)_{i=0}^3$ are no longer linearly independent, and thus do not provide a basis for the curve. One remedy for this is to subdivide the curve using the de Casteljau algorithm, and treat the two subdivided curves separately. In the case of coincident points, two subdivisions are necessary to completely remove the collinearity.

So far, all the methods in this paper have been independent of the parametric form, and it would be nice to find a method of dealing with these cases without resorting to parametric subdivision. Unfortunately, explicit methods which incorporate all collinear configurations seem to be rather more complicated than the simple methods presented in this paper. It appears that different configurations of points require different basis functions and coefficients. For example, end point coindidence ($\mathbf{c}_1 = \mathbf{c}_2$), mid point coincidence ($\mathbf{c}_1 = \mathbf{c}_2$) and mixed point coincidence ($\mathbf{c}_0 = \mathbf{c}_1$, or $\mathbf{c}_2 = \mathbf{c}_3$, or $\mathbf{c}_0 = \mathbf{c}_2$, or $\mathbf{c}_1 = \mathbf{c}_3$) all appear to require separate treatment. In addition, there are also several cases of collinearity to consider.

Experimentally, it appears that the following basis functions support all cases, including collinear and coincident control points.

$$L_{03}^3, \quad L_{01}L_{13}^2, \quad L_{02}^2L_{23}, \quad L_{01}L_{12}L_{23}, \quad L_{12}L_{03}^2, \quad L_{02}L_{23}, \quad L_{01}L_{13}.$$

However, the explicit formula for the coefficients no longer holds, and the symmetries that were apparent earlier in this paper, appear to be lost. Due to the number of different cases and the comparative complexity of an explicit formula when trying to incorporate collinear points, we feel that parametric subdivision of the curve is currently the best option. However, this is the subject of ongoing research.

It may be noted that the test for cases of collinearity is not a difficult one. Since, during the implicitization we are using the λ_is as coefficients, if one is computed to be zero, we can instruct the algorithm to deal with that case appropriately. In the case of running the algorithm in floating point precision, we would specify collinearity to within given tolerance.

5.2 Zero weights

In the preceding sections we have mostly assumed that the weights are non-zero. In the CAGD community, it is fairly common to define Bézier curves as having non-zero, or positive weights. Indeed, if we allow some of the weights of a cubic curve to be zero, the curve often degenerates to a conic or a line. In such cases it would be better to model the curve as a lower degree parametric curve. However, if either $w_1 = 0$ or $w_2 = 0$ (but not both), then the curve does not degenerate to a conic section. These cases were treated in Proposition 5. The implicit representation given by Theorem 1, is still valid in these cases.

5.3 Numerical issues

When the curve has control points which are close to collinear, or the curve is close to a degenerate conic, issues with numerical stability need to be considered. Heuristically, it seems that issues with numerical stability are not too great; the methods appear to work well even when these 'close to degenerate' cases occur. However, when implementing in a given floating point precision, the tolerances required to define when a case is considered degenerate should be investigated further.

6 Examples

In this section we consider several examples for which the computations can easily be done by hand. Figure 5 shows five different curves with various properties. We include two cases which fail using the general method and need to be treated separately.

6.1 Three simple examples

For simplicity, the control points of the first three examples of Figure 5 are rearrangements of the points $(0,0)^T$, $(0,1)^T$, $(1,1)^T$ and $(1,0)^T$. For each case we compute the quantities $(\lambda_i)_{i=0}^3$ and $(\phi_i)_{i=0}^3$, the coefficients $(b_i)_{i=0}^3$, the double point $\mathbf{s}$, and whether or not the curve exhibits an unwanted singularity. For full clarity we describe the first example in detail.

We have

$$\mathbf{c}_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix},\ \mathbf{c}_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix},\ \mathbf{c}_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix},\ \mathbf{c}_3 = \begin{pmatrix} 1 \\ 0 \end{pmatrix},$$

and

$$w_0 = w_1 = w_2 = w_3 = 1.$$

The quantities λ_i are thus all equal to ± 1 and we have $u_0 = u_3 = 1$ and $u_1 = u_2 = 3$. Using the formula (5), we obtain

$$\phi_1 = -6,\ \phi_2 = -6 \text{ and } \phi_3 = -8$$

and thus

$$b_0 = 72,\ b_1 = -18,\ b_2 = -18,\ b_3 = 8.$$

The double point of the curve can be computed from (13) as

$$\mathbf{s} = \frac{-324(1,1)^T + 432(1,0)^T}{216} = \left(\frac{1}{2}, -\frac{3}{2}\right).$$

Clearly the double point does not lie within the convex hull of the control points. For the implicit equation in a Cartesian system we can write

$$q(x,y) = 72x(y-1)(x-1) - 18x(1-x-y)^2 - 18(x-y)^2(x-1) + 8y^3.$$

Since $\phi_1\phi_2 > 0$, we know, by Proposition 9, that there is no unwanted branch in the region of interest.

	$(\lambda_0, \lambda_1, \lambda_2, \lambda_3)$	(b_0, b_1, b_2, b_3)	(ϕ_1, ϕ_2, ϕ_3)	$\mathbf{s}$	$\phi_1\phi_2$
Ex. 1	$(1, -1, 1, -1)$	$(72, -18, -18, 8)$	$(-6, -6, -8)$	$(\frac{1}{2}, \frac{-3}{2})$	36
Ex. 2	$(-1, -1, 1, 1)$	$(72, -36, -36, 8)$	$(12, 12, -8)$	$(\frac{1}{2}, \frac{3}{4})$	144
Ex. 3	$(-1, 1, 1, -1)$	$(72, -36, -36, 8)$	$(12, 12, 8)$	(∞, ∞)	144
Ex. 4	$(1/3, -1, 1, -1/3)$	$(0, 0, 0, 0)$	$(0, 0, 0)$	n/a	0
Ex. 5	$(-1/2, 0, 1, -1/2)$	$(0, 0, -9/2, 9/16)$	$(9/2, 3, 9/4)$	n/a	n/a

Table 1: The computed quantities for a range of curves pictured in Figure 5. All examples use the same weights $u_0 = u_3 = 1$ and $u_1 = u_2 = 3$.

We summarize the quantities for the five examples of Figure 5, in Table 1. Note that in the third example we have a double point at infinity. In this case, the denominator in the formula (13) vanishes, as expected.

6.2 Example of conic degeneration

This is the fourth example of Figure 5. Having detected that both $\phi_1 = 0$ and $\phi_2 = 0$, Theorem 10 tells us we have a conic section. We thus use the formula of Proposition 12 for the implicit representation. This gives,

$$\begin{aligned} q(x, y) &= u_0 u_3 L_{03}(x, y)^2 - u_1 u_2 L_{01}(x, y) L_{23}(x, y), \\ &= y^2 + 9(x - y/3)(x + y/3 - 1). \end{aligned}$$

Since we have a conic section the double point computations are not applicable.

6.3 Example with collinear control points

The disappearance of λ_1 in this example indicates that a collinearity occurs between $\mathbf{c}_0$, $\mathbf{c}_2$ and $\mathbf{c}_3$. We see that there appears a linear dependency in the basis functions, between K_2 and K_3, and the coefficients b_0 and b_1 become zero. Following the suggestion of Section 5, we therefore subdivide the curve a single time, at the parameter value $t = 1/2$. This gives two curves with the following control points

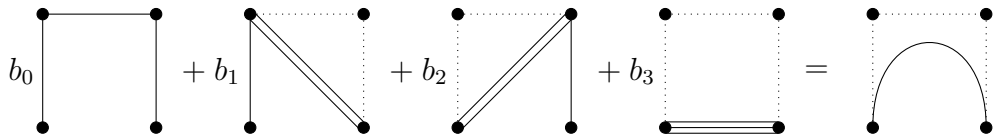
$$\mathbf{c}_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \mathbf{c}_1 = \begin{pmatrix} 0 \\ 1/2 \end{pmatrix}, \mathbf{c}_2 = \begin{pmatrix} 1/8 \\ 1/2 \end{pmatrix}, \mathbf{c}_3 = \begin{pmatrix} 5/16 \\ 3/8 \end{pmatrix},$$

and

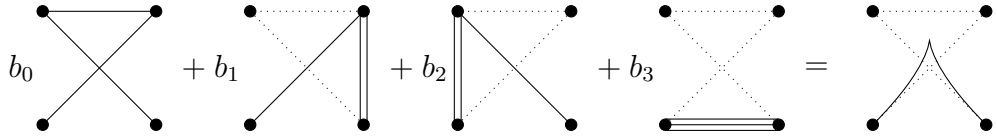
$$\mathbf{c}_0 = \begin{pmatrix} 5/16 \\ 3/8 \end{pmatrix}, \mathbf{c}_1 = \begin{pmatrix} 1/2 \\ 1/4 \end{pmatrix}, \mathbf{c}_2 = \begin{pmatrix} 3/4 \\ 0 \end{pmatrix}, \mathbf{c}_3 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

These can each be treated in the same way as the previous examples, finding in both cases that $\mathbf{s} = (-8, 36)^T$; an acnode.

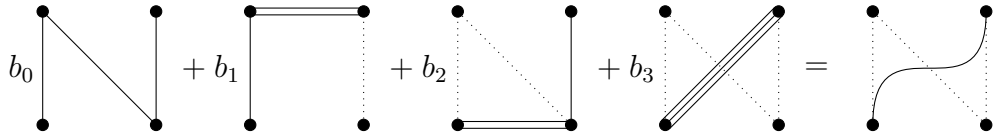
It may be noted that although the implicit equation vanishes identically, the double point can still be computed without subdivision if we choose the correct formula; that is by choosing the barycentric formula (14), with respect to the three non-collinear points $\mathbf{c}_0$, $\mathbf{c}_1$ and $\mathbf{c}_3$. We



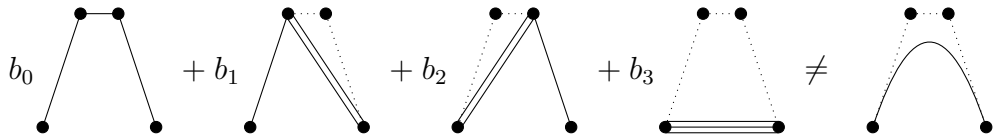
Example 1: Non-singular segment



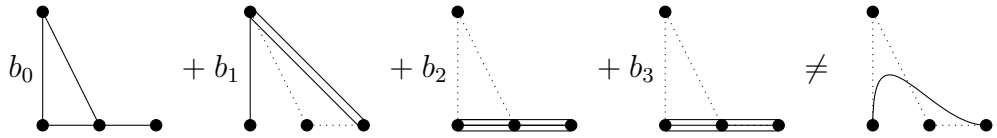
Example 2: Singular segment (cusp)



Example 3: Double point at infinity



Example 4: Degeneration to a conic



Example 5: Collinear control points

Figure 5: The quantities computed for each of these examples are shown in Table 1. The fourth and fifth examples fail. The fourth is a curve which degenerates to a conic, whereas the fifth has three collinear points. These cases need to be treated separately.

then get

$$\mathbf{s} = \frac{-243/2(0,1)^T + 27(1,0)}{-27/8} = \begin{pmatrix} -8 \\ 36 \end{pmatrix}.$$

6.4 Example with an unwanted singularity

For this example we use the control points for the curve pictured in Figure 3, given by

$$\mathbf{c}_0 = \begin{pmatrix} 1/4 \\ 0 \end{pmatrix},\ \mathbf{c}_1 = \begin{pmatrix} 9/8 \\ 1/2 \end{pmatrix},\ \mathbf{c}_2 = \begin{pmatrix} 13/16 \\ 3/4 \end{pmatrix},\ \mathbf{c}_3 = \begin{pmatrix} 17/32 \\ 19/24 \end{pmatrix},$$

with $w_0 = w_1 = w_2 = w_3 = 1$. Using the same computations as in the previous examples we get

$$\begin{aligned}
(\lambda_i)_{i=0}^3 &= (11/192, 15/64, 53/96, 3/8), \\
(b_i)_{i=0}^3 &= (312435/4194304, -66285/2097152, 220957/18874368, 1441/1048576), \\
(\phi_i)_{i=1}^3 &= (-491/4096, 379/3072, 131/2048), \\
\mathbf{s} &= (363241/470596, 146294/352947), \\
\phi_1\phi_2 &= -186089/12582912.
\end{aligned}$$

The negative value of $\phi_1\phi_2$ indicates that we have an unwanted singularity, the location of which is given by $\mathbf{s}$. Since the curve exhibits an unwanted singularity, we compute the lines $\tilde{S}_1$ and $\tilde{S}_2$, which intersect the double point:

$$\begin{aligned}
\tilde{S}_1(x,y) &= 965/8192x - 1215/8192y - 965/32768, \\
\tilde{S}_2(x,y) &= -12773/49152x - 10865/65536y + 17649/65536.
\end{aligned}$$

For a given point (x, y), we can then use boolean operations on the signs of $q(x, y)$, $\tilde{S}_1(x, y)$ and $\tilde{S}_2(x, y)$ in order to define which points lie 'inside' and 'outside' the curve.

7 Discussion and conclusion

For the sake of brevity, we have omitted extended discussions of interesting features that arise in using this method in the previous sections. In this section we mention some of these features before concluding the paper.

An alternative representation for the implicit coefficients is to divide through by the non-

zero factor $u_0u_1u_2u_3\lambda_0\lambda_1\lambda_2\lambda_3$:

$$\begin{aligned}
\tilde{b}_0 &= \frac{\lambda_1\lambda_2}{\lambda_0\lambda_3} - \frac{u_1u_2}{u_0u_3}, \\
\tilde{b}_1 &= \frac{\lambda_1^2}{\lambda_0\lambda_2} - \frac{u_1^2}{u_0u_2}, \\
\tilde{b}_2 &= \frac{\lambda_2^2}{\lambda_1\lambda_3} - \frac{u_2^2}{u_1u_3}, \\
\tilde{b}_3 &= \frac{\lambda_0\lambda_3}{\lambda_1\lambda_2} - \frac{u_0u_3}{u_1u_2}.
\end{aligned}$$

This form highlights more clearly the symmetry between the u_is and the λ_is. It also appears that this approach aids the numerical stability of the implementation, being only quadratic in the numerators and denominators.

It is interesting to note that the exponents which appear in the formulas for the coefficients (3), are reminiscent of the exponents of the terms of the discriminant of a univariate cubic polynomial in monomial form[2]. The relationship between the coefficients and the cubic discriminant should be the subject of further research.

Experiments show that parts of the method appear to be extensible to higher degrees. In particular, it is not difficult to define basis functions for quartic curves using the same heuristic reasoning as in Section 2. However, the number of basis functions appears to increase exponentially, and attempts to find an explicit formula for the coefficients appear to be more difficult. An extension of the theory to surfaces also appears to be much more difficult due to the complicated limiting control surfaces involved. However, this could be a direction for future research.

7.1 Conclusion

This paper has shown that it is possible to represent the implicit form of all non-degenerate rational planar cubic Bézier curves as a linear combination of four basis functions, when no three control points are collinear. The method has been described in terms of purely geometric quantities and symmetries have been highlighted. The resulting coefficients of the implicit polynomial lead naturally to a geometric characterization of several aspects of the curve. The method has a compact representation and can easily be implemented on a GPU, as an alternative to the methods in [8, 9]. Additionally, the formulas which aid in locating the singularity, and whether or not it is unwanted, are simple and computationally inexpensive.

Acknowledgements

I would like to thank Tor Dokken for reading through the manuscript and for helpful suggestions. The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement n° PITN-GA-2008-214584 (SAGA), and from the Research Council of Norway (IS-TOPP).

[2]There is a fifth term in the cubic discriminant that would correspond to $b_4 = u_0u_1u_2u_3\lambda_0\lambda_1\lambda_2\lambda_3 - u_0u_1u_2u_3\lambda_0\lambda_1\lambda_2\lambda_3 \equiv 0$.

Bibliography

[1] Oliver J.D. Barrowclough and Tor Dokken. Approximate implicitization using linear algebra. *Journal of Applied Mathematics*, 2012. doi:10.1155/2012/293746.

[2] Falai Chen and Wenping Wang. The μ-basis of a planar rational curve: properties and computation. *Graph. Models*, 64(6):368–381, November 2002.

[3] Robert M. Corless, Mark Giesbrecht, Ilias S. Kotsireas, and Stephen M. Watt. Numerical implicitization of parametric hypersurfaces with linear algebra. In John Campbell and Eugenio Roanes-Lozano, editors, *Artificial Intelligence and Symbolic Computation*, volume 1930 of *Lecture Notes in Computer Science*, pages 174–183. Springer Berlin/Heidelberg, 2001. 10.1007/3-540-44990-6_13.

[4] Tor Dokken. Approximate implicitization. In *Mathematical methods for curves and surfaces*, pages 81–102. Vanderbilt Univ. Press, Nashville, TN, 2001.

[5] Ioannis Z. Emiris and Ilias S. Kotsireas. Implicitization exploiting sparseness. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 67:281, 2005.

[6] Gerald Farin. *Curves and surfaces for CAGD: a practical guide*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[7] Michael S. Floater. Rational cubic implicitization. In M. Daehlen, T. Lyche, and L.L. Schumaker, editors, *Mathematical Methods for Curves and Surfaces*, pages 151–159. Vanderbilt University Press, 1995.

[8] Charles Loop and Jim Blinn. Resolution independent curve rendering using programmable graphics hardware. *ACM Trans. Graph.*, 24(3):1000–1009, July 2005.

[9] Ron Pfeifle. Rendering cubic curves on a GPU with Floater's implicitization. *Journal of Graphics Tools*, 16(2):105–122, 2012.

[10] Thomas W. Sederberg, David C. Anderson, and Ron N. Goldman. Implicitization, inversion, and intersection of planar rational cubic curves. *Computer Vision, Graphics, and Image Processing*, 31:89–102, 1985.

[11] Thomas W. Sederberg and Falai Chen. Implicitization using moving curves and surfaces. In *SIGGRAPH 95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 301–308, New York, NY, USA, 1995. ACM.

[12] Thomas W. Sederberg and Scott R. Parry. Comparison of three curve intersection algorithms. *Computer-Aided Design*, 1986.

[13] Maureen C. Stone and Tony D. DeRose. A geometric characterization of parametric cubic curves. *ACM Trans. Graph.*, 8(3):147–163, July 1989.

[14] Jan B. Thomassen. Self-intersection problems and approximate implicitization. In *Computational Methods for Algebraic Spline Surfaces*, pages 155–170. Springer Berlin Heidelberg, 2005.

[15] Guo-Jin Wang and Guo-Zhao Wang. The rational cubic Bézier representation of conics. *Computer Aided Geometric Design*, 1992.

Appendix 4.A Some geometric properties

Here we state some simple geometric properties that are used in the proofs of the next section.

Proposition 13. *For any four points* $\mathbf{c}_0$, $\mathbf{c}_1$, $\mathbf{c}_2$ *and* $\mathbf{c}_3$ *we have the following:*

$$\lambda_0 + \lambda_1 + \lambda_2 + \lambda_3 = 0.$$

Proof. This can be verified by simply writing out the expression using Definition 2, and checking that all terms cancel out. □

This proposition shows that there is some degeneracy in the representation; that is, one of the λ_is can always be written as a combination of the other three. This is reflected in the simplified forms presented in the paper. However, for the sake of symmetry, we have proceeded for the most part, to use all four λ_i values.

Proposition 14. *Assume we are given four points* $(\mathbf{c}_i)_{i=0}^3$ *with no three collinear. Then, when the respective denominators are non-zero, we can define*

$$\begin{aligned}
\mathbf{m}_1 &= \frac{\mathbf{c}_0\lambda_0 + \mathbf{c}_1\lambda_1 - \mathbf{c}_2\lambda_2 - \mathbf{c}_3\lambda_3}{\lambda_0 + \lambda_1 - \lambda_2 - \lambda_3} = \frac{\mathbf{c}_0\lambda_0 + \mathbf{c}_1\lambda_1}{\lambda_0 + \lambda_1} = \frac{\mathbf{c}_3\lambda_3 + \mathbf{c}_2\lambda_2}{\lambda_3 + \lambda_2}, \\
\mathbf{m}_2 &= \frac{\mathbf{c}_0\lambda_0 - \mathbf{c}_1\lambda_1 + \mathbf{c}_2\lambda_2 - \mathbf{c}_3\lambda_3}{\lambda_0 - \lambda_1 + \lambda_2 - \lambda_3} = \frac{\mathbf{c}_0\lambda_0 + \mathbf{c}_2\lambda_2}{\lambda_0 + \lambda_2} = \frac{\mathbf{c}_1\lambda_1 + \mathbf{c}_3\lambda_3}{\lambda_1 + \lambda_3}, \\
\mathbf{m}_3 &= \frac{\mathbf{c}_0\lambda_0 - \mathbf{c}_1\lambda_1 - \mathbf{c}_2\lambda_2 + \mathbf{c}_3\lambda_3}{\lambda_0 - \lambda_1 - \lambda_2 + \lambda_3} = \frac{\mathbf{c}_0\lambda_0 + \mathbf{c}_3\lambda_3}{\lambda_0 + \lambda_3} = \frac{\mathbf{c}_1\lambda_1 + \mathbf{c}_2\lambda_2}{\lambda_1 + \lambda_2}.
\end{aligned} \tag{4.17}$$

If any of these points do exist, they define the intersection of the lines L_{01} *and* L_{23}, L_{02} *and* L_{13}, *or* L_{03} *and* L_{12} *respectively.*[3]

Proof. That the various equalities hold, when the denominators are non-zero, is a consequence of Proposition 13. The fact that they intersect at the respective lines is then a triviality, since

[3] At least one of these points exists in the affine plane, since if two of the denominators vanish, then the two pairs of corresponding lines are parallel; but then the third point will be the intersection of the two lines passing through opposite vertices of the parallelogram thus formed. These lines must necessarily be non-parallel, thus the point of intersection is finite.

each point can be written as a scaled linear combination of the points which define the line. For example, $\mathbf{m}_1$ must lie on the line L_{01}, by the identity $\mathbf{m}_1 = \frac{\mathbf{c}_0\lambda_0+\mathbf{c}_1\lambda_1}{\lambda_0+\lambda_1}$, and on the line L_{23}, by $\mathbf{m}_1 = \frac{\mathbf{c}_3\lambda_3+\mathbf{c}_2\lambda_2}{\lambda_3+\lambda_2}$. □

Appendix 4.B Linear independence and proofs of Theorems

4.B.1 Linear independence

In the following proposition we establish linear independence of the basis functions, in the applicable cases.

Proposition 15. *Suppose no three of the points $(\mathbf{c}_i)_{i=0}^3$ are collinear. Then the functions K_0, K_1, K_2, K_3 are linearly independent.*

Proof. Assume that

$$\rho(x,y) = \sum_{i=0}^{3} b_i K_i(x,y) = 0, \quad \text{for all } (x,y) \in \mathbb{R}^2.$$

We prove linear independence by evaluating ρ at four distinct points. Assume that the points $\mathbf{m}_1$ and $\mathbf{m}_2$ defined by (4.17) exist. Then we can evaluate ρ at $\mathbf{c}_1, \mathbf{c}_2, \mathbf{m}_1$ and $\mathbf{m}_2$. For example, at $\mathbf{m}_2$ we have

$$\rho(\mathbf{m}_2) = \frac{\lambda_0^2\lambda_1\lambda_2\lambda_3^2 b_0 - \lambda_1^3\lambda_2^3 b_3}{\lambda_0+\lambda_2}.$$

After rearranging the rows to obtain a triangular matrix and dividing through by any common factors, we can set up the linear system with respect to evaluation at the four points as follows:

$$\begin{pmatrix} \lambda_0^2\lambda_3^2 & 0 & 0 & -\lambda_1^2\lambda_2^2 \\ 0 & \lambda_0^2\lambda_3 & 0 & -\lambda_1^3 \\ 0 & 0 & \lambda_3^2\lambda_0 & -\lambda_2^3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = 0.$$

Now, the determinant of the matrix can be computed as

$$\lambda_0^5\lambda_3^5$$

which never vanishes since the control points are not collinear. Care needs to be taken in the case when L_{01} and L_{23}, (resp. L_{02} and L_{13},) are parallel, as the denominator of $\mathbf{m}_1$ (resp. $\mathbf{m}_2$) vanishes. However, a similar linear system can be set up by using homogeneous coordinates, in which case the vanishing denominator is not a problem. Thus, the proof holds in all cases. □

4.B.2 Proof of Theorem 1

The proof of Theorem 1 is essentially a long exercise in expanding the rational function $q \circ \mathbf{p}$, in order to show that it is identically zero. We assume the conditions of Theorem 1 for the entirety of this section (i.e., that the cubic is non-degenerate and no three control points are collinear).

We know, by Proposition 15 that the basis functions $(K_i)_{i=0}^3$ are linearly independent, and by Proposition 10 that not all the coefficients $(b_i)_{i=0}^3$ are zero. Thus the polynomial q is not identically zero, and the theorem is proved if we can show that $q \circ \mathbf{p}$ vanishes identically.

We first consider the composition of $L_{ij}(\mathbf{p}(t))$ for all $i \neq j$.

Lemma 16. *The rational cubic function $L_{ij}(\mathbf{p}(t))$ can be given in Bernstein form by*

$$L_{ij}(\mathbf{p}(t)) = \frac{1}{w(t)} \sum_{\substack{k=0 \\ k \neq i,j}}^{3} \lambda_{ijk} w_k B_k(t),$$

where $B_k(t) = \binom{3}{k} t^k (1-t)^{3-k}$ and $w(t)$ denotes the denominator of (1).

Proof. We first note that by Definition 1,

$$L_{ij}(\mathbf{p}(t)) = \frac{1}{w(t)} \begin{vmatrix} p_0(t) & p_1(t) & w(t) \\ c_{i,0} & c_{i,1} & 1 \\ c_{j,0} & c_{j,1} & 1 \end{vmatrix}.$$

Now, by expanding the determinant, we have that

$$\begin{aligned} w(t)L_{ij}(\mathbf{p}(t)) &= (c_{i1} - c_{j1}) \sum_{k=0}^{3} c_{k0} w_k B_k(t) + (c_{j0} - c_{i0}) \sum_{k=0}^{3} c_{k1} w_k B_k(t) \\ &\qquad + (c_{i0}c_{j1} - c_{i1}c_{j0}) \sum_{k=0}^{3} w_k B_k(t), \\ &= \sum_{k=0}^{3} ((c_{i1} - c_{j1})c_{k0} + (c_{i0} - c_{j1})c_{k1} + (c_{i0}c_{j1} - c_{i1}c_{j0})) w_k B_k(t), \\ &= \sum_{k=0}^{3} \lambda_{ijk} w_k B_k(t). \end{aligned} \tag{4.18}$$

Clearly, when $k = i$ or $k = j$, the corresponding term in the sum is zero meaning we only need sum over $k \neq i, j$. □

A consequence of summing only over $k \neq i, j$ is that we can remove factors of t and $1 - t$ when certain coefficients disappear. That is, we can write

$$\begin{aligned} w(t)L_{01}(\mathbf{p}(t)) &= t^2(u_2\lambda_3(1-t) - u_3\lambda_2 t), \\ w(t)L_{12}(\mathbf{p}(t)) &= (u_0\lambda_3(1-t)^3 - u_3\lambda_0 t^3), \\ w(t)L_{23}(\mathbf{p}(t)) &= (1-t)^2(u_0\lambda_1(1-t) - u_1\lambda_0 t), \\ w(t)L_{02}(\mathbf{p}(t)) &= t(u_1\lambda_3(1-t)^2 - u_3\lambda_1 t^2), \\ w(t)L_{13}(\mathbf{p}(t)) &= (1-t)(u_0\lambda_2(1-t)^2 - u_2\lambda_0 t^2), \\ w(t)L_{03}(\mathbf{p}(t)) &= t(1-t)(u_1\lambda_2(1-t) - u_2\lambda_1 t). \end{aligned} \tag{4.19}$$

Thus by Lemma 16 and the above identities, we can express the compositions $(K_i \circ \mathbf{p})_{i=0}^3$ as follows:

Lemma 17. *For each $i = 0, 1, 2, 3$, we can express $K_i(\mathbf{p}(t))$ in the form*

$$K_i(\mathbf{p}(t)) = \frac{t^2(1-t)^2}{w(t)^3} G_i(t),$$

where

$$\begin{aligned}
G_0(t) &= (u_2\lambda_3(1-t) - u_3\lambda_2 t)(u_0\lambda_3(1-t)^3 - u_3\lambda_0 t^3)(u_0\lambda_1(1-t) - u_1\lambda_0 t),\\
G_1(t) &= (u_2\lambda_3(1-t) - u_3\lambda_2 t)(u_0\lambda_2(1-t)^2 - u_2\lambda_0 t^2)^2,\\
G_2(t) &= (u_1\lambda_3(1-t)^2 - u_3\lambda_1 t^2)^2(u_0\lambda_1(1-t) - u_1\lambda_0 t),\\
G_3(t) &= t(1-t)(u_1\lambda_2(1-t) - u_2\lambda_1 t)^3.
\end{aligned}$$

The common factor of $\frac{t^2(1-t)^2}{w(t)^3}$ can be ignored in showing that $q \circ \mathbf{p} \equiv 0$; it is thus sufficient to show that

$$\sum_{i=0}^{3} b_i G_i(t) \equiv 0. \tag{4.20}$$

It is a simple, yet lengthy exercise to compute the coefficients of this polynomial in the degree five Bernstein basis, in order to show that they are all zero. We compute the coefficient of $B_0^5(t) = (1-t)^5$ as an example. By observation, the coefficients $g_{i,0}$ of $B_0^5(t)$ of each of the functions $(G_i)_{i=0}^3$ are as follows:

$$\begin{aligned}
g_{0,0} &= u_0^2 u_2 \lambda_1 \lambda_3^2,\\
g_{1,0} &= u_0^2 u_2 \lambda_2^2 \lambda_3,\\
g_{2,0} &= u_0 u_1^2 \lambda_1 \lambda_3^2,\\
g_{3,0} &= 0.
\end{aligned}$$

Thus, the coefficient of $B_0^5(t)$ of (4.20), is given by

$$\sum_{i=0}^{3} b_i g_{i,0} = 0.$$

We can perform similar computations to show that the other coefficients (of $B_j^5(t) = \binom{5}{j} t^j (1-t)^{n-j}$, $j = 0, \ldots, 5$) are all zero, thus proving the theorem.

4.B.3 Proof of Proposition 12

We first prove the following Lemma

Lemma 18. *Let $q_2(x, y) = u_0 u_3 L_{03}(x, y)^2 - u_1 u_2 L_{01}(x, y) L_{23}(x, y)$. Then for any cubic curve $\mathbf{p}(t)$, given by (1), we have*

$$q_2(\mathbf{p}(t)) = \frac{t^2(1-t)^2}{w(t)^2} r(t),$$

where $r(t)$ is given by (15).

Proof. Using (4.19) we can write

$$q_2(\mathbf{p}(t)) = \frac{t^2(1-t)^2}{w(t)^2}(a_0(1-t)^2 + 2a_1t(1-t) + a_2t^2)$$

where

$$\begin{aligned} a_0 &= u_0u_1^2u_3\lambda_2^2 - u_0u_1u_2^2\lambda_1\lambda_3 = \Phi_2, \\ a_1 &= u_1^2u_2^2\lambda_0\lambda_3 + u_0u_1u_2u_3\lambda_1\lambda_2 - 2u_0u_1u_2u_3\lambda_1\lambda_2 = \frac{1}{2}\Phi_3, \\ a_2 &= u_0u_2^2u_3\lambda_1^2 - u_1^2u_2u_3\lambda_0\lambda_2 = \Phi_1. \end{aligned} \tag{4.21}$$

Thus, the terms inside the parentheses are given by $r(t)$. □

The proof of Proposition 12 is then immediate, since $r(t) \equiv 0$ when the curve degenerates to a conic, by Theorem 10.