Bernard Mourrain
Scott Schaefer
Guoliang Xu (Eds.)

LNCS 6130

Advances in Geometric Modeling and Processing

6th International Conference, GMP 2010
Castro Urdiales, Spain, June 2010
Proceedings

Springer

Lecture Notes in Computer Science 6130

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Bernard Mourrain Scott Schaefer
Guoliang Xu (Eds.)

Advances in Geometric Modeling and Processing

6th International Conference, GMP 2010
Castro Urdiales, Spain, June 16-18, 2010
Proceedings

Springer

Volume Editors

Bernard Mourrain
GALAAD, Inria Méditerranée
2004 route des Lucioles, 06902 Sophia Antipolis Cedex, France
E-mail: bernard.mourrain@sophia.inria.fr

Scott Schaefer
Texas A&M University, Department of Computer Science
College Station, TX 77843-3112, USA
E-mail: schaefer@cs.tamu.edu

Guoliang Xu
Chinese Academy of Science
Institute of Computational Mathematics and Scientific/Engineering Computing
Beijing 100080, China
E-mail: xuguo@lsec.cc.ac.cn

Library of Congress Control Number: 2010927597

CR Subject Classification (1998): I.3.5, G.2, I.5, I.4, F.2, F.2.2

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-642-13410-6 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-13410-4 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180

Preface

This volume contains the papers presented at 6th Conference on Geometric Modeling and Processing (GMP 2010) held in Castro Urdiales, Spain during June 16–18, 2010. Geometric Modeling and Processing is a biannual international conference series on geometric modeling, simulation and computing. Previously, GMP has been held in Hong Kong (2000), Saitama, Japan (2002), Beijing, China (2004), Pittsburgh, USA (2006) and Hangzhou, China (2008).

GMP 2010 received a total of 30 submissions that were reviewed by three to four Program Committee members on average. While the number of submissions dropped significantly from previous years, the quality did not and was still quite high overall. Based on the reviews received, the committee decided to accept 20 papers for inclusion in the proceedings. Additionally, extended versions of selected papers were considered for a special issue of *Computer-Aided Design (CAD)* and *Computer-Aided Geometric Design (CAGD)*. The paper topics spanned a wide variety and include:

- Solutions of transcendental equations
- Volume parameterization
- Smooth curves and surfaces
- Isogeometric analysis
- Implicit surfaces
- Computational geometry

Many people helped make this conference happen and we are grateful for their help. We would especially like to thank the Conference Chair, all of the authors who submitted papers, the Program Committee members who reviewed the papers and all of the participants at the conference.

Bernard Mourrain
Scott Schaefer
Guoliang Xu

Conference Organization

Conference Chair

Laureano Gonzalez-Vega University of Cantabria, Spain

Program Chairs

Bernard Mourrain	GALAAD, INRIA Mediterranée, France
Scott Schaefer	Texas A&M University, USA
Guo Liang Xu	Chinese Academy of Science Beijing, China

Program Committee

Marc Alexa
Alexander Belyaev
Mirela Ben-Chen
Tamy Boubekeur
Marie-Paule Cani
Falai Chen
Jiansong Deng
Tamal Dey
Neil Dodgson
Gershon Elber
Ioannis Z. Emris
Bianca Falcidieno
Gerald Farin
Andre Galligo
Xiao-Shan Gao
Ron Goldman
Xianfeng Gu
Stefanie Hahmann
Kai Hormann
ShiMin Hu
Tao Ju
Bert Juettler
Takashi Kanai
Misha Kazhdan
Young J Kim
Tae-wan Kim
Leif Kobbelt
Seungyong Lee
Bruno Levy
Guiqing Li
Hua Li
Ligang Liu
Weiyin Ma
Takashi Maekawa
Steve Mann
Dinesh Manocha
Ralph Martin
Knut Morken
Ashish Myles
Ahmad Nasri
Gregory Nielson
Nicholas M. Patrikalakis
Jorg Peters
Konrad Polthier
Hong Qin
Ulrich Reif
Jarek Rossignac
Malcom Sabin
Zbynek Sir
Luiz Velho
Johannes Wallner
Guozhao Wang
Guojin Wang
Wenping Wang

Joe Warren
Hongbin Zha
Qin Zhang
Caiming Zhang
Kun Zhou

External Reviewers

Bert Buchholz
Daniela Giorgi
Xian-Ying Li
Michela Mortara
Olga Sorkine
Jean-Marc Thiery
Guo-xin Zhang

Table of Contents

Global Solutions of Well-Constrained Transcendental Systems Using Expression Trees and a Single Solution Test

Maxim Aizenshtein, Michael Bartoň, and Gershon Elber

Department of Computer Science, Technion, Haifa, 32000, Israel
sniffer@t2.technion.ac.il,
{barton,gershon}@cs.technion.ac.il

Abstract. We present an algorithm which is capable of globally solving a well-constrained transcendental system over some sub-domain $D \subset \mathbb{R}^n$, isolating all roots. Such a system consists of n unknowns and n regular functions, where each may contain non-algebraic (transcendental) functions like sin, exp or log. Every equation is considered as a hyper-surface in $\mathbb{R}^n$ and thus a bounding cone of its normal field can be defined over a small enough sub-domain of D. A simple test that checks the mutual configuration of these bounding cones is used that, if satisfied, guarantees at most one zero exists within the given domain. Numerical methods are then used to trace the zero. If the test fails, the domain is subdivided. Every equation is handled as an expression tree, with polynomial functions at the leaves, prescribing the domain. The tree is processed from its leaves, for which simple bounding cones are constructed, to its root, which allows to efficiently build a final bounding cone of the normal field of the whole expression. The algorithm is demonstrated on curve-curve and curve-surface intersection problems.

1 Introduction and Previous Work

Solving nonlinear algebraic and/or transcendental systems of equations is a crucial problem in many fields such as computer-aided design, manufacturing, robotics, kinematics and many others. Robust and efficient algorithms that solve such systems are in strong demand. For instance, the problem of intersecting a parametric space curve with a parametric surface leads to a system consisting of three equations and three unknowns. Similarly, the problem of computing the closest point(s) on a curve or surface to a given point leads to a *well-constrained* polynomial/transcendental system (see, e.g., [17,18]). In these and similar applications, all solutions of a system of equations within a certain domain D, which is typically a box in $\mathbb{R}^n$, are sought for.

For *polynomial* systems, various methods exist. The symbolically oriented approaches like *Gröbner bases* and similar elimination-based techniques [3] map the original system to a simpler one, preserving the solution set. *Polynomial continuation methods* start at roots of a suitable simple system and transform it continuously to the desired one [15]. These methods handle the system in a

B. Mourrain, S. Schaefer, and G. Xu (Eds.): GMP 2010, LNCS 6130, pp. 1–18, 2010.

purely algebraic manner and give a general information about the solution set. These methods are typically not well-suited if only *real* roots are required.

Contrary to this, a family of solvers which focuses only on real roots has been introduced. These subdivision based schemes handle the polynomials as hyper-surfaces in $\mathbb{R}^n$ and exploit the convex hull property of its Bernstein-Bézier representations [4,8,10,12,14]. The domain is subdivided, sub-domains which can not contain a root are clipped away and (a set of) sub-domain(s) which may contain roots are returned. The subdivision is usually stopped if some numerical threshold is reached. In [8], a termination criterion which guarantees *at most one* root within a sub-domain has been proposed for isolating roots. Many others polynomial root-finding techniques exist. One survey can be found in [9].

In contrast, for the case of *transcendental* solvers, schemes which also support trigonometric and transcendental terms, the literature is not so extensive. Local root tracing techniques, such as Newton-Raphson iterations, are clearly employed once a close-to-a-root guess is available. One family of solvers is based on the reduction of the original n-dimensional system to one-dimensional nonlinear equations, see [6] and related work cited therein. Every function of the system is evaluated at $n-1$ variables and solved with respect to the remaining one. An approximation of the root is obtained and the process is iteratively repeated, converging quadratically to the root. Even though these methods use a reduction to a single equation, a good initial guess of the root is again needed and the detection of all the roots is not guaranteed.

Another iterative method was proposed in [7]. Every function of the system is considered as an *objective* function. The goal is to minimize these functions and the problem is essentially reduced to a multiobjective optimization problem. An evolutionary algorithm is proposed and a sequence of candidates that approximate the root is created. Again, similarly to [6], the process is iterative and a good initial guess is required to reach the root.

A different subdivision based approach that handles *some* transcendental systems was presented in [5]. The method presented therein is capable of solving an Extended Chebyshev (EC) systems and is well suited for systems consisting of exp function(s). In contrast, "if sin or cos functions are involved, its difficult to decide whether the system is EC or not." [5], (p. 94, section 4.3 and p. 82, 1st paragraph of Section 4.2). Making this decision fully automatic is an even more complex task. In contrast, our approach requires to subdivide only *polynomial* leaves that contain the variable which needs to be subdivided (and can be therefore efficiently achieved by using deCasteljeau algorithm). The numerical subdivision in [5] is based on multiplying a subdivision matrix, which is expensive and the numerical stability is guaranteed only for low-dimensional systems, see [5], (p. 82).

Another family of subdivision based solvers that support also transcendental functions relies on *interval arithmetic* [11]. These methods typically construct an interval bound on values that given function may attain over given domain. If the bound of some function of the system is no-zero containing, the particular

domain is discarded. Again, these schemes are difficult to guarantee numerical stability during subdivision and no root isolations are offered.

A major drawback of all the subdivision solvers stems from its exponential dependency on the dimension of the problem. In [4], an alternative representation of the equations, in a form of expression trees, is shown to present only polynomial dependency. Herein, we exploit another advantage of expression trees and show how they can be used to find the roots of sets of transendental functions.

In this paper, we present a "divide and conquer" algorithm which is capable of solving *transcendental* (non-algebraic), *well-constrained* system over some domain $D \subset \mathbb{R}^n$. Such a system consists of n unknowns and n regular functions, where each may contain transcendental functions like sin, exp or log. Every equation is considered as a hyper-surface in $\mathbb{R}^n$ and thus a bounding cone of its normal field can be defined over a small enough sub-domain of D. The termination criterion of [8] is exploited to check the mutual configuration of these bounding cones which, if satisfied, guarantees at most one zero within the given sub-domain, and hence offers a robust scheme to isolate all roots, globally. A multivariate Newton-Raphson method is then used to converge to the zero. Moreover, such a condition guarantees that the subdivision is not terminated until *all* roots are isolated, with the possibility of terminating at the permissible subdivision tolerance, in cases such as multiple roots.

The rest of the paper is organized as follows. Section 2 briefly recalls notions as transcendental systems, single solution criterion and expression trees. In section 3, the transcendental system's solver is presented. The construction of bounding cones is explained and its arithmetic is introduced. In section 4, the application of the proposed solver is demonstrated on curve-curve and curve-surface intersection problems. Finally, Section 5 identifies some possible future improvements of the presented method and concludes.

2 Preliminaries

The presented solver exploits both the expression trees representation [4] and the single solution termination test of [8]. A brief survey on these topics will be given. In 2.1, a non-algebraic system is defined and the single root termination criterion for such a system is introduced in 2.2. The notion of expression trees is recalled in 2.3.

2.1 Solving Well-Constrained Transcendental Systems

Definition 1. *Function* $f : \mathbb{R} \to \mathbb{R}$ *is* algebraic *over* $\mathbb{Q}$ *if there exist a polynomial* $p(x, y)$ *with integer coefficients* y *such that* $p(x, f(x)) = 0$. *Functions which are non-algebraic are called* transcendental.

Definition 2. *Consider the mapping* $\mathcal{F} : \mathbb{R}^n \to \mathbb{R}^n$, *such that at least one component* f_i, $i = 1, \ldots, n$ *of* $\mathcal{F}(\mathbf{x}) = \{f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_n(\mathbf{x})\}$ *is a transcendental function in variables* $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. *Then, every solution* $\mathbf{x}$ *of the system,*

$$\mathcal{F}(\mathbf{x}) = 0, \tag{1}$$

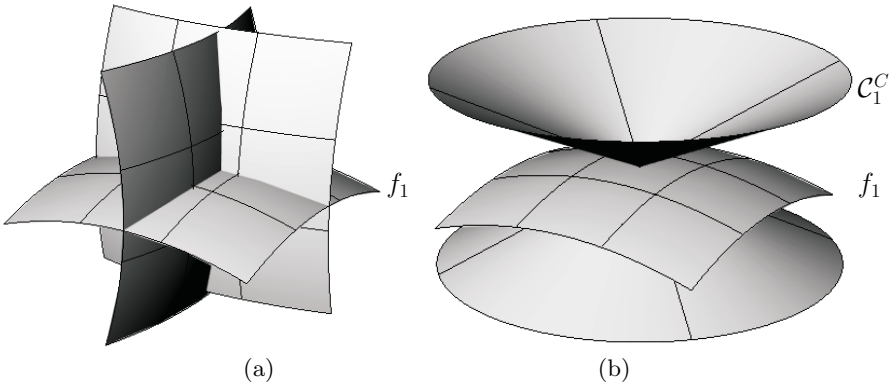


Fig. 1. (a) System (1) for $n = 3$, with single solution over some domain $D \subset \mathbb{R}^n$. (b) Complementary (tangent) circular bounding hyper-cone $\mathcal{C}_1^C$ of hyper-surface $f_1 = 0$.

is called a root *of* $\mathcal{F}$ *and the set of all roots is known as the* zero set *of the* transcendental mapping $\mathcal{F}$. *The determinant of Jacobian matrix,* $(\frac{\partial f_i}{\partial x_j}(\mathbf{x}))$, *is referred to as a* Jacobian *of system (1) at* $\mathbf{x}$.

In general, system (1) has a zero set of dimension zero. Assume system (1) is *well-constrained* in some sub-domain $D \subseteq \mathbb{R}^n$ and $\mathbf{a} = (a_1, a_2, \ldots, a_n) \in D$ is a root. By well-constrained we mean that Jacobian of system (1) never vanishes in (the vicinity of) any its root. If there is a guarantee that $\mathbf{a}$ is the *only root* of (1) in D, some numerical technique, like the multivariate Newton–Raphson method [16], can be used to try and robustly converge on that root. Hence, such a criterion is strongly desired.

2.2 Single Solution Termination Criterion for Transcendental Systems

In [8], a single solution criterion was formulated for (piecewise) *polynomial* systems. Considering every $f_i(\mathbf{x})$, $i = 1, \ldots, n$, from system (1) as hyper-surface in $\mathbb{R}^n$, its bounding hyper-cone of the normal field, and subsequently bounding hyper-cone of the complementary (tangential) field, was created. From the mutual position of all n complementary hyper-cones, an existence of at most one root can be determined. See Fig. 1 and [8] for more.

Since this idea is general, regardless of the type of the system (polynomial, transcendental), we adopt this approach and, in a similar manner, test the mutual position of all corresponding tangent bounding hyper-cones. Obviously, the construction of these hyper-cones, unlike the polynomial case, can not be accomplished from the control points of hyper-surfaces $f_i(\mathbf{x}) = 0$ (there are no control points anymore) and it will be explained later, in Section 3. Since the proposed technique is based on the bound of normal fields (gradients), all hyper-surfaces

are required to be regular and C^1 continuous over the domain of interest. We start by formulating two definitions:

Definition 3. *Consider implicit function* $f_i(\mathbf{x}) = 0$, $\mathbf{x} \in \mathbb{R}^n$ $i = 1, \ldots, n$ *over some (rectangular) sub-domain* $D \subset \mathbb{R}^n$. *We define the* normal field *of the implicit function* $f_i(\mathbf{x}) = 0$ *over sub-domain* D *by*

$$\mathcal{N}_i = \{\nabla f_i(\mathbf{x}), \mathbf{x} \in D\}, \tag{2}$$

where $\nabla f_i(\mathbf{x}) = (\frac{\partial f_i}{\partial x_1}, \frac{\partial f_i}{\partial x_2}, \ldots, \frac{\partial f_i}{\partial x_n})$ *is the gradient of* f_i.

Definition 4. *Consider circular hyper-cone in* $\mathbb{R}^n$ *with the axis in the direction of unit vector* $\boldsymbol{v}_i$ *and an opening angle* α_i *as*

$$\mathcal{C}_i^N(\boldsymbol{v}_i, \alpha_i) = \{\boldsymbol{u} | \langle \boldsymbol{u}, \boldsymbol{v}_i \rangle = \|\boldsymbol{u}\| \cos \alpha_i\}, \tag{3}$$

We say that $\mathcal{C}_i^N$ *is a* bounding normal hyper-cone *of function* f_i *if*

$$\langle \boldsymbol{u}, \boldsymbol{v}_i \rangle \geq \|\boldsymbol{u}\| \cos \alpha_i, \quad \forall \boldsymbol{u} \in \mathcal{N}_i. \tag{4}$$

By a complementary (or tangent) bounding hyper-cone, $\mathcal{C}_i^C$, *we denote*

$$\mathcal{C}_i^C(\boldsymbol{v}_i, \alpha_i) = \mathcal{C}_i^N(\boldsymbol{v}_i, 90^{\text{o}} - \alpha_i). \tag{5}$$

Remark 1. In the remainder of the paper, if no misunderstanding can occur, we call the circular bounding normal hyper-cone as *bounding normal cone* and the circular complementary bounding hyper-cone as *bounding tangent cone*.

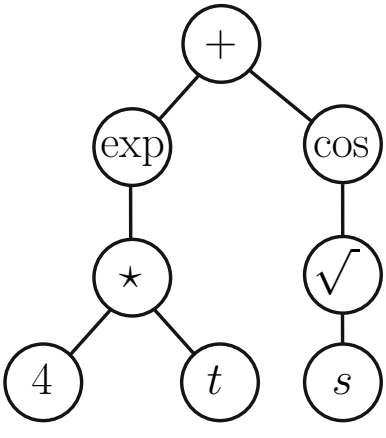


Fig. 2. Binary tree for $f(s,t) = e^{4t} + \cos(\sqrt{s})$. The bounding normal cone of the whole expression f is constructed by parsing the tree from the leaves (lower row) to the upper node, the root, applying the bounding cones' arithmetic.

2.3 Expression Trees

We recall the notion of a *binary tree* as a structure that uniquely corresponds to some, not necessarily algebraic, expression. The leaf *nodes* of the tree are constants and unknowns, expressed as polynomial parametric forms and the interior nodes are unary/binary operators, including, in this case, transcendental functions. In the case of a binary operator, its two sub-nodes are the two operands whereas an unary operator is descended by only one sub-node, see Fig. 2.

In the context of solving transcendental system (1), we have n functions (represented as expression trees) $f_i(\mathbf{x})$ and our aim is to construct a bounding normal cone of every $f_i(\mathbf{x})$ in order to decide whether there is a single zero inside some sub-domain D. For every particular tree, $f_i(\mathbf{x})$, we start to construct bounding normal cones bottom-up from every leaf and, using the bounding cones' arithmetic described in Section 3, the tree is parsed all the way up to the root of the tree, resulting with the bounding cone of the whole expression of $f_i(\mathbf{x})$.

3 Bounding Cones' Construction and Arithmetic

In this section, we explain how the bounding normal cone of an expression – an interior node of an expression tree – is constructed, and introduce the bounding cones' arithmetic which is used when two leaves are merged together at some binary (or even unary) node.

3.1 Truncated Bounding Cones and Their Bounding Polytopes

Definition 5. *Consider bounding normal cone, $\mathcal{C}_g^N(\boldsymbol{v}, \alpha)$, of g, over some sub-domain $D \subset \mathbb{R}^n$. Similarly, consider two positive numbers ∇min and ∇max, such that for all $\mathbf{x} \in D$,*

$$\nabla min \leq \|\nabla g(\mathbf{x})\| \leq \nabla max, \tag{6}$$

holds and

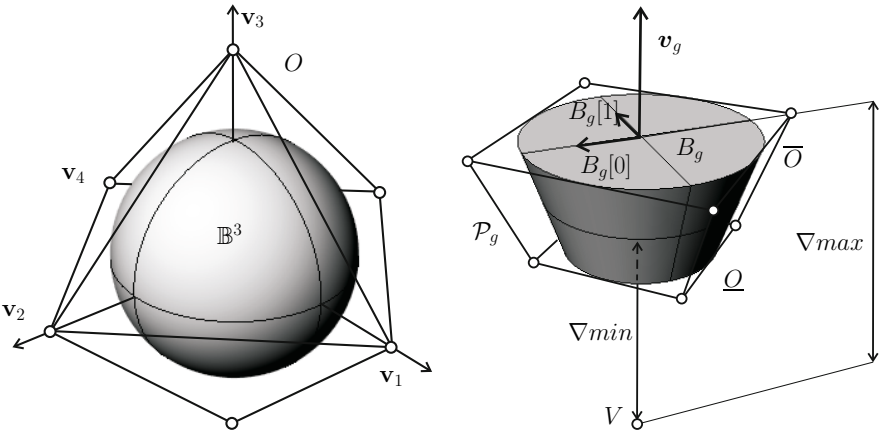
$$\langle \nabla g, \boldsymbol{v} \rangle > \alpha \|\nabla g\|. \tag{7}$$

We further denote by $\overline{\mathcal{C}_g^N} = (\mathcal{C}_g^N, \nabla min, \nabla max)$ the truncated bounding normal cone *of function g.*

Observe that the bounding normal cone, as defined in Def. 4, always contains the origin of the coordinate system (the apex of the cone), see Fig. 3(b). As will be seen from the definition of arithmetic operations on bounding cones, a tighter bound which does not contain the origin, is needed. The truncated cone defined in Def. 5 bounds both the direction and the magnitude of the gradients of g.

The *upper* and *lower caps* of $\overline{\mathcal{C}_g^N}$ (see Fig. 3) are $n-1$ dimensional balls, since their boundaries are $n-2$ dimensional spheres, the result of intersections of the hyper-cone with two hyper-planes perpendicular to its axis.

In order to perform operations with truncated normal cones, we now introduce polygonal bounding regions to these cones, which are referred to as *bounding*



(a) (b)

Fig. 3. (a) $n = 4$, a cap of a truncated cone in $\mathbb{R}^4$, ball $\mathbb{B}^3$, with its wire-frame bounding orthoplex (an octahedron for $\mathbb{B}^3$) consisting of $2(n-1) = 6$ axis-aligned vertices. (b) $n = 3$, truncated normal cone in $\mathbb{R}^3$ with axis $\boldsymbol{v}_g$ and apex V at the origin. The orthogonal complement of $\boldsymbol{v}_g$, B_g, and its orthonormal basis $\{B_g[0], B_g[1]\}$ is computed to construct a pair of bounding orthopleces $\underline{O}$, $\overline{O}$ and subsequently the bounding polytope $\mathcal{P}_g$.

polytopes. Such a polytope follows the shape of the truncated cone, conservatively bounds it, and is easy to construct once a polygonal bound on both caps of the cone are given. Direct operations on (exact) truncated cones would be very difficult to handle, whereas these polytopes discretize the problem to treating only a finite number of points (the vertices of the polytope).

Definition 6. *Consider an* $(n-1)$*-dimensional ball,* $\mathbb{B}^{n-1}$*, of radius* r*. An* orthoplex[1] O *of ball* $\mathbb{B}^{n-1}$ *is the set*

$$O = \{\mathbf{x} \in \mathbb{R}^{n-1}, \|\mathbf{x}\|_1 \leq r\sqrt{n-1}\}, \tag{8}$$

where $\|.\|_1$ *is the* L^1 *norm.*

Lemma 1. *The orthoplex* O *from Def. 6 bounds* $\mathbb{B}^{n-1}$*.*

Proof. By definition, all the points of $\mathbb{B}^{n-1}$ satisfy $\|\mathbf{x}\|_2 \leq r$. Due to the equivalence of norms in a finite dimensional space, $\|\mathbf{x}\|_1 \leq \lambda\|\mathbf{x}\|_2$, for some $\lambda \in \mathbb{R}^+$. Hölder inequality

$$\sum_{i=1}^{n-1} |x_i y_i| \leq (\sum_{i=1}^{n-1} x_i^2)^{\frac{1}{2}} \cdot (\sum_{i=1}^{n-1} y_i^2)^{\frac{1}{2}}, \tag{9}$$

for $y_i = 1$, $i = 1, \ldots, n-1$ gives $\lambda = \sqrt{n-1}$, which yields in $\|\mathbf{x}\|_1 \leq r\sqrt{n-1}$. □

[1] See, e.g., `wikipedia.org/wiki/Cross-polytope`

Obviously, the bounding orthoplex O was defined in a way that it bounds $\mathbb{B}^{n-1}$. Note the advantage that O possesses only $2(n-1)$ vertices, compared to another natural bounding region – the $(n-1)$-dimensional cube, which consists of 2^{n-1} vertices. Since the operations on truncated cones are reduced to the vertices of bounding polytopes, the *linear growth* with respect to the dimension is definitely beneficial.

The construction of an orthoplex is straightforward. Consider ball $\mathbb{B}^{n-1}$ with its center at the origin of the Cartesian system. Then, all the vertices $\mathbf{v}_1$, $\mathbf{v}_2$, $\ldots$, $\mathbf{v}_{2n-2}$ of the bounding orthoplex are located on the $n-1$ axes, at a distance of $\pm r\sqrt{n-1}$ from the origin, see Fig. 3(a), so they can be expressed as all permutations over $(\pm r\sqrt{n-1}, 0, \ldots, 0)$.

Then, ball $\mathbb{B}^{n-1}$, along with all the vertices $\mathbf{v}_i$ of its bounding orthoplex, is transformed (rotated and translated) from the origin-related position to the proper location such that it caps the truncated cone. In order to achieve this transformation, the destination position of the system is needed. Since the ball's basis is the orthogonal complement of the cone's axis, see Fig. 3, the construction of the basis is achieved by a Gramm-Schmidt process.

Definition 7. *Let $\overline{\mathcal{C}_g^N}$ be the truncated bounding normal cone of g and let $\underline{O}$, $\overline{O}$ be the bounding orthopleces of the lower and upper caps, respectively. The convex hull of $\{\underline{O}, \overline{O}\}$ is referred to as the* bounding polytope *of $\overline{\mathcal{C}_g^N}$, denoted by $\mathcal{P}_g$, see Fig. 3.*

Apparently, the closed polyhedron $\mathcal{P}_g$ that bounds $\overline{\mathcal{C}_g^N}$ consists of (at most) $4(n-1)$ vertices, as the convex hull of $\underline{O}$ and $\overline{O}$, each of which has $2(n-1)$ vertices. Therefore, the arithmetic operations on the bounding truncated cones can be efficiently accomplished on their bounding polytopes.

3.2 Bounding Cone's Arithmetic

We introduce the arithmetic rules for the computation of the resulting bounding cone at a node of an expression tree. In this work, we consider the following binary operations: $+$, $-$, $\star$, $\cdot$ and transcendental functions: sin, cos, exp, and log, that can act at any node. The idea is general and can be applied to arbitrary trigonometric function. However, current implementation handles only the above mentioned functions.

Let f and g be two expressions, two neighboring leaves that are being merged at some node of an expression tree, and let $\overline{\mathcal{C}_f^N}(\boldsymbol{v}_f, \alpha_f)$ and $\overline{\mathcal{C}_g^N}(\boldsymbol{v}_g, \alpha_g)$ be their truncated bounding normal cones. Sections 3.2 to 3.2 explain the action taken for the different operators, as part of the execution of the bounding cones' arithmetic.

Addition. Let $h = f + g$ and let $\overline{\mathcal{C}_h^N}$ be the sought truncated normal cone of h. As we already mentioned, the exact construction of $\overline{\mathcal{C}_h^N}$ from $\overline{\mathcal{C}_f^N}$ and $\overline{\mathcal{C}_g^N}$ would be complicated. Instead, we use their bounding polytopes $\mathcal{P}_f$ and $\mathcal{P}_g$, as the following holds

$$\overline{\mathcal{C}_h^N} \subseteq \overline{\mathcal{C}_f^N} \oplus \overline{\mathcal{C}_g^N} \subseteq \mathcal{P}_f \oplus \mathcal{P}_g \subseteq \mathcal{C}^*, \tag{10}$$

where the $\oplus$ denotes the Minkowski sum, and $\mathcal{C}^*$ is a cone that contains the Minkowski sum of both polytopes. This construction is reduced to only adding all possible pairs of vertices of both polytopes. Algorithm 1 summarizes this process. An explanation of some of its steps follows in more detail:

- Since each of the bounding polytopes consists of at most $4(n-1)$ vertices, their Minkowski sum is obtained by processing all possible pairs, $16(n-1)^2$ in all. See line 1.1.
- $\nabla minmax$ is a vector of size 2 holding $(\nabla min, \nabla max)$.
- The radii of the orthopleces is set by $\sqrt{n-1}\tan(\alpha)$ and $\overline{\mathcal{C}^N}.\nabla minmax[i]$ in lines 1.7 and 1.9 and lines 1.10 and 1.10, respectively.
- In lines 1.2 and 1.3, the orthogonal complements of both axis vectors are constructed. These orthonormal bases are used to build vertices $\mathbf{v}_f$, $\mathbf{v}_g$ of the bounding polytopes, in lines 1.10 and 1.11.
- Bounding cone $\mathcal{C}^*$ that contains the Minkowski sum of both polytopes is computed, possibly using the method of [1], and returned in line 1.13.

Subtraction. Since

$$\nabla(f-g) = \nabla f + \nabla(-g) \tag{11}$$

and the bounding cone of $-g$ is achieved simply by flipping $\mathcal{C}_g$, the problem is easily reduced to addition.

Scaling. Scaling (scalar multiplication) by a non-zero coefficient $\lambda \in \mathbb{R}$ is achieved by scaling all polytope's vertices.

Algorithm 1. AddNormalCone($\overline{\mathcal{C}_f^N}, \overline{\mathcal{C}_g^N}, n$)

input : $\overline{\mathcal{C}_f^N}(\boldsymbol{v}_f, \alpha_f)$, $\overline{\mathcal{C}_g^N}(\boldsymbol{v}_g, \alpha_g)$, truncated normal cones of f and g; n, dimension;
output : $\mathcal{C}^*$, bounding normal cone of $f+g$, see Eq. (10);

1.1 PointsList $\leftarrow$ List to hold $16(n-1)^2$ elements;
1.2 OrthoSystem$B_f \leftarrow$ `HyperplaneOrthoSystem`($\boldsymbol{v}_f$, n);
1.3 OrthoSystem$B_g \leftarrow$ `HyperplaneOrthoSystem`($\boldsymbol{v}_g$, n);
1.4 **for** $i \leftarrow 0$ **to** 1 **do**
1.5 **for** $j \leftarrow 0$ **to** 1 **do**
1.6 **for** k $\leftarrow 1$ **to** $n-1$ **do**
1.7 $\boldsymbol{u}_f \leftarrow \sqrt{n-1}\tan(\alpha_f) \cdot$ OrthoSystemB_f[k];
1.8 **for** m $\leftarrow 1$ **to** $n-1$ **do**
1.9 $\boldsymbol{u}_g \leftarrow \sqrt{n-1}\tan(\alpha_g) \cdot$ OrthoSystemB_g[m];
1.10 $\mathbf{v}_f \leftarrow \overline{\mathcal{C}_f^N}.\nabla minmax[i] \cdot (\boldsymbol{v}_f \pm \boldsymbol{u}_f)$;
1.11 $\mathbf{v}_g \leftarrow \overline{\mathcal{C}_g^N}.\nabla minmax[j] \cdot (\boldsymbol{v}_g \pm \boldsymbol{u}_g)$;
1.12 `AppendToList`(PointsList, $\mathbf{v}_f + \mathbf{v}_g$);
1.13 $\mathcal{C}^* \leftarrow$ `BoundingConeOfVectors`(PointsList);
1.14 **return** $\mathcal{C}^*$;

Multiplication. Since the gradient of a product is

$$\nabla(f \star g) = g \cdot \nabla f + f \cdot \nabla g, \tag{12}$$

the bounding cone is obtained by applying the above discussed operations, where $g \cdot \nabla f$ is accomplished by computing the minimum and maximum of g, in the sub-domain D. Note that a constant sign of both f and g over D is required. If not, the bounding cone would span a whole $\mathbb{R}^n$ and the solver subdivides in that case.

Transcendental Functions. exp: Since the exponential function attains only positive values and

$$\nabla(e^f) = e^f \cdot \nabla f, \tag{13}$$

this case is similar to scaling by $\min_{\mathbf{x}\in D} e^{f(\mathbf{x})}$ and $\max_{\mathbf{x}\in D} e^{f(\mathbf{x})}$.

log: We obtain

$$\nabla(\log f) = \frac{1}{f} \cdot \nabla f, \tag{14}$$

the problem is again reduced to scaling and f is required to have a strictly monotone sign on D.

sin & **cos:** Analogously,

$$\nabla(\sin f) = \cos f \cdot \nabla f, \tag{15}$$

and scaling by constants $\max_{\mathbf{x}\in D} \cos f(\mathbf{x})$ and $\min_{\mathbf{x}\in D} \cos f(\mathbf{x})$ gives the result under the assumption that $\cos f$ does not change sign on D.

In the case of a polynomial leaf, $g(\mathbf{x})$, the bounds of $\min_{\mathbf{x}\in D} g(\mathbf{x})$ and $\max_{\mathbf{x}\in D} g(\mathbf{x})$ are directly obtained from its control points exploiting the convex hull property. Min/max bounds of interior nodes are computed following the same rules of interval arithmetic for simple arithmetic and Equations (13) to (15), in case of transendental functions. In the latter case, f is required to be monotone over D and $f(g(\mathbf{x}))$ is evaluated at $\min g(\mathbf{x})$ and $\max g(\mathbf{x})$.

3.3 No Root Exclusion Test

In order to eliminate domains which contain no roots, the sign variation of every function $f_i(\mathbf{x})$, $i = 1, \ldots, n$, is tested over a given domain D. If $f_i(\mathbf{x}) > 0$, (or $f_i(\mathbf{x}) < 0$) for some i, for all $\mathbf{x} \in D$, D is discarded. For any polynomial leaf, this test is easily achieved by checking the signs of corresponding Bernstein coefficients. If all are positive (negative), the convex hull property guarantees that no roots exist. For every f_i, all its polynomial leaves are tested and the *minimum* and *maximum* of the Bernstein coefficients define a bounding interval of values that the (polynomial) leaves may attain. Since f_i is treated as an expression tree, an interval arithmetic is applied at every interior node of the tree, giving a new bound on the merged expression. Parsing the tree from its leaves to the root, only to provide a bound on f_i itself.

3.4 Numerical Improvements Stage

Once a domain, which contains at most one root is isolated, a numerical improvement stage is commenced, and techniques, such as the ones presented in [6] or [7] could be clearly employed. Herein, we use simple Newton Raphson iterations, starting with an initial solution guess of x_0 at the mid point of the obtained domain. The expression tree structure of the equations also allows for an efficient computation of $\frac{\partial f_i}{\partial x_j}$, necessary for the Newton Raphson iterations, by using the derivative rules (such as the addition and multiplication rules, in Section 3.2).

If the iterations do not converge or go outside of the domain, we declare that there is no root in the domain. One can, in that case, continue the subdivision steps in the hope that a closer initial guess will be more successful. This, until some prescribed subdivision tolerance is met. The last case typically hints on non simple roots, which are prevented by passing the single solution test, or in some cases, on roots on the boundary of the domain. However, if the Newton-Rapshon fails, one can further subdivide to get a closer initial guess or, in the no-root case, to eliminate that domain by the exclusion test.

3.5 Algorithm – Summary

Every function f_i of the system (1) is represented by an expression tree. The solver parses the tree, starts with the simple bounding cones of polynomial functions at the leaves of the expression tree, and ends up at the root of the tree with a bounding normal cone of f_i. If in some node, the merging process fails to produce valid bounding cone (i.e. the cone spans the whole $\mathbb{R}^n$), the solver simply subdivides. Recall from [4] that the advantage of subdividing using expression trees stems from the fact that only the leaves in the direction of the subdivision are required to be subdivided.

Once the bounding cones of all the f_i functions are built, the complementary bounding cones are constructed (recall Section 2.2) and the single solution test of [8] is executed. If this test succeeds, guaranteeing at most one isolated root within the domain, a multivariate Newton-Rapshon method is applied. Otherwise, the solver subdivides further, up to the permissible tolerance.

3.6 Analysis of the Bounding Cone's Tightness

In this section, we discuss the quality of the bound which was introduced in Section 3.1. The polyhedral bound, the bounding polytope, of the truncated normal cone is based on bounds of the cap(s), the $(n-1)$ dimensional ball(s) $\mathbb{B}^{n-1}$. For convenience, we shift the index to n, in this section.

Since the tightness of the polytope with respect to the truncated cone follows the quality of the bound of the orthoplex with respect to $\mathbb{B}^n$, we discuss the quality of this bound. Several criteria of the bound can be considered:

(1) Number of vertices of the bounding polyhedron,
(2) distance of the farthest vertex from the center of the ball,
(3) the ratio between the volumes of the bound and the original ball.

Table 1. Cube vs. Orthoplex as a bound on unit ball $\mathbb{B}^n$ with respect to the dimension n. The numbers of vertices, volumes and relative volumes with respect to the ball $\mathbb{B}^n$ are shown.

n	$NumOfVert$		$Volume$		$Volumetric\ Ratio$	
	$Orth.$	$Cube$	$Orth.$	$Cube$	$\frac{Sphere}{Orth.}$	$\frac{Sphere}{Cube}$
2	4	4	4	4	$\frac{\pi}{4}$	$\frac{\pi}{4}$
4	8	16	$\frac{32}{3}$	16	$\frac{3\pi^2}{64}$	$\frac{\pi^2}{32}$
6	12	64	$\frac{96}{5}$	64	$\frac{5\pi^3}{576}$	$\frac{\pi^3}{384}$
8	16	256	$\frac{8192}{315}$	256	$\frac{105\pi^4}{65536}$	$\frac{\pi^4}{6144}$
10	20	1024	$\frac{16000}{567}$	1024	$\frac{189\pi^5}{640000}$	$\frac{\pi^5}{122880}$

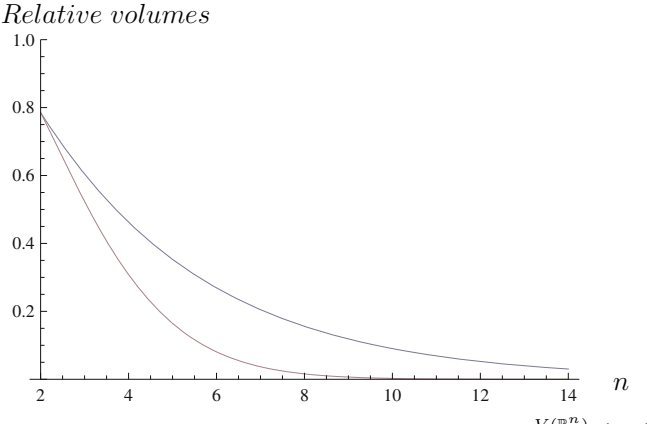


Fig. 4. Comparison of the bounding tightness: Relative volumes $\frac{V(\mathbb{B}^n)}{V(C^n)}$ (red) and $\frac{V(\mathbb{B}^n)}{V(O^n)}$ (blue), as a function of dimension n, are depicted

As a natural alternative to an orthoplex, an n-dimensional cube comes up in mind. A comparison of these two bounds follows.

(1) As already mentioned in Section 3.1, an orthoplex consists of only $2n$ vertices in contrast to 2^n vertices of the cube.
(2) Any vertex of both bounding objects is at the distance of $r\sqrt{n}$ from the center of $\mathbb{B}^n$.
(3) The volume of a cube is $V(C^n) = (2r)^n$ and the volume of the inscribed ball[2], $\mathbb{B}^n$, is given by

$$V(\mathbb{B}^n) = \frac{(r\sqrt{\pi})^n}{\Gamma(\frac{n}{2}+1)}, \tag{16}$$

[2] See, e.g., `mathworld.wolfram.com/Ball.html`

which can be rewritten using Stirling's approximation as

$$\frac{(r\sqrt{\pi})^n}{\Gamma\left(\frac{n}{2}+1\right)} \approx \left(\frac{2\pi e}{n}\right)^{\frac{n}{2}} \frac{r^n}{\sqrt{n\pi}}. \tag{17}$$

The direct computation of the volume[3] of the orthoplex gives

$$V(O^n) = \int_O d\mathbf{x} = (r\sqrt{n})^n \int_{\|\mathbf{x}\|_1 \le 1} d\mathbf{x} = \frac{(2r\sqrt{n})^n}{n!}. \tag{18}$$

Table 1 displays the comparison of the criteria for various n.

Observe also the asymptotic behavior of $\frac{V(\mathbb{B}^n)}{V(O^n)}$ and $\frac{V(\mathbb{B}^n)}{V(C^n)}$, in Fig. 4. Using Stirling's approximation again, we get

$$\frac{V(\mathbb{B}^n)}{V(O^n)} = \frac{\frac{(r\sqrt{\pi})^n}{\Gamma(\frac{n}{2}+1)}}{\frac{(2r\sqrt{n})^n}{n!}} \approx \frac{(\frac{2\pi e}{n})^{\frac{n}{2}} \frac{r^n}{\sqrt{n\pi}}}{(\frac{2e}{\sqrt{n}})^n \frac{r^n}{\sqrt{2n\pi}}} = \sqrt{2}(\frac{\pi}{2e})^{\frac{n}{2}}, \tag{19}$$

whereas

$$\frac{V(\mathbb{B}^n)}{V(C^n)} \approx (\frac{e\pi}{2n})^{\frac{n}{2}} \frac{1}{\sqrt{n\pi}}, \tag{20}$$

which converges to zero much faster.

As observed from the above analysis, the bounding orthoplex is not only more efficient to process but it also offers a satisfactory bound on $\mathbb{B}^n$, that is better than the bounding cube. Hence, the use of the orthoplex as a bounding volume results in a tight bound of the truncated cone.

4 Examples

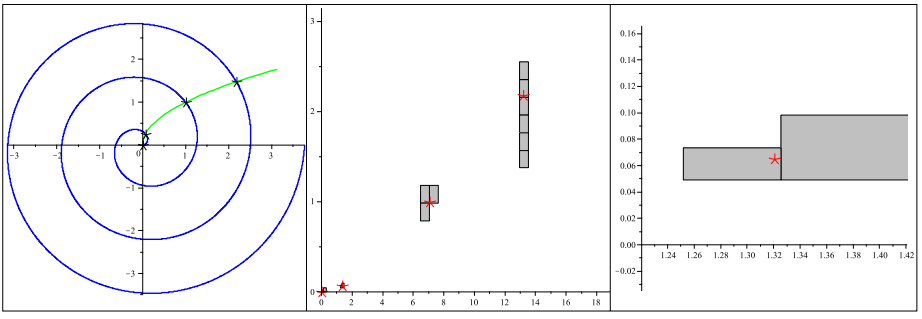
Demonstrating the proposed solver on intersection problems, in this section, we present several examples of solving non-polynomial well-constrained systems. In the first example, an Archimedean spiral is intersected with a parabola, (see Fig. 5), resulting in four single roots.

In the next example, circle $C_1(t) = [10\cos(t), 10\sin(t)]$, $t \in [0, 2\pi]$ is intersected with cycloid $C_2(s) = [10\cos(s) + 2\cos(10s), 10\sin(s) + 2\sin(10s)]$, $s \in [0, 2\pi]$ yielding the system

$$\begin{aligned} 10\cos(t) - 10\cos(s) - 2\cos(10s) &= 0, \\ 10\sin(t) - 10\sin(s) - 2\sin(10s) &= 0, \end{aligned} \tag{21}$$

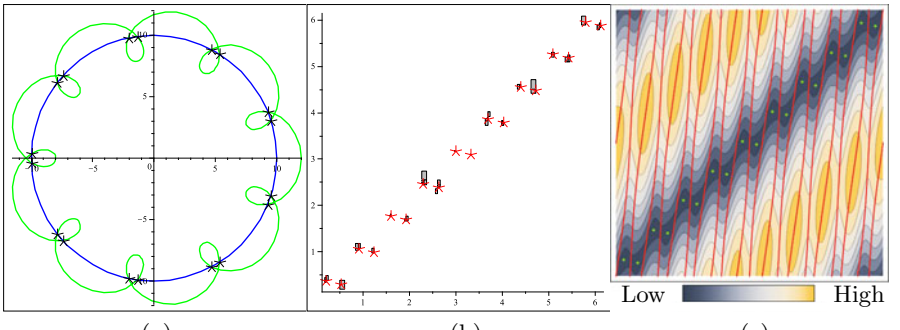
over the Cartesian product of their parametric domains, $[0, 2\pi] \times [0, 2\pi]$, see Fig. 6. A modification of the circle's radius gives the tangent and near-to-tangent configuration, see Fig. 7. Whereas the first case forces the solver to subdivide until the subdivision tolerance is reached, and the centers of the may-be-root domains are returned, in the latter case, the roots are isolated by the single solution test [8].

[3] See, e.g., `wikipedia.org/wiki/Cross-polytope`



(a) (b) (c)

Fig. 5. (a) Archimedean spiral $C_1(t) = [\frac{1}{5}t\cos(t), \frac{1}{5}t\sin(t)]$, $t \in [0, 6\pi]$ (blue) vs. a segment of parabola $C_2(s) = [s, \sqrt{s}]$, $s \in [0, \pi]$, (green) gives 4 intersection points. (b) Corresponding points (red asterisks) in the parametric st-space $[0, 6\pi] \times [0, \pi]$. The grey domains, rectangles with black polylines as boundaries, report when the subdivision was stopped with a guarantee of at most one root within the domain. (c) A zoom-in on the second root. In the case of the root at the origin $[0, 0]$, the subdivision tolerance of $\varepsilon_{sub} = 10^{-3}$ was reached. Once the subdivision is stopped, a Newton-Raphson scheme is applied to numerically reach the root.

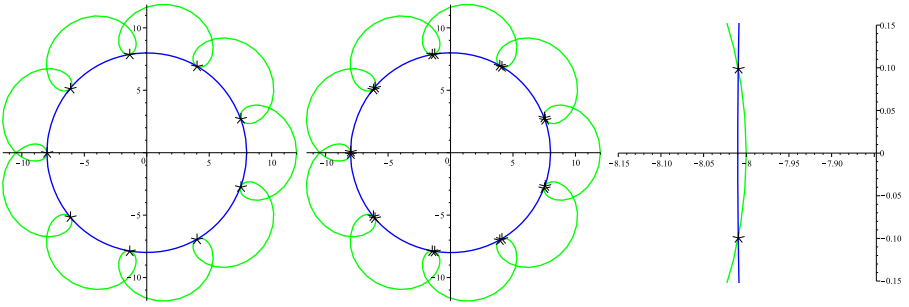


(a) (b) (c)

Fig. 6. (a) A cycloid $C_2(s) = [10\cos(s) + 2\cos(10s), 10\sin(s) + 2\sin(10s)]$, $s \in [0, 2\pi]$ (green) is intersected with a circle (blue), giving 18 intersection points. (b) Corresponding points (red asterisks) in the solution (preimage) st-space $[0, 2\pi] \times [0, 2\pi]$ and sub-domains, that contain at most one root (grey). (c) Red isocurves indicate locations where the Jacobian of System (21) is zero. Color coding depicts the L^2 norm of the system, dark color corresponds to low values. Green dots are the roots.

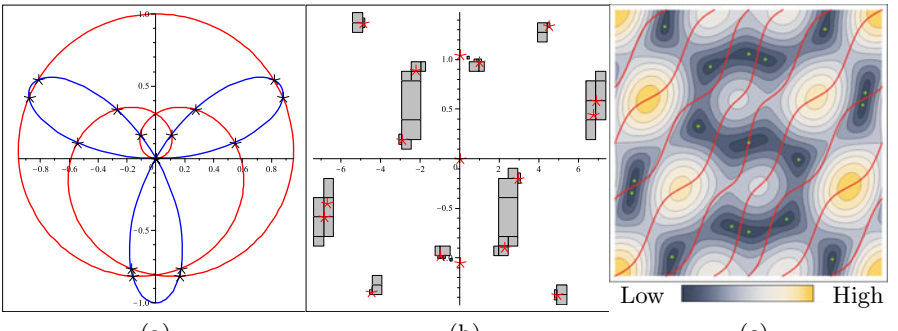
More complex example is shown at Fig. 8. Two cycloidal curves $C_1(t) = [\sin(\frac{t}{5})\cos(t), \sin(\frac{t}{5})\sin(t)]$, $t \in [-\frac{5\pi}{2}, \frac{5\pi}{2}]$ and $C_2(s) = [\sin(3s)\cos(s), \sin(3s)\sin(s)]$, $s \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ are intersected, having fourteen single roots and a triple root at point $[0, 0]$.

An example that corresponds to a 4×4 system is shown at Fig. 9. Vertices $V_1, \ldots, V_4$ of a square (of an unknown size) are constrained to lie in turn on four



(a) (b) (c)

Fig. 7. (a) A cycloid $C_2(s) = [10\cos(s) + 2\cos(10s), 10\sin(s) + 2\sin(10s)]$, $s \in [0, 2\pi]$ (green) and a circle (blue) of radius $r = 8$ possess a tangent contact along 9 points (black asterisks). (b) A semi-tangent configuration for $r = 8.01$ with 18 pairwise grouped intersection points and a zoom on one such a pair (c).



(a) (b) (c)

Fig. 8. (a) Two cycloidal curves are intersected, giving 17 intersection points (black asterisks). The solution point at the origin has multiplicity 3. (b) Corresponding points (red asterisks) in the solution (preimage) st-space $[-\frac{5\pi}{2}, \frac{5\pi}{2}] \times [-\frac{\pi}{2}, \frac{\pi}{2}]$ and sub-domains, that contain the at most one root (grey rectangles). (c) Red isocurves indicate locations where the Jacobian of the system is zero. Color coding depicts the L^2 norm of the system, dark color corresponds to low values. Green dots are the roots.

algebraic curves $\alpha_1 : (x-3)^2 + (y-0.4)^2 = 1$, $\alpha_2 : (x-3)^2 + (x-3)(y-3) + (y-3)^2 = 1$, $\alpha_3 : x^2 - x(y-3) + (y-3)^2 = 1$ and $\alpha_4 : x^2 + xy + y^2 = 1$. Consider V_i, $i = 1, \ldots, 4$ as $V_i = [c_1 + k\cos(\phi + \frac{(i-1)\pi}{2}), c_2 + k\sin(\phi + \frac{(i-1)\pi}{2})]$, where $C = [c_1, c_2]$ is the center of the square, k is the scaling factor, and ϕ is the angle between $V_1 - C$ and the positive x-axis. Substituting the coordinates of V_i into α_i gives the system. Solving for $[c_1, c_2, k, \phi]$ over domain $[0, 3] \times [0, 3] \times [0, 2.5] \times [-\pi/2, \pi/2]$ gives 6 solutions, see Fig. 9. Observe that the configuration of four blue curves offers many almost-solution positions of the square, which makes the system time demanding.

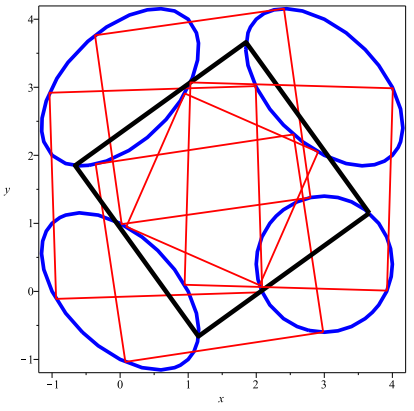
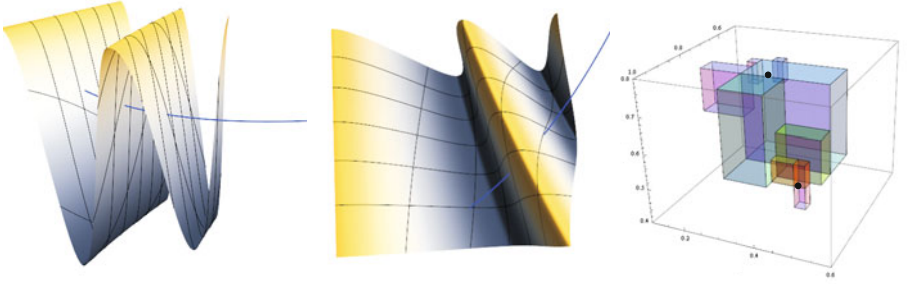


Fig. 9. A solution of a time demanding 4×4 system: A square of an unknown edge's length is sought such that each its vertex lies on one of four particular implicit curves (blue). Six solutions were found (red). One solution (square) is highlighted (bold black).



(a) (b) (c)

Fig. 10. (a) A curve-surface intersection with 2 solutions. (b) The same arrangement seen from the top. (c) A zoom on a root-containing segment of the solution, $uv \times t$, space. Colored voxels report when the solver stops subdivision, while the two thick black dots are the roots of the system.

As a last example, space curve $C(t) = [e^t, e^{-2t}, \frac{t}{2}]$, $t \in [0, 1]$ is intersected with surface $S(u, v) = [\ln(1 + u), \ln(1 + v), \cos(2\pi(uv + v))$, $[u, v] \in [0, 1] \times [0, 1]$, giving a system of dimension three.

Table 2 gives a timing summary of all given examples. These timings are obtained from a 2.67 Mhz IBM PC running Windows XP. Every example was run 60 times and the timings were averaged.

Table 2. All the parametric spaces were scaled to the unit box, tested with subdivision tolerance $\varepsilon_{sub} = 10^{-3}$. Times are for an 2.67 MHz IBM PC running Windows XP.

Example	*Num Of Roots*	*Time(secs)*
Fig. 5	4	0.0049
Fig. 6	18	0.0487
Fig. 7(a)	9	0.1011
Fig. 7(b)	18	0.0998
Fig. 8	17	0.0276
Fig. 9	6	100.2
Fig. 10	2	0.0187

5 Conclusion and Future Work

In this work, we have presented a solver that robustly solves well-constrained $n \times n$ transcendental systems. Exploiting the expression trees to construct bounding normal cones of n transcendental constraints, the subdivision based solver detects all sub-domains, where at most one root can exist. The root is then numerically improved by a multivariate Newton-Raphson scheme.

The presented solver guarantees to isolate and return *all* single roots of the system within a given domain. Other roots are only isolated. This, in contrast of commercial software such as Maple[4], Mathematica[5], or Matlab[6], which – to our best knowledge – can locally isolate and provide only one root or a few of them.

As a future work, an improved algorithm is intended, which better handles multiple roots. As of now, the solver only subdivides to such points, up to the permissible tolerance. In addition, the numerical stage of the algorithm deserves some further research. For once, under the current scheme, there is no guarantee that a multivariate Newton-Raphson scheme will converge to the root. Also, if the system is underconstrained and (at least) one-dimensional solution space is expected like in [2], a special treatment of the system is more favorable.

Despite the use of expression trees, which are highly efficient during the subdivision stage, the growth in the number of subdivisions is required to be minimal with respect to the dimension n. Hence, the handling of higher-dimensional systems is within the scope of our interest.

Acknowledgments

This research was partly supported by the Israel Science Foundation (grant No. 346/07), and in part by the New York metropolitan research fund, Technion.

[4] http://www.maplesoft.com/
[5] http://www.wolfram.com/products/mathematica
[6] http://www.mathworks.com/

References

1. Barequet, G., Elber, G.: Optimal bounding cones of vectors in three and higher dimensions. Information Processing Letters 93, 83–89 (2005)
2. Bartoň, M., Hanniel, I., Elber, G.: Topologically guaranteed univariate solutions of underconstrained polynomial systems via no–loop and single–component tests (in preparation)
3. Cox, D.A., Little, J.B., O'Shea, D.: Using algebraic geometry. Springer, Heidelberg (2005)
4. Elber, G., Grandine, T.: Efficient solution to systems of multivariate polynomials using expression trees. In: Tenth SIAM Conference on Geometric Design and Computing (2007)
5. Gaukel, J.: Efficient solving of polynomial and nonpolynomial systems using subdivision (in German), PhD thesis, TU Darmstadt (2003)
6. Graspa, T.N., Vrahatis, M.N.: Dimension reducing methods for systems of nonlinear equations and unconstrained optimization: A review. Recent Adv. Mech. Related Fields, 215–225 (2003)
7. Grosan, C., Abraham, A.: A new approach for solving nonlinear equations systems. IEEE Transactions on systems, man and cybernetics: Systems and humans 38(3) (2008)
8. Hanniel, I., Elber, G.: Subdivision termination criteria in subdivision multivariate solvers using dual hyperplanes representations. Computer Aided Design (39), 369–378 (2007)
9. McNamee, J.M.: Bibliographies on roots of polynomials. J. Comp. Appl. Math (47), 391–394, 78(1-1), (110), 305–306, (142) (433–434) (1993–2002)
10. Mourrain, B., Pavone, J.-P.: Subdivision methods for solving polynomial equations. J. of Symbolic Computation 44(3), 292–306 (2009)
11. Neumaier, A.: Introduction to Numerical Analysis. Cambridge Univ. Press, Cambridge (2001)
12. Reuter, M., Mikkelsen, T.S., Sherbrooke, E.C., Maekawa, T., Patrikalakis, N.M.: Solving nonlinear polynomial systems in the barycentric Bernstein basis. Visual Computer (24), 187–200 (2008)
13. Rheinboldt, W.C.: Methods for solving systems of nonlinear equations. In: Regional Conference Series in Applied Mathematics, 2nd edn. Society for Industrial and Applied Mathematics (SIAM), Philadelphia (1998)
14. Sherbrooke, E.C., Patrikalakis, N.M.: Computation of solution of nonlinear polynomial systems. Computer Aided Geometric Design 5(10), 379–405 (1993)
15. Sommese, A.J., Wampler, C.W.: The numerical solution of systems of polynomials arising in engineering and science. World Scientific, Singapore (2005)
16. Stewart, J.: Multivariate Calculus (2002)
17. Wang, H., Kearney, J., Atkinson, K.: Robust and efficient computation of the closest point on a spline curve. In: Lyche, T., et al. (eds.) Curve and Surface Design, Saint Malo 2002, pp. 397–405. Nashboro Press, Brentwood (2002)
18. Zhou, J., Sherbrooke, E.C., Patrikalakis, N.M.: Computation of stationary points of distance functions. Engineering with Computers 9(4), 231–246 (1993)

Surfaces with Rational Chord Length Parameterization

Bohumír Bastl[1], Bert Jüttler[2], Miroslav Lávička[1], and Zbyněk Šír[1]

[1] University of West Bohemia, Faculty of Applied Sciences,
Department of Mathematics, Plzeň, Czech Republic
{bastl,lavicka,zsir}@kma.zcu.cz
[2] Johannes Kepler University of Linz, Institute of Applied Geometry,
Linz, Austria
bert.juettler@jku.at

Abstract. We consider a rational triangular Bézier surface of degree n and study conditions under which it is rationally parameterized by chord lengths (RCL surface) with respect to the reference circle. The distinguishing property of these surfaces is that the ratios of the three distances of a point to the three vertices of an arbitrary triangle inscribed to the reference circle and the ratios of the distances of the parameter point to the three vertices of the corresponding domain triangle are identical. This RCL property, which extends an observation from [10,13] about rational curves parameterized by chord lengths, was firstly observed in the surface case for patches on spheres in [2]. In the present paper, we analyze the entire family of RCL surfaces, provide their general parameterization and thoroughly investigate their properties.

1 Introduction

Recently, chord length parametrization has become an active research area in Computer Aided Geometric Design. This approach was motivated by the use of chord length parameterization for interpolation and approximation of discrete point data. It can be seen as an alternative to arc-length parameterizations because analogously to arc-length parameter, the chord-length parameter is also uniquely given by the loci of the curve.

The investigation of rational curves with chord length parameterization was initiated by [6] by formulating the observation that rational quadratic circles in standard Bézier form are parameterized by chord length. Earlier, a geometric proof of this fact was given in [11], along with an application to a circle-preserving variant of the four-point subdivision scheme. A thorough analysis followed in [10,13], where two independent constructions for general rational curves of this type were presented. In some sense, rational curves with chord length parameterizations (shortly RCL curves) are a chord-length analogy to the so called Pythagorean-hodograph curves characterized by closed form expressions for their arc-lengths, cf. [7,9].

Curves with RCL property are worth studying mainly because of the following advantages. First they provide a simple inversion formula, which can be

B. Mourrain, S. Schaefer, and G. Xu (Eds.): GMP 2010, LNCS 6130, pp. 19–28, 2010.

e.g. used for computing their implicit equations. Second, it is simple to perform point-curve testing. Finally, these curves do not possess self-intersections. In addition to straight lines and circles in standard form, this class of RCL curves also contain e.g. equilateral hyperbola, Bernoulli's lemniscate and Pascal's Limaçon. Curves with chord-length parameterization were also mentioned among remarkable families of curves admitting a complex rational form in [12].

Motivated by RCL curves, it is natural to extend this approach also to rational surfaces. A promising result was presented in [2] where the equal chord property of quadratic rational Bézier patches describing a segment of a sphere was proved. For this, the well-known construction of spherical quadratic patches by stereographic projection was used, cf. [1,4,5]. This result directly extends the planar result for circles, see [6]. As a byproduct, it was shown in [2] how to characterize this property using tripolar coordinates in space, which extend the observations of [13] concerning the relation between bipolar coordinates (see [3,8] for more details) and curves with chord-length parameterization.

The present paper is devoted to the equal chord property of rational triangular Bézier surfaces of degree n, thus extending the results of [10,13] to the case of surfaces. We present a general construction of rational chord length parameterizations (RCL surfaces) and study their attractive geometric properties. The introduced approach is then demonstrated by several examples of RCL surfaces.

2 Preliminaries

We consider a rational surface of degree n, which is described by its triangular Bernstein–Bézier representation

$$\mathbf{P}(\mathbf{X}) = \frac{\displaystyle\sum_{i,j,k\in\mathbb{Z}^+,\ i+j+k=n} w_{ijk}\,\mathbf{b}_{ijk}\,\frac{n!}{i!j!k!}\lambda^i\mu^j\nu^k}{\displaystyle\sum_{i,j,k\in\mathbb{Z}^+,\ i+j+k=n} w_{ijk}\,\frac{n!}{i!j!k!}\lambda^i\mu^j\nu^k}, \quad \mathbf{X}\in\mathbb{R}^2 \tag{1}$$

with respect to a non-degenerate reference triangle $\triangle(\mathbf{A}_1,\mathbf{A}_2,\mathbf{A}_3)\subset\mathbb{R}^2$ with vertices $(\mathbf{A}_\ell)_{\ell=1,2,3}$. Its argument

$$\mathbf{X} = \lambda\mathbf{A}_1 + \mu\mathbf{A}_2 + \nu\mathbf{A}_3, \quad \lambda+\mu+\nu = 1, \tag{2}$$

is expressed by barycentric coordinates with respect to the reference triangle.

The shape of the surface is determined by the $\binom{n+1}{2}$ control points $\mathbf{b}_{ijk}$ with the associated weights w_{ijk}. In particular, the control net of the patch has the three vertices

$$\mathbf{v}_1 = \mathbf{b}_{n00}, \quad \mathbf{v}_2 = \mathbf{b}_{0n0}, \quad \text{and} \quad \mathbf{v}_3 = \mathbf{b}_{00n} \tag{3}$$

which are the images of the vertices of the reference triangle.

Let

$$R_\ell(\mathbf{X}) = ||\mathbf{X}-\mathbf{A}_\ell||^2 \quad \text{and} \quad r_\ell(\mathbf{X}) = ||\mathbf{P}(\mathbf{X})-\mathbf{v}_\ell||^2 \tag{4}$$

be the squared distances of the point $\mathbf{X}$ and its image $\mathbf{P}(\mathbf{X})$ to the vertices of the domain triangle and to the vertices of the patch, respectively.

Definition 1. *The surface* (1) *is a rational chord length parameterization (RCL) with respect to the reference triangle, if*

$$r_1 : r_2 : r_3 = R_1 : R_2 : R_3, \quad \textit{or, equivalently,}$$
$$\forall (i,j) \in \{(1,2),(2,3),(3,1)\} : \quad r_i R_j = r_j R_i \tag{5}$$

holds for all points $\mathbf{X} \in \mathbb{R}^2$.

The squares of the chord lengths are quadratic polynomial functions of the barycentric coordinates. On the other hand, for any point of a RCL surface, the barycentric coordinates of the argument can be computed from the chord lengths by solving a quadratic equation, see [2] for more details.

We first analyze the relation between the reference triangle and the triangle spanned by the vertices of the control net.

Lemma 1. *If the surface is a rational chord length parameterization, then the triangles* $\triangle(\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3)$ *and* $\triangle(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$ *are similar.*

Proof. We evaluate the three relations (5) at the three vertices $\mathbf{A}_\ell$ of the domain triangle. Six of these 9 equations are trivially satisfied, since one of the r_i and R_i vanish at each vertex. The remaining three equations guarantee the similarity of the triangles. □

In the remainder of the paper, we identify the reference triangle $\triangle(\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3)$ with the vertex triangle $\triangle(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$ and the domain $\mathbb{R}^2$ containing it with the plane spanned by the vertex triangle. Consequently, *the domain of the surface is the plane spanned by the vertex triangle.*

For any point $\mathbf{Y} \in \mathbb{R}^3$, we denote with

$$\varrho_\ell(\mathbf{Y}) = ||\mathbf{Y} - \mathbf{v}_\ell||^2, \quad \ell = 1,2,3, \tag{6}$$

the squared distances to the vertices of the patch.

Lemma 2. *The set of all points* $\mathbf{Y}$ *satisfying*

$$\forall (i,j) \in \{(1,2),(2,3),(3,1)\} : \quad \varrho_i(\mathbf{Y}) R_j(\mathbf{X}) = \varrho_j(\mathbf{Y}) R_i(\mathbf{X}) \tag{7}$$

is a circle which passes through $\mathbf{X}$ *and is perpendicular to any sphere containing the vertices of the patch. If* $\mathbf{X}$ *lies on the circumcircle of the vertex triangle, then the circle* $\mathbf{Y}$ *shrinks to the single point* $\mathbf{X}$.

Proof. Recall that for any two points $\mathbf{M}$, $\mathbf{N}$ in the plane, the set of all points $\mathbf{Z}$ satisfying

$$||\mathbf{Z} - \mathbf{M}||^2 = c\,||\mathbf{Z} - \mathbf{N}||^2 \tag{8}$$

for some positive constant c is a circle (Apollonius' definition) which intersects any circle through $\mathbf{M}$ and $\mathbf{N}$ orthogonally. Consequently, for a given point $\mathbf{X}$, the set of all points $\mathbf{Y}$ satisfying

$$\varrho_i(\mathbf{Y})R_j(\mathbf{X}) = \varrho_j(\mathbf{Y})R_i(\mathbf{X}) \tag{9}$$

is a sphere whose center lies on the line through $\mathbf{v}_i$ and $\mathbf{v}_j$. Moreover, any sphere containing these two vertices intersects this sphere orthogonally. Indeed, if we consider the intersection with the common symmetry plane of both spheres, which is spanned by the sphere's center and the line through $\mathbf{v}_i$ and $\mathbf{v}_j$, then we obtain the two families of circles which appear in Apollonius' definition of a circle.

Clearly, the three spheres (9) obtained for $(i,j) \in \{(1,2),(2,3),(3,1)\}$ intersect in one circle, since the equations defining them are not independent. Moreover, since these spheres intersect any sphere through the three points $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$, orthogonally, so does the intersection curve, cf. Fig. 1.

If $\mathbf{X}$ belongs to the circumcircle of the vertex triangle, then any two of the three spheres (9) touch each other at this point and the circle degenerates into a single point. □

Corollary 1. *If* $\mathbf{P}$ *is a rational chord length parameterization, then its restriction to the circumcircle of the reference triangle is the identity. Moreover, the surface is a rational chord length parameterization with respect to any reference triangle which possesses the same circumcircle.*

Proof. The surface $\mathbf{P}$ is a RCL surface if and only if any point $\mathbf{P}(\mathbf{X})$ lies on the circle described in Lemma 2. On the one hand, if $\mathbf{X}$ is on the circumcircle of

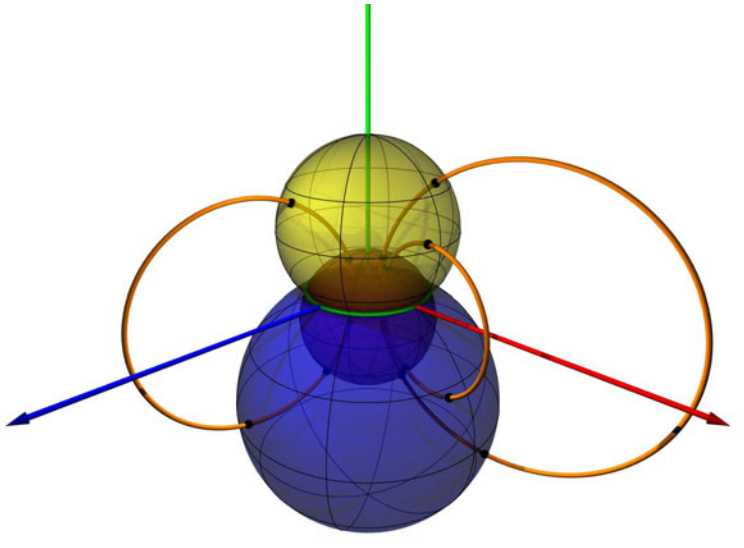


Fig. 1. Examples of circles perpendicular to any sphere containing the reference circle

the reference triangle, then this circle shrinks to the point $\mathbf{X}$ itself. On the other hand, the family of circles described in Lemma 2 does not depend on choice of the reference triangle. □

Consequently, the RCL surface always contains the circumcircle of its reference triangle, and its definition depends solely on this circle. The latter fact can also be concluded from Corollary 4 of [2]. This observation motivates the following extended definition.

Definition 2. *A surface* $\mathbf{P}$ *is said to be a rational chord length parameterization with respect to a circle, if it is a rational chord length parameterization with respect to a reference triangle possessing this circle as its circumcircle.*

3 Construction of RCL Surfaces

In order to simplify the formulas, we choose the reference circle as the unit circle $\mathcal{C}$ in the xy-plane. Consequently, the arguments of the rational surface $\mathbf{P}$ are all points of the form $\mathbf{X} = (u, v, 0)^\top$.

Theorem 1. *A surface* $\mathbf{P}$ *is a rational chord length parameterization with respect to the reference circle* $\mathcal{C}$ *if and only if there exists a rational function* $q : (u, v) \mapsto q(u, v)$ *such that*

$$\mathbf{P}(u,v) = \left(\frac{(1+q^2)u}{1+q^2(u^2+v^2)}, \frac{(1+q^2)v}{1+q^2(u^2+v^2)}, \frac{q(1-u^2-v^2)}{1+q^2(u^2+v^2)} \right)^\top. \tag{10}$$

Proof. Without loss of generality, we consider the reference triangle with the vertices $\mathbf{A}_1 = (1,0,0)^\top$, $\mathbf{A}_2 = (0,1,0)^\top$, $\mathbf{A}_3 = (0,-1,0)^\top$ on the reference circle $\mathcal{C}$. The surface $\mathbf{P}$ is RCL if and only if there exists a rational function λ such that the squared distances R_ℓ and r_ℓ are related by

$$\forall (u,v): \ \lambda(u,v) R_\ell(u,v) = r_\ell(u,v), \quad \ell = 1,2,3. \tag{11}$$

A short computation confirms that the intersection points of the three spheres with centers $\mathbf{A}_i$ and radii $\sqrt{r_i}$ has the coordinates

$$\begin{aligned} \mathbf{P}_\pm(u,v) = \frac{1}{4}\big(& -2r_1 + r_2 + r_3, -r_2 + r_3, \\ & \pm\sqrt{2}\cdot\sqrt{4(r_2+r_3) - [(r_1-r_2)^2 + (r_1-r_3)^2] - 8}\ \big)^\top. \end{aligned} \tag{12}$$

Using (11) and the identities $R_1 = (u-1)^2 + v^2$, $R_2 = u^2 + (v-1)^2$, $R_3 = u^2 + (v+1)^2$, which follow from the definition (4), this can be rewritten as

$$\mathbf{P}_\pm(u,v) = \big(\lambda u, \lambda v, \pm\sqrt{(1-\lambda)(\lambda u^2 + \lambda v^2 - 1)}\ \big)^\top. \tag{13}$$

This surface has a rational parameterization with respect to u, v if and only if the argument of the square root is a perfect square. This is equivalent to the condition on the existence of a rational function $q(u, v)$ such that

$$1 - \lambda = q^2(\lambda u^2 + \lambda v^2 - 1). \tag{14}$$

Solving (14) for λ we arrive at

$$\lambda(u,v) = \frac{1+q(u,v)^2}{1+q(u,v)^2(u^2+v^2)}. \tag{15}$$

Finally, we substitute λ into (13). The two possible choices of the sign of the third coordinate can be obtained by specifying the sign of the rational function q. □

We provide a geometric meaning for this result.

Proposition 1. *Consider the angle $\alpha(u,v) \in [-\pi,\pi]$ which satisfies*

$$\tan\frac{\alpha(u,v)}{2} = q(u,v). \tag{16}$$

If $u^2+v^2 \neq 1$, then α is the angle between the xy-plane and the sphere which passes through the point $\mathbf{P}(u,v)$ and the reference circle $\mathcal{C}$. If $u^2+v^2=1$, then $\mathbf{P}(u,v)$ lies on the reference circle $\mathcal{C}$ and α is the angle between the xy-plane and the tangent plane of the surface $\mathbf{P}$ at this point.

Proof. We consider a surface (10). In the first case, the unique sphere which passes through the reference circle and through the point $\mathbf{P}(u,v)$ has the center $\mathbf{C} = (0,0,(q^2-1)/(2q))^\top$ and the radius $r = (q^2+1)/(2|q|)$. The oriented angle α between the sphere and the xy-plane is equal to the angle between the vectors $(\mathbf{C}-\mathbf{A}_1)$ and $(0,0,1)^\top$, which gives $\tan\alpha = \frac{2q}{1-q^2}$. The second case can be proved similarly by a direct computation. □

Remark 1. The angle α is equal to the angle which is used in the definition of tripolar coordinates, as introduced in [2].

The following observation provides an alternative geometric interpretation of the characterization result (10).

Proposition 2. *Any RCL surface (10) with the reference circle $\mathcal{C}$ can be obtained by composing*

(i) the inversion M with respect to the sphere centered at $(0,-1,0)^\top$ with radius $\sqrt{2}$,
(ii) the rotation R_α about the x-axis through the angle $\alpha(u,v)$, where q satisfies (16), and
(iii) the same inversion as in (i) ,

and applying this transformation to the parameterization $(u,v,0)^\top$ of the plane containing $\mathcal{C}$.

Proof. The rotation (ii) and the inversion (i,iii) are described by

$$R_\alpha(x,y,z) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1-q^2}{1+q^2} & -\frac{2q}{1+q^2} \\ 0 & \frac{2q}{1+q^2} & \frac{1-q^2}{1+q^2} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{17}$$

and

$$M(x,y,z) = \frac{1}{x^2+(y+1)^2+z^2} \begin{pmatrix} 2x \\ 1-x^2-y^2-z^2 \\ 2z \end{pmatrix}. \tag{18}$$

A direct computation now confirms that

$$\mathbf{P}(u,v) = (M \circ R_\alpha \circ M)(u,v,0), \tag{19}$$

cf. (16) and (10). □

Remark 2. The characterization (19) of RCL surfaces can be derived directly, as follows. The inversion M maps the reference circle to the x-axis and the circles of constant chord-length ratios described in Lemma 2 to coaxial circles around it. Consequently, $M(\mathbf{P}(u,v))$ can be obtained by applying the rotation R_α to the point $M(u,v,0)$. This leads to (19), since $M = M^{-1}$. All RCL surfaces can be obtained in this way, since M is a birational mapping. Proposition 1 can also be derived from this construction, since the spherical inversion M is a conformal transformation.

4 Properties and Examples of RCL Surfaces

In this section we will review some attractive properties of RCL surfaces and demonstrate them on some interesting examples which are computed using (10) for different choices of $q(u,v)$. Obviously, by choosing a constant function $q(u,v)$, we obtain a sphere, cf. [2].

Proposition 3. *Any RCL surface $\mathbf{P}(u,v)$ has a rational unit normal field along the reference circle. On the other hand, any rational unit normal field along the reference circle can be extended to an RCL surface. Finally, two RCL surfaces given by (10) with functions q_1, q_2 have the same normals along the reference circle if and only if*

$$q_1 - q_2 = (1-u^2-v^2)f, \tag{20}$$

where $f(u,v)$ is a rational function.

Proof. Under the condition $u^2+v^2=1$, the unit normal of $\mathbf{P}$ can be computed from (10) as

$$\left(\frac{2qu}{1+q^2}, \frac{2qv}{1+q^2}, \frac{1-q^2}{1+q^2}\right)^\top. \tag{21}$$

This gives also the second statement. Finally, the third part is a direct consequence. □

Let I denotes the circle inversion with respect to the reference circle in the u,v plane, i.e.,

$$I(u,v) = \left(\frac{u}{u^2+v^2}, \frac{v}{u^2+v^2}\right)^\top.$$

The following proposition can be verified by a straightforward computation.

Proposition 4. *The two surfaces* $\mathbf{P}_1(u,v)$, $\mathbf{P}_2(u,v)$ *obtained for* $q(u,v)$ *and* $-1/q(I(u,v))$, *respectively, are identical up to the reparameterization via* I, *i.e.,*

$$\mathbf{P}_1(u,v) = \mathbf{P}_2(I(u,v)).$$

Definition 3. *For a given* q *let us call the restriction of* $\mathbf{P}_1(u,v)$, *or* $\mathbf{P}_2(u,v)$ *to the reference disc (i.e., to the interior of the reference circle) the* first branch, *or the* second branch *of the associated RCL surface.*

Figures 2 and 3 (left) present examples of the surfaces mentioned in Proposition 4, where red and blue patches correspond to $\mathbf{P}_1(u,v)$ and $\mathbf{P}_2(u,v)$, respectively.

Proposition 5. *If* $q(u,v)$ *(or* $1/q(I(u,v))$*) does not possess a pole at* $(0,0)$, *then the first branch (or the second branch) is smooth and bounded. In particular, if both these conditions hold, then the entire RCL surface is a closed bounded smooth surface.*

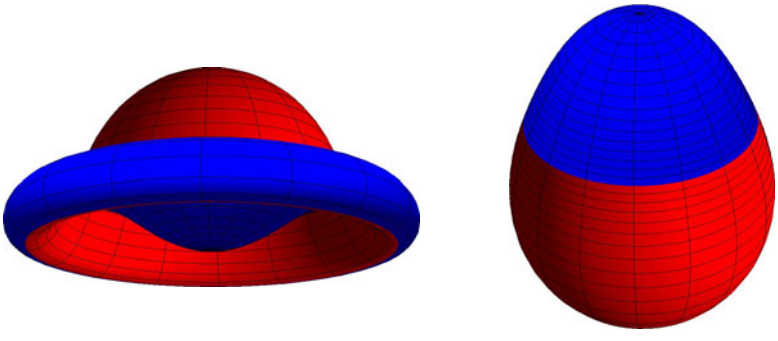


Fig. 2. Left: $q(u,v) = 1 - u^2 - v^2$; Right: $q(u,v) = 1/(1+u^2+v^2) + 1$

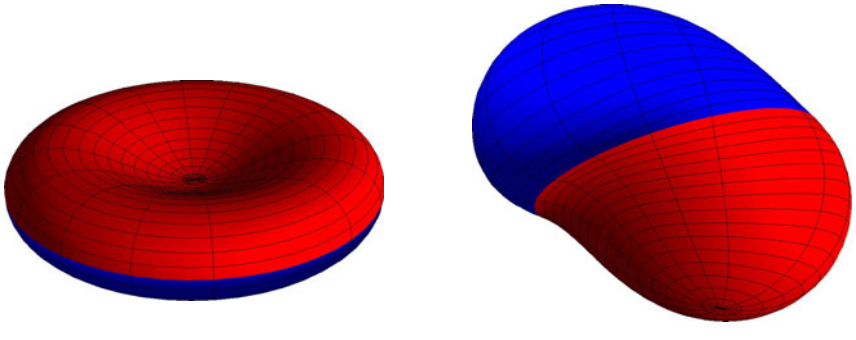


Fig. 3. Left: $q(u,v) = u^2 + v^2$; Right: $q(u,v) = u + 1$

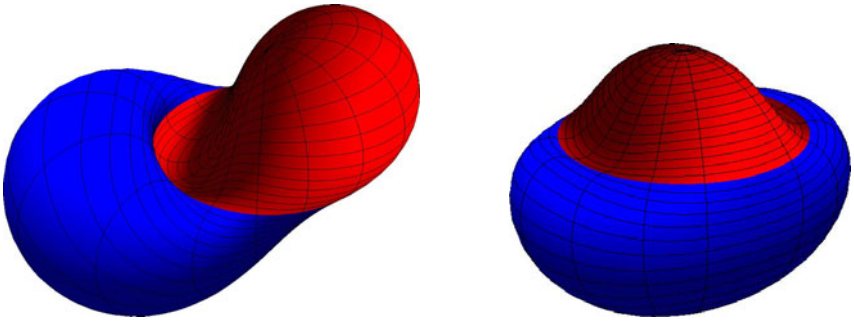


Fig. 4. Left: $q(u,v) = 2u + v + 1$; Right: $q(u,v) = u^2 - 2/3$

Proof. If there is no pole for q at $(0,0)$, then $\mathbf{P}_q(0,0) = (0,0,q)^\top$ is well defined and finite. By a continuity argument the same holds for some neighborhood of $(0,0)$. The remainder of the first branch is also bounded, since each point must lie on the corresponding circle – see Lemma 2. The same argument holds for the second branch and $-1/q(I(0,0))$. □

Proposition 6. *The first branches* $\mathbf{P}_1$, $\tilde{\mathbf{P}}_1$ *of two RCL surfaces join with* G^1 *continuity along the reference circle if and only if* $q\tilde{q} = -1$ *for* $u^2 + v^2 = 1$.

Proof. The first branches $\mathbf{P}_1$ and $\tilde{\mathbf{P}}_1$ join with G^1 continuity along the reference circle iff $\tilde{\alpha} = -(180° - \alpha)$. Hence, $\tilde{q} = \tan\frac{\tilde{\alpha}}{2} = -\tan\left(90° - \frac{\alpha}{2}\right) = -\cot\frac{\alpha}{2} = -1/q$. □

Figures 3 (right) and 4 show examples of surfaces described in Proposition 6, where red and blue patches correspond to $\mathbf{P}_1$ and $\tilde{\mathbf{P}}_1$, respectively.

5 Conclusion

We described a class of rational triangular Bézier surfaces possessing a parameterization which preserves the distance ratios to the vertices of the domain triangle inscribed to the reference circle. This extends the property of chord-length parameterization of rational curves, which was studied in [10] and [13], to the case of surfaces. We identified a family of RCL surfaces, characterized their general parameterization and studied their properties. The future research will be focused mainly on modeling with surface patches of this type.

Acknowledgments

B. Bastl, M. Lávička and Zbyněk Šír were supported by Research Plan MSM 4977751301. All authors were supported by grant 2009/05 of AKTION Österreich–Tschechische Republik. We thank all referees for their comments which helped us to improve this paper.

References

1. Albrecht, G.: Determination and classification of triangular quadric patches. Computer Aided Geometric Design 15, 675–697 (1998)
2. Bastl, B., Jüttler, B., Lávička, M., Schicho, J., Šír, Z.: Spherical quadratic Bézier triangles with rational chord lengths parameterization and tripolar coordinates in space, Computer Aided Geometric Design, submitted. Available as Report no. 90 (2009) (submitted), http://www.industrial-geometry.at/techrep.php
3. Bateman, H.: Spheroidal and bipolar coordinates. Duke Mathematical Journal 4(1), 39–50 (1938)
4. Dietz, R., Hoschek, J., Jüttler, B.: An algebraic approach to curves and surfaces on the sphere and on other quadrics. Computer Aided Geometric Design 10, 211–229 (1993)
5. Farin, G.: Curves and Surfaces for CAGD. Morgan Kaufmann, San Francisco (2002)
6. Farin, G.: Rational quadratic circles are parametrized by chord length. Computer Aided Geometric Design 23, 722–724 (2006)
7. Farouki, R.: Pythagorean-Hodograph Curves: Algebra and Geometry Inseparable. Springer, Berlin (2008)
8. Farouki, R., Moon, H.P.: Bipolar and multipolar coordinates. In: Cippola, R., Martin, R. (eds.) The Mathematics of Surfaces IX, pp. 348–371. Springer, Heidelberg (2000)
9. Farouki, R., Sakkalis, T.: Real rational curves are not unit speed. Computer Aided Geometric Design 8, 151–158 (1991)
10. Lü, W.: Curves with chord length parameterization. Computer Aided Geometric Design 26, 342–350 (2009)
11. Sabin, M.A., Dodgson, N.A.: A circle-preserving variant of the four-point subdivision scheme. In: Lyche, T., Schumaker, L. (eds.) Mathematical Methods for Curves and Surfaces, pp. 275–286. Nashboro Press (2005)
12. Sánchez-Reyes, J.: Complex rational Bézier curves. Computer Aided Geometric Design 26, 865–876 (2009)
13. Sánchez-Reyes, J., Fernández-Jambrina, L.: Curves with rational chord-length parametrization. Computer Aided Geometric Design 25, 205–213 (2008)

Support Function of Pythagorean Hodograph Cubics and G^1 Hermite Interpolation

Eva Černohorská and Zbynek Šír

Faculty of Mathematics and Physics, Charles University in Prague,
Sokolovská 83, 186 75 Praha 8
evajs@atlas.cz,zbynek.sir@mff.cuni.cz

Abstract. The Tschirnhausen cubic represents all non-degenerate Pythagorean Hododgraph cubics. We determine its support function and represent it as a convolution of a centrally symmetrical curve and a curve with linear normals. We use the support function to parametrize the Tschirnhausen cubic by normals. This parametrization is then used to an elegant and complete solution of the G^1 Hermite interpolation by Pythagorean Hodograph cubics. We apply the resulting algorithm to various examples and extend it to the interpolation by offsets of PH cubics.

1 Introduction

The support function representation describes a curve as the envelope of its tangent lines, where the distance between the tangent line and the origin is specified by a function of the unit normal vector. This representation is one of the classical tools in the field of convex geometry [3,10,11]. In this representation offsetting and convolution of curves correspond to simple algebraic operations of the corresponding support functions. In addition, it provides a computationally simple way to extract curvature information [9]. Applications of this representation to problems from Computer Aided Design were foreseen in the classical paper [16] and developed in several recent publications, see e.g. [18,8,19,2,1,20].

Pythagorean hodograph (PH) curves form an important subclass of polynomial parametric curves. The distinguishing property is that their arc length function is piecewise polynomial and, in the planar case, they possess rational offset curves. Since their introduction by Farouki and Sakkalis [5], planar and spatial PH curves have been thoroughly studied, see [6,7] and the references cited therein. Due to the special algebraic properties of PH curves, all constructions, such as interpolation or approximation, which are linear in the case of standard spline curves become quadratic, see e.g. [12,15,17].

The simplest nontrivial polynomial PH curves are polynomial PH cubics. The problem of G^1 Hermite interpolation with PH cubics was first studied in [14] and later analyzed in [4]. In these papers the control polygon of the interpolants is constructed directly, while certain condition on its legs and angles is required, in order to ensure the PH condition.

The support function representation is particularly well suited for geometrical interpolation. In the present paper we exploit the fact, that up to similarities and reparameterization there is only one PH cubic, called the Tschirnhausen cubic. We determine its

B. Mourrain, S. Schaefer, and G. Xu (Eds.): GMP 2010, LNCS 6130, pp. 29–42, 2010.

support function and then fully describe all possible data which can be interpolated and give the number of solutions. We thus solve the interpolation problem in a top-down way and extend results of [14].

The remainder of this paper is organized as follows. In Section 2 we recall some basic facts about PH curves and support function. In Section 3 we determine the support function and suitable parameterization of the Tschirnhausen cubic. In Section 4 we carefully analyze all Hermite data occurring on the Tschirnhausen cubic. Section 5 is devoted to the interpolation algorithm and examples. Finally we conclude the paper.

2 Preliminaries

In this section we will recall some basic facts about Pythagorean Hodograph curves and the support function representation.

Recall, that a polynomial planar curve $\mathbf{c} = (x(t), y(t))^\top$ is called *Pythagorean-Hodograph curve* (PH), if there exists a polynomial σ so that

$$x'^2(t) + y'^2(t) = \sigma^2(t) \tag{1}$$

see [6] and citations therein. All polynomial PH curves can be constructed using the following lemma proved for general unique factorization domains by Kubota in [13].

Lemma 1. *The condition (1) is satisfied if and only if there exist polynomials $u(t)$, $v(t)$, $w(t)$ such that*

$$x'(t) = 2u(t)v(t)w(t) \quad \textit{and} \quad y'(t) = [u^2(t) - v^2(t)]w(t). \tag{2}$$

We are in particular interested in cubic PH curves. We can suppose that $w(t) = 1$, since any non-constant w would lead to a cubic which is just a (singular) reparameterization of a straight line. In order to obtain a cubic, the maximal degree of polynomials u, v must be 1. Consider the following PH cubic.

Definition 1. *The PH curve*

$$T(t) = \left(-t^2, t - \frac{t^3}{3}\right)^\top, \tag{3}$$

obtained setting $w(t) = 1$, $u(t) = -1$ and $v(t) = t$ in (2), is called the Tschirnhausen cubic *see Figure 1.*

We use a somewhat unusual position of the Tschirnhausen cubic in order to obtain a nice support function later on. In fact, this curve is a typical representative of PH cubics as shown in the following lemma which is proved in [5].

Lemma 2. *Any (segment of) PH cubic can be obtained from (a segment of) the Tschirnhausen cubic via scaling, rotation, translation and linear reparameterization.*

For an (oriented) planar curve $\mathbf{c}$ we define its support function h as (possibly multivalued) function defined on the (subset of the) unit circle

$$h : \mathbb{S}^1 \supset U \to \mathbb{R}^1$$

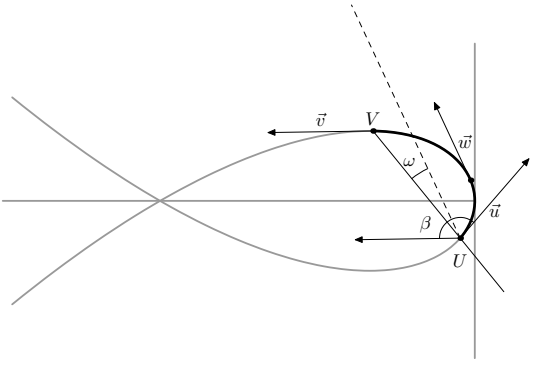


Fig. 1. The Tschirnhausen cubic and description of the Hermite data on its segment U, V using vectors β and ω

which to any unit normal $\mathbf{n} = (n_1, n_2)^\top$ associates the distance(s) from the origin to the corresponding tangent line(s) of the curve. The curve $\mathbf{c}$ can be recovered from h as the envelope of the system $\{\mathbf{n} \cdot \mathbf{x} - h(\mathbf{n}) = 0 : \mathbf{n} \in U\}$ of these lines via the envelope formula

$$\mathbf{c}(\mathbf{n}) = h(\mathbf{n})\mathbf{n} + \nabla_{\mathbb{S}_1} h(\mathbf{n}), \tag{4}$$

where $\nabla_{\mathbb{S}_1}$ denotes the intrinsic gradient with respect to the unit circle. Rotation, scaling and translation of $\mathbf{c}$ correspond to rotation of h, multiplication of h by a constant and adding a linear term to h, respectively. Moreover the convolution of curves corresponds to the addition of their support functions, see [18] for more details.

3 Support Function of the Tschirnhausen Cubic

In this section we will determine the support function of the Tschirnhausen cubic and describe it as a convolution of a curve with odd rational support function and a curve with even rational support function. We will also parametrize the Tschirnhausen cubic by its normals.

Theorem 1. *The support function of the Tschirnhausen cubic is the restriction of the function* (5) *to the unit circle.*

$$h(n_1, n_2) = \frac{n_1^3 - 3n_1 + 2}{3n_2^2} \tag{5}$$

Proof. Consider the two equations

$$\mathbf{n} \cdot T(t) = -n_1 t^2 + n_2 (t - \frac{t^3}{3}) = h \quad \text{and} \quad \mathbf{n} \cdot T'(t) = -n_1 2t + n_2 (1 - t^2) = 0, \tag{6}$$

where the former express that $T(t)$ lies on a line and the latter forces its normal vector $\mathbf{n}$ to be perpendicular to the tangent vector at the same point $T(t)$. After elimination

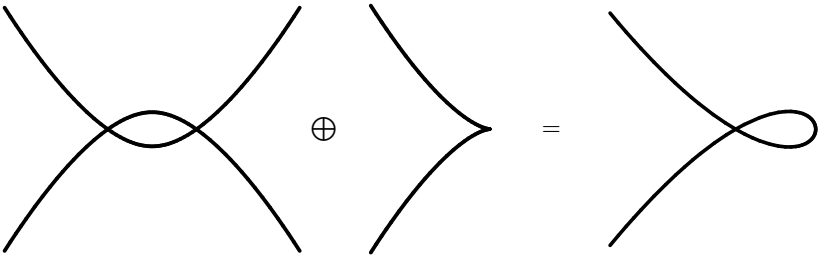


Fig. 2. Tschirnhausen cubic as a convolution of a LN curve and a curve with an even rational support function

of t from (6) we get one quadratic equation for h. Each solution corresponds to one orientation of T and we choose one of them by fixing the normal $(1,0)^\top$ at the point $(0,0)^\top$. After simplification of the corresponding solution using $n_1^2 + n_2^2 = 1$ we obtain the result (5). See [8] for more details about support functions obtained as restrictions of rational functions. □

For later reference purposes we substitute the parameterization $\mathbf{n} = (\cos\theta, \sin\theta)^\top$ of the unit circle into (5) obtaining

$$h(\theta) = \frac{(1-\cos\theta)(2+\cos\theta)}{3(1+\cos\theta)}. \tag{7}$$

The envelope formula (4) then becomes

$$T(\theta) = h(\theta)\mathbf{n}(\theta) + h'(\theta)\mathbf{n}'(\theta) = \left(-\frac{1-\cos 2\theta}{2(1+\cos\theta)^2}, \frac{2\sin\theta + 2\sin 2\theta}{3(1+\cos\theta)^2} \right)^\top, \tag{8}$$

which is the parametrization of the Tschirnhausen cubic by its normal, i.e. the normal at $T(\theta)$ is simply $\mathbf{n}(\theta) = (\cos\theta, \sin\theta)^\top$. In this case, the two parameterizations $T(t)$ and $T(\theta)$ are related by simple equation

$$t = \tan\frac{\theta}{2}. \tag{9}$$

Theorem 1 also leads to the natural decomposition of the support function of the Tschirnhausen cubic into a sum of an even and an odd rational function

$$\frac{n_1^3 - 3n_1 + 2}{3n_2^2} = \frac{2}{3n_2^2} + \frac{n_1^3 - 3n_1}{3n_2^2}. \tag{10}$$

Consequently, the Tschirnhausen cubic is a convolution (see Figure 2) of a centrally symmetrical curve (of degree 6) with even rational support and a cubic curve with linear normals, see [8] for general theory.

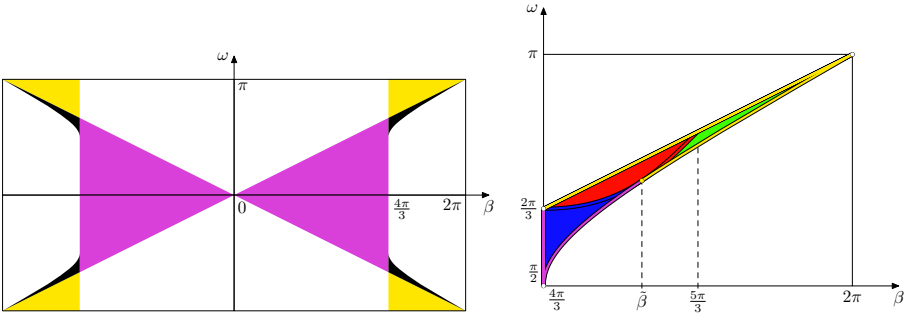


Fig. 3. All pairs of angles obtained from the segments on the Tschirnhausen cubic (left) and the detail of one of the four symmetrically placed components of the domain Ω (right), which is black on the left figure. Different colors represent: no segment (white), one segment without loop (violet), one segment with loop (yellow), two segments without loop (blue), two segments with loop (green) and two segments of which precisely one has a loop (red). When printed in black and white the colors can be distinguished by the decreasing brightness: yellow, green, violet, red, blue. Note than only two colors (yellow and violet) appear on the left figure.

4 G^1 Data on the Tschirnhausen Cubic

In this section we will describe G^1 Hermite data generated by boundary points of all possible segments of Tschirnhausen cubic. Consider a segment with the end–points U,V. The G^1 boundary data are (up to similarity) fully described by a pair of oriented angles $\beta \in (-2\pi, 2\pi)$ and $\omega \in [-\pi, \pi]$, where β is the rotation angle traveled by the tangent vector between points U, V and ω is the oriented angle between the difference vector $\overrightarrow{UV}$ and the bisector of β (more precisely the middle tangent vector $\overrightarrow{w}$ which appears in the segment UV), see Fig. 1.

Clearly the obtained pairs (β, ω) do not depend on the scaling, rotation and translation of the cubic. On the other hand, symmetrical results will occur for its two possible orientations.

Lemma 3. *Considering all segments on the Tschirnhausen cubic, there is*

i) one segment without loop for any pair of angles satisfying

$$|\beta| < \frac{4\pi}{3}, \quad |\omega| < \frac{|\beta|}{2} \tag{11}$$

ii) one segment with loop for any pair of angles

$$\frac{4\pi}{3} < |\beta| < 2\pi, \quad \frac{|\beta|}{2} < |\omega| < \pi \tag{12}$$

iii) two segments for any pair of angles in the interior of the domain

$$\Omega := \{(\beta, \omega) : \frac{4\pi}{3} \le |\beta| \le 2\pi, \quad |\pi - \arctan \frac{2 \sin \frac{|\beta|}{2}}{\sqrt{1 + 2\cos\beta}}| \le |\omega| \le \frac{|\beta|}{2}\}. \tag{13}$$

and one segment for angles on its boundary curves.[1]

Other pairs of angles can not be obtained, including corner points of Ω, see Figure 3.

Proof. Two different orientations of the Tschirnhausen cubic will produce symmetrical data ($-\beta$, ω instead of β, ω). We will therefore suppose $\beta > 0$ and use only the parametrization (8) which is very suitable for our purpose, since θ is the oriented angle between the tangent vector and the vector $(0,1)^\top$. For this reason, any segment for given β can be obtained from $T(\theta)$ for $\theta \in (-\beta/2+\alpha, \beta/2+\alpha)$. Since $T(\theta)$ is defined only for $\theta \in (-\pi, \pi)$, we get following domains for β, α

$$\beta \in (0, 2\pi), \qquad \alpha \in \left(-\pi + \frac{\beta}{2}, \pi - \frac{\beta}{2}\right). \tag{14}$$

In order to simplify the computation of ω we will consider rotated Tschirnhausen cubic

$$T_\alpha(\tilde{\theta}) := T(\tilde{\theta} + \alpha)$$

and segments obtained by taking $T_\alpha(\tilde{\theta})$ for $\tilde{\theta} \in (-\beta/2, \beta/2)$. In this case, ω is simply the angle between the vectors $\overrightarrow{UV}$ and $(0,1)^\top$, which is the bisector of the tangent vectors at U and V. By a direct computation, we get

$$\overrightarrow{UV} = T_\alpha\left(\tfrac{\beta}{2}\right) - T_\alpha\left(-\tfrac{\beta}{2}\right) = \left(-\frac{8\sin\alpha\sin^3\frac{\beta}{2}}{3(\cos\alpha+\cos\frac{\beta}{2})^3}, \frac{4(3\cos\frac{\beta}{2}+(2+\cos\beta)\cos\alpha)\sin\frac{\beta}{2}}{3(\cos\alpha+\cos\frac{\beta}{2})^3}\right)^\top. \tag{15}$$

and taking the ratio of both components of this vector we get

$$\tan\omega = -\frac{2\sin\alpha\sin^2\frac{\beta}{2}}{3\cos\frac{\beta}{2} + (2+\cos\beta)\cos\alpha} =: F(\alpha,\beta). \tag{16}$$

The angle ω can be deduced from (16) considering, that $|\omega| < \pi/2$ or $|\omega| > \pi/2$ depending on the sign of the y-component of $\overrightarrow{UV}$. In order to understand the behavior of ω as a function of β and α we have to analyze the function $F(\alpha,\beta)$ within the domain (14). It is easy to check that $F(\alpha,\beta)$ is odd in α and even in β. We will discuss different cases depending on β. The denominator of (16) is zero exactly for

$$\alpha = \pm \underbrace{\arccos\left(\frac{3\cos\frac{\beta}{2}}{2+\cos\beta}\right)}_{\bar{\alpha}} \tag{17}$$

and due to (14), $F(\alpha,\beta)$ will have a discontinuity for $\beta \in (\pi, 4\pi/3)$. At the same time, $\bar{\alpha}$ is the only value, for which the y-component of the vector $\overrightarrow{UV}$ can be 0 and change sign.

We also obtain

$$\frac{\partial F(\alpha,\beta)}{\partial\alpha} = -\frac{2(2+3\cos\alpha\cos\frac{\beta}{2}+\cos\beta)\sin^2\frac{\beta}{2}}{(3\cos\frac{\beta}{2}+(2+\cos\beta)\cos\alpha)^2}, \tag{18}$$

[1] The exact behavior for angles in $\tilde{\Omega}$ is discussed in Corollary 1.

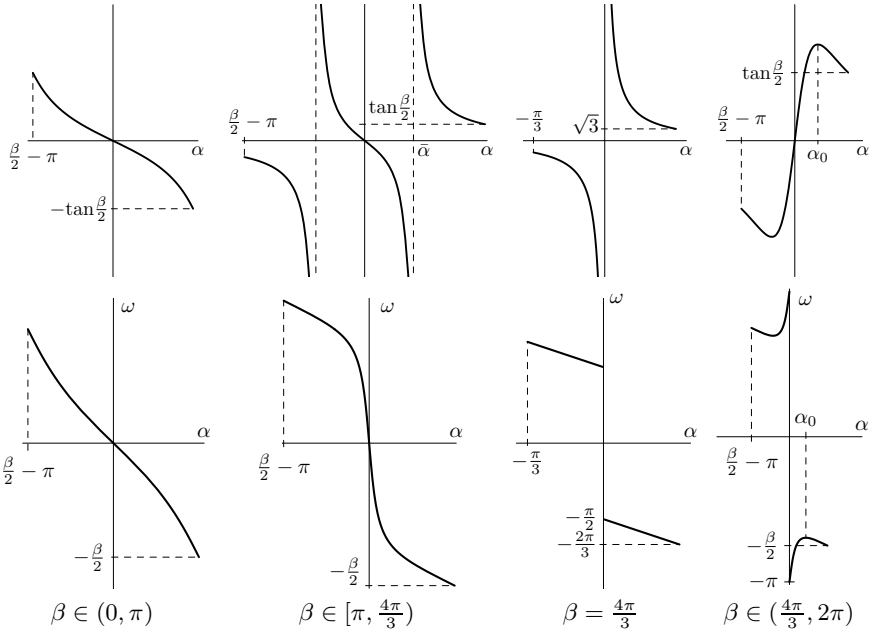


Fig. 4. For fixed β we display F (upper row) and ω (lower row) as functions of $\alpha \in (-\pi + \frac{\beta}{2}, \pi - \frac{\beta}{2})$

which is zero for

$$\alpha = \pm \underbrace{\arccos\left(\frac{2+\cos\beta}{3\cos\frac{\beta}{2}}\right)}_{\alpha_0}. \tag{19}$$

A segment on Tschirnhausen cubic cubic $T(\theta)$ has a loop if and only if the both values $\theta = \pm 2\pi/3$ are contained in its domain. This gives following loop condition for the rotation angle α

$$|\alpha| \geq \frac{\beta}{2} - \frac{2\pi}{3}, \tag{20}$$

which can be satisfied only for $\beta \geq 4\pi/3$.

Case $\beta \in (0, \pi)$:
In this case ω is strictly decreasing function of α, see Fig. 4, lower row. Indeed both $\bar{\alpha}$ and α_0 are not contained in the domain $\alpha \in (-\pi + \frac{\beta}{2}, \pi - \frac{\beta}{2})$ and function F is continuous and strictly decreasing and takes its values from the interval $(-\tan\frac{\beta}{2}, \tan\frac{\beta}{2})$. Figure 4, upper row, shows the behavior of F. Moreover the y-component of the vector $\overrightarrow{UV}$ is in this case always positive and therefore $\omega = \arctan(F)$ yielding (11) for $\beta \in (0, \pi)$.

Case $\beta \in [\pi, \frac{4\pi}{3})$:
In this case ω is again decreasing function of α, see Fig. 4, lower row. In this case, $\bar{\alpha} \in (-\pi + \frac{\beta}{2}, \pi - \frac{\beta}{2})$ and F has two discontinuities. However, $\alpha_0 \notin (-\pi + \frac{\beta}{2}, \pi - \frac{\beta}{2})$ and F has negative derivative everywhere. The values of F cover all $\mathbb{R}$. Figure 4, upper row, shows typical behavior of F. At the same time, the y-component of the vector $\overrightarrow{UV}$ is positive for $\alpha \in (-\bar{\alpha}, \bar{\alpha})$ and negative otherwise. For this reason

$$\omega = \begin{cases} \arctan(F) + \pi, & \text{for } \alpha \in (-\pi + \frac{\beta}{2}, -\bar{\alpha}) \\ \arctan(F), & \text{for } \alpha \in (-\bar{\alpha}, \bar{\alpha}) \\ \arctan(F) - \pi, & \text{for } \alpha \in (\bar{\alpha}, \pi - \frac{\beta}{2}) \end{cases}$$

and we get (11) for $\beta \in [\pi, \frac{4\pi}{3})$.

Case $\beta = \frac{4\pi}{3}$:
This is the limit situation of the previous case. $0 = \bar{\alpha} \in (-\pi + \frac{\beta}{2}, \pi - \frac{\beta}{2})$, but $\alpha_0 \notin (-\pi + \frac{\beta}{2}, \pi - \frac{\beta}{2})$. Therefore F has one discontinuity and its derivative is always negative. The behavior of F can be seen in Fig. 4, upper row, and it takes values from intervals $\in (-\infty, -\sqrt{3}) \cup (\sqrt{3}, \infty)$. The y-component is negative everywhere, so

$$\omega = \begin{cases} \arctan(F) + \pi, & \text{for } \alpha \in (-\pi + \frac{\beta}{2}, 0) \\ \arctan(F) - \pi, & \text{for } \alpha \in (0, \pi - \frac{\beta}{2}), \end{cases}$$

see Figure 4, lower row. We thus obtain result for one boundary curve of Ω, see (13) and Fig 3, right. For $\alpha = 0$ we get degenerated data, since $U = V$. For other α the loop condition (20) is not satisfied and the segments are without loop.

Case $\beta \in (\frac{4\pi}{3}, 2\pi)$:
In this case, $\bar{\alpha} \notin (-\pi + \frac{\beta}{2}, \pi - \frac{\beta}{2})$, but $\alpha_0 \in (-\pi + \frac{\beta}{2}, \pi - \frac{\beta}{2})$, F is continuous and has two extremes, see Figure 4, upper row. At the same time, the y-component of the vector $\overrightarrow{UV}$ is always negative and

$$\omega = \begin{cases} \arctan(F) + \pi, & \text{for } \alpha \in (-\pi + \frac{\beta}{2}, 0] \\ \arctan(F) - \pi, & \text{for } \alpha \in (0, \pi - \frac{\beta}{2}), \end{cases}$$

see Figure 4, lower row. Note, that this function is in fact smooth due to the periodic behavior of ω, (i.e. values π and $-\pi$ are identified. The values at endpoints are $\pm\beta/2$ and we obtain (12). The extreme values of F can be computed as $\pm\frac{2\sin\frac{\beta}{2}}{\sqrt{1+2\cos\beta}}$ and we obtain boundary curves of Ω in (13).

Let us denote $\underline{\alpha} = \arctan\beta/2$ the left point of the interval where F cease to be proper. The loop condition value $\alpha_l = \frac{\beta}{2} - \frac{2\pi}{3}$, see (20) will always fall in the interval $[\underline{\alpha}, \pi - \beta/2]$. For this reason the segments of (12) will always have a loop. The position of α_l with respect to α_0 leads to various cases of presence of loops for segments in Ω. □

The previous proof, in particular the last two cases, yields also an information about the data in Ω, which we summarize in following corollary.

Corollary 1. *The following pairs of angles on the boundary $\partial\Omega$ are obtained (see figure 3, right).*

i) from one segment without loop

$$\left\{|\beta| = \frac{4\pi}{3}, \frac{\pi}{2} < |\omega| < \frac{2\pi}{3}\right\} \cup \left\{\frac{4\pi}{3} < |\beta| < \tilde{\beta}, |\omega| = F_0\right\} \tag{21}$$

ii) from one segment with a loop

$$\left\{|\beta| \in \left(\frac{4\pi}{3}, 2\pi\right), |\omega| = \frac{|\beta|}{2}\right\} \cup \left\{|\beta| \in \left[\tilde{\beta}, 2\pi\right), |\omega| = F_0\right\}. \tag{22}$$

Following pairs in the interior of Ω are obtained

iii) from one segment without loop and one segment with a loop

$$\left\{|\beta| \in \left(\frac{4\pi}{3}, \frac{5\pi}{3}\right), |\omega| \in \left(F_l, \frac{|\beta|}{2}\right)\right\} \tag{23}$$

iv) from two segments without loop

$$\left\{|\beta| \in \left(\frac{4\pi}{3}, \tilde{\beta}\right), \quad |\omega| \in (F_0, F_l)\right\} \tag{24}$$

v) from two segments with loop

$$\left\{|\beta| \in \left[\frac{5\pi}{3}, 2\pi\right), |\omega| \in \left(F_0, \frac{|\beta|}{2}\right)\right\} \cup \left\{|\beta| \in \left(\tilde{\beta}, \frac{5\pi}{3}\right), |\omega| \in (F_0, F_l)\right\} \tag{25}$$

where

$$\tilde{\beta} = 2\arccos\left(-\frac{2}{\sqrt{7}}\right), \quad F_0 = \pi - \arctan\frac{2\sin\frac{|\beta|}{2}}{\sqrt{1+2\cos\beta}},$$

$$F_l = \pi + \frac{2\cos\left(\frac{|\beta|}{2} + \frac{\pi}{6}\right)\sin^2\frac{\beta}{2}}{3\cos\frac{\beta}{2} - (2+\cos\beta)\sin\left(\frac{|\beta|}{2} + \frac{\pi}{6}\right)}.$$

5 Hermite Interpolation with PH Cubics and Their Offsets

In this section we will apply the previous results to the inverse problem, i.e. to the Hermite interpolation by PH cubics. We will discuss the number of solutions, provide an algorithm for their computation and give a number of examples. We will also extend the interpolation algorithm to the offsets of PH cubics.

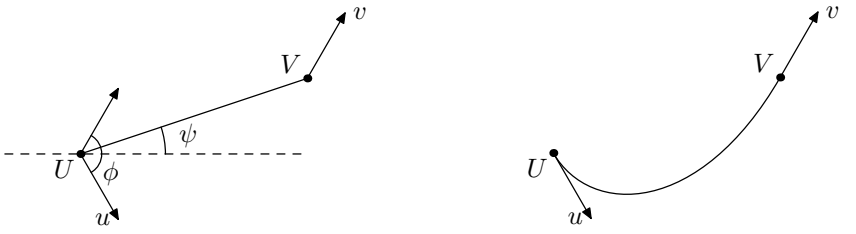


Fig. 5. Given data and a G^1 interpolant

5.1 Existence and Number of Solutions

We have the situation as in the Figure 5, i.e. we are given two points U, V along with unit tangent vectors $\overrightarrow{u}$ and $\overrightarrow{v}$ and we want to find a PH cubic which interpolates these data.

Up to similarities these data are described by two angles $\phi \in (-\pi, \pi)$, $\psi \in [-\pi, \pi]$, where ϕ is the oriented angle between $\overrightarrow{u}$ and $\overrightarrow{v}$ and ψ the oriented angle between $\overrightarrow{UV}$ and the bisector $(\overrightarrow{u} + \overrightarrow{v})/2$ (see Fig. 1). For simplicity we exclude the case $\overrightarrow{u} = -\overrightarrow{v}$ which allows for no interpolants.

Suppose we have an interpolant of the data by a segment of the Tschirnhausen cubic. Then angles ϕ, ψ are related to the angles β, ω in one of the two following way. Either the curve tangent travels the shorter angle ϕ or it travels the complementary angle. Thus we get

$$\beta = \phi, \omega = \psi \quad \text{or} \quad \beta = \phi - 2\pi, \omega = \psi - \pi \tag{26}$$

Applying Lemma 3 and combining these two cases, we obtain immediately following result.

Theorem 2. *For given G^1 Hermite data described by ϕ, ψ there exists*

i) two interpolants (one of them with a loop) if

$$0 < |\phi| < \frac{2\pi}{3} \quad \textit{and} \quad |\psi| < \frac{|\phi|}{2} \tag{27}$$

ii) one interpolant with a loop if

$$|\phi| > \frac{2\pi}{3} \quad \textit{and} \quad |\psi| \neq \frac{|\phi|}{2} \tag{28}$$

iii) two interpolants for (ϕ, ψ) in the interior of the domain

$$\tilde{\Omega} := \{(\phi, \psi) : |\phi| \leq \frac{2\pi}{3}, \quad |\pi - \arctan \frac{2 \sin \frac{|\phi|}{2}}{\sqrt{1 + 2\cos\phi}}| < |\psi| < |\frac{\phi}{2}|\}. \tag{29}$$

and one segment for angles on its boundary curves.[2]

[2] Exact behavior for angles in Ω follows from Corollary 1.

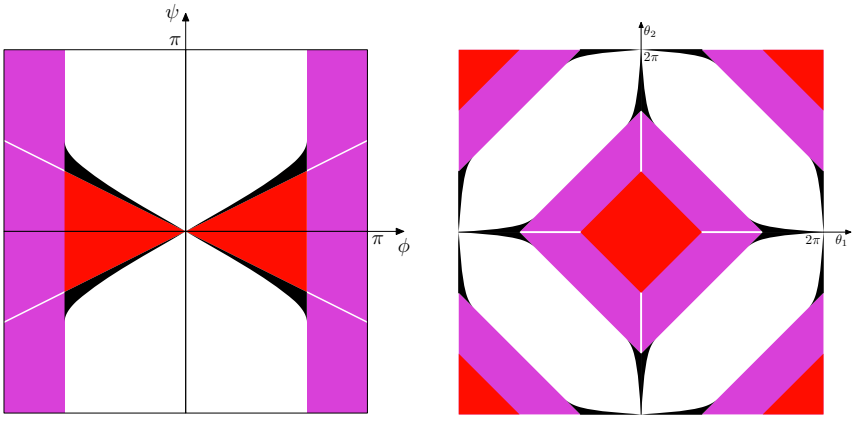


Fig. 6. Number of interpolants for various Hermite data expressed by angles ϕ, ψ (left figure) or equivalently by angles θ_1, θ_2 (right figure). Different colors represent: white: no segment, violet (or light gray in b/w): one segment without loop, red (or dark gray in b/w): one segment with loop and one without loop. For a detail of black regions see Figure 3, right.

Figure 6, left represents graphically the previous theorem. It is simply obtained from the Figure 3 by keeping the part $\beta < \pi$ corresponding to the first case of (26) and adding symmetrical images of the remaining parts corresponding to the second case of (26).

In order to make our results compatible with [14], we transform the angles ϕ, ψ to the angles θ_1, θ_2 used therein. These are oriented angles between $\overrightarrow{UV}$ and vectors $\overrightarrow{u}$ and $\overrightarrow{v}$ respectively, where θ_1 is measured anticlockwise and θ_2 clockwise. These angles are related to ϕ, ψ by a simple linear transformation and the results of the interpolation problem are displayed on the figure 6, right.

5.2 Computation of Interpolants and Examples

In order to compute interpolants for concrete G^1 Hermite data $U, V, \overrightarrow{u}, \overrightarrow{v}$ we design the following procedure.

1. Determine ϕ and ψ.
2. Using Theorem 2 decide whether to set $\beta = \phi, \omega = \psi$ or $\beta = \phi - 2\pi, \omega = \psi - \pi$, or to consider both cases.
3. Compute α from equation (16).
4. From (9) compute values of t_1, t_2 corresponding to $\theta = \alpha - \beta/2$ and $\theta = \alpha + \beta/2$.
5. Compute control points for the segment $t \in [t_1, t_2]$ of (3).
6. Transform the control points by the unique direct Euclidean similarity transformation which maps the first control point to U and the last one to V.

Let us comment on some steps of this procedure to argue, that they are extremely simple and fast. Steps 1 and 2 are trivial. Step 3 might seem too complicated, but (16) with unknown α is essentially of the form

$$A \sin(\alpha) + B \cos(\alpha) = C, \tag{30}$$

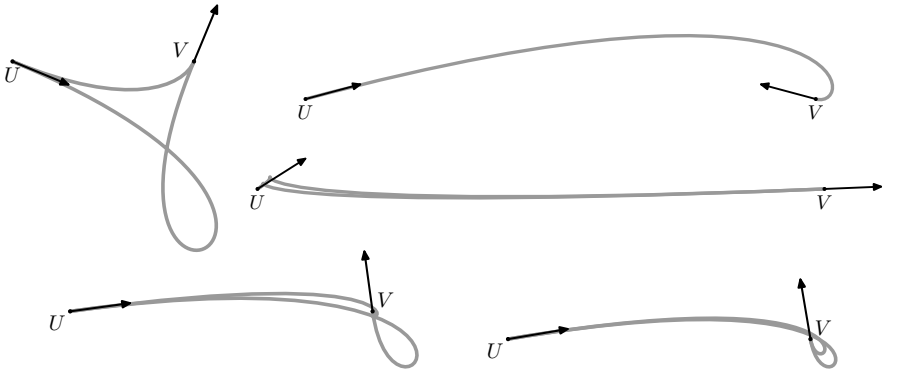


Fig. 7. Hermite data and PH cubic interpolants, see Table 1 for numerical values. Example 1 (left, top), example 2 (right, top), example 3 (right, middle), example 4 (left, bottom), example 5 (right, bottom).

Table 1. Examples of various input data and computed segments determined by α, see Figure 7 for resulting interpolants. The color corresponds to color code of Figures 6 and 3.

example number	(ϕ, ψ)	(β, ω)	computed α	presence of a loop	color code
1	$(\frac{\pi}{2}, \frac{\pi}{8})$	$(\frac{\pi}{2}, \frac{\pi}{8})$ $(\frac{3\pi}{2}, \frac{7\pi}{8})$	-1.435 -3.090	yes no	red
2	$(\frac{5\pi}{6}, -\frac{\pi}{2})$	$(-\frac{7\pi}{6}, \frac{\pi}{2})$	-2.325	no	violet
3	$(\frac{\pi}{6}, -\frac{289\pi}{3000})$	$(\frac{55\pi}{30}, \frac{2711\pi}{3000})$	0.137 0.160	yes yes	green
4	$(\frac{\pi}{2}, -\frac{88\pi}{300})$	$(-\frac{3\pi}{2}, \frac{212\pi}{300})$	-2.649 -2.908	no yes	red
5	$(\frac{\pi}{2}, -\frac{91\pi}{300})$	$(-\frac{3\pi}{2}, \frac{209\pi}{300})$	-2.649 -2.908	no no	blue

which is well known simple goniometric equation. Moreover, for the next step we really need only $\sin(\alpha)$ and $\cos(\alpha)$ which can be computed from (30) via solving a quadratic equation similar to [14]. Step 5 can be efficiently performed applying the de Casteljau algorithm to a precomputed polygon of the segment $t \in [0, 1]$. Step 6 is trivial and occurs in most interpolation algorithms requiring some kind of canonical position of the data.

We will now present several examples obtained by this procedure. Figure 7 shows input data (end-points and vectors) and interpolants to these data. Table 1 displays numerical values of angles occurring in the algorithm. Example 1 describes interpolants for data treated in [14]. The other examples show interpolants for data excluded [14] and differ by the number of loops.

The interpolation procedure can be used for approximation a given curve with a G^1 continuous spline composed of PH cubics. Figure 8, right, shows the graph of the function sinus approximated by a spline composed of 4 segments of PH cubics. The

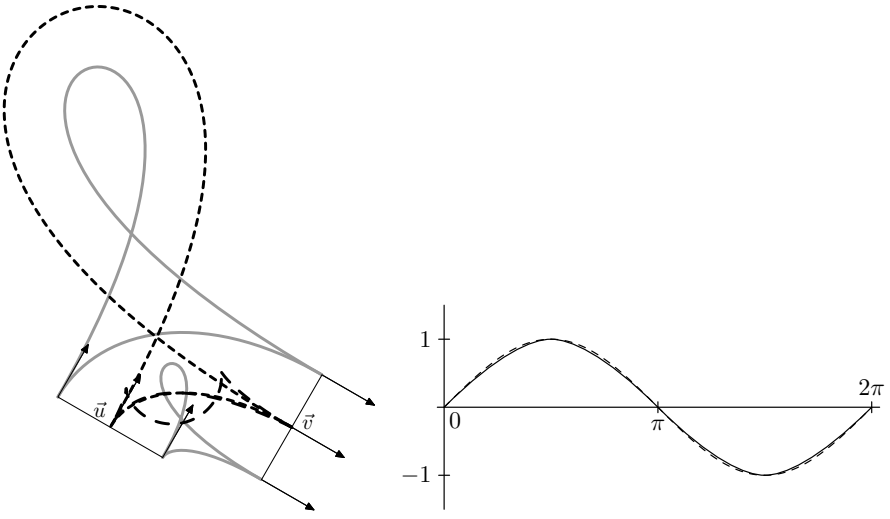


Fig. 8. Left:Hermite data and interpolanats (dashed) consisting of offsets at given distance to PH cubics (gray). Right: graph of the function sinus (dashed) approximated by a PH cubic spline (solid).

approximation order is 4 (error decreases 16 times after one subdivision), since in the limit case the analysis of [14] applies.

The interpolation algorithm can be simply extended to offsets of PH cubics. This problem naturally occurs when we want to produce certain shape with a circular tool and we want the center of the tool to follow a PH spline curve permitting to simply control its speed. Given G^1 data and the offset distance d, we can easily obtain the corresponding data for the PH cubic by shifting the end points perpendicularly to the end point vectors (angle ϕ will not change). We will not analyze in details this problem and limit ourselves to one example of data and four Hermite interpolants (two for the left and two for the right offset), see Figure 8, left.

6 Conclusion

We have determined the support function representation of the Tschirnhausen cubic and applied it to the computation of PH cubic Hermite interpolants. This approach allowed us to solve the interpolation problem in a top-down way based on the analysis of all possible segments of the Tschirnhausen cubic. We have presented a full discussion of existence and number of solutions, which is analogous to [4] and remove data restriction of [14]. We have also represented the Tschirnhausen cubic as the convolution of a centrally symmetrical sextic curve and a cubic with linear normals. In the future we plan to apply similar techniques to the G^2 interpolation with special curves.

Acknowledgment

This research was supported by the project MSM 0021620839 of the Czech Ministry of Education and by the project no. 201/08/0486 of the Czech Science Foundation.

References

1. Aigner, M., Gonzalez-Vega, L., Jüttler, B., Schicho, J.: Parameterizing surfaces with certain special support functions, including offsets of quadrics and rationally supported surfaces. Symbolic Comput. 44(2), 180–191 (2009)
2. Aigner, M., Gonzalez-Vega, L., Jüttler, B., Sampoli, M.L.: Computing isophotes on free-form surfaces based on support function approximation. In: Hancock, E.R., Martin, R.R., Sabin, M.A. (eds.) Mathematics of Surfaces XIII. LNCS, vol. 5654, pp. 1–18. Springer, Heidelberg (2009)
3. Bonnesen, T., Fenchel, W.: Theory of convex bodies. BCS Associates, Moscow (1987)
4. Byrtus, M., Bastl, B.: G1 Hermite interpolation by PH cubics revisited. Submitted to Computer Aided Geometric Design, 20xx
5. Farouki, R.T., Sakkalis, T.: Pythagorean hodographs. IBM J. Res. Develop. 34, 736–752 (1990)
6. Farouki, R.T.: Pythagorean hodograph curves. In: Farin, G., Hoschek, J., Kim, M.-S. (eds.) Handbook of Computer Aided Geometric Design, pp. 405–427. North-Holland, Amsterdam
7. Farouki, R.T.: Pythagorean-hodograph curves: Algebra and Geometry Inseparable. In: Geometry and Computing. Springer, Heidelberg (2008)
8. Gravesen, J., Jüttler, B., Šír, Z.: On rationally supported surfaces. Comput. Aided Geom. Design 5(4-5), 320–331 (2008)
9. Gravesen, J.: Surfaces parametrised by the normals. Computing 79, 175–183 (2007)
10. Groemer, H.: Geometric Applications of Fourier Series and Spherical Harmonics. Cambridge University Press, Cambridge (1996)
11. Gruber, P.M., Wills, J.M.: Handbook of convex geometry. North–Holland, Amsterdam (1993)
12. Jüttler, B.: Hermite interpolation by Pythagorean hodograph curves of degree seven. Math. Comp. 70, 1089–1111 (2001)
13. Kubota, K.K.: Pythagorean Triples in Unique Factorization Domains. Amer. Math. Monthly 79, 503–505 (1972)
14. Meek, D.S., Walton, D.J.: Geometric Hermite interpolation with Tschirnhausen cubics. Journal of Computational and Applied Mathematics 81, 299–309 (1997)
15. Moon, H.P., Farouki, R.T., Choi, H.I.: Construction and shape analysis of PH quintic Hermite interpolants. Comp. Aided Geom. Design 18, 93–115 (2001)
16. Sabin, M.: A Class of Surfaces Closed under Five Important Geometric Operations, Technical report no. VTO/MS/207, British aircraft corporation (1974), http://www.damtp.cam.ac.uk/user/na/people/Malcolm/vtoms/vtos.htm
17. Šír, Z., Jüttler, B.: Constructing acceleration continuous tool paths using pythagorean hodograph curves. Mech. Mach. Theory 40(11), 1258–1272 (2005)
18. Šír, Z., Gravesen, J., Jüttler, B.: Curves and surfaces represented by polynomial support functions. Theor. Comput. Sci. 392(1-3), 141–157 (2008)
19. Šír, Z., Gravesen, J., Jüttler, B.: Computing Minkowski sums via Support Function Representation. In: Chenin, P., Lyche, T., Schumaker, L. (eds.) Curve and Surface Design: Avignon 2006, pp. 244–253. Nashboro Press, Brentwood (2007)
20. Šír, Z., Bastl, B., Lávička, M.: Hermite interpolation by hypocycloids and epicycloids with rational offsets. Comput. Aided Geom. Design (2010), doi:10.1016/j.cagd.2010.02.001

Piecewise Tri-linear Contouring for Multi-material Volumes

Powei Feng[1], Tao Ju[2], and Joe Warren[1]

[1] Rice University
{pfeng,jwarren}@rice.edu
[2] Washington University in St. Louis
taoju@cse.wustl.edu

Abstract. The ability to model objects composed of multiple materials has become increasingly more demanded in scientific applications. The visualization of a discrete multi-material volume often suffers from voxelization of the boundary between materials. We propose a contouring method that can be efficiently implemented on the GPU to reduce the artifacts and jaggedness along the material boundaries. Our method extends naturally from the standard tri-linear contouring in a signed volume, and further provides sub-voxel accuracy for representing three or more materials.

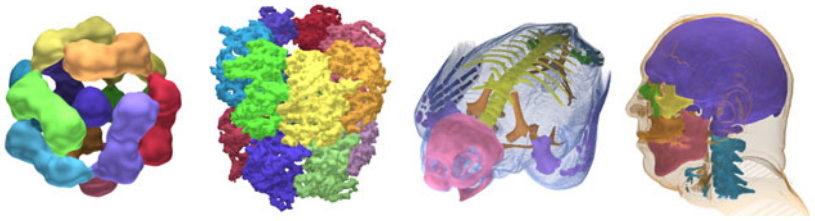
1 Introduction

Many scientific modeling applications require the ability to model objects composed of multiple materials. Probably the most common examples are found in bio-medicine, where researchers are often interested in the decomposition of a biological structure, obtained by imaging techniques like MRI or EM, into individual function units. Figure 1 shows two such examples, a human skull segmented into anatomical subdivisions, and a molecular complex decomposed into protein subunits. Modeling of these smaller units helps biologists and medical researchers to understand the function of the entire entity.

One of the simplest ways to represent multiple materials in a grid volume is to attach an integer material label to each grid point. While this approach is fairly simple to implement, its drawbacks are obvious. The discrete labeling leads to blocky, voxelized material boundaries that are hard to shade in a natural manner [7] (see Figures 2(a) and 2(c)).

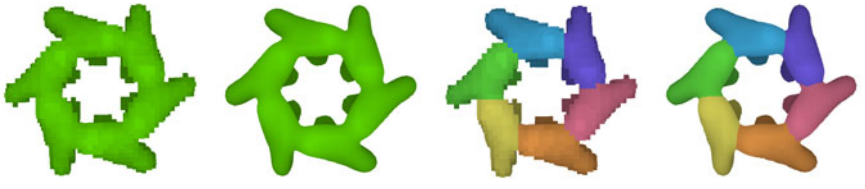
When only two materials (e.g., inside and outside) are present in the volume, a standard solution is implicit modeling [1]. In this approach, each grid point is associated with a positive or negative floating-point scalar, where the sign indicates whether the grid point lies inside or outside the object. These scalars can be considered as samples of a continuous function $f(x, y, z)$, and the boundary surface is defined as the set of all points where $f(x, y, z) = 0$. There are numerous contouring algorithms that can produce a polygonal approximation of

B. Mourrain, S. Schaefer, and G. Xu (Eds.): GMP 2010, LNCS 6130, pp. 43–56, 2010.



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

Fig. 1. Multi-material volumes representing structure of Hsp 26 (EMDB 1226) (a), subunits of the molecular chaperone GroEL (b), bones of a salamander (c), and the anatomical regions of a human head (d) visualized using tri-linear contours



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

Fig. 2. Comparison of the typical voxelized material boundaries (a,c) with the same materials visualized using our tri-linear contours (b,d). This example is the replicative helicase G40P molecular structure, before (a,b) and after (c,d).

this surface, such as Marching Cubes [11], Dual Contouring [9] and others [5,10]. Alternatively, the continuous surface can be directly rendered on GPU [18,3,12]. The key idea behind these approaches are that the signed grid can be stored as a 3D texture and that a single texture fetch can be used to evaluate $f(x, y, z)$ via tri-linear interpolation at an arbitrary point. In practice, this tri-linear boundary surface provides better normals (for shading) and better silhouettes than either polygonal contours or voxelized boundaries (see Figure 2(b)).

While the use of two signs to model two materials is simple and elegant, the idea of using three or more labels to represent a partition of space into multiple materials has received only limited attention. Most existing works in this direction focus on producing polygonal inter-material boundaries. For example, Dual Contouring [9] creates polygons from point and normal data stored on edges in the grid, and the works of [6,15] use a generalized Marching Cubes look-up table to polygonalize a multi-labeled cell.

A work on smooth boundaries among multiple materials that is similar to our own is by Stalling et al. [16]. In their approach, a tri-linear function $f^k(x, y, z)$ is defined for each material k, and the boundary surfaces are located where the values of two or more functions are identical and higher than the remaining functions. To define $f^k(x, y, z)$, each grid point is associated with an array of

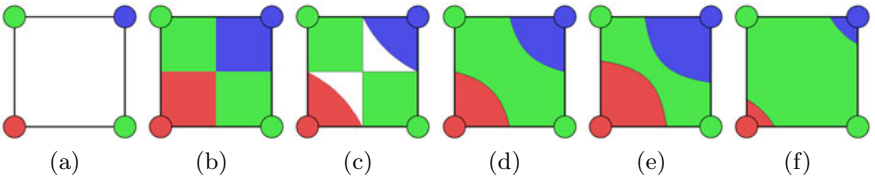


Fig. 3. Two-dimensional comparison of methods. (a) is a multi-material voxel with no intensity information. (b) is the naive approach of classifying by nearest neighbor. (c) Tiede et al. proposes a linear filter for classification [17], but it leaves points unclassified. (d) We propose a tri-linear representation that classifies all points within a voxel. (e) and (f) are examples of our approach that demonstrate the flexibility in representing contours.

scalars representing the "probabilities" that this grid point is classified as each material present in the volume. While this approach gives smooth material interfaces, the need to store multiple scalars per point not only increases the memory consumption, in comparison to the signed scalar representation of two materials, but also hampers fast GPU implementations as more texture fetches are needed. We build on their approach by providing a more compact representation that allows for fast GPU implementation.

Tiede et al. introduced a multi-material classification scheme for volume ray-casting [17]. Hadwiger et al. integrated this classification scheme into their hardware implementation of high-quality volume rendering [8]. The classification scheme described by Tiede et al. focuses on segments produced by thresholding. In the case where the threshold ranges of multiple materials overlap, Tiede et al.'s approach is to linearly interpolate the binary mask associated with the material. For each material A, space where the interpolated value (with respect to A's tri-linearly interpolated binary mask) is greater than 0.5 is classified as A. Although linear filtering resolves the overlap of threshold ranges, it also produces unclassified regions within a single voxel (see Figure 3(c)). In the case where the input is a segmented volume without intensity information, Tiede et al's approach would produce a classification that has ripple-like effect (see Figure 4(b)). Our method guarantees classification for all points within a voxel (see Figure 3(d)) and provides greater flexibility in sub-voxel classification (see Figure 3(e)), hence capable of representing smooth inter-material boundary (see Figure 4(c)) . Also note that both Tiede et al. and Hadwiger et al. tackled the problem from a visualization perspective, where they improved multi-material rendering for one particular visualization technique. Our approach is to present a geometric representation for multi-material volume that can be used for various visualization methods.

In this paper, we propose an alternative generalization of the idea of two-sign tri-linear contouring to that of multi-labeled tri-linear contouring with the goal of creating smooth multi-material boundary surfaces that can be efficiently rendered. The key difference between our generalization and that in [16] is that we only require a single scalar and a single integer label to be stored at each grid

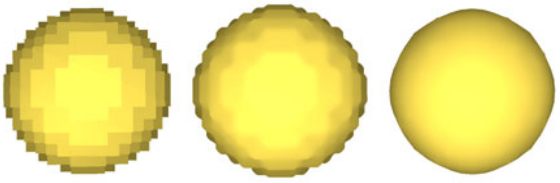


Fig. 4. Three-dimensional comparison of methods. (a) is the input of a sphere-like segment. (b) is rendered using Tiede et al.'s classification scheme [17]. Note that it produces a bumpy surface. (c) is our representation for the segment contour. Details for constructing (c) from the segment (a) is described in Section 4.

point. We show that the multi-material contours defined this way enjoys a number of properties, such as being piece-wise tri-linear and reproducing two-sign tri-linear contours where only two materials are present. The compact volume representation allows fast GPU-based rendering of the smooth contours. We demonstrate the use of tri-linear contouring in several examples of multi-labeled volumes.

Contributions. In the context of implicit modeling and multi-material modeling, our work makes several novel contributions:

- We introduce a generalization of the standard two-sign tri-linear contouring to multi-labeled volumes, and demonstrate the properties of the resulting contours.
- We present an efficient GPU implementation for rendering the tri-linear multi-material contours.
- We generalize the common set operations, union and intersection, from two-signed volume to a multi-material volume.
- We demonstrate the use of our multi-material representation and routines in several examples.

2 Multi-material Contouring

Conventionally, the tri-linear contour in a two-material grid is defined by signed scalars associated with the grid points. Our approach for multi-material contouring is to replace the signed scalar at each grid point with a *scalar* and a *material label*. In the two material case, our method would reproduce the standard tri-linear contours defined by signed scalars. In the case of three or more materials, the contouring method would generate piecewise tri-linear contours that form a continuous surface. Our method adds small overhead over the traditional signed volumes, and allows efficient hardware-accelerated rendering.

In the following, we first introduce the definition of contours in our volume representation. We next present a number of properties of such contours. We end this section by a discussion of means to evaluate the contours.

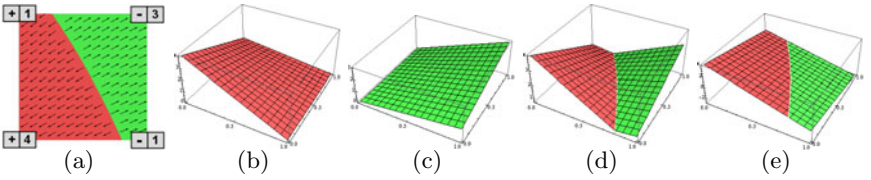


Fig. 5. Two material example: material classification in a 2D cell with +/− labels (the arrows denote the gradient) (a), the bi-linear function for each material label (b,c) and their maximum (d). (e) shows the bi-linear function defined by treating the two-labeled scalars at cell corners as signed scalars, and the function's zero contour.

2.1 Defining contours

To define the contour surfaces that partition the space into regions with different materials, we first consider the dual problem of *classifying* the material of an arbitrary point in space. Given a grid cell whose corners have associated non-negative scalars s_i and material labels m_i, the following method can be used to determine the material label of a point x inside the cell. Here, index i ranges from 0 to 7, representing the eight corners of a cell.

- For each distinct material label k present in the cell, construct a set of scalars t^k associated with the corners of the cell via the following rule:

$$t_i^k = s_i \quad \text{if } k = m_i$$
$$t_i^k = 0 \quad \text{otherwise}$$

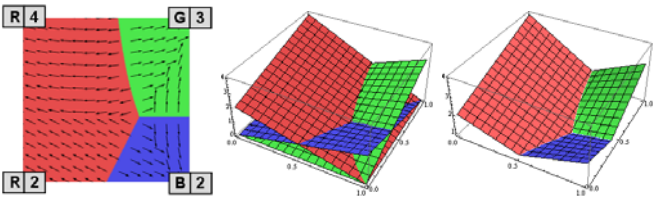
- Compute the values of the tri-linear interpolant $t^k(x)$ for each distinct label k. The trilinear coefficients are t_i^k for $i = 0, 1, \ldots 7$.
- Return the material label k for which $t^k(x)$ is maximum.

With this classification, the contour between two regions with material labels k and j is simply where tri-linear functions $t^k(x), t^j(x)$ both reach maximum. This contour is an iso-surface of the form:

$$t^k(x) = t^j(x) \geq t^m(x), \quad \forall m \neq k, j \tag{1}$$

Figure 5(a) illustrates the classification method and the resulting contours in a 2D cell. In this example, only two materials are present at the cell corners, which we label as + (red) and − (green). Figure 5(b) and 5(c) shows the two bi-linear functions $t^+(x)$ and $t^-(x)$, respectively. Figure 5(d) shows a plot of the maximum of these two functions, where the white curve indicates the contour.

Figure 6(a) shows another 2D example in which the four corner of the cell have three distinct materials, red, green and blue. Figure 6(b) show plots of the three bi-linear functions associated with the materials. Finally, Figure 6(c) shows a plot of the maximum of these functions and the associated partition of the cell into three distinct materials via three contours that meet at a common point.



(a) (b) (c)

Fig. 6. Three material example: material classification in a 2D cell with $R/G/B$ labels (the arrows denote the gradient) (a), the three bi-linear functions, one for each material label (b), and their maximum (c)

2.2 Characterization of the Contours

The multi-material contours produced by this method have several important properties.

Piecewise tri-linear and continuous surfaces. By Definition 1, the multi-material contours defined within each cell are piecewise contours of various tri-linear functions. The contours are also continuous across neighboring cells. This fact follows from the observation that two cells sharing a common grid point, edge or face have the same scalars and material labels on that common grid element. Since the restriction of the tri-linear functions used in defining our multi-material contour on each grid point, edge or face depend only the scalar and material labels on that grid element, the multi-material contours must agree across adjacent grid elements.

Reproducing tri-linear contours on signed grids. A key property of our contour definition is that it can exactly reproduce the contours defined by the standard tri-linear interpolation on a signed grid. Consider a cell whose corners are associated with signed scalars $\hat{s}_i$. The tri-linear interpolant $\hat{s}(x)$ defined by these signed scalars is either positive or negative, and the standard tri-linear contour is defined as the surface $\hat{s}(x) = 0$.

To reproduce this surface using our method, we construct the scalars and labels at the cell corners as follows. If $\hat{s}_i$ is positive at corner i, we let $s_i = \hat{s}_i$ and label m_i as $+$. Otherwise, we let $s_i = -\hat{s}_i$ and label m_i as $-$. Note that this coefficient set satisfies the relation

$$\hat{s}_i = t_i^+ - t_i^- \tag{2}$$

Hence the associated tri-linear functions $t^+(x)$ and $t^-(x)$ also satisfy

$$\hat{s}(x) = t^+(x) - t^-(x). \tag{3}$$

Therefore, the zero contour of the function $\hat{s}(x)$ coincides with the locations of x where $t^+(x)$ equals $t^-(x)$, which is the contour defined by our classification method.

Figure 5(e) plots the bi-linear function $\hat{s}(x)$ for the 2D cell configuration in Figure 5(a), treating each labeled scalar as a signed scalar. Observe that the zero contour of this bi-linear function is identical to the contour defined using our method as where the two bi-linear functions are identical (Figure 5(d)).

Gradient. One useful property from standard implicit modeling is that the gradient of the implicit function is normal to the contours of that function. In the multi-material case, a similar property holds. Given a contour formed by the iso-surface $t^k(x) = t^j(x)$, the gradient of the function $t^k(x) - t^j(x)$ is simply the normal to this surface. The key observation here is that the pair of material labels j and k change as the point x varies over the cell. For the three material case, $t^k(x)$ and $t^j(x)$ denote the largest and second largest tri-linear interpolant at x. This will produce the exact gradient field since we can view the local neighborhood of a point on the two-material boundary as being defined by the two dominant tri-linear functions. Note that points where more than two materials meet are degenerate with unknown gradients. Figure 5(a) show the gradient field for a two material cell while Figure 6(a) shows the gradient field for a three material cell.

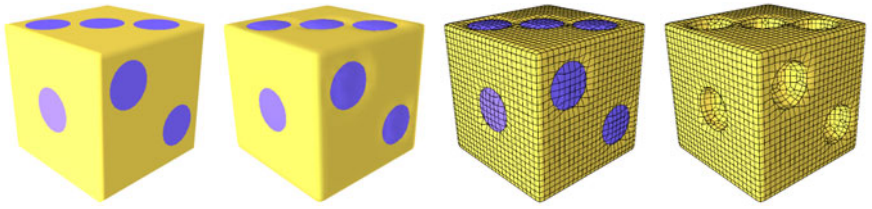
2.3 Evaluating Contours

We will discuss two ways to evaluate the multi-material contours, one utilizing the graphics hardware for direct surface rendering, and the other resorting to polygonization of the contours. Note that all examples in this paper are presented using the first approach.

Direct rendering. A key motivation of our multi-material representation is to utilize graphics hardware for efficient rendering of the contours. We use texture-slicing volume rendering as our algorithm for visualizing multi-material volumes, as done in standard implicit modeling on a signed grid [18,2]. Texture-slicing volume rendering approximates the ray-integral in traditional ray-casting volume rendering by rendering view perpendicular slices and compositing the slices using hardware blending.

The scalars and material labels, along with auxiliary data such as a density map, are stored as 3D textures. Coloring a single screen fragment involves a number of texture fetches to determine the density, color, and shade of the fragment in texture space, utilizing the underlying tri-linear interpolation capability of GPU. For our classification algorithm, we need to fetch an additional 8 scalar values and 8 integer labels as part of the fragment shader program. Texture fetches are typically expensive operations in shader programming. We note that, however, these 16 fetches can be reduced to 4 fetches by packing the values into the RGBA channels for a single texel. As we shall see in the Results section, our algorithm achieves interactive rendering rates, even for volumes with complex material composition like those in Figure 1.

Both the scalars and material labels are 8-bit textures, which allows up to 256 number of materials. With only 8-bits of precision for the scalars, we need



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

Fig. 7. The perspective view of a multi-material volume of size 33^3 rendered using GPU tri-linear contouring (a) and as polygonal contours generated by Dual Contouring (b), showing the grid structure (c). (d) depicts the mesh generated from Dual Contouring without the dots.

to ensure that the precision is not spent on non-essential portions of the representation. We note that the scalars are used for arbitration only on the border between different materials. This implies that the scalars only need to be accurate for cells that intersect the inter-material boundary.

Lastly, we use a single 1D transfer function that maps the voxel density to an opacity value. The color of each pixel is determined by the classification, where each material label is associated with a single color. Note that our method is a general classification scheme, and it can be extended to include multiple transfer functions (one for each material label) as described by Hadwiger et al. [8].

Polygonization. Besides GPU-based rendering, an alternative way to evaluate the contours is using polygonal methods such as Dual Contouring. Under Dual Contouring, we create a vertex within each cell that exhibits a label change, and then form a polygon for each grid edge that exhibits a material label change by connecting the vertices created within the cells sharing that edge.

The vertex in a cell should be located closest to the intersection of all the pairwise tri-linear surfaces within the cell. More formally, let M be the set of materials in the cell and let x be a point inside the cell. Consider the function

$$E(x) = \sum_{j \neq k \in M} (t^k(x) - t^j(x))^2 \tag{4}$$

The minimum of this function describes a point that is close to the intersection of all the surfaces that satisfy $t^k(x) = t^j(x)$. This is a non-linear optimization problem that can be costly to compute. We approximate the solution using an QEF-based approach that was described by Schaefer et al [14]. Using this method, we first locate the intersections, p_i, of the surfaces along the cell edges. These intersections and the gradient directions (as described in Section 2.2), n_i, at these points describe a set of planes per cell. We then find a point that minimizes

$$E'(x) = \sum_i (n_i \cdot (x - p_i))^2 \tag{5}$$

This minimization gives an approximation of Eq. 4 that is reasonable for our purpose. Note that this is an outline for computing the contour point. Please refer to the work of Schaefer et al. for more implementation details [14]. An example of the Dual Contouring polygonization is shown in Figure 7.

3 Set Operations on Multi-material Contours

One of the primary attractions of implicit modeling is the ease with which it can model Boolean operations from constructive solid geometry [13]. To enable interactive construction of multi-material models in our representation, we next develop analogs of the set operations Union and Intersection for multi-material contours. These operations can be applied in several context with regards to interactive segmentation. Due to space limitation, we direct the reader to [4] for more detail.

In the signed (two-material) case, the typical convention is to represent a solid as the set of solutions to the inequality $f(x, y, z) < 0$. Now, given two solids $f(x, y, z) < 0$ and $g(x, y, z) < 0$, the union of these two solids is simply the set $\min(f(x, y, z), g(x, y, z)) < 0$ while the intersection of two solids is the set $\max(f(x, y, z), g(x, y, z)) < 0$.

If the functions f and g are represented by signed grids, a standard technique for approximating their union or intersection is to take the min or max of their associated sign grids. Our goal is to develop equivalent rules for the two-material case that generalize to the multi-material case in a natural manner.

Our approach is as follows; consider two materials A and $\neg A$ (not A). A can be interpreted as being the inside of a solid (i.e; negative in the implicit model) and $\neg A$ can be interpreted as being the outside of a solid (i.e; positive in the implicit model.). Given a multi-material map consisting of only these two materials, we can attempt to construct rules for computing new non-negative scalars and material labels on the grid that reproduce the operations Union and Intersection.

In particular, given a grid point with two associated pairs (s_1, k_1) and (s_2, k_2) (where both the s_i are non-negative), our goal is to compute a scalar/label pair (s, k) for the union of the material S. This new pair can be compute using the case look-up given in Table 1.

Note that the rule for computing k is straightforward. For Union, the new material label is A if and only if at least one of the material labels is A. For Intersection, the new material label is A if and only if both of the material labels are A. The rule for computing the new scalar s is only slightly more involved. The key is to convert back to the signed case and then return the result of taking the minimum of the converted scalars. For example, if both material labels are A, we take the negative of both scalars s_1 and s_2, computed their min and then negate the result. These three operations are simply the equivalent of taking the max of the original scalars. In particular, if both s_1 and s_2 are non-negative,

$$\max(s_1, s_2) = -\min(-s_1, -s_2) \tag{6}$$

Table 1. Rules for performing Intersection and Union operations. We consider the pairs (s_1, k_1) and (s_2, k_2) as the input. The output of Union and Intersection is denoted as pair (s, k).

Union			
k_1	k_2	k	s
A	A	A	$\max(s_1, s_2)$
A	$\neg A$	A	s_1
$\neg A$	A	A	s_2
$\neg A$	$\neg A$	$\neg A$	$\min(s_1, s_2)$

Intersection			
k_1	k_2	k	s
A	A	A	$\min(s_1, s_2)$
A	$\neg A$	$\neg A$	s_2
$\neg A$	A	$\neg A$	s_1
$\neg A$	$\neg A$	$\neg A$	$\max(s_1, s_2)$

Similar argument can be used to derive the given formulas for the remaining cases.

Another interpretation of these operations on the scalars s_1 and s_2 is to view these numbers as estimate of the distance from the grid point to the boundary of the region A. In the case of Union, the rule is that if both grid points lie in A, a good estimate of the distance from the grid point to the boundary of the union is the maximum of these two distances. Similar arguments again apply in the other cases.

3.1 Operations for Three or More Materials

Given the method for union and intersection defined above, the generalization of these operations to the three or more material case is relatively easy. We suggest two operations analogous to Union and Intersection for the multi-material case. The first operation *Overwrite* takes a multi-material map and a two-material map (with material A and $\neg A$) and performs the multi-material analog of Union. In particular, it treats the material in the first multi-material map as being either A or $\neg A$ and applies the two material rules for Union described above. The result of an Overwrite operation is that the material A in the second map overwritten onto any existing materials in the first map. The resulting map contains the union of the materials A in both maps.

The second operation *Restrict* again takes as input a multi-material map and a two-material map (with materials A and $\neg A$). In this case, the Restrict operation modifies the second map to return the intersection of the first map (viewed as materials A and $\neg A$) and the second map. Essentially, the second map is restricted to only those regions where the material A exists in the first map.

4 Implementation

We first consider visualizing a given multi-material volume where grid points are attached with only integer labels. Direct visualization of the voxelized material boundaries gives a blocky look (see Figure 8(a)). To create piece-wise smooth tri-linear contours, our method needs additional scalars besides the material labels.

Fig. 8. Implementation details. Viewing a multi-material volume with only integer labels: direct rendering of the voxelized boundaries (a), tri-linear contouring using uniform assignment of scalars (b) and using Guassian-filtered scalars (c). Viewing nested iso-surfaces in a density volume using transfer function (d) as a multi-material volume (e).

A simple approach is to assign $s_i = 1$ uniformly for every grid point. Although locally smooth, such contours "wobble" a lot and do not represent a globally smooth surface (see Figure 8(b)).

To alleviate local surface undulations, we use an improved scalar assignment based on blurring. Recall in our classification method that t_i^k is a scalar at grid point i for material label k, determined by the scalar s_i and material label m_i at that point. First, we assign $s_i = 1$ for every grid point, and hence t_i^k would be binary (0 or 1). Next, for each label k, we compute a blurred value, $\bar{t}_i^k$, by applying a truncated $3 \times 3 \times 3$ gaussian filter to t_g^k at neighboring grid points g as mentioned in [7]. Finally, we assign s_i to be

$$s_i^{new} = \bar{t}_i^k - \bar{t}_i^j \tag{7}$$

where $\bar{t}_i^k$ and $\bar{t}_i^j$ are the largest and second largest values among the blurred scalars for all material labels. Accordingly, the material label is set to be $m_i = k$.

This heuristic is guided by the same intuition given in the *gradient* discussion of Section 2.2. In the two-material case, the contour resulted from the blurred assignment reproduces the standard tri-linear contour after blurring a signed scalar grid. In the case of three or more materials, the contour is formed by the top two dominant tri-linear interpolants ($\bar{t}^k(x)$ and $\bar{t}^j(x)$), and the assignment in Eq. 7 results in an approximation of that contour in the cell. Figure 8(c) shows the result of our heuristic. Note that the resulting contours improve over those of uniform assignment. More in-depth implementation details, including segmentation generation, can be found in [4].

5 Results

Our method has been implemented and tested on an Intel Xeon 5150 machine with 2 dual-core CPUs running at 2.66GHz. We use an nVidia GTX280 graphics card with 1GB of video RAM. The shaders were written in GLSL. We use OpenMP to enable multi-core processing for easily parallelizable portions of the code.

Table 2. The rendering speed for a selection of datasets. All results were measured in frames per second (fps).

model	size	binary (fps)	tri-linear (fps)
Hsp26	$128 \times 128 \times 128$	69	23
GroEL	$240 \times 240 \times 240$	44	18
head	$128 \times 256 \times 256$	46	13
engine	$256 \times 256 \times 256$	40	17
sea turtle	$256 \times 256 \times 397$	29	11

We gather rendering times for a selection of our test cases. The models are displayed in Figure 1. The running time is largely dependent on the maximum dimension of the volume as we use that to determine the number of quads to use as proxies for rendering. The rendering screen is 512×512 pixels. In addition, shading only occurs for visible fragments, and intensive computation only occurs for inhomogeneous cells. The variability of computing load in the fragment shader accounts for the differences in rendering times for volumes such as the "head" and "engine" datasets.

It is also worthy to note that the rendering speed is also dependent on the pixel estate required to display each volume. In other words, the smaller the volume appears on the screen, regardless of input size, the faster the rendering will be, which is an expected result. Our results are taken from the slowest

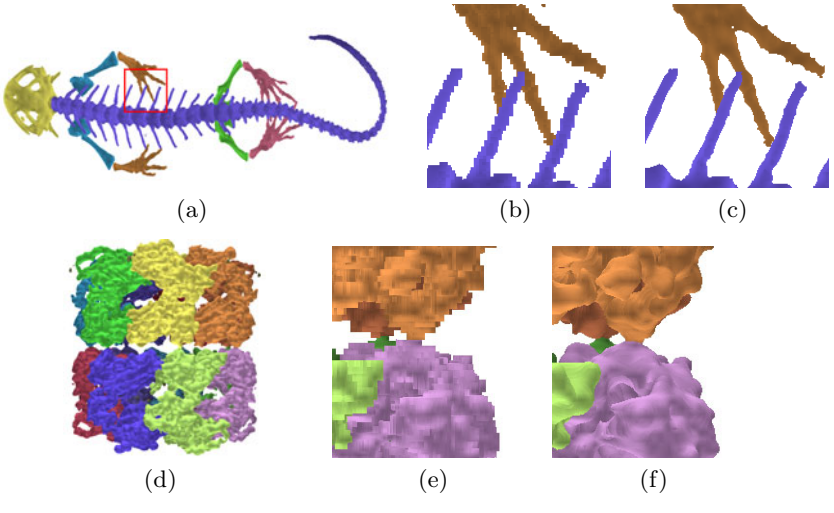


Fig. 9. A close up example of rendering using binary classification (b,e) and our piecewise tri-linear representation (c,f). The top example is the CT scan of Tarich Torosa (salamander) provided by the Digitial Morphology Library. The bottom example is the GroEL structure, which was obtained from Electron Microscopy Data Bank (EMDB) under entry number 5002.

rendering time for each of the test sets. Our method maintains a reasonable frame-rate when rendering the tri-linear contours. Lastly, Figures 9 illustrates the rendering improvement of the tri-linear contours versus binary classification.

6 Conclusions and Future Work

We present a contouring technique for multi-material volumes, aimed at providing both a smoother inter-material boundary than in typical voxelized approaches and efficient GPU-based rendering. The technique is a generalization of the standard tri-linear contouring in a signed volume, and only requires a small overhead to offer sub-voxel representations of multiple materials. Our technique can be used to improve the visualization of an existing multi-labeled volume and in interactive painting of a density volume [4].

For our future work, we will experiment with using higher level interpolants such as B-splines for classification. This will give us greater flexibility and accuracy in defining the boundary between materials.

Acknowledgement

The work of Powei Feng is supported in part by the NIH grant R01GM079429. The work of Tao Ju is supported in part by NSF grants IIS-0705538, IIS-0846072, CCF-0702662 and DBI-0743691. We thank Matt Baker, Wah Chiu, Lu Liu, and Travis McPhail for helpful discussions. We thank Timothy Rowe and Jennifer Maisano of Digital Morphology Library for providing the salamander and sea turtle datasets. We thank The Volume Library, volvis.org, Electron Microscopy Data Bank, and the Protein Data Bank for providing the rest of the datasets.

References

1. Bloomenthal, J.: Implicit surfaces. Computer Aided Geometric Design 5, 341–355 (1997)
2. Cullip, T.J., Neumann, U.: Accelerating volume reconstruction with 3d texture hardware. Tech. rep. Chapel Hill, NC (1994)
3. Engel, K., Kraus, M., Ertl, T.: High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In: HWWS 2001: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware, pp. 9–16. ACM, New York (2001)
4. Feng, P.: Segmentation and Visualization of Volume Maps. Master's thesis, Rice University, Texas, United States (2010)
5. Frisken, S.F., Perry, R.N., Rockwood, A.P., Jones, T.R.: Adaptively sampled distance fields: a general representation of shape for computer graphics. In: SIGGRAPH 2000: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, pp. 249–254. ACM Press/Addison-Wesley Publishing Co., New York (2000)

6. Fujimori, T., Suzuki, H.: Surface extraction from multi-material ct data. In: Ninth International Conference on Computer Aided Design and Computer Graphics, pp. 319–324 (December 2005)
7. Gibson, S.F.F.: Using distance maps for accurate surface representation in sampled volumes. In: IEEE Symposium on Volume Visualization and Graphics, vol. 0, pp. 23–30 (1998)
8. Hadwiger, M., Berger, C., Hauser, H.: High-quality two-level volume rendering of segmented data sets on consumer graphics hardware. In: VIS 2003: Proceedings of the 14th IEEE Visualization 2003 (VIS 2003), p. 40. IEEE Computer Society Press, Washington (2003)
9. Ju, T., Losasso, F., Schaefer, S., Warren, J.: Dual contouring of hermite data. In: SIGGRAPH 2002: Proceedings of the 29th annual conference on Computer graphics and interactive techniques, pp. 339–346. ACM, New York (2002)
10. Kobbelt, L.P., Botsch, M., Schwanecke, U., Seidel, H.P.: Feature sensitive surface extraction from volume data. In: SIGGRAPH 2001: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pp. 57–66. ACM, New York (2001)
11. Lorensen, W.E., Cline, H.E.: Marching cubes: A high resolution 3d surface construction algorithm. In: SIGGRAPH 1987: Proceedings of the 14th annual conference on Computer graphics and interactive techniques, pp. 163–169. ACM, New York (1987)
12. Rezk-Salama, C., Engel, K., Bauer, M., Greiner, G., Ertl, T.: Interactive volume on standard pc graphics hardware using multi-textures and multi-stage rasterization. In: HWWS 2000: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware, pp. 109–118. ACM, New York (2000)
13. Ricci, A.: A Constructive Geometry for Computer Graphics. The Computer Journal 16(2), 157–160 (1973)
14. Schaefer, S., Warren, J.: Dual contouring: "the secret sauce", rice University, Department of Computer Science Technical Report (2003)
15. Shammaa, M.H., Suzuki, H., Ohtake, Y.: Extraction of isosurfaces from multi-material ct volumetric data of mechanical parts. In: SPM 2008: Proceedings of the 2008 ACM symposium on Solid and physical modeling, pp. 213–220. ACM, New York (2008)
16. Stalling, D., Zckler, M., Hege, H.C.: Interactive segmentation of 3d medical images with subvoxel accuracy. In: Proc. CAR 1998 Computer Assisted Radiology and Surgery, pp. 137–142 (1998)
17. Tiede, U., Schiemann, T., Höhne, K.H.: High quality rendering of attributed volume data. In: VIS 1998: Proceedings of the conference on Visualization 1998, pp. 255–262. IEEE Computer Society Press, Los Alamitos (1998)
18. Wilson, O., VanGelder, A., Wilhelms, J.: Direct volume rendering via 3d textures. Tech. rep., Santa Cruz, CA, USA (1994)

An Efficient Algorithm for the Sign Condition Problem in the Semi-algebraic Context

Rafael Grimson[1,2]

[1] Dept. of Mathematics, University of Buenos Aires
rgrimson@dm.uba.ar
[2] Dept. of Computer Science, Hasselt University

Abstract. We study algebraic complexity of the sign condition problem for any given family of polynomials. Essentially, the problem consists in determining the sign condition satisfied by a fixed family of polynomials at a query point, performing as little arithmetic operations as possible. After defining precisely the sign condition and the point location problems, we introduce a method called *the dialytic method* to solve the first problem efficiently. This method involves a *linearization* of the original polynomials and provides the best known algorithm to solve the sign condition problem. Moreover, we prove a lower bound showing that the dialytic method is almost optimal. Finally, we extend our method to the point location problem.

The dialytic method solves (non-uniformly) the sign condition problem for a family of s polynomials in $\mathbf{R}[X_1, ..., X_n]$ given by an arithmetic circuit $\Gamma_{\mathcal{F}}$ of non-scalar complexity L performing $\mathcal{O}((L+n)^5 \log(s))$ arithmetic operations.

If the polynomials are given in dense representation and d is a bound for their degrees, the complexity of our method is $\mathcal{O}(d^{5n} log(s))$. Comparable bounds are obtained for the point location problem.

1 Introduction

Given a partition $\mathcal{S}$ of $\mathbf{R}^n$ into disjoint regions, the *point-location problem* for the partition $\mathcal{S}$ asks to determine the region containing a query point. Point location is a basic problem in computational geometry and has inspired several data structures (see [Sno04]). It has applications in different domains, including geographic information systems (GIS) and robot motion planning. We give now a precise definition of this problem.

Definition 1. *An algorithm taking as input a point in $\mathbf{R}^n$ and with a finite set of possible outputs $\{O_1, ..., O_k\}$ solves the point location problem for a given partition $\mathcal{S}$ of $\mathbf{R}^n$ if it satisfies the following condition:*

for any pair of points $x, y \in \mathbf{R}^n$, the algorithm returns the same output on both inputs x and y, if and only if x and y belong to the same element of the partition $\mathcal{S}$ of $\mathbf{R}^n$.

B. Mourrain, S. Schaefer, and G. Xu (Eds.): GMP 2010, LNCS 6130, pp. 57–76, 2010.

The output of a point location algorithm can be seen as a *label* identifying the region containing the query point. The regions in the given partition may have very complex descriptions. Using labels instead of these descriptions, we obtain algorithms whose query time is independent of their size. Using the terminology from database theory, we are measuring the point location search time and not its report time.

Definition 2. *Let $\mathcal{P} \subset \mathbf{R}[X_1, ..., X_n]$ be a finite family of polynomials. The realizations of the $\mathcal{P}$-sign condition form a partition of $\mathbf{R}^n$ denoted by $\mathcal{S}(\mathcal{P})$. The elements of $\mathcal{S}(\mathcal{P})$ are not necessarily connected subsets of $\mathbf{R}^n$. We define the* arrangement induced by *$\mathcal{P}$ as the partition of $\mathbf{R}^n$ consisting of the connected components of the realization of the sign conditions of the family $\mathcal{P}$ and denote it by $\mathcal{A}(\mathcal{P})$. The elements of $\mathcal{A}(\mathcal{P})$, are called the* faces *of the arrangement induced by $\mathcal{P}$.*

We first study the point location problem for the partition $\mathcal{S}(\mathcal{P})$, called *the sign condition problem for the family $\mathcal{P}$*. The point location problem for the partition $\mathcal{A}(\mathcal{P})$ is called the *point location problem for the family $\mathcal{P}$* and will be studied afterwards.

In Section 2, we introduce the computational models used in this article and the different representations of polynomials that we consider: circuit, dense arithmetic and dense bit representations. In Section 3 we introduce the *dialytic method*; it solves the sign condition problem for a given family of polynomials in any of the mentioned representations (see Theorem 2 and Corollaries 1 and 2). In Section 4 we present sharp lower bounds for this problem. Finally, in Section 5, we extend our method to the point location problem.

1.1 Basic Observations

The simplest instance of point location is list searching. Given different points $x_1, x_2, ..., x_s \in \mathbf{R}$, consider indices $1 \leq i_1, ..., i_s \leq s$ such that $x_{i_1} < ... < x_{i_s}$. Then, a partition of $\mathbf{R}$ into disjoint regions is determined by these points and the intervals $(-\infty, x_{i_1}), (x_{i_1}, x_{i_2}), ..., (x_{i_{s-1}}, x_{i_s}), (x_{i_s}, +\infty)$. The list searching problem already illustrates several aspects of the general point location problem. On the one hand, without any preprocessing, the point location query for this partition of $\mathbf{R}$ can be answered in time $\mathcal{O}(s)$ performing a linear search. On the other hand, if we order the points in a preprocessing stage (using $\mathcal{O}(s \log(s))$ operations), the query can be answered performing a binary search involving only $\mathcal{O}(\log(s))$ operations. In what follows, we generalize this second method to higher dimensions and higher degrees.

The space $\mathbf{R}^n$ can be divided in 2^n regions by n hyperplanes. If we consider $s > n$ hyperplanes in $\mathbf{R}^n$, we will no longer obtain 2^s regions determined. Some *implications* appear; its associated system of equations is overdetermined. It is easy to see this in the plane: two lines divide the plane in four different regions, but no three lines divide the plane in eight regions. Not all syntactically possible sign conditions are simultaneously geometrically realizable by any family of hyperplanes.

An analogous phenomenon can be observed for algebraic hypersurfaces of higher degree. In 1968, Warren [War68] proved that the number of connected components of the realizations of strict sign conditions of a family of s polynomials in n variables of degree at most d, is bounded by $(4esd/n)^n$, where e is the base of the natural logarithm (see also [Mil64, Gri88, HRS90b, JS00, LB01, BPR10]).

For a fixed n, the number of syntactically definable sign conditions, 3^s, grows exponentially with s, while the number of simultaneously geometrically realizable sign conditions grows only polynomially in s and d. Moreover, the number of faces of the induced arrangement is also polynomial in s and d, for any fixed n. The best bound known today [BPR10] for the number of faces of $\mathcal{A}(\mathcal{P})$ is

$$\frac{(2d)^n}{n!}s^n + \mathcal{O}(s^{n-1}).$$

Observing this bound, it is natural to try to design an algorithm that solves the point location problem performing a number of arithmetic operations that grows logarithmically in s, the number of polynomials in the family $\mathcal{P}$.

1.2 Related Work

Linear Case. Let $\mathcal{P} \subset \mathbf{R}[X_1, ..., X_n]$ be a family of s linear polynomials. We remark that, since the non-empty realizations of $\mathcal{P}$-sign conditions are convex sets, the sign condition and point location problems for the family $\mathcal{P}$ coincide.

Dobkin and Lipton [DL76] were the first to present an algorithm solving the point location problem, in this context, whose query time is logarithmic in the numbers s of polynomials; the size of the associated data structure is $\mathcal{O}(s^{2^n-2})$. Clarkson [Cla87] improved the space complexity to $\mathcal{O}(s^{n+\varepsilon})$; in both cases the query time is exponential in n. Meyer auf der Heide [MadH84] solved a particular instance of this problem (he considered hyperplanes with integer coefficients only), that allowed him to derive the existence of a non-uniform polynomial-time solution to the Knapsack Problem (see also [MadH88]). Finally, in 1993, Meiser [Mei93] gave a solution with running time $\mathcal{O}(n^5 \log(s))$ and space bound $\mathcal{O}(s^{n+\varepsilon})$, for arbitrary $\varepsilon > 0$. The preprocessing is done in expected time $\mathcal{O}(s^{n+1+\varepsilon})$, for arbitrary $\varepsilon > 0$. This last algorithm allowed Meiser to derive a strongly polynomial non-uniform algorithm for the NP-complete Knapsack problem (see also Chapter 3 in [BCS97]). After the next paragraph, we give a brief description of Meiser's algorithm.

Polynomial case. Chazelle and Sharir [CS90] (see also [CEGS91]) proposed an algorithm, based on Collins' Cylindrical Algebraic Decomposition [Col75], for the general algebraic point location problem in the traditional unit-cost RAM model. The complexity of their method is logarithmic in the number of polynomials, but in the complexity analysis they ignore the dependency of their method on the degree of the polynomials and on the dimension of the ambient space. This is a usual practice in computational geometry, where the degree of the polynomials and the dimension of the ambient space are assumed to be bounded by a constant.

Grigoriev [Gri00] bounded the branching (or topological) complexity of the sign condition problem from above by the logarithm of the number of faces of the arrangement $\mathcal{A}(\mathcal{P})$. Nevertheless, the algebraic complexity of Grigoriev's method depends linearly on s. See also [Koi00] for further details and for its relation with the P = NP question over the reals.

Before presenting our work, we briefly summarize Meiser's algorithm for the linear case.

Meiser's algorithm. For future reference, we state Meiser's result [Mei93] precisely in our model (see Section 2 for some details on the model).

Theorem 1. *Given a family, $\mathcal{P} \subset \mathbf{R}[X_1, ..., X_n]$, containing s linear polynomials, there exists an algebraic computation tree $\Gamma_{\mathcal{P}}$ that solves the sign condition problem for the family $\mathcal{P}$ in time $\mathcal{O}(n^5 \log(s))$.*

The size of the tree $\Gamma_{\mathcal{P}}$ is bounded by $\mathcal{O}(s^{n+\varepsilon})$ and it can be constructed in expected time $\mathcal{O}(s^{n+1+\varepsilon})$, for arbitrary $\varepsilon > 0$. □

Meiser's original algorithm uses the *trie* data structure. The conversion of this algorithm to the context of algebraic computation trees is straight-forward. We observe that the upper bound stated by Meiser for his algorithm is not optimal. He claims an $\mathcal{O}(n^5 \log(s))$ bound, but more precisely it is $n^4 \log(n)^{\mathcal{O}(1)} \log(s)$. Meiser's method behaves well also in the bit model; see his article for further details.

Roughly, in a first *step* the algorithm evaluates at the query point all the polynomials in a subset $\mathcal{R}$ of $\mathcal{P}$ and determines the *degenerated simplex* (see [Mei93]) in a triangulation $\triangle\mathcal{A}(\mathcal{R})$ of $\mathcal{A}(\mathcal{R})$ containing the query point. The set $\mathcal{R}$ and the triangulation $\triangle\mathcal{A}(\mathcal{R})$ are precomputed and have the following key properties:

- The cardinality of $\mathcal{R}$ is bounded by a polynomial in n,
- the degenerated simplex in the triangulation $\triangle\mathcal{A}(\mathcal{R})$ containing the query point can be determined in polynomial time in n, and
- only a constant fraction ε, $0 < \varepsilon < 1$ of the polynomials in $\mathcal{P}$ change their sign in each degenerated simplex in $\triangle\mathcal{A}(\mathcal{R})$.

In this way, after a logarithmic number of *steps* ($\log(s)$), the problem is reduced to a number of equations whose number depends on n but not on s. Then, the sign condition is determined by direct evaluation.

We remark that Meiser's algorithm is completely linear, *i.e.*, it does not perform any non-scalar multiplication.

2 Computational Models and Representations of Polynomials

Our algorithms are represented by algebraic computation trees over the real numbers (see [BCS97]; *cf.* [Str81], [BO83] and [Lic90]). We measure the number

of arithmetic operations performed by an algorithm and call it its *algebraic complexity*.

In some cases, we are also interested in the *bit* or *binary complexity* of our algorithms. To measure this within our computational model, we restrict the arithmetic operations performed by our algorithms to integer numbers. Rational and algebraic numbers are represented by tuples of integers and we measure, besides the number of arithmetic operations, the bitsize of the integers involved in these operations. For the sake of definiteness, we assume that real algebraic points in $\mathbf{R}^n$ are represented using a Triangular Thom encoding (see [BPR06]).

In this sense, algorithms (like that of Khachiyan [Kha79] for linear programming) that belong to the bit model but are not based on arithmetic operations, are out of the scope of our model.

Roughly, given a finite family of polynomials we construct, in a preprocessing stage, a data structure. Then, using this data structure, we answer some queries about the original family efficiently. Within the model of algebraic computation trees, the data structure is the tree itself.

The performance of a data structure is measured by the time spent in answering a query (called the *query time*), the time needed to construct the data structure (called the *preprocessing time*) and the *size* of the data structure. Since the data structure is constructed only once, its query time and size are more important than its preprocessing time. If a data structure supports insertion and deletion operations, the *update time* is also relevant, but we shall not consider this situation.

The complexity of some queries depend on the output size—consider, for instance, the sign condition query. We divide the query time in two parts: the *search time* and the *reporting time*. In some applications, it is important to distinguish different answers but not to write them down explicitly; in these cases, the search time plays a fundamental role.

2.1 Computational Models

We introduce the computational models considered in this article along with their associated complexity measures.

We start defining the *algebraic decision tree* model, that is a simple model used to prove lower complexity bounds (see Exercises 3.15 and 11.4 in [BCS97]). We then introduce the *algebraic computation tree* model.

A *tree* is a finite set T of nodes such that

- there is one specially designated node called the *root* of the tree;
- the remaining nodes are partitioned into $m \geq 0$ disjoint nonempty sets $T_1, ..., T_m$, and each of these sets is in a tree. These trees are call the *subtrees* of the root.

The number m of subtrees of a node $v \in T$ is called the *outdegree* of the node. The root of a tree is called the *parent* of the roots of its subtrees. The *predecessor* relation is defined as the transitive closure of the *parent* relation. Hence, a node $v_1 \in T$ is called a *predecessor* of a node $v_2 \in T$ if v_1 is the root of a subtree of T

containing v_2. A *ternary tree* is a tree in which each internal node has outdegree one, two or three.

Algebraic Decision Trees over the Reals. Let n be a positive integer. An *algebraic decision tree over the reals* is a ternary tree together with a function that assigns to each of its inner nodes v a polynomial F_v in $\mathbf{R}[X_1, ..., X_n]$, and to each of its leaves a label (for instance, these labels could be "accept" or "reject").

The semantics of algebraic decision trees is defined as follows. To any *input* $x \in \mathbf{R}^n$, we assign a unique path in the tree from the root to a leaf by continuing with the left son of a node v if $F_v(x) < 0$, with the middle son if $F_v(x) = 0$, and with the right son if $F_v(x) > 0$. The *output* of the algebraic decision tree is the label of the leaf where the path ends.

The number of steps of an algebraic decision tree is defined as the depth of the underlying tree.

Algebraic decision trees are used to give lower bounds for the *branching* (also called *topological*) complexity of semi-algebraic problems.

Algebraic Computation Trees over the Reals. We now briefly describe the notion of *algebraic computation trees* used in this article. Our definitions are based on the formulation of [BCS97] (see also [Str81], [BO83] and [Lic90]).

Analogous to Strassen [Str81], we define computation trees as consisting of *a subjacent tree*, an *instruction function* and an *inference partition* of the leaves.

Syntax of computation trees.

Definition 3. *Let $\mathcal{T}$ be a finite tree with four types of nodes:* assignment nodes *(outdegree 1),* arithmetic nodes *(outdegree 1),* test nodes *(outdegree 3) and* leaf nodes *(outdegree 0). An* algebraic computation tree *with variables $X_1, ..., X_n$ is a such a tree $\mathcal{T}$ together with a function (the* instruction function*) that associates*

- *to each assignment node v, a real constant or a variable;*
- *to each arithmetic node v, an arithmetic operation $\circ_v \in \{+, -, \times, /\}$ and two predecessor nodes, $p_1(v)$ and $p_2(v)$, of v in $\mathcal{T}$;*
- *to each test node v, two predecessor nodes, $p_1(v)$ and $p_2(v)$, of v in $\mathcal{T}$;*
- *to each leaf node l, a label.*

Semantics of computation trees. Let $\mathcal{T}$ be an algebraic computation tree with variables $X_1, ..., X_n$. We associate to each internal node $v \in \mathcal{T}$, a rational function $comp_v \in \mathbf{R}[X_1, ..., X_n]$, as follows:

- for an assignment node v, with an associated real constant c, we define $comp_v := c$;
- for an assignment node v, with an associated variable X_i, we define $comp_v := X_i$;
- for an arithmetic node v, we define $comp_v := comp_{p_1(v)} \circ_v comp_{p_2(v)}$;
- for a test node v, we define $comp_v := comp_{p_1(v)} - comp_{p_2(v)}$.

If v is an internal node of $\mathcal{T}$, we say that it *computes* the rational function $comp_v$. For any $x = (x_1, ..., x_n) \in \mathbf{R}^n$ we associate to v the real value $comp_v(x, u)$ if $comp_v$ is defined on x. Otherwise, we say that $comp_v$ is undefined on x.

Let v be an arithmetic node that computes the product of two preceding nodes. If one of these two preceding nodes computes a constant from $\mathbf{R}$, we say that v performs a *scalar* multiplication Otherwise, we say that it performs a *non-scalar* multiplication.

Definition 4 (Computation path, output). *The* computation path *followed in an algebraic tree $\mathcal{T}$ on input $x = (x_1, ..., x_n) \in \mathbf{R}^n$, is the unique path in $\mathcal{T}$ that satisfies the following properties:*

- *the path starts at the root of $\mathcal{T}$;*
- *the successor of an assignment node v in this path is its unique immediate successor in the tree $\mathcal{T}$;*
- *an arithmetic node v has a successor in this path only if $comp_v(x)$ is defined: in this case the successor of v in the path is its unique successor in $\mathcal{T}$; otherwise, the computation path ends in v;*
- *the successor of a test node v in this path corresponds to the first, second or third immediate successor of the node v in $\mathcal{T}$, according to whether $comp_v(x)$ is less, equal or greater than zero, respectively*—i.e., *according to whether $comp_{p_1(v)}(x)$ is lower, equal or greater than $comp_{p_2(v)}(x)$;*
- *leaf nodes have no successor in a computation path.*

We denote by $\mathcal{T}(x)$ the last node reached by this computation path. If $\mathcal{T}(x)$ is a leaf, say l, of $\mathcal{T}$, then its label is the result of the execution *of $\mathcal{T}$ on input x and we say that the algebraic tree $\mathcal{T}$ is executable on x.*

Pragmatics of computation trees. Algebraic computation trees provide an excellent model to prove lower bounds for the algebraic complexity of some problems. Lower bounds are usually proved for the branching complexity or the non-scalar complexity of any algebraic computation tree solving a fixed problem.

Let $\mathcal{T}$ be an algebraic computation tree. Its *total complexity* is defined as the length of the longest path in the tree. Its *branching complexity* is defined as the maximum number of branching (test) nodes in a single path of the tree. Its *non-scalar complexity* is defined as the maximum number of non-scalar multiplication nodes in a single path of the tree. Finally, its *multiplicative-branching complexity* is defined as the maximum number of non-scalar multiplication nodes and branching nodes in a single path of the tree.

In order to prove lower bounds, the main drawback of this model is that it does not include a notion of *uniformity*. On the other hand, this allows to describe non-uniform algorithms in this model. For instance, Meyer auf der Heide [MadH84] described a non-uniform polynomial time solution to the NP-complete Knapsack problem using a similar (linear) model.

Arithmetic Circuits over the Reals. An *arithmetic circuit* (see [vzG86, BCS97]), C, is a finite directed acyclic multigraph (a multigraph is a graph in

which more than one edge is allowed between a pair of vertices) where each node has in-degree zero (*input nodes or constant nodes*) or in-degree two (*gate nodes*). Some nodes of the graph are marked as *output* nodes.

Every gate node is labeled with an arithmetic operation $(\times, +, -, /)$. We consider only *division free arithmetic circuits*, *i.e.*, arithmetic circuits where no gate node is labeled with $/$. Each input nodes has an associated variable name and each constant node an associated constant from $\mathbf{R}$.

We define the size of C as the number of nodes in the circuit (see also [Weg87]). We will use this number as our measure of *sequential complexity*, whereas the *parallel complexity* is determined by the *depth* of the circuit (*i.e.*, the length of the longest path in the subjacent graph, from an input node to an output node). The *non-scalar complexity* of C is defined as the number of non-scalar multiplication nodes in the circuit.

We will not define the semantics of arithmetic circuits. We simply remark that they furnish a versatile data structure to represent polynomials.

2.2 Representation of the Polynomials

Now, we describe the data types used in this article to represent polynomials. Let us first introduce some notation.

Definition 5. *A* sign condition *is an element of* $\{0, 1, -1\}$. *For* $x \in \mathbf{R}$ *we define*

$$sgn(x) := \begin{cases} -1 \text{ if } x < 0; \\ 0 \text{ if } x = 0; \\ 1 \text{ if } x > 0. \end{cases}$$

Let $\mathcal{P} \subset \mathbf{R}[X_1, ..., X_n]$. *A* $\mathcal{P}$-sign condition, σ, *is an element of* $\{-1, 0, 1\}^{\mathcal{P}}$. *We say that* $\mathcal{P}$ *realizes the sign condition* σ *at* $x \in \mathbf{R}^n$, *or that* x *satisfies the sign condition* σ *if, for every* $P \in \mathcal{P}$, $sgn(P(x)) = \sigma(P)$. *We denote the sign condition realized by* $\mathcal{P}$ *at* x *by* $sgn(\mathcal{P}, x)$.

Let us denote by $H(m)$ the *height* (or absolute value) of an integer $m \in \mathbf{Z}$ and by $h(m)$ its *logarithmic height* (or bitsize) defined as $h(m) := \lceil \log(H(m) + 1) \rceil$.

For a polynomial $P \in \mathbf{Z}[X_1, ..., X_n]$, we denote by $H(P)$ its *height* defined as the maximal height of all its coefficients and, analogously, by $h(P)$ its *logarithmic height* defined as the maximal logarithmic height of all its coefficients.

Let be given a polynomial $F \in \mathbf{R}[X_1, ..., X_n]$. We shall consider the following different representations of it. Besides the number n of variables, each of these representations has associated some natural parameters measuring the complexity of the representation.

1. **Arithmetic-circuit representation**. The polynomial F is represented by a division-free arithmetic circuit Γ over $\mathbf{R}$ that computes it. Let us denote by L the non-scalar complexity of Γ and observe that the degree of F is bounded by 2^L. The parameters associated with this representation are n and L.

2. **Dense arithmetic representation**. Suppose that the polynomial F has degree d. The dense arithmetic representation of F consists on the tuple in $\mathbf{R}^{\binom{d+n}{n}}$ of its coefficients in the monomial basis. The parameters associated with this representation are n and d.
3. **Dense bit representation**. We assume that the polynomial F has integer coefficients. If F has logarithmic height τ and degree d, its dense bit representation is the tuple in $\mathbf{Z}^{\binom{d+n}{n}}$ of its coefficients in the monomial basis, where each integer is represented by its bit encoding (of size at most τ). The parameters associated with this representation are n, d and τ.

Given a family $\mathcal{F} := \{F_1, ..., F_s\}$ of polynomials in $\mathbf{R}[X_1, ..., X_n]$, the dense (arithmetic or bit) representation of $\mathcal{F}$ is simply the collection of the dense (arithmetic or bit) representations of each polynomial in the family $\mathcal{F}$.

On the other hand, the arithmetic-circuit representation the family $\mathcal{F}$ is division-free arithmetic circuit Γ over $\mathbf{R}$ that computes all the polynomials in $\mathcal{F}$. Let us denote by L the non-scalar size of Γ and observe that the degrees of the polynomials in $\mathcal{F}$ are bounded by 2^L. The parameters associated with this representation are s, n and L.

We observe that the dense representation can be seen as a special case of the arithmetic-circuit representation with L equal to $\binom{d+n}{n} - n - 1$, the number of monomials of degree between two and d in n variables.

The *size* of a Triangular Thom encoding $(T_1, \sigma_1, ..., T_n, \sigma_n)$ (where, for $1 \leq i \leq n$, T_i is a polynomial and σ_i a sign condition on the derivatives of T_i, see [BPR06]) of a real algebraic point $x \in \mathbf{R}^n$ is defined as the pair (d, τ), where d is the maximum of the degrees of the polynomials $T_1, ..., T_n$ and τ us the maximum of their logarithmic heights.

3 The Dialytic Method to Solve the Sign Condition Problem

In this section we introduce the dialytic method[1] and show how it enables us to reduce the sign condition problem to the linear case. We assume the arithmetic-circuit representation of polynomials. We recall that the dense arithmetic representation can be seen as a particular case of this representation.

Let us consider a family $\mathcal{F} := \{F_1, ..., F_s\}$ of polynomials in $\mathbf{R}[X_1, ..., X_n]$ and suppose that $\Gamma_{\mathcal{F}}$ is a division-free arithmetic circuit of non-scalar complexity L that computes the family $\mathcal{F}$.

Suppose given a family of polynomials $\mathcal{G} = \{G_1, ..., G_k\} \subset \mathbf{R}[X_1, ..., X_n]$ that generate an $\mathbf{R}$-subspace $\mathbb{V}_{\mathcal{G}}$ of $\mathbf{R}[X_1, ..., X_n]$ that contains $\mathcal{F}$. Let us also suppose that the family $\mathcal{G}$ can be evaluated by a division-free arithmetic circuit of non-scalar complexity $L_{\mathcal{G}}$. We consider the following three different examples of the family $\mathcal{G}$, called *the family of generators*:

[1] The word *dialytic* comes from the Greek word $\delta\iota\acute{\alpha}\lambda\upsilon\sigma\iota\varsigma$, meaning separation [LSJM40]. This term was used by Sylvester [Syl42] when he introduced the resultant of a monic polynomial treating each monomial as a different variable.

- As the family of generators we can take the family $\mathcal{G}_P$ composed of the polynomials computed by the non-scalar multiplication nodes in $\Gamma_{\mathcal{F}}$ together with a basis for the linear polynomials in $\mathbf{R}[X_1, ..., X_n]$. In this case, we have $k = L + n + 1$ and $L_{\mathcal{G}_P} = L$. It is clear that any polynomial in $\mathcal{F}$ can be written as a linear combination of the polynomials in $\mathcal{G}_P$.
- Alternatively, as the family of generators we can take a maximal, $\mathbf{R}$-linearly independent subset $\mathcal{G}_{\mathcal{F}}$ of $\{F_1, ..., F_s\}$. In this case, we have k equal to the dimension of the $\mathbf{R}$-subspace generated $\mathcal{F}$ (bounded by $L + n + 1$, as the previous example shows) and $L_{\mathcal{G}_{\mathcal{F}}} \leq L$.
- Finally, if the polynomials $F_1, ..., F_s$ have degree bounded by d we can also take, as the family of generators, the monomial basis $\mathcal{G}_B$ of $\mathbf{R}[X_1, ..., X_n]$ obtaining $k = \binom{n+d}{n}$ and $L_{\mathcal{G}_B} = \binom{n+d}{n} - (n + 1)$.

We assume fixed any such family of generators $\mathcal{G} = \{G_1, ..., G_k\}$. Then, for $1 \leq i \leq s$ and $1 \leq j \leq k$, there exist constants $\alpha_j^{(i)} \in \mathbf{R}$ such that

$$F_i = \Sigma_{j=1}^{k} \alpha_j^{(i)} G_j.$$

Let $Z_1, ..., Z_k$ be new indeterminates and consider, for $1 \leq i \leq s$, the polynomials $\overline{F_i} := \Sigma_{j=1}^{k} \alpha_j^{(i)} Z_j \in \mathbf{R}[Z_1, ..., Z_k]$. Let us denote by $G : \mathbf{R}^n \to \mathbf{R}^k$ the function defined by $G(x) := (G_1(x), ..., G_k(x))$.

Remark 1. For any $x \in \mathbf{R}^n$ and for $1 \leq i \leq s$, the value of $F_i(x)$ is the same as the value of $\overline{F_i}(G(x))$.

In particular, this implies that a solution for the sign condition problem for the family of linear polynomials $\{\overline{F_1}, ..., \overline{F_1}\}$ induces a solution for the sign condition problem for the original family $\{F_1, ..., F_s\}$.

Theorem 2. *Let $\mathcal{F} := \{F_1, ..., F_s\} \subset \mathbf{R}[X_1, ..., X_n]$ be a family of polynomials and suppose given an arithmetic circuit $\Gamma_{\mathcal{F}}$ of non-scalar complexity L that computes the family $\mathcal{F}$.*

Then, there exists an algebraic computation tree Γ that solves the sign condition problem for the family $\mathcal{F}$ performing $\mathcal{O}((L + n)^5 \log(s))$ arithmetic operations.

The size of Γ is bounded by $s^{\mathcal{O}(n+L)}$ and it can be constructed in expected time $s^{\mathcal{O}(n+L)}$.

To prove this theorem, we need the following lemma that relates the total complexity of evaluating some polynomials in a given family to the non-scalar complexity of the family.

Lemma 1. *Let $\mathcal{F} := \{F_1, ..., F_s\} \subset \mathbf{R}[X_1, ..., X_n]$ be a family of polynomials represented by a division-free arithmetic circuit Γ of non-scalar complexity L. Let k be a positive integer, $1 \leq k \leq s$, and let $1 \leq i_1 \leq ... \leq i_k \leq s$ be integers. Then, $F_{i_1}, ..., F_{i_k}$ can be computed with total complexity $\mathcal{O}((L + k)(L + n))$.*

Proof. Let us denote by $n_1, ..., n_L$ the non-scalar multiplication nodes in Γ and by $P_1, ..., P_L$ the polynomials in $\mathbf{R}[X_1, ..., X_n]$ computed by these nodes. We assume, without loss of generality, that $\deg(P_1) \leq \deg(P_2) \leq ... \leq \deg(P_L)$.

It is easy to see that, for $1 \leq i \leq L$, the node n_i computes the product of two polynomials of the form

$$\Sigma_{j=1}^{i-1}\gamma_j P_j + \Sigma_{j=1}^{n}\beta_j X_j + \beta_0,$$

where the greek letters represent real numbers. Rewriting the circuit if necessary, each of these linear combinations can be computed from the preceding non-scalar multiplication nodes and input variables without non-scalar multiplications and a total complexity of $\mathcal{O}(n+i)$. Thus, there exists a division-free arithmetic circuit, namely Γ_P, that computes the family $\{P_1, ..., P_L\}$ with total complexity $\mathcal{O}(L^2 + nL)$.

We remark that, for any $1 \leq i \leq s$,

$$F_i = \Sigma_{j=1}^{L}\gamma_j P_j + \Sigma_{j=1}^{n}\beta_j X_j + \beta_0,$$

where the greek letters represent real numbers. Hence, each F_i can be computed from $\{P_1, ..., P_L\}$ performing $\mathcal{O}(L+n)$ arithmetic operations. Thus, the polynomials $F_{i_1}, ..., F_{i_k}$ can be computed with a total complexity $\mathcal{O}(L^2+nL)+\mathcal{O}(k(L+n))$. This completes the proof.

Proof (Proof of Theorem 2). Let $\mathcal{G} = \{G_1, ..., G_k\}$ be one of the families of generators $\mathcal{G}_P$ or $\mathcal{G}_{\mathcal{F}}$ defined above, and denote by $G : \mathbf{R}^n \to \mathbf{R}^k$ the associated function.

Using Meiser's result (see Theorem 1) we construct an algebraic computation tree $\Gamma_{\overline{\mathcal{F}}}$ that solves the point location problem for the family $\overline{\mathcal{F}} := \{\overline{F_1}, ..., \overline{F_s}\}$ of linear polynomials in $\mathbf{R}[Z_1, ..., Z_k]$, with query time $\mathcal{O}(k^5 \log(s))$.

Then, given a query point $x \in \mathbf{R}^n$ the sign condition satisfied by the family $\{F_1, ..., F_s\}$ at x can be determined using Meiser's algorithm for the family $\overline{\mathcal{F}}$ at the point $G(x)$.

The correctness of this method follows from Remark 1. The number of arithmetic operations needed to compute $G(x)$ is bounded by $\mathcal{O}((L+n)^2)$ by Lemma 1 since k is bounded by $L+n+1$ by construction. Thus, the total complexity is bounded by $\mathcal{O}((L+n)^5 \log(s) + (L+n)^2) = \mathcal{O}((L+n)^5 \log(s))$.

If we use the monomials basis $\mathcal{G}_B$ as the family of generators, we obtain the following result.

Corollary 1. *Let $\mathcal{F} := \{F_1, ..., F_s\} \subset \mathbf{R}[X_1, ..., X_n]$ be a family of polynomials of degree bounded by d.*

Then, there exists an algebraic computation tree Γ that solves the sign condition problem for the family $\mathcal{F}$ performing $\mathcal{O}(\binom{d+n}{n}^5 \log(s)) = \mathcal{O}(d^{5n} \log(s))$ arithmetic operations

The size of Γ is bounded by $s^{\mathcal{O}(d^n)}$ and it can be constructed in expected time $s^{\mathcal{O}(d^n)}$.

Proof. Ordering the monomials in $\mathbf{R}[X_1, ..., X_n]$ of degree at most d by ascending degree, each monomial in $\mathcal{G}_B$ can be computed as a product of two preceding monomials. Hence, the family $\mathcal{G}_B$ can be computed by a division-free arithmetic circuit of non-scalar complexity $\binom{n+d}{n} - n - 1$. Thus, the result follows from last theorem.

If we restrict our algorithms to perform arithmetic operations on integers, using Algorithm 11.8 (Sign Determination Algorithm) from [BPR06], we obtain the following result.

Corollary 2. *Let $\mathcal{F} := \{F_1, ..., F_s\} \subset \mathbf{Q}[X_1, ..., X_n]$ be a family of polynomials of total degree bounded by d and logarithmic height bounded by τ.*

Then, there exists an algebraic computation tree Γ that allows to determine, for any algebraic point $x \in \mathbf{R}^n$ given by a triangular Thom encoding of size (d', τ'), the sign conditions satisfied by the polynomials in $\mathcal{F}$ at x performing $\log(s)\overline{d}^{\mathcal{O}(n)}$ arithmetic operations between integers of logarithmic height bounded by $\overline{\tau}\overline{d}^{\mathcal{O}(n)}$, where $\overline{\tau} = \max\{\tau, \tau'\}$ and $\overline{d} = \max\{d, d'\}$. The size of the algebraic computation tree Γ is $\mathcal{O}(\tau s^{(d+1)^n})$ and it can be constructed in expected time $\mathcal{O}(\tau s^{(d+1)^n+1})$. □

Evaluation of First-Order Quantifier-Free Formulas. Let us consider $\mathcal{L}$, the first-order language defined as the usual first-order language of the reals but allowing only unary predicates $t > 0$ and $t = 0$ for any term t in the language, instead of the usual binary predicates $s = t$ and $s > t$ for arbitrary terms s and t. Of course, this does not change the expressive power of the language.

Let φ be a quantifier-free formula in this language with n free variables. The truth value of φ evaluated at $x \in \mathbf{R}^n$ depends only on the signs taken at x by the polynomials involved in φ. Hence, as a consequence of the Corollary 1 we obtain the following result.

Proposition 1. *Let φ be a quantifier-free formula with n free variables in the language $\mathcal{L}$ containing s polynomials of degree bounded by d.*

Then, there exists an algebraic computation tree Γ that solves the membership problem for the set $\{x \in \mathbf{R}^n \mid \mathbf{R} \models \varphi(x)\}$ performing $\mathcal{O}(d^{5n}\log(s))$ arithmetic operations.

The size of Γ is bounded by $s^{\mathcal{O}(d^n)}$ and it can be constructed in expected time $s^{\mathcal{O}(d^n)}$. □

We remark that the dialytic method performs non-scalar multiplications only to evaluate the function G (defined before Remark 1) at the input point. The rest of the algorithm is free from non-scalar multiplications, *i.e.*, it performs only linear operations on these results and branches according to their signs.

We can ask now: are these complexity bounds reasonable for the simple sign condition problem? In the next section we shall prove some lower bounds related to this problem.

4 Lower Bounds for the Sign Condition Problem

We shall analyze the cost of solving the sign condition problem for different examples of families of polynomials. Each example leads to a different lower complexity bound for the depth of any algebraic computation tree solving this

problem. In this way, we obtain lower bounds for the worst case complexity of the sign condition problem in terms of natural parameters of the given family.

In Example 1 we construct, for any positive integers s and n, a family of s linear forms in $\mathbf{R}[X_1, ..., X_n]$ that leads to the lower bound $\Omega(n \cdot \log(s))$ for the branching complexity of any algebraic computation tree solving the sign condition problem for this family.

In Example 2 we construct, for any positive integers s, L and n with $s \geq n^2$, a family of non-scalar complexity L, containing $s+1$ polynomials in $\mathbf{R}[X_1, ..., X_n]$, that leads to the lower bound $l(L, n, s) := \max\{L, n^{\frac{\log_3(s)}{2}}\}$ for the multiplicative branching complexity of any algebraic computation tree solving the sign condition problem for this family. If we denote by $u(L, n, s) := \mathcal{O}((L+n)^5 \log(s))$ the upper bound given by the dialytic method, we obtain that

$$u(L, n, s) = \mathcal{O}(l(L, n, s)^6) = l(L, n, s)^{\mathcal{O}(1)}.$$

Hence, this example shows that the dialytic method is almost optimal.

The main drawback of Example 2 is that the degrees of the polynomials involved in it are exponential on L. Example 3 is a modification of it. We obtain, under a suitable hypothesis, the same results as in the referred example with the additional property that the polynomials in the constructed family have degree bounded by $\mathcal{O}(L^2)$.

In Example 4, we consider a very restricted model: algebraic decision trees that can only test the sign of the polynomials in the family $\mathcal{F}$ at the input point. Our algorithms do not fit in this model since they evaluate also polynomials that do not belong to the original family. For this model, we show an $\Omega(s)$ lower bound.

4.1 The Algebraic Model

Now, we concentrate on the multiplicative branching complexity of any algebraic computation tree solving the sign condition problem for a given family of polynomials, *i.e.*, we take into account non-scalar multiplications and comparisons.

First we give a lower bound for the linear case, showing that Meiser's original algorithm is almost optimal.

Example 1. This example is a simplified linear version of the example used in [JS00] to prove a lower bound on the number of sign conditions satisfied by a family of polynomials.

For any positive integers s and n with $s > n$, consider s linear forms $l_1, ..., l_s \in \mathbf{R}[X_1, ..., X_n]$ satisfying:

- for every subset $\{l_{i_1}, ..., l_{i_n}\}$ of $\{l_1, ..., l_s\}$ consisting of n different linear forms, the linear equation system $l_{i_1}(X) = 0, ..., l_{i_n}(X) = 0$ has exactly one solution in $\mathbf{R}^n$, and
- for every subset $\{l_{i_1}, ..., l_{i_{n+1}}\}$ of $\{l_1, ..., l_s\}$ consisting of $n+1$ different linear forms, the linear equation system $l_{i_1}(X) = 0, ..., l_{i_{n+1}}(X) = 0$ has no solutions in $\mathbf{R}^n$.

We remark that these conditions define a non-empty open set (complementary to determinantal varieties, see [JS00]) in the space $\mathbf{R}^{s\times(n+1)}$ of coefficient of the linear forms. Hence, it is legitimate to assume the existence of a family $\{l_1, ..., l_s\}$ with the stated properties. The linear forms $l_1, ..., l_s$ are said to be in *general position*. Let us denote by C_s the number of sign conditions realized by this family in $\mathbf{R}^n$.

The two preceding conditions guarantee that for any two different subsets, $\{l_{i_1}, ..., l_{i_n}\}$ and $\{l_{j_1}, ..., l_{j_n}\}$, of $\{l_1, ..., l_s\}$ consisting each of n linear forms, the unique solution of the linear equation system $l_{i_1}(X) = 0, ..., l_{i_n}(X) = 0$ is different from the unique solution of the linear equation system $l_{j_1}(X) = 0, ..., l_{j_n}(X) = 0$. In particular, we obtain that the family $\{l_1, ..., l_s\}$ satisfies at least $\binom{s}{n}$ different sign conditions in $\mathbf{R}^n$, *i.e.*, $C_s \geq \binom{s}{n}$.

The following proposition follows immediately and plays an important role in our lower-bound results.

Proposition 2. *If an algebraic computation tree computes a partition π of $\mathbf{R}^n$, then its branching complexity is at least* $\log_3(\#\pi)$.

Proof. We recall that, in any algebraic computation tree, the only nodes with more than one immediate successor are the branching nodes, that have three immediate successors. Taking into account that a computation tree has at least one output node (*i.e.*, one leaf of the subjacent tree) for each element of π, the proof follows easily by induction.

Suppose that Γ is an algebraic computation tree that solves the sign condition problem for the family $\{l_1, ..., l_s\}$. Hence, it computes a partition of cardinality C_s. We conclude, from Proposition 2, that its branching complexity is at least $\log_3(C_s) > n(\log_3(s) - \log_3(n))$. In particular, if we take $s > n^2$, we obtain that the branching complexity of Γ is $n\frac{\log_3(s)}{2} = \Omega(n \cdot \log(s))$.

We remark that the upper bound given by Meiser's algorithm is $\mathcal{O}(n^5 \log(s))$. Thus, the upper bound is bounded by a polynomial function of the lower bound, which is satisfactory.

Meiser's algorithm does not use non-scalar multiplications. From our lower bound, we conclude that using non-scalar multiplications would not help to improve essentially the point location algorithm in the linear case.

For the discussion of the next example we need the following technical lemma that is the key to bound the non-scalar complexity of any algebraic computation tree that solves the sign condition problem.

Lemma 2. *Assume that $\{F_0, ..., F_s\}$ is a family of different irreducible polynomials defining real algebraic hypersurfaces in $\mathbf{R}^n$ and that Γ is an algebraic computation tree solving the sign condition problem for this family.*

Then, for every $i \in \mathbf{N}$, $0 \leq i \leq s$, there exists a branching node of Γ testing the sign of a multiple of F_i evaluated at the input point.

Proof. Let $G_1, ..., G_k$ be the non-zero irreducible factors of the polynomials intervening in the branching nodes of Γ and suppose, for the sake of definiteness,

that F_0 is not associated with any of them. We remark that $G_1, ..., G_k$ and F_0 are irreducible.

Then, a particular form of the real Nullstellensatz for principal ideas (see Theorem 4.5.1 in [BCR98]) implies that there exists an $x \in \mathbf{R}^n$ such that $F_0(x) = 0$ and $G_i(x) \neq 0$ for $1 \leq i \leq k$. Choose $\varepsilon \in \mathbf{R}, \varepsilon > 0$ such that the polynomials $G_1, ..., G_k$ do not vanish anywhere in the ball $B = B_\varepsilon(x)$.

Therefore, the signs of $G_1, ..., G_k$ are constants in B. In particular, the computation path followed by Γ for any two input points of B is exactly the same.

Since $\{x \in \mathbf{R}^n \mid F_0(x) = 0\}$ is an hypersurface that cuts B, we conclude that there are two points $y, z \in B$ satisfying the conditions $F_0(y) = 0$ and $F_0(z) \neq 0$. Hence, Γ does not solve the sign condition problem for the family $\{F_0, ..., F_s\}$. This contradicts the assumption that Γ solves the sign condition problem for the family $\{F_0, ..., F_s\}$.

Example 2. In this example we show that, for any positive integers n, s and L with $s > n^2$ it is possible to construct a family $\mathcal{F}$ of $s+1$ polynomials in $\mathbf{R}[X_1, ..., X_n]$ such that $L(\mathcal{F}) = L$ and any algebraic computation tree solving the sign condition problem for this family has multiplicative branching complexity at least $\max\{L, n \cdot \log(s)\}$.

Given positive integers s, n and L with $s > n^2$, consider, as in Example 1, s linear forms $F_1, ..., F_s \in \mathbf{R}[X_1, ..., X_n]$ in general position and let us define the polynomial $F_0 := X_1^{2^L} - X_2$. We denote by $\mathcal{F}$ the family $\mathcal{F} := \{F_0, F_1, ..., F_s\}$ composed of these polynomials.

Suppose that Γ is an algebraic computation tree that solves the sign condition problem for this family. Since the family $\mathcal{F}$ has at least the same number of realizable sign conditions as the family $\{F_1, ..., F_s\}$, we conclude, as in Example 1, that the branching complexity of Γ is at least $n^{\frac{\log_3(s)}{2}}$.

Clearly, the polynomial $F_0 = X_1^{2^L} - X_2$ is irreducible and takes positive and negative values in $\mathbf{R}^n$. Hence, it defines a real algebraic hypersurface. Whence, the family $\mathcal{F}$ satisfies the assumptions of Lemma 2. Thus, Γ evaluates a multiple of F_0. Since the degree of any multiple of F_0 is at least 2^L, we conclude that the non-scalar complexity of Γ is at least L.

Summarizing, we have that the branching complexity of Γ is at least $n^{\frac{\log_3(s)}{2}}$ and that its non-scalar complexity is at least L. Hence, its multiplicative branching complexity is at least $l(L, n, s) := \max\{L, n^{\frac{\log_3(s)}{2}}\}$.

The upper bound obtained from the dialytic method is $u(L, n, s) := \mathcal{O}((L + n)^5 \log(s))$. In order to compare both bounds, we remark that $l(L, n, s) = \max\{L, n^{\frac{\log_3(s)}{2}}\} = \Omega(L + n \cdot \log(s))$. Hence, we obtain that

$$u(L, n, s) \leq l(L, n, s)^6 = l(L, n, s)^{\mathcal{O}(1)}.$$

This proves that the dialytic method behaves very well for the chosen parameters.

Discussion. Let us consider a new parameter $M \in \mathbf{N}$ defined as the maximum of the non-scalar complexity of each polynomial in the given family. We remark that while the upper bound given by the dialytic method depends intrinsically on L, our lower bound would depend on M instead of L.

The family of polynomials constructed in this example has two characteristics that allowed us to derive the lower complexity bound:

1. the non-scalar complexity of some polynomials in the family is *close to* the non-scalar complexity of the whole family ($L = M^{\mathcal{O}(1)}$), and
2. the family defines enough different sign conditions ($s^{\mathcal{O}(n)}$).

What is not satisfactory about this lower bound is that the degrees of the polynomials involved are exponential on L. In the following example, we show that, under a suitable assumption, it is possible to modify our construction to obtain a polynomial F_0 whose degree is quadratic in its non-scalar complexity.

Example 3. This example is a modification of Example 2 and we shall use the notation introduced there. For any $d > 0$ sufficiently large, there exists, following Corollary 3.1 in [BH99], a univariate polynomial $P_d \in \mathbf{R}[X]$ of degree d such that the non-scalar complexity of any multiple of P_d is at least $\frac{1}{3}d^{\frac{1}{2}}$. In particular, $L(P_d) \geq \frac{1}{3}d^{\frac{1}{2}}$. On the other hand, Horner's rule give the upper bound, $L(P_d) \leq d - 1$.

We make the following (unproven) assumption: For any $d > 0$ sufficiently large and for any constant $c \in \mathbf{R}$ the non-scalar complexity of any multiple of $P_d - c$ is at least $\frac{1}{3}d^{\frac{1}{2}}$.

We assume that this conjecture is true and continue with the following construction.

Consider the polynomial $\widetilde{F_0} := P_d(X_1) - X_2 \in \mathbf{R}[X_1, ..., X_n]$. We claim that the non-scalar complexity of any non-zero multiple of $\widetilde{F_0}$ in $\mathbf{R}[X_1, ..., X_n]$ is at least $\frac{1}{3}d^{\frac{1}{2}}$. To prove the claim, consider a straight line program (SLP) γ of non-scalar complexity L that computes a non-zero multiple P of $\widetilde{F_0}$. Take $x = (x_1, ..., x_n) \in \mathbf{R}^n$ such that $P(x) \neq 0$, evaluate the SLP γ in $(X, x_2, ..., x_n)$ and denote by γ' the resulting SLP. Then, γ' computes a non-zero multiple of $P_d - x_2 \in \mathbf{R}[X]$ and we conclude from our conjecture that its non-scalar complexity is at least $\frac{1}{3}d^{\frac{1}{2}}$. Since the non-scalar complexity of γ is at least that of γ', our claim follows.

We define now the family $\widetilde{\mathcal{F}}$ as in the last example but using the polynomial $\widetilde{F_0}$ instead of F_0. We immediately obtain the following result.

For any three positive integers n, s and d with $s > n^2$ and $d > \mathcal{O}(1)$ there exists a family $\mathcal{F}$ of non-scalar complexity $L(\mathcal{F}) = d^{\mathcal{O}(1)}$ containing $s+1$ polynomials in $\mathbf{R}[X_1, ..., X_n]$ of degree bounded by d such that any algebraic computation tree solving the sign condition problem for this family has multiplicative branching complexity $\Omega(d + n \cdot \log(s))$.

Let us consider now another model.

Algebraic decision tree model. This is the simplest model where the sign condition problem for a family $\{F_1, ..., F_s\}$ of polynomials can be solved. In this model, an algorithm is an *algebraic decision tree* (that should not be confused with the *algebraic computation trees* of the previous examples) whose tests can only be based on the sign satisfied by some polynomial in $\{F_1, ..., F_s\}$ evaluated at the input point.

Example 4. In this example, we construct a family of linear polynomials in $\mathbf{R}[X_1, X_2]$ and show that any algebraic decision tree in our restricted model that solves the sign condition problem for this family must have depth s.

Consider the unit circle in the plane $S^1 \subset \mathbf{R}^2$ and s different points on it, $p_1, ..., p_s \in S^1$. For $1 \leq i \leq s$, let us denote by F_i the linear equation $p_i \cdot (x_1, x_2) - 1$ representing the tangent line to S^1 passing through p_i. We remark that inside the unit circle all these equations take negative values. Also, for $1 \leq i \leq s$, all these linear polynomials take negative values at p_i except F_i which is zero.

Suppose now that Γ is an algebraic decision tree satisfying that any of its decisions (branchings) is based on the sign of some polynomial in the family $F_1, .., F_s$ evaluated at the input point. We claim that for any input whose computation path follows the *negative sign* branch of each test in this path, all the polynomials in the family $F_1, ..., F_s$ must be evaluated before the sign taken by all of them at the input is completely determined.

To prove the claim, suppose that $F_1, ..., F_{s-1}$ are evaluated but not F_s. Consider the point p_s; as remarked before, $F_1, ..., F_{s-1}$ take negative values at p_s and $F_s(p_s) = 0$. Then, there exists a small open ball, $B_\varepsilon(p_s) \subset \mathbf{R}^2$ such that $F_1, ..., F_{s-1}$ take negative values inside this ball. Since $F_s = 0$ describes a line, we conclude that there exist two different points, namely x and y, in the ball $B_\varepsilon(p_s)$ such that $F_s(x) > 0$ and $F_s(y) < 0$. Hence, the sign of an input point cannot be determined evaluating a proper subset of $\{F_1, ..., F_s\}$ at this point. Thus, the depth of Γ is at least s.

Discussion. This example can be easily generalized to higher dimensions and, with some more work, to polynomials of any given non-scalar complexity.

This example shows that other polynomials than $F_1, ..., F_s$ must be evaluated in order to obtain an upper bound that depends logarithmically on s. Inspecting Meiser's algorithm and the dialytic method, we see that it is enough to admit to test the sign of linear combinations of the polynomials evaluated in previous tests.

5 The Dialytic Method for Point Location

Now, we show how the dialytic method can be used to solve the point location problem for a given family of polynomials.

To solve the point location problem for a family of polynomials, we use the following proposition, that is an immediate consequence of Theorem 16.18 in [BPR06] (see [HRS90a, GHR+90, HRS94b, CGV91, GV92, HRS94a, Can93, GR93, BPR99] for the historical development of this result).

Proposition 3. *Let $\mathcal{P}$ be a family of s polynomials in $\mathbf{R}[X_1, ..., X_n]$ of degree bounded by d. Then, there exists a family $\widetilde{\mathcal{P}}$ containing $s^n d^{\mathcal{O}(n^4)}$ polynomials in $\mathbf{R}[X_1, ..., X_n]$ of degree bounded by $d^{\mathcal{O}(n^3)}$, such that the partition $\mathcal{S}(\widetilde{\mathcal{P}})$ of $\mathbf{R}^n$ induced by the realization of the sign conditions on the family $\widetilde{\mathcal{P}}$ is finer than the partition $\mathcal{A}(\mathcal{P})$ induced by the connected components of the realization of the sign conditions on the family $\mathcal{P}$.*

Moreover, there exists an algorithm that, on input $\mathcal{P}$, computes a family $\widetilde{\mathcal{P}}$ with the stated properties in time bounded by $s^{n+1}d^{\mathcal{O}(n^4)}$; if the input polynomials have integer coefficients whose bitsize is bounded by τ, the bitsize of the coefficients of the output is $\tau d^{\mathcal{O}(n^3)}$. □

The last proposition implies that a solution of the sign condition problem for the family $\widetilde{\mathcal{P}}$ leads to a solution of the point location problem for the family $\mathcal{P}$. Combining the Corollary 2 with the last proposition, and identifying different outputs corresponding to a same face of the arrangement $\mathcal{A}(\mathcal{P})$, we obtain following corollary.

Corollary 3. *Let $\mathcal{P} := \{P_1, ..., P_s\}$ be a family polynomials in $\mathbf{R}[X_1, ..., X_n]$ of total degree bounded by d and logarithmic height bounded by τ. Then, there exists an algebraic computation tree of size $\tau s^{d^{\mathcal{O}(n^4)}}$ that solves the point location problem for the family $\mathcal{P}$. For any $x \in \mathbf{R}^n$ given by a triangular Thom encoding of size (d', τ'), the point location query at x is answered performing $\log(s)\overline{d}^{\mathcal{O}(n^4)}$ arithmetic operations between integers of logarithmic height bounded by $\overline{\tau}\overline{d}^{\mathcal{O}(n)}$, where $\overline{\tau} = \max\{\tau, \tau'\}$ and $\overline{d} = \max\{d^{\mathcal{O}(n^3)}, d'\}$. The algebraic computation tree can be constructed in expected time $\tau s^{d^{\mathcal{O}(n^4)}}$.* □

References

[BCR98] Bochnak, J., Coste, M., Roy, M.F.: Real Algebraic Geometry. Modern Surveys in Mathematics, vol. 36. Springer, Heidelberg (1998)

[BCS97] Bürgisser, P., Clausen, M., Shokrollahi, M.A.: Algebraic complexity theory. Grundlehren der mathematischen Wissenschaften. Springer, Heidelberg (1997)

[BH99] Baur, W., Halupczok, H.: On lower bounds for the complexity of polynomials and their multiples. Computational Complexity 8(4), 309–315 (1999)

[BO83] Ben-Or, M.: Lower bounds for algebraic computation trees. In: STOC 1983: Proceedings of the fifteenth annual ACM symposium on Theory of computing, pp. 80–86. ACM, New York (1983)

[BPR10] Basu, S., Pollack, R., Roy, M.F.: An asymptotically tight bound on the number of connected components of realizable sign conditions. Combinatorica (2009/2010) (to appear)

[BPR99] Basu, S., Pollack, R., Roy, M.-F.: Computing roadmaps of semi-algebraic sets on a variety. J. AMS 3(1), 55–82 (1999)

[BPR06] Basu, S., Pollack, R., Roy, M.F.: Algorithms in real algebraic geometry. In: Algorithms and Computation in Mathematics, 2nd edn., vol. 10. Springer, Secaucus (2006)

[Can93] Canny, J.F.: Computing roadmaps of general semi-algebraic sets. Comput. J. 36(5), 504–514 (1993)

[CEGS91] Chazelle, B., Edelsbrunner, H., Guibas, L.J., Sharir, M.: A singly-exponential stratification scheme for real semi-algebraic varieties and its applications. Theoretical Computer Science 84(1), 77–105 (1991)

[CGV91] Canny, J.F., Grigoriev, D., Vorobjov, N.: Finding connected components of a simialgebraic set in subexponential time. Appl. Algebra Eng. Commun. Comput. 2, 217–238 (1991)

[Cla87] Clarkson, K.L.: New applications of random sampling in computational geometry. Discrete & Computational Geometry 2(2), 195–222 (1987)

[Col75] Collins, G.E.: Hauptvortrag: Quantifier elimination for real closed fields by Cylindrical Algebraic Decomposition. Automata Theory and Formal Languages, 134–183 (1975)

[CS90] Chazelle, B., Sharir, M.: An algorithm for generalized point location and its applications. J. Symb. Computation 10(3-4), 281–309 (1990)

[DL76] Dobkin, D.P., Lipton, R.J.: Multidimensional searching problems. SIAM J. Comput. 5(2), 181–186 (1976)

[GHR$^+$90] Grigoriev, D., Heintz, J., Roy, M.F., Solern, P., Vorobjov, N.: Comptage des composantes connexes d'un ensemble semi-algbrique en temps simplement exponentiel. C.R. Acad. Sci. Paris 311, 879–882 (1990)

[GR93] Gournay, L., Risler, J.J.: Construction of roadmaps in semi-algebraic sets. Appl. Algebra Eng. Commun. Comput. 4, 239–252 (1993)

[Gri88] Grigoriev, D.: Complexity of deciding tarski algebra. J. Symb. Computation 5(1/2), 65–108 (1988)

[Gri00] Grigoriev, D.: Topological complexity of the range searching. J. Complexity 16(1), 50–53 (2000)

[GV92] Grigoriev, D., Vorobjov, N.: Counting connected components of a semialgebraic set in subexponential time. Computational Complexity 2, 133–186 (1992)

[HRS90a] Heintz, J., Roy, M.F., Solern, P.: Single exponential path finding in semialgebraic sets. Part 1: The case of a regular bounded hypersurface. In: Sakata, S. (ed.) AAECC 1990. LNCS, vol. 508, pp. 180–196. Springer, Heidelberg (1991)

[HRS90b] Heintz, J., Roy, M.F., Solern, P.: Sur la complexit du principe de Tarski-Seidenberg. Bull. SMF 118, 101–126 (1990)

[HRS94a] Heintz, J., Roy, M.F., Solern, P.: Description of the connected components of a semialgebraic in single exponential time. Discrete & Computational Geometry 11, 121–140 (1994)

[HRS94b] Heintz, J., Roy, M.F., Solern, P.: Single exponential path finding in semialgebraic sets II: The general case, Algebraic geometry and its applications, collections of papers from Abhyankar's 60-th birthday conference, Purdue University, West-Lafayette (1994)

[JS00] Jeronimo, G., Sabia, J.: On the number of sets definable by polynomials. J. of Algebra 227(2), 633–644 (2000)

[Kha79] Khachiyan, L.G.: A polynomial algorithm in linear programming. Soviet Mathematics Doklady 20, 191–194 (1979)

[Koi00] Koiran, P.: Circuits versus trees in algebraic complexity. In: Reichel, H., Tison, S. (eds.) STACS 2000. LNCS, vol. 1770, pp. 35–52. Springer, Heidelberg (2000)

[LB01] Ganapathy, M.K., Babai, L., Ranyai, L.: On the number of zero-patterns of a sequence of polynomials. J. American Math. Soc. 14, 717–735 (2001)

[Lic90] Lickteig, T.: On semialgebraic decision complexity, Technical Report TR-90-052, Int. Comput. Sc. Inst, Berkeley. Habilitationsschrift, Universitat Tubingen (1990)

[LSJM40] Liddell, H.G., Scott, R., Jones, H.S., Mckenzie, R.: A greek-english lexicon. Clarendon Press, Oxford (1940)

[MadH84] Meyer auf der Heide, F.: A polynomial linear search algorithm for the n-dimensional knapsack problem. J. ACM 31(3), 668–676 (1984)

[MadH88] Meyer auf der Heide, F.: Fast algorithms for n-dimensional restrictions of hard problems. J. ACM 35(3), 740–747 (1988)

[Mei93] Meiser, S.: Point location in arrangements of hyperplanes. Inf. Comput. 106(2), 286–303 (1993)

[Mil64] Milnor, J.: On the Betti numbers of real varieties. Proc. AMS 15, 275–280 (1964)

[Sno04] Snoeyink, J.: Point location. In: Goodman, J.E., O'Rourke, J. (eds.) Handbook of Discrete and Computational Geometry. CRC Press LLC, Boca Raton (2004)

[Str81] Strassen, V.: The computational complexity of continued fractions. In: SYMSAC 1981: Proceedings of the fourth ACM symposium on Symbolic and algebraic computation, pp. 51–67. ACM, New York (1981)

[Syl42] Sylvester, J.J.: Memoir on the dialytic method of elimination. Part I. Philosophical Magazine XXI, 534–539 (1842)

[vzG86] von zur Gathen, J.: Parallel arithmetic computations: a survey. In: Wiedermann, J., Gruska, J., Rovan, B. (eds.) MFCS 1986. LNCS, vol. 233, pp. 93–112. Springer, Heidelberg (1986)

[War68] Warren, H.: Lower bounds for approximation of nonlinear manifolds. Trans. AMS 133, 167–178 (1968)

[Weg87] Wegener, I.: The complexity of boolean functions. B. G. Teubner, and J. Wiley & Sons (1987)

Constraints on Curve Networks Suitable for G^2 Interpolation

Thomas Hermann[1], Jorg Peters[2], and Tim Strotman[1]

[1] Parasolid Components, Siemens PLM Software
[2] University of Florida
{tamas.hermann,tim.strotman}@siemens.com, jorg@cise.ufl.edu

Abstract. When interpolating a network of curves to create a C^1 surface from smooth patches, the network has to satisfy an algebraic condition, called the vertex enclosure constraint. We show the existence of an additional constraint that governs the admissibility of curve networks for G^2 interpolation by smooth patches.

1 Introduction

One much-studied paradigm of geometric design is surface interpolation of a given network of C^2 curve segments (see Figure 1). While many C^2 constructions exist that join n patches (e.g. [Hah89, GH95, Ye97, Rei98, Pra97, YZ04, LS08, KP09]), these constructions *generate* the boundary curves that emanate from the common point, i.e. rely on full control of these curves. In many practical scenarios, however, the curves are feature curves. That is, they are *given* and may only be minimally adjusted. It is well-known, that interpolating a network of curves by smooth patches to create a C^1 surface is not always possible when the number of curves is even, since an additional algebraic constraint must hold for the *normal component* of the curve expansion at the common point. This is the *first-order* vertex enclosure constraint [Pet91, DS91, HPS09]. Here we discuss whether curve nets have to meet additional *second-order vertex enclosure constraints* to allow for their G^2 interpolation by smooth surface patches. The two papers on this subject we are aware of are [DS92] which sketches how one might solve the G^2 constraints but does not discuss whether they can be solved and [Pet92] which analyzes the case when curves join with equal angles.

In particular, we want to determine constraints, if any, on n boundary curves $\mathbf{y}^j$, $j = 1, \ldots, n$ so that consecutive patches surrounding a vertex can join with C^2 continuity after reparameterization by some regular map

$$\mathbf{\Phi}^j : \mathbb{R}^2 \to \mathbb{R}^2, \qquad \mathbf{\Phi}^j(u,v) =: \left[\begin{smallmatrix}\sigma^j \\ \tau^j\end{smallmatrix}\right]. \tag{1}$$

We show constructively under what constraints smooth interpolating surfaces can be constructed, but we do not discuss how to obtain fair surfaces. Nor are we suggesting heuristics for the generation of curve networks.

2 Smooth Network Interpolation

As illustrated in Figure 1, we consider n curves $\mathbf{y}^j : \mathbb{R} \to \mathbb{R}^3$ that start at a point $\mathbf{p} \in \mathbb{R}^3$, and we aim at filling-in between the curves using patches $\mathbf{x}^j : \mathbb{R}^2 \to \mathbb{R}^3$,

B. Mourrain, S. Schaefer, and G. Xu (Eds.): GMP 2010, LNCS 6130, pp. 77–87, 2010.

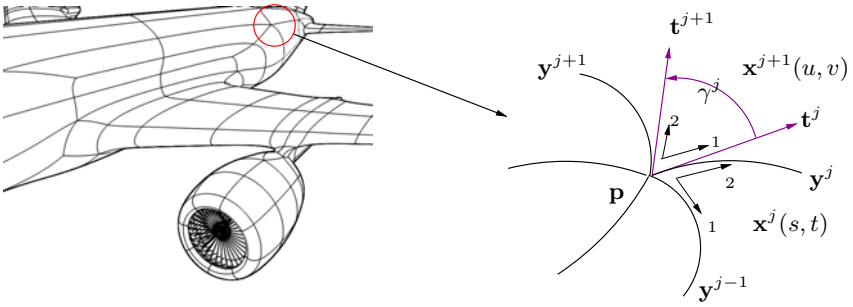


Fig. 1. (*left*) Network of curve segments. This paper focuses on (*right*) local network interpolation (see also Definition 1): curves $\mathbf{y}^j$, $j \in \mathbb{Z}_n$, meeting at a point $\mathbf{p}$ are given and pairwise interpolating patches $\mathbf{x}^j$ are sought. The arrow-labels $_1$ and $_2$ indicate the domain parameters associated with the boundary curves of the patches, e.g. $\partial_1 \mathbf{x}^{j+1}(\nu, 0) = \partial_2 \mathbf{x}^j(0, \nu)$.

$j \in \mathbb{Z}_n$. We note that the angle γ^j corresponds to patch $\mathbf{x}^{j+1}$ and assume for notational simplicity that the curves are arclength-parameterized at $\nu = 0$ so that each tangent vector $\mathbf{t}^j := (\mathbf{y}^j)'(0)$ is a unit vector. Differential geometry provides us with two fundamental properties that the curve network $\{\mathbf{y}^j\}$ must satisfy to be part of a C^2 surface. There must exist a vector $\mathbf{n}$, the normal at $\mathbf{p}$, and $\mathbb{I}(\cdot,\cdot)$, the second fundamental form acting on the tangent plane components of its two arguments, such that

$$\mathbf{t}^j \cdot \mathbf{n} = 0, \quad \text{and} \quad \mathbb{I}(\mathbf{t}^j, \mathbf{t}^j) = \mathbf{y}_2^j \cdot \mathbf{n}, \quad j \in \mathbb{Z}_n. \tag{2}$$

(We note that $\mathbf{n}$ is unique iff there are two linearly independent $\mathbf{t}^j$ and $\mathbb{I}$ is unique iff there are three pair-wise linearly independent $\mathbf{t}^j$. When the tangents form an X, i.e. when there are just two pair-wise linearly independent $\mathbf{t}^j$, then there is a one-parameter family of second fundamental forms.)

Definition 1 (Smooth Network Interpolation). *Let*

$$\mathbf{y}^j : \mathbb{R} \to \mathbb{R}^3, \nu \mapsto \mathbf{y}^j(\nu), \qquad j \in \mathbb{Z}_n = \{1, \ldots, n\} \tag{3}$$

be a sequence of n regular, $C^{2\mathtt{k}}$ continuous curves in $\mathbb{R}^3$ that meet at a common point $\mathbf{p}$ in a plane with oriented normal $\mathbf{n}$ and at angles γ^j less than π (cf. Figure 1):

$$\mathbf{y}^j(0) = \mathbf{p}, \mathbf{t}^j := (\mathbf{y}^j)'(0) \perp \mathbf{n}, \quad 0 < \gamma^j := \angle(\mathbf{t}^j, \mathbf{t}^{j+1}) < \pi. \tag{4}$$

A $G^{\mathtt{k}}$ surface network interpolation *of $\{\mathbf{y}^j\}$ is a sequence of patches*

$$\mathbf{x}^j : \mathbb{R}^2 \to \mathbb{R}^3, (s,t) \mapsto \mathbf{x}^j(s,t), \qquad j \in \mathbb{Z}_n, \tag{5}$$

that are regular and $C^{2\mathtt{k}}$ at $\mathbf{p}$, that interpolate the curve network according to

$$\mathbf{x}^j(\nu, 0) = \mathbf{y}^{j-1}(\nu), \ \mathbf{x}^j(0, \nu) = \mathbf{y}^j(\nu), \tag{6}$$

(with index modulo n) and that connect pairwise so that $G^{\mathtt{k}}$ constraints (see e.g. [PBP02] or [Pet02]) hold:

$$\text{at } (u,0) \qquad \partial_1^{k_1} \partial_2^{k_2} \mathbf{x}^{j+1} = \partial_1^{k_1} \partial_2^{k_2} (\mathbf{x}^j \circ \mathbf{\Phi}^j), \quad \text{for } 0 \le k_i \le \mathtt{k}. \tag{7}$$

Smooth Network Interpolation restricted to the neighborhood of $\mathbf{p}$ *is called* local network interpolation.

Note that the increased smoothness at vertices is natural for spline constructions but (intentionally) rules out out Gregory's rational constructions [Gre74, MW91, Her96] and that, by [HLW99], (7) is equivalent to $\partial_2^i \mathbf{x}^{j+1}(u,0) = \partial_2^i(\mathbf{x}^j \circ \mathbf{\Phi}^j)(u,0)$ for $0 \le i \le \mathtt{k}$. Since the reparameterization appears only on one side, the formulation may appear asymmetric; but with $\mathbf{\Phi}^j$ regular, we can invert the relationship – so this formulation is as general and powerful as reparameterizing both $\mathbf{x}^{j+1}$ and $\mathbf{x}^j$.

Section 3 introduces the constraints for $\mathtt{k} = 2$, resulting from expanding (7) at $(0,0)$. Section 4 then classifies the G^2 constraints at the vertex and analyzes their solvability for a fixed curve network. Theorem 1 establishes the existence of second-order vertex enclosure constraints. We conclude with a conjecture on the properties of a matrix that holds the key to the complete characterization of second-order vertex enclosure constraints.

3 Notation and Constraints

Since our focus is on curvature continuity at $\mathbf{p} = \mathbf{x}^j(0,0)$, we abbreviate the kth derivative of $\mathbf{y}^j$ evaluated at 0 as $\mathbf{y}^j_k$ and write

$$\mathbf{x}^j_{k_1k_2} := (\partial_1^{k_1}\partial_2^{k_2}\mathbf{x}^j)(0,0), \quad \tau^j_{k_1k_2} := (\partial_1^{k_1}\partial_2^{k_2}\tau^j)(0,0),\ \sigma^j_{k_1k_2} := (\partial_1^{k_1}\partial_2^{k_2}\sigma^j)(0,0). \tag{8}$$

We *drop superscripts whenever the context makes them unambiguous*, e.g. we write

$$\mathbf{x}_{k_1k_2} := \mathbf{x}^j_{k_1k_2},\ \mathbf{x}^-_{k_1k_2} := \mathbf{x}^{j-1}_{k_1k_2}, \ldots, \tag{9}$$

$$\mathbf{y}_k := \mathbf{y}^j_k = \mathbf{x}_{0k},\ \mathbf{y}^-_k := \mathbf{y}^{j-1}_k = \mathbf{x}_{k0},\ \mathbf{y}^+_k := \mathbf{y}^{j+1}_k = \mathbf{x}^+_{0k}. \tag{10}$$

That is $\mathbf{x}_{k_1k_2}$ is a vector in $\mathbb{R}^3$ and *not* a vector of vectors $[\ldots, \mathbf{x}^j_{k_1k_2}, \ldots]$.

We also *tag the equations arising from* (7) for a specific choice of (k_1,k_2) and j as $(k_1,k_2)^j$. Again, to minimize ink, we leave out the superscript when possible. By (6), $\mathbf{\Phi}^j$ has the expansion

$$\mathbf{\Phi}^j(u,v) := \begin{bmatrix} (\sigma^j_{01} + \sigma^j_{11}u + \ldots)v + (\sigma^j_{02} + \sigma^j_{12}u + \ldots)\frac{v^2}{2} + \ldots \\ u + (\tau^j_{01} + \tau^j_{11}u + \ldots)v + (\tau^j_{02} + \tau^j_{12}u + \ldots)\frac{v^2}{2} + \ldots \end{bmatrix}. \tag{11}$$

Substituting the curves according to (6), we obtain from (7) at $(0,0)$, via the chain rule, the G^1 constraints

$$\mathbf{y}^+_1 = \mathbf{y}^-_1\sigma_{01} + \mathbf{y}_1\tau_{01} \tag{0,1}$$

$$\mathbf{x}^+_{11} = \mathbf{y}^-_1\sigma_{11} + \mathbf{x}_{11}\sigma_{01} + \mathbf{y}_2\tau_{01} + \mathbf{y}_1\tau_{11} \tag{1,1}$$

$$\mathbf{x}^+_{21} = 2\mathbf{x}_{11}\sigma_{11} + 2\mathbf{y}^-_1\sigma_{21} + \mathbf{x}_{12}\sigma_{01} + \mathbf{y}_3\tau_{01} + 2\mathbf{y}_2\tau_{11} + 2\mathbf{y}_1\tau_{21} \tag{2,1}$$

$$\mathbf{x}^+_{31} = 3\mathbf{x}_{12}\sigma_{11} + 6\mathbf{x}_{11}\sigma_{21} + 6\mathbf{y}^-_1\sigma_{31} + \mathbf{x}_{13}\sigma_{01} + \mathbf{y}_4\tau_{01} + 3\mathbf{y}_3\tau_{11} + 6\mathbf{y}_2\tau_{21} + 6\mathbf{y}_1\tau_{31} \tag{3,1}$$

and the G^2 constraints

$$\begin{aligned}
\mathbf{y}_2^+ &= \mathbf{y}_2^- \sigma_{01}^2 + 2\sigma_{01}\mathbf{x}_{11}\tau_{01} + \mathbf{y}_1^- \sigma_{02} + \mathbf{y}_2 \tau_{01}^2 + \mathbf{y}_1 \tau_{02} && \text{(0,2)}\\
\mathbf{x}_{12}^+ &= 2\sigma_{11}\mathbf{y}_2^- \sigma_{01} + 2\sigma_{11}\mathbf{x}_{11}\tau_{01} + \mathbf{y}_1^- \sigma_{12} + \mathbf{x}_{21}\sigma_{01}^2 + 2\sigma_{01}\mathbf{x}_{12}\tau_{01}\\
&\quad + \mathbf{x}_{11}\sigma_{02} + \mathbf{y}_3\tau_{01}^2 + \mathbf{y}_2\tau_{02} + 2\tau_{11}\mathbf{x}_{11}\sigma_{01} + 2\tau_{11}\mathbf{y}_2\tau_{01} + \mathbf{y}_1\tau_{12} && \text{(1,2)}\\
\mathbf{x}_{22}^+ &= 4\tau_{21}\mathbf{y}_2\tau_{01} + 4\tau_{11}\mathbf{y}_3\tau_{01} + 4\sigma_{11}\mathbf{x}_{11}\tau_{11} + 4\tau_{21}\mathbf{x}_{11}\sigma_{01} + 4\sigma_{11}\mathbf{x}_{12}\tau_{01}\\
&\quad + 4\sigma_{21}\mathbf{x}_{11}\tau_{01} + 2\sigma_{01}\mathbf{x}_{13}\tau_{01} + 4\sigma_{21}\mathbf{y}_2^-\sigma_{01} + 4\sigma_{11}\mathbf{x}_{21}\sigma_{01} + 2\mathbf{y}_2\tau_{12}\\
&\quad + 4\tau_{11}\mathbf{x}_{12}\sigma_{01} + 2\mathbf{x}_{11}\sigma_{12} + 2\mathbf{y}_1^-\sigma_{22} + \mathbf{x}_{12}\sigma_{02} + \mathbf{y}_3\tau_{02} + 2\mathbf{y}_1\tau_{22}\\
&\quad + 2\mathbf{y}_2^-\sigma_{11}^2 + \mathbf{x}_{22}\sigma_{01}^2 + \mathbf{y}_4\tau_{01}^2 + 2\mathbf{y}_2\tau_{11}^2. && \text{(2,2)}
\end{aligned}$$

Lemma 1 (equivalence of $\mathbb{I}$ and normal twist). *Let $\mathbf{x}(u,v)$ be a patch interpolating the curves $\mathbf{y}^0(u)$ and $\mathbf{y}^1(v)$ with tangents $\mathbf{t}^0$ and $\mathbf{t}^1$ respectively and $\mathbf{x}^+$ its consecutive patch interpolating $\mathbf{y}^1$, and $\mathbf{y}^2$ with tangent $\mathbf{t}^2$. If the tangents $\mathbf{t}^j, j = 0,1,2$ are pairwise linearly independent then defining a unique second fundamental form $\mathbb{I}(\cdot,\cdot)$ at $(0,0)$ is equivalent to defining $\mathbf{x}_{11}(0,0)\cdot\mathbf{n}$, the normal component of the corner twist.*

Proof. Let $\mathbf{t}^2 = a\mathbf{t}^0 + b\mathbf{t}^1$. Since the second fundamental form is identical for adjacent patches, at $(0,0)$,

$$\begin{aligned}
&a^2\mathbb{I}(\mathbf{t}^0,\mathbf{t}^0) + 2ab\mathbb{I}(\mathbf{t}^0,\mathbf{t}^1) + b^2\mathbb{I}(\mathbf{t}^1,\mathbf{t}^1) = \mathbb{I}(a\mathbf{t}^0 + b\mathbf{t}^1, a\mathbf{t}^0 + b\mathbf{t}^1) = \mathbb{I}(\mathbf{t}^2,\mathbf{t}^2) && \text{(12)}\\
&=^{(2)} \mathbf{x}_{02}^+ \cdot \mathbf{n} = (a^2\mathbf{x}_{20} + 2ab\mathbf{x}_{11} + b^2\mathbf{x}_{02})\cdot\mathbf{n}\\
&=^{(2)} a^2\mathbb{I}(\mathbf{t}^0,\mathbf{t}^0) + 2ab\mathbf{x}_{11}\cdot\mathbf{n} + b^2\mathbb{I}(\mathbf{t}^1,\mathbf{t}^1).
\end{aligned}$$

If $a, b \neq 0$ then comparing terms shows $\mathbb{I}(\mathbf{t}_0,\mathbf{t}_1) = \mathbf{x}_{11}\cdot\mathbf{n}$ as claimed. ∎

If the tangents form an X, we can define a consistent second fundamental form for the surface network by choosing the value of

$$w_{11} := \mathbf{x}_{11}^0\cdot\mathbf{n} = \mathbf{x}_{11}^2\cdot\mathbf{n} = -\mathbf{x}_{11}^1\cdot\mathbf{n} = -\mathbf{x}_{11}^3\cdot\mathbf{n}. \tag{13}$$

By (1,1), fixing $\mathbf{x}_{11}$ determines $\tau_{ik}^j, \sigma_{ik}^j$ for $0 \le i,k \le 1$.

4 Constraints on Boundary Curves Arising from G^2 Continuity

Let us call the G^1 and the G^2 constraints $(i,s)^j$ for $i+s \le 4$ listed in the previous section, G^2 *vertex constraints*. First we show that, if we can find a solution satisfying the G^2 vertex constraints then there exists a solution to the local network interpolation. Later, we analyze under what conditions a solution exists.

Given a network of curves and a solution to the G^2 vertex constraints, we construct a local network interpolation as follows.

Lemma 2. *If the G^2 vertex constraints hold then there exists a* local network interpolation $\{\mathbf{x}^k\}$.

Proof. Dropping as usual the superscript k, we define a network of surfaces

$$\mathbf{x}(s,t) := \mathbf{y}^-(s) + \mathbf{y}(t) - \mathbf{y}_0 + \sum_{ij\in I} \mathbf{x}_{ij} \frac{s^i}{i!}\frac{t^j}{j!} + s^3\big(t\mathbf{l}_1(s) + \frac{t^2}{2}\mathbf{l}_2(s)\big), \tag{14}$$

where $I := \{(1,1),(1,2),(1,3),(2,1),(2,2)\}$, $J := I \cup \{(1,0),(2,0)\}$ and

$$\mathbf{x}_{ij}, ij \in I, \quad \sigma(u,v) := \sum_{ij\in J} \sigma_{ij}\frac{u^i}{i!}\frac{v^j}{j!}, \quad \tau(u,v) := u + \sum_{ij\in J} \tau_{ij}\frac{u^i}{i!}\frac{v^j}{j!} \tag{15}$$

satisfy the G^2 vertex constraints. By definition, for any choice of polynomials $\mathbf{l}_1$, $\mathbf{l}_2$, $\hat{\mathbf{x}}(u,v) := \mathbf{x}(\sigma(u,v),\tau(u,v))$ joins $\mathbf{x}$ in a G^2 fashion along $\mathbf{y}$; and so does

$$\mathbf{x}^+(u,v) := \hat{\mathbf{x}}(u,0) + \partial_v\hat{\mathbf{x}}(u,0)v + \partial_v^2\hat{\mathbf{x}}(u,0)\frac{v^2}{2} + \hat{\mathbf{y}}^+(v), \tag{16}$$
$$\hat{\mathbf{y}}(\nu) := \mathbf{y}(\nu) - \mathbf{y}_0 - \mathbf{y}_1\nu - \mathbf{y}_2\frac{\nu^2}{2}$$

since $\mathbf{x}^+$ agrees up to second order with $\hat{\mathbf{x}}$ along $\mathbf{y}$. Therefore $\mathbf{x}^+$ and $\mathbf{x}$ meet in a G^2 fashion along $\mathbf{y}$. Also $\mathbf{x}^+$ interpolates $\mathbf{y}$ and $\mathbf{y}^+$ and the Taylor coefficients of $\mathbf{x}^+$ are designed to be those of $\mathbf{x}$ with the superscript increased by 1. ∎

We note that $\mathbf{l}_1$ and $\mathbf{l}_2$ are not directly involved in the definition of $\mathbf{x}^+$ but rather are defined via (1) for the next curve:

$$\mathbf{l}_1(s) := \frac{\partial_t\mathbf{x}(s,0) - \mathbf{y}_1 - \sum_{i=1}^2 \mathbf{x}_{i1}\frac{s^i}{i!}}{s^3}, \quad \mathbf{l}_2(s) := \frac{\partial_t^2\mathbf{x}(s,0) - \mathbf{y}_2 - \sum_{i=1}^2 \mathbf{x}_{i2}\frac{s^i}{i!}}{s^3}.$$

Now we focus on solvability of the G^s constraints, $s = 0,1,2$. The solvability of $(k_1,k_2)^j$ for $k_1+k_2 = 2$ follows from Lemma 1. Our main goal is therefore to find the local G^s constraints $(k_1,k_2)^j$ of Section 2 for $3 \le k_1+k_2 \le 4$ in terms of the higher-order derivatives, $\mathbf{x}_{21}^j$, $\mathbf{x}_{12}^j$, $\mathbf{x}_{31}^j$, $\mathbf{x}_{13}^j$, $\mathbf{x}_{22}^j$ and for the reparameterizations' derivatives $\tau_{k_1k_2}^j$, $\sigma_{k_1k_2}^j$ for $i,k > 1$. We first consider the equations $(k_1,k_2)^j$ when $k_1+k_2 = 4$.

Lemma 3. *The equations $(k_1,k_2)^j$, where $k_1+k_2 = 4$, can always be solved in terms of $\mathbf{x}_{31}^j$, $\mathbf{x}_{13}^j$, $\mathbf{x}_{22}^j$.*

Proof. We have more vector-valued variables, $\mathbf{x}_{31}$, $\mathbf{x}_{13}$, $\mathbf{x}_{22}$, than constraints: (3,1), (2,2). Equation (3,1) expresses $\mathbf{x}_{31}^+$ in terms of $\mathbf{x}_{13}$ so that we can focus on solving (2,2) in terms of $\mathbf{x}_{13}$ and $\mathbf{x}_{22}$. Equation (2,2) can be arranged as

$$\mathbf{x}_{22}^+ = \sigma_{01}^2\mathbf{x}_{22} + 2\tau_{01}\sigma_{01}\mathbf{x}_{13} + f(\mathbf{y},\mathbf{x}_{11},\mathbf{x}_{12},\mathbf{x}_{21}), \tag{2,2}$$

where $f(\mathbf{y},\mathbf{x}_{11},\mathbf{x}_{12},\mathbf{x}_{12})$ is the collection of terms on the boundary or appearing in lower-order equations. Clearly, we can solve $n-1$ of these equations for $\mathbf{x}_{22}^+$. In general, this is all we can hope for since, for equal angles γ^j, the analysis in [Pet92] shows that the constraint matrix for solving (2,2) in terms of just $\mathbf{x}_{22}$ is rank-deficient by 1.

If the tangents do not form an X configuration, i.e. not all consecutive pairs of angles add to π, then at least one $\tau_{01}^j \neq 0$. Let $\tau_{01}^1 \neq 0$. Then, for any choice of $\mathbf{x}_{22}^1$, we can solve (2,2), for $\mathbf{x}_{22}^2, \ldots \mathbf{x}_{22}^n, \mathbf{x}_{13}^1$.

If the tangents form an X then the valence must be $n = 4$ and $\tau_{01} = 0$ and we solve the tangential component for $\mathbf{x}_{22}^2, \ldots \mathbf{x}_{22}^4, \mathbf{x}_{11}^1, \sigma_{12}$ (we may need σ_{12} to choose $w_{11} \neq 0$ for Lemma 6). ∎

Our analysis therefore focusses on the case of $k_1 + k_2 = 3$ derivatives. If the corresponding constraints are solvable then no second-order vertex enclosure constraint exists and a construction is always possible. However, the situation is not that simple as the next lemma shows.

Lemma 4. *The equations $(k_1, k_2)^j$, where $k_1 + k_2 = 3$, can be solved in terms of $\mathbf{x}_{12}^j$ and $\mathbf{x}_{21}^j$ if and only if the following $n \times n$ system of equations has a solution:*

$$\mathbf{Mh} = \mathbf{r}, \qquad \mathbf{M}_{jk} := \begin{cases} \sin\gamma, & k = j-1 \\ 2\sin(\gamma^- + \gamma), & k = j \\ \sin\gamma^-, & k = j+1 \\ 0, & else, \end{cases} \tag{17}$$

$$\begin{aligned} \mathbf{r}^j :=& \frac{1}{\sigma_{01}}(2\tau_{01}\sigma_{11} + 2\tau_{11}\sigma_{01} + \sigma_{02})\mathbf{x}_{11} \\ &+ \frac{1}{\sigma_{01}}((\tau_{01})^2\mathbf{y}_3 + 2\tau_{01}\tau_{11}\mathbf{y}_2 + 2\sigma_{01}\sigma_{11}\mathbf{y}_2^- + \tau_{02}\mathbf{y}_2 + \tau_{12}\mathbf{y}_1 + \sigma_{12}\mathbf{y}_1^-) \\ &+ \sigma_{01}(2\sigma_{11}^-\mathbf{x}_{11}^- + \tau_{01}^-\mathbf{y}_3^- + 2\tau_{11}^-\mathbf{y}_2^- + 2\tau_{21}^-\mathbf{y}_1^- + 2\sigma_{21}^-\mathbf{y}_1^{j-2}). \end{aligned} \tag{18}$$

Proof. We eliminate $\mathbf{x}_{21}$ by substituting $(2,1)^{j-1}$ into $(1,2)^j$ to obtain

$$\mathbf{x}_{12}^+ = \mathbf{x}_{12}^-\sigma_{01}^-\sigma_{01}^2 + 2\sigma_{01}\mathbf{x}_{12}\tau_{01} + g(\mathbf{y}, \mathbf{x}_{11}), \tag{19}$$

where $g(\mathbf{y}, \mathbf{x}_{11})$ collects the terms depending on $\mathbf{y}$ and $\mathbf{x}_{11}$. We divide both sides by $-\sigma_{01} := \frac{\sin\gamma}{\sin\gamma^-}$ to obtain for $\mathbf{h}^j := -\dfrac{\mathbf{x}_{12}^j}{\sin\gamma^{j-1}}$

$$\sin\gamma^j\mathbf{h}^{j-1} + 2\sin(\gamma^{j-1} + \gamma^j)\mathbf{h}^j + \sin\gamma^{j-1}\mathbf{h}^{j+1} = \frac{1}{\sigma_{01}}g(\mathbf{y}, \mathbf{x}_{11}) =: \mathbf{r}^j. \tag{20}$$

This is Equation (17). ∎

Although, generically, we can freely choose all $\tau_{k_1k_2}^j$ and $\sigma_{k_1k_2}^j$ for $k_1 + k_2 > 1$, rank deficiency of the matrix $\mathbf{M}$ could lead to an additional constraint on the boundary curves when we consider a *higher-order saddle point*. For a higher-order saddle point, $\mathbf{n} \cdot \mathbf{y}_k^j = 0$ for $k = 1, 2$ and this can force $\mathbf{n} \cdot \mathbf{x}_{11}^j = 0$ so that

$$\mathbf{n} \cdot \mathbf{r}^j = \frac{(\tau_{01})^2}{\sigma_{01}}\mathbf{n} \cdot \mathbf{y}_3 + \sigma_{01}\tau_{01}^-\mathbf{n} \cdot \mathbf{y}_3^-.$$

If $\ell \in \mathbb{R}^n$ is a left null-vector of $\mathbf{M}$, i.e. $\ell\mathbf{M} = \mathbf{0}$, then we obtain the *second-order vertex enclosure constraint*

$$\sum_j \tau_{01}^j \left(\frac{\tau_{01}^j \ell^j}{\sigma_{01}^j} + \sigma_{01}^{j+1}\ell^{j+1}\right)\mathbf{n}\cdot\mathbf{y}_3^j. \tag{21}$$

We therefore focus on the rank of $\mathbf{M}$. The next lemma partly characterizes $\mathtt{rank}(\mathbf{M})$ and hence explains in what cases a second-order vertex enclosure constraint *can* exists or where no second-order vertex enclosure constraint exists because $\mathbf{M}$ is of full rank.

Lemma 5 (rank of $\mathbf{M}$). *The rank of $\mathbf{M}$ is at least $n-2$. The matrix $\mathbf{M}$ is of full rank ($\mathtt{rank}(\mathbf{M}) = n$) if either all angles are equal, and $n \notin \{3,4,6\}$; or if all angles are less than $\pi/3$.*

Proof. Since all $\sin\gamma^j > 0$, we can solve (20) for $j = 1,\ldots,n-2$, i.e. the rank-deficiency in the general case is at most 2. Discrete Fourier analysis in [Pet92] shows $\mathbf{M}$ to be of full rank if all angles are equal, and $n \notin \{3,4,6\}$. If all angles are less than $\pi/3$ then the matrix is strictly diagonally dominant and therefore invertible. ∎

We will see below that, for $n = 4$ and equal angles, $\mathtt{rank}(\mathbf{M}) = 2$; and for $n = 5$, when three angles are $\pi/2$, then $\mathtt{rank}(\mathbf{M}) = 3$. Discrete Fourier analysis in [Pet92] showed $\mathtt{rank}(\mathbf{M}) = n-1$ if $n \in \{3,6\}$ and all angles are equal; and $\mathtt{rank}(\mathbf{M}) = n-2$ when all angles are equal and $n = 4$. In the general case, however, the analysis is more complex.

As for first-order vertex enclosure constraint, we can focus exclusively on the normal component of the constraints since, in the tangent plane, we can always choose τ_{11} and σ_{11} to solve (1,1) for an arbitrary choice of $\mathbf{x}_{11}^j$. Then we can use the tangent component of $\mathbf{x}_{11}$ and the free choice of σ_{02} to solve the tangent component of $(20)^1$ and $(20)^n$ (while $(20)^j$ is solved in terms of $\mathbf{x}_{12}^j$, $j = 2,\ldots,n-1$). Focussing on the normal component, we note that the right hand side simplifies to

$$\begin{aligned}\mathbf{n}\cdot\mathbf{r}^j =& \frac{1}{\sigma_{01}}(2\tau_{01}\sigma_{11} + 2\tau_{11}\sigma_{01} + \sigma_{02})\mathbf{n}\cdot\mathbf{x}_{11} + 2\sigma_{01}\sigma_{11}^-\mathbf{n}\cdot\mathbf{x}_{11}^- \\ &+\frac{1}{\sigma_{01}}((\tau_{01})^2\mathbf{n}\cdot\mathbf{y}_3 + (2\tau_{01}\tau_{11} + \tau_{02})\mathbf{n}\cdot\mathbf{y}_2) \\ &+\sigma_{01}\tau_{01}^-\mathbf{n}\cdot\mathbf{y}_3^- + 2(\sigma_{11} + \sigma_{01}\tau_{11}^-)\mathbf{n}\cdot\mathbf{y}_2^-.\end{aligned} \tag{22}$$

Lemma 6. *If $n = 3$ or $n = 4$, no second-order vertex enclosure constraint exists for any choice of γ^j.*

Proof. For $n = 3$, $\sin(\gamma^- + \gamma) = -\sin\gamma^+$ and $\mathbf{M}$ simplifies to

$$\mathbf{M} = \begin{bmatrix} -2\sin\gamma^2 & \sin\gamma^3 & \sin\gamma^1 \\ \sin\gamma^2 & -2\sin\gamma^3 & \sin\gamma^1 \\ \sin\gamma^2 & \sin\gamma^3 & -2\sin\gamma^1 \end{bmatrix}. \tag{23}$$

Since $0 < \sin\gamma^j \le 1$, multiples of $\ell := [1,1,1]$ are the only null-vectors of $\mathbf{M}$; that is, $\mathtt{rank}(\mathbf{M}) = 2$. We have a solution iff

$$\mathbf{r}^1 + \mathbf{r}^2 + \mathbf{r}^3 = \mathbf{0}. \tag{24}$$

If we choose $\tau_{kl}^j = \sigma_{kl}^j = 0$ for $k + l > 1$ then we have a solution since

$$\begin{aligned}\mathbf{r}^1 + \mathbf{r}^2 + \mathbf{r}^3 &= \sum_{j=1}^{3} \tau_{01}^j \left(\frac{\tau_{01}^j}{\sigma_{01}^j} + \sigma_{01}^{j+1}\right)\mathbf{y}_3^j \\ &= \sum_{j=1}^{3} \tau_{01}^j \frac{\sin(\gamma^- + \gamma) + \sin\gamma^+}{-\sin\gamma}\mathbf{y}_3^j = 0.\end{aligned}$$

That is, we can choose $\mathbf{x}_{12}^1$ freely and enforce all $(1,2)^j$ by choice of $\mathbf{x}_{12}^2$ and $\mathbf{x}_{12}^3$. Then $\mathbf{x}_{21}^j$ is uniquely determined by $(2,1)^{j-1}$ and all constraints for $k_1 + k_2 = 3$ hold.

If $n = 4$, the determinant of $\mathbf{M}$ is

$$D = \begin{vmatrix} 2\sin(\gamma^4+\gamma^1) & \sin\gamma^4 & 0 & \sin\gamma^1 \\ \sin\gamma^2 & 2\sin(\gamma^1+\gamma^2) & \sin\gamma^1 & 0 \\ 0 & \sin\gamma^3 & 2\sin(\gamma^2+\gamma^3) & \sin\gamma^2 \\ \sin\gamma^3 & 0 & \sin\gamma^4 & 2\sin(\gamma^3+\gamma^4) \end{vmatrix} \tag{25}$$

$$= \left(4\sin(\gamma^4+\gamma^1)\sin(\gamma^1+\gamma^2) + \sin\gamma^1\sin\gamma^3 - \sin\gamma^2\sin\gamma^4\right)^2 \tag{26}$$

$$= \left(3\sin\frac{\gamma^1-\gamma^2-\gamma^3+\gamma^4}{2}\sin\frac{\gamma^1+\gamma^2-\gamma^3-\gamma^4}{2}\right)^2 \tag{27}$$

$$= 9\sin^2(\gamma^2+\gamma^3)\sin^2(\gamma^1+\gamma^2). \tag{28}$$

The last equation holds because $\sum \gamma^j = 2\pi$. That is $D = 0$ if and only if $\gamma^1 + \gamma^2 = \pi$ and therefore $\gamma^3 + \gamma^4 = \pi$; or $\gamma^2 + \gamma^3 = \pi$ and therefore $\gamma^4 + \gamma^1 = \pi$. That is $D = 0$ if and only if at least one pair of tangents, $\mathbf{t}^1$,$\mathbf{t}^3$ or $\mathbf{t}^2$,$\mathbf{t}^4$, is parallel.

If $\gamma^1 + \gamma^2 = \pi$ and $\gamma^2 + \gamma^3 = \pi$, i.e. the tangents form an X then $\sin\gamma^j = s$, $j = 1,2,3,4$, for some scalar $0 < s \leq 1$. The matrix

$$\mathbf{M} = \begin{bmatrix} 0 & s & 0 & s \\ s & 0 & s & 0 \\ 0 & s & 0 & s \\ s & 0 & s & 0 \end{bmatrix} \tag{29}$$

is of rank 2 and has left null-vectors $[1,\ -c,\ -1,\ c]$ and $[-c,\ -1,\ c,\ 1]$ for any c, for example $c := 2\cos\gamma^4$. Without loss of generality, we choose $\ell_1 := [1,\ 0,\ -1,\ 0]$ and $\ell_2 := [0,\ -1,\ 0,\ 1]$. Since $\sigma_{01} = 1$ and $\tau_{01} = 0$ and, by (1,1)$^{j-1}$, $\mathbf{n}\cdot\mathbf{x}_{11}^- = -\mathbf{n}\cdot\mathbf{x}_{11}$

$$\mathbf{n}\cdot\mathbf{r}^j := (2\tau_{11} + \sigma_{02} - 2\sigma_{11}^-)\mathbf{n}\cdot\mathbf{x}_{11} + \tau_{02}\mathbf{n}\cdot\mathbf{y}_2 + 2(\sigma_{11} + \tau_{11}^-)\mathbf{n}\cdot\mathbf{y}_2^-. \tag{30}$$

Choosing, for example, $w_{11} \neq 0$ in (13), we can enforce $\mathbf{n}\cdot\mathbf{r}^j = 0$ by choice of σ_{02} and the constraints can be satisfied.

If $\gamma^1+\gamma^2 = \pi$ but $\gamma^2+\gamma^3 \neq \pi$ then $s_1 := \sin\gamma^2 = \sin\gamma^1$ and $s_4 := \sin\gamma^3 = \sin\gamma^4$ and hence

$$\mathbf{M} = \begin{bmatrix} 2\sin(\gamma^4+\gamma^1) & \sin\gamma^4 & 0 & \sin\gamma^1 \\ \sin\gamma^1 & 0 & \sin\gamma^1 & 0 \\ 0 & \sin\gamma^4 & -2\sin(\gamma^1+\gamma^4) & \sin\gamma^1 \\ \sin\gamma^4 & 0 & \sin\gamma^4 & 0 \end{bmatrix}. \tag{31}$$

For this $\mathbf{M}$, $\mathtt{rank}(\mathbf{M}) = 3$. Since $\sin(\gamma^4 + \gamma^1) = \cos\gamma^4 \sin\gamma^1 + \cos\gamma^1 \sin\gamma^4$,

$$\ell\mathbf{M} = 0 \quad \text{for } \ell := [1,\ -2\cos\gamma^4,\ -1,\ -2\cos\gamma^1]. \tag{32}$$

By (13) one $\mathbf{n} \cdot \mathbf{x}_{11}^j$ can be chosen freely and we can set

$$\ell\mathbf{n} \cdot \mathbf{r} = \mathbf{n} \cdot \mathbf{r}^1 - 2\cos\gamma^4 \mathbf{n} \cdot \mathbf{r}^2 - \mathbf{n} \cdot \mathbf{r}^3 - 2\cos\gamma^1 \mathbf{n} \cdot \mathbf{r}^4 = 0 \tag{33}$$

by judicious choice of σ_{02}^j. ∎

Our main result, however, proves that a second-order vertex enclosure constraint exists for a higher valence for some choice of γ.

Theorem 1 (second-order vertex enclosure constraint). *For $n = 5$ and some choice of γ^j, a second-order vertex enclosure constraint exists.*

Proof. For $n = 5$, we compute

$$\det \mathbf{M} = 18 \prod \sin(\gamma^j + \gamma^{j+1}). \tag{34}$$

Abbreviating $s_j := \sin\gamma^j$ and $c_j := \cos\gamma^j$ and assuming, without loss of generality that $\gamma_1 + \gamma_2 = \pi$ and therefore $\gamma_3 + \gamma_4 + \gamma_5 = \pi$, we get

$$\ell := [s_3, 2s_5(c_2 s_3/s_2 + c_3) - s_4, -s_5, s_2, -s_2]. \tag{35}$$

If we choose all $\mathbf{y}_2^j$ so that $\mathbf{n} \cdot \mathbf{y}_2^j = 0$ then (2) and Lemma 1 imply $\mathbf{n} \cdot \mathbf{x}_{11}^j = 0$. With $s_{j-1,j} := \sin(\gamma^{j-1} + \gamma^j)$, the second-order vertex enclosure constraint (21) has to hold (note again that $s_j > 0$ for all j):

$$\begin{aligned} 0 &= \sum_j \tau_{01}^j \Big(\frac{\tau_{01}^j \ell^j}{\sigma_{01}^j} + \sigma_{01}^{j+1}\ell^{j+1}\Big)\mathbf{n} \cdot \mathbf{y}_3^j \\ &= \sum_j \frac{s_{j-1,j}}{-s_j s_{j-1}}(s_{j-1,j}\ell^j + s_{j+1}\ell^{j+1})\mathbf{n} \cdot \mathbf{y}_3^j. \end{aligned} \tag{36}$$

Specifically, for $\gamma = \frac{\pi}{6}[3,3,2,2,2]$

$$\begin{aligned} [\ldots, s_j, \ldots] &= [\,1\ 1\ \tfrac{\sqrt{3}}{2}\ \tfrac{\sqrt{3}}{2}\ \tfrac{\sqrt{3}}{2}\,],\ [\ldots, c_j, \ldots] = [\,0\ 0\ \tfrac{1}{2}\ \tfrac{1}{2}\ \tfrac{1}{2}\,], \\ [\ldots, s_{j-1,j}, \ldots] &= [\,\tfrac{1}{2}\ 0\ \tfrac{1}{2}\ \tfrac{\sqrt{3}}{2}\ \tfrac{\sqrt{3}}{2}\,],\ \ell = [\,\tfrac{\sqrt{3}}{2}\ 0\ \tfrac{\sqrt{3}}{2}\ 1\ -1\,]. \end{aligned}$$

Then the second-order vertex enclosure constraint is

$$0 = [1,0,1,0,0][\ldots, \mathbf{n} \cdot \mathbf{y}_3^j, \ldots]^\mathsf{t} = \mathbf{n} \cdot \mathbf{y}_3^1 + \mathbf{n} \cdot \mathbf{y}_3^3. \tag{37}$$

That is, for the two terms corresponding to the curves with opposing tangents, $\mathbf{n} \cdot \mathbf{y}_3^1 = -\mathbf{n} \cdot \mathbf{y}_3^3$ has to hold. ∎

We note that the case $n = 5$ yields a doubly rank-deficient matrix $\mathbf{M}$ when $\gamma = \frac{\pi}{4}[2,2,2,1,1]$.

5 Higher Valences

Theorem 1 established the existence of a second-order vertex enclosure constraint. An explicit proof for valences $n \geq 6$ requires exhibiting the null-vector ℓ and hence a full understanding of the rank of $\mathbf{M}$ in its general form. We have not been able to establish the rank in generality. But we hazard a conjecture.

Conjecture 1. If, for some j both $|2\sin(\gamma^j+\gamma^{j-1})| < \sin\gamma^j+\sin\gamma^{j-1}$ and $|2\sin(\gamma^j+\gamma^{j-1})| < \sin\gamma^{j+1}+\sin\gamma^{j-2}$ then there is a choice of the remaining angles for which $\mathbf{M}$ is rank-deficient.

The conjecture draws on Lemma 5 which proves full rank when $\mathbf{M}$ is diagonally dominant. Above, we conjecture that when both the row and column of an index are not diagonally dominant then additional angles can be found so that the determinant of $\mathbf{M}$ is zero.

We conclude with some examples supporting the conjecture. The following choices of n angles γ^j, yield a matrix $\mathbf{M}$ with zero determinant:

Examples supporting Conjecture 1.

n	$[\ldots,\gamma^j,\ldots] =$	
6	$\frac{\pi}{6}[2,3,1,3,h,3-h]$,	$h := \frac{6}{\pi}\text{atan}\frac{2\sqrt{3}}{3} \approx 1.636886845$
7	$\frac{\pi}{6}[2,2,2,1,1,1,3]$,	
7	$\frac{\pi}{6}[3,2,1,2,2,h,2-h]$,	$h := \frac{6}{\pi}\text{atan}\frac{\sqrt{3}}{29} \approx 0.1139327031$
8	$\frac{\pi}{6}[2,2,1,1,1,1,h,4-h]$,	$h := -\frac{6}{\pi}\text{atan}\frac{483\sqrt{3}}{-147-672\sqrt{6}} \approx 0.8337394914$
12	$\frac{\pi}{12}[4,1,\ldots,1,h,11-h]$,	$h \approx 2.237657840$

6 Conclusion

We established the existence of a second-order vertex enclosure constraint that governs the admissibility of curve networks for G^2 interpolation by smooth patches. We fully analyzed the practically important cases of valence 3,4 and 5 and characterized the second-order vertex enclosure constraint for valence 5. In all other cases, lacking an exact characterization of the null-space of $\mathbf{M}$, Lemma 5 establishes bounds on the angle distribution that guarantee admissability of any curve network for G^2 interpolation. Conversely, we showed that a solution to the G^2 vertex constraints allows constructing a G^2 local network interpolation.

Acknowledgements. The work was supported in part by NSF grant CCF-0728797. Jianhua Fan helped to bring the G^2 vertex constraints into readable form.

References

[DS91] Du, W.-H., Schmitt, F.J.M.: G^1 smooth connection between rectangular and triangular Bézier patches at a common corner. In: Laurent, P.-J., Le Méhauté, A., Schumaker, L.L. (eds.) Curves and Surfaces, pp. 161–168. Academic Press, London (1991)

[DS92] Du, W.-H., Schmitt, F.J.M.: On the G^2 continuity of piecewise parametric surfaces. In: Lyche, S. (ed.) Mathematical Methods in CAGD II, pp. 197–207 (1992)

[GH95] Grimm, C.M., Hughes, J.F.: Modeling surfaces of arbitrary topology using manifolds. In: Computer Graphics. Annual Conference Series, vol. 29, pp. 359–368 (1995)

[Gre74] Gregory, J.A.: Smooth interpolation without twist constraints. In: Barnhill, R.E., Riesenfeld, R.F. (eds.) Computer Aided Geometric Design, pp. 71–87. Academic Press, London (1974)

[Hah89] Hahn, J.: Filling polygonal holes with rectangular patches. In: Theory and practice of geometric modeling (Blaubeuren, 1988), pp. 81–91. Springer, Berlin (1989)

[Her96] Hermann, T.: G^2 interpolation of free form curve networks by biquintic Gregory patches. Computer Aided Geometric Design 13, 873–893 (1996)

[HLW99] Hermann, T., Lukács, G., Wolter, F.E.: Geometrical criteria on the higher order smoothness of composite surfaces. Computer Aided Geometric Design 17, 907–911 (1999)

[HPS09] Hermann, T., Peters, J., Strotman, T.: A geometric criterion for smooth interpolation of curve networks. In: Keyser, J. (ed.) SPM 2009: 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling, pp. 169–173. ACM, New York (2009)

[KP09] Karčiauskas, K., Peters, J.: Guided spline surfaces. Computer Aided Geometric Design 26(1), 105–116 (2009)

[LS08] Loop, C.T., Schaefer, S.: G^2 tensor product splines over extraordinary vertices. Comput. Graph. Forum 27(5), 1373–1382 (2008)

[MW91] Miura, K.T., Wang, K.K.: C^2 Gregory patch. In: Post, F.H., Barth, W. (eds.) EUROGRAPHICS 1991, pp. 481–492. North-Holland, Amsterdam (1991)

[PBP02] Prautzsch, H., Boehm, W., Paluzny, M.: Bézier and B-Spline Techniques. Springer, Heidelberg (2002)

[Pet91] Peters, J.: Smooth interpolation of a mesh of curves. Constructive Approximation 7, 221–246 (1991)

[Pet92] Peters, J.: Joining smooth patches at a vertex to form a C^k surface. Computer-Aided Geometric Design 9, 387–411 (1992)

[Pet02] Peters, J.: Geometric continuity. In: Handbook of Computer Aided Geometric Design, pp. 193–229. Elsevier, Amsterdam (2002)

[Pra97] Prautzsch, H.: Freeform splines. Computer Aided Geometric Design 14(3), 201–206 (1997)

[Rei98] Reif, U.: TURBS—topologically unrestricted rational B-splines. Constructive Approximation 14(1), 57–77 (1998)

[Ye97] Ye, X.: Curvature continuous interpolation of curve meshes. Computer Aided Geometric Design 14(2), 169–190 (1997)

[YZ04] Ying, L., Zorin, D.: A simple manifold-based construction of surfaces of arbitrary smoothness. ACM TOG 23(3), 271–275 (2004)

Computing the Distance between Canal Surfaces

Yanpeng Ma[1], Changhe Tu[1], and Wenping Wang[2]

[1] School of Computer Science and Technology
Shandong University, Jinan, China
chtu@sdu.edu.cn
[2] University of Hong Kong

Abstract. A canal surface is the envelope of a one-parameter set of moving spheres. We present an accurate and efficient method for computing the distance between two canal surfaces. First, we use a set of cone-spheres to enclose a canal surface. A cone-sphere is a surface generated by sweeping a sphere along a straight line segment with the radius of the sphere changing linearly; thus it is a truncated circular cone capped by spheres at the two ends. Then, for two canal surfaces we use the distances between their bounding cone-spheres to approximate their distance; the accuracy of this approximation is improved by subdividing the canal surfaces into more segments and use more cone-spheres to bound the segments, until a pre-specified threshold is reached. We present a method for computing tight bounding cone-spheres of a canal surface, which is an interesting problem in its own right. Based on it, we present a complete method for efficiently computing the distances between two canal surfaces using the distances among all pairs of their bounding cone-spheres. The key to its efficiency is a novel pruning technique that can eliminate most of the pairs of cone-spheres that do not contribute to the distance between the original canal surfaces. Experimental comparisons show that our method is more efficient than Lee et al's method [13] for computing the distance between two complex objects composed of many canal surfaces.

Keywords: canal surface, distance computation, cone-spheres, bounding volume, distance interval.

1 Introduction

A canal surface is the envelope of a one-parameter set of spheres with radii $r(t) > 0$ and centers $P(t)$, $t \in [0, 1]$. It can be regarded as a surface generated by sweeping a sphere of radius $r(t)$ along $P(t)$, called the *center curve*. Canal surfaces are widely used in the CAD/CAM, computer graphics, computer games and animations. Research on the canal surface abounds, including rendering[10,17,21], modeling [8,14,5], parameterization [3,4], etc.

Distance computation between two objects refers to computing the distance between two disjoint objects. It is a major research topic in CAD/CAM, NC verification, robotics, computer animation, and haptic rendering. There are many

B. Mourrain, S. Schaefer, and G. Xu (Eds.): GMP 2010, LNCS 6130, pp. 88–103, 2010.

methods for computing the distance between two objects, with the majority of these methods for polyhedral models (e.g. [1,6,7,11,12,15,18]). Distance computation for free form surfaces is more difficult [2,19]. Some straightforward algorithms solve a set of polynomial equations, which can be quite time consuming to solve.

Using line geometry, Sohn et al [20] reformulate the distance computation as a simple instance of a surface-surface intersection problem, and present an approach to computing the distance between two ellipsoids or the distance between an ellipsoid and a simple surface, such as a cylinder, cone, and torus. Johnson and Cohen [6] present an approach using the convex hulls of control nets–two closest NURBS patches are detected and their minimum distance is computed by recursive subdivision to the NURBS patches. Kim et al [9] compute the distance between a canal surface and a simple surface, such as a cylinder, cone, or torus, by reducing the distance computation to solving a polynomial equation in one variable, which can be computed quickly.

We study the distance computation between two canal surfaces. Throughout we will assume that $P(t)$ and $r(t)$ are rational functions. Several methods have been proposed for computing the distance between two canal surfaces recently. Chen et al [2] shrink or grow one surface to touch another one, and obtain equations for calculating one of the two closest points. It require that the two surfaces do not intersect and at least one must be implicit. In [13], taking a canal surface as one parameter family of spheres, Lee et al reduce the distance computation to computing the minimum distance between two moving spheres. Both methods need to solve a system of polynomial equations. The degree of these equations are usually very high and therefore time-consuming to solve even when the sphere center $P(t)$ and radius $r(t)$ have moderate degrees.

We propose a new method for computing the distance between two disjoint canal surface. Our contributions are:

- We propose to use *cone-spheres* as bounding volumes of canal surfaces and present a method for computing a set of cone-spheres to tightly bound a canal surface. Computing a cone-sphere bounding volume of canal surfaces is an important problem in its own right, since such tight bounding volumes can be applied to distance computation, collision detection, and ray-tracing, etc. that involve canal surfaces.
- We present a robust algorithm for computing the distance between two canal surfaces based on iterative subdivision of the canal surfaces and computation of the bounding cone-spheres of the resulting canal surface segments. For speeding up the search for the distance we propose an effective pruning technique based on the notion of *distance intervals* to eliminate most of the cone-spheres pairs and the canal surface sections contained therein that do not contribute to the distance between the two canal surfaces.

The remainder of this paper is organized as follows. In Section 2 we consider distance computation between two cone-spheres. In Section 3, a method is presented for computing tight bounding cone-spheres of a given canal surface. In

Section 4, we describe our algorithm for computing the distance between two canal surfaces, based on recursive subdivision and pruning using distance intervals. In Section 5 we give experimental results and comparisons with Lee's method [13]. We conclude the paper in Section 6.

2 Preliminaries

2.1 Cone-Spheres and Their Distance Computation

As is shown in Fig. 1 and Fig. 2, a *cone-sphere* consists a truncated right circular cone (drawn with red lines) and two spheres tangent to it at its two ends [16]. It can also be viewed as a simple canal surface generated by a sphere moving along a straight line segment with its radius varying linearly.

The distance between two disjoint cone-spheres is the Euclidean distance between two closest points that come from the surfaces of the two cone-spheres, respectively. This distance can be realized in three different cases as shown in Fig. 2. Note that the distance between two intersecting cone-spheres is zero.

The distance between two cone-spheres can be computed as follows. Suppose that we have two cone-spheres $\mathcal{C}_p$ and $\mathcal{C}_q$, generated by moving spheres $\mathcal{O}_p(u)$ $(u \in [0,1])$ and $\mathcal{O}_q(v)$ $(v \in [0,1])$, respectively. Then the center curve and radius function of $\mathcal{O}_p$ are

$$P(u) = P_1 + u(P_2 - P_1) \tag{1}$$

$$r_p(u) = r_{p1} + u(r_{p2} - r_{p1}) \tag{2}$$

where P_1 and P_2 are the centers of the spheres at the two ends of the cone-sphere $\mathcal{C}_p$, and r_{p1} and r_{p2} the radii of these two spheres. Similarly, the center curve and radius function of $\mathcal{O}_q$ are

$$Q(v) = Q_1 + v(Q_2 - Q_1) \tag{3}$$

$$r_q(v) = r_{q1} + v(r_{q2} - r_{q1}) \tag{4}$$

Here, $0 \le u \le 1$ and $0 \le v \le 1$.

The distance between the two moving spheres $\mathcal{O}_p$ and $\mathcal{O}_q$ is

$$d(u,v) = ||P(u) - Q(v)|| - r_p(u) - r_q(v) \tag{5}$$

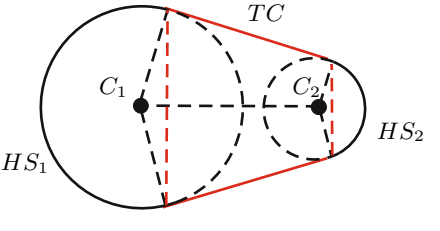
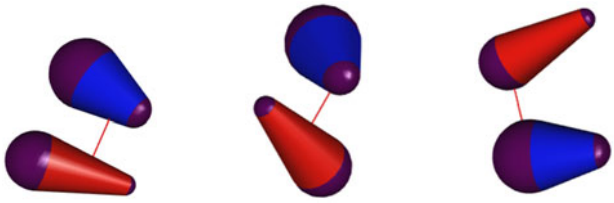


Fig. 1. An example of cone-spheres



(a) (b) (c)

Fig. 2. Three cases of distance between two cone-spheres, the distance formed by (a) cone-cone, or (b) cone-sphere, or (c) sphere-sphere

Then the distance of $\mathcal{C}_p$ and $\mathcal{C}_q$ is the minimum of $d(u,v)$ over $(u,v) \in [0,1]^2$. We first fix some notation to simplify derivation. We denote

$$a = ||\overrightarrow{P_1Q_1}||^2, \quad b = ||\overrightarrow{P_1P_2}||^2, \quad c = ||\overrightarrow{Q_1Q_2}||^2, \quad d = \overrightarrow{P_1Q_1} \cdot \overrightarrow{P_1P_2},$$
$$e = \overrightarrow{P_1Q_1} \cdot \overrightarrow{Q_1Q_2}, \quad f = \overrightarrow{P_1P_2} \cdot \overrightarrow{Q_1Q_2}, \quad g = r_{p2} - r_{p1}, \quad h = r_{q2} - r_{q1}.$$

As illustrated in Fig. 2, the distance between $\mathcal{C}_p$ and $\mathcal{C}_q$ can be realized in three cases: 1) between two cones; 2) between a cone and a sphere; and 3) between two spheres. These cases will be considered below, one by one.

Case 1: cone vs. cone– In this case we compute the minimum of $d(u,v)$ with $0 < u < 1$ and $0 < v < 1$. This entails seeking the zeros of the partial derivatives of $d(u,v)$, that is

$$\frac{\partial d(u,v)}{\partial u} = \frac{bu - d - fv}{\sqrt{a + bu^2 + cv^2 - 2du + 2ev - 2fuv}} - g = 0$$

$$\frac{\partial d(u,v)}{\partial v} = \frac{cu - e - fv}{\sqrt{a + bu^2 + cv^2 - 2du + 2ev - 2fuv}} - h = 0$$

This system of equations can be written as

$$v = Au - B \tag{6}$$

$$l_1 u^2 + m_1 u + n_1 = 0 \tag{7}$$

where

$$A = \frac{bh + fg}{fh + cg}, \quad B = \frac{dh + eg}{fh + cg},$$
$$l_1 = (b - Af)^2 - g^2(b + A^2c - 2Af),$$
$$m_1 = -2(b - Af)(d - Bf) - 2g^2(Ae + Bf - d - ABc),$$
$$n_1 = (d - Bf)^2 - g^2(a + B^2c - 2Be).$$

The local minima of $d(u,v)$ can be found from Eqs. (7) and (6) by solving a quadratic equation. Assuming that the two cone-spheres are disjoint, then by

geometric observation, the only relevant solution is the one with exactly one local minimizer (u_0, v_0) of $d(u, v)$ in $(0, 1)$.

Case 2: cone vs. sphere– In this case, the distance between the two cone-spheres is realized between one capping sphere of one cone-sphere and the truncated cone of the other cone-sphere. That is, $d(u, v)$ reaches it minimum with u or v being 0 (or 1). Without loss of generality, assume $v = 0$. Then we can find the minimum by solving the equation

$$\frac{\partial d(u,0)}{\partial u} = \frac{bu - d}{\sqrt{a + bu^2 - 2du}} - g = 0$$

It can be re-written as

$$l_2 u^2 + m_2 u + n_2 = 0 \tag{8}$$

where $l_2 = b(b - g^2)$, $m_2 = 2d(g^2 - b)$, and $n_2 = d^2 - ag^2$. Note that only a root u_0 in $(0, 1)$ is relevant in this case.

Case 3: sphere vs. sphere– Distances between two capping spheres are the values of $d(0, 0), d(0, 1), d(1, 0)$ or $d(1, 1)$, which can easily be evaluated and compared to give the minimum values.

Finally, the distance between the two cone-spheres is computed as the smallest of all the minima of $d(u, v)$ in the three cases above.

3 Computing Bounding Cone-Spheres

In this section we discuss how to compute bounding cone-spheres of a canal surface. Fig. 3(a) shows a canal surface from Q_1 to Q_2. Using the line segment

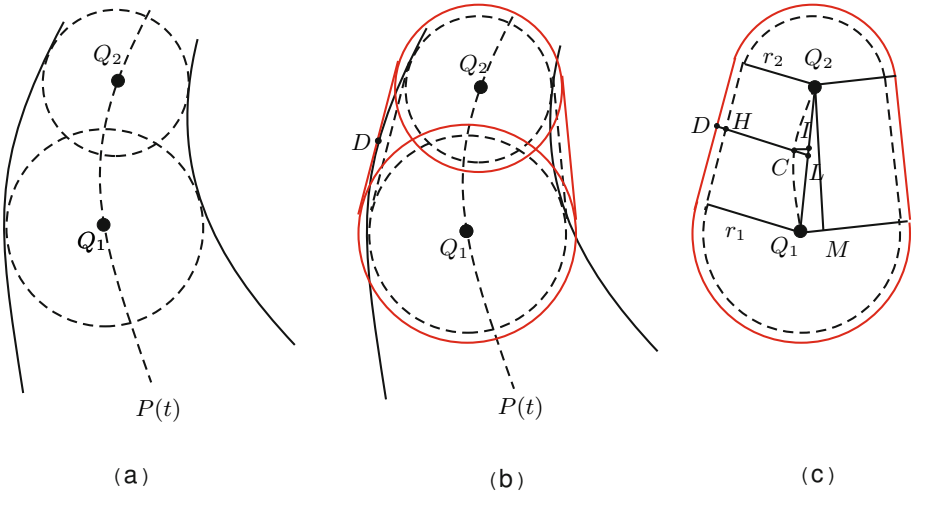


Fig. 3. Computation of the bounding cone-sphere of a segment of canal surface

$\overline{Q_1Q_2}$ and the two spheres centered at Q_1 and Q_2 we obtain a cone-sphere, as shown by the dashed outline in Fig. 3(b). Our task is to enlarge (that is, offset) this cone-sphere to obtain another cone-sphere, as shown by the red, solid outline in Fig. 3(b), that encloses the canal surface.

Now we are going to determine the necessary offset distance to obtain a bounding cone-sphere. Refer to Fig. 3(c). Suppose that $C = P(t)$ is a point on the center curve where the moving sphere touches the bounding cone-sphere at D. Let the line CD intersect the original cone-sphere at H and the line segment $\overline{Q_1Q_2}$ at L. Let I be the closest point on $\overline{Q_1Q_2}$ to C. Then $\overrightarrow{CI}$ is perpendicular to $\overline{Q_1Q_2}$.

The offset distance $\|HD\|$ can be expressed as

$$\|\overrightarrow{HD}\| = \|\overrightarrow{CD}\| + \|\overrightarrow{CL}\| - \|\overrightarrow{HL}\| \tag{9}$$

where

$$\|\overrightarrow{CD}\| = r(t) \tag{10}$$

$$\|\overrightarrow{HL}\| = r_2 + \frac{\|\overrightarrow{Q_2L}\|}{\|\overrightarrow{Q_1Q_2}\|}(r_1 - r_2) \tag{11}$$

$$\|\overrightarrow{CL}\| = \frac{\|\overrightarrow{CI}\|}{\sin\alpha} = \frac{\sqrt{\overrightarrow{CI}\cdot\overrightarrow{CI}}}{\sin\alpha} \tag{12}$$

with $\alpha = \angle CLI$. Furthermore, we have

$$\overrightarrow{CI} = \overrightarrow{CQ_2} - \frac{\overrightarrow{Q_2Q_1}\cdot\overrightarrow{CQ_2}}{\|\overrightarrow{Q_2Q_1}\|^2}\overrightarrow{Q_2Q_1} \tag{13}$$

To reduce the degree of the equation that we have to solve, in Eqn. (11) we replace $\|\overrightarrow{Q_2L}\|$ by $\|\overrightarrow{Q_2I}\| = \frac{\overrightarrow{Q_2C}\cdot\overrightarrow{Q_2Q_1}}{\|\overrightarrow{Q_2Q_1}\|}$ to obtain an approximation of $\|\overrightarrow{HL}\|$ as

$$G_{HL} = r_2 + \frac{\|\overrightarrow{Q_2I}\|}{\|\overrightarrow{Q_1Q_2}\|}(r_1 - r_2) \tag{14}$$

For most practical cases the difference between $\|\overrightarrow{Q_2L}\|$ by $\|\overrightarrow{Q_2I}\|$ is small – it is zero when $r_1 = r_2$ or when the center curve coincide with $\overline{Q_1Q_2}$. Furthermore, by an elementary argument, it can be shown that $G_{HL}(t) \le \|\overrightarrow{HL}\|$ for any $t \in [0,1]$. Then, defining

$$f(t) = \|\overrightarrow{CD}\| + \|\overrightarrow{CL}\| - G_{HL}(t) \tag{15}$$

we have $f(t) \ge \|\overrightarrow{HD}\|$ for any $t \in [0,1]$. Therefore, the maximum of $f(t)$ in $[0,1]$ can be used as the offset distance to obtain a bounding cone-sphere.

To compute the maximum of $f(t)$, we consider the equation $f'(t) = 0$, which is found to be

$$\frac{1}{\sin\alpha}\cdot\frac{(\overrightarrow{CI}\cdot\overrightarrow{CI})'}{2\sqrt{\overrightarrow{CI}\cdot\overrightarrow{CI}}} + r'(t) - \frac{(\|\overrightarrow{Q_2I}\|)'}{\|\overrightarrow{Q_1Q_2}\|}(r_1 - r_2) = 0$$

Clearing the square root yields

$$[r'(t) - \frac{(||\overrightarrow{Q_2I}||)'}{||\overrightarrow{Q_1Q_2}||}(r_1 - r_2)]^2 = \frac{1}{\sin^2\alpha} \cdot \frac{[(\overrightarrow{CI} \cdot \overrightarrow{CI})']^2}{4\,\overrightarrow{CI} \cdot \overrightarrow{CI}}$$

which is

$$4\sin^2\alpha\,[r'(t) - \frac{(||\overrightarrow{Q_2I}||)'}{||\overrightarrow{Q_1Q_2}||}(r_1 - r_2)]^2\,(\overrightarrow{CI} \cdot \overrightarrow{CI}) - [(\overrightarrow{CI} \cdot \overrightarrow{CI})']^2 = 0 \qquad (16)$$

We solve this equation to find the maximum of $f(t)$ and use it as the offset distance to obtain a bounding cone-sphere. If $P(t)$ and $r(t)$ are polynomials with $\deg(P(t)) = m$ and $\deg(r(t)) = n$, then the degree of this equation is $\max\{4m-2, 2m+2n-2\}$. For example, if both $P(t)$ and $r(t)$ are cubic, Eqn. (16) has degree 10.

When we compute the bounding cone-sphere, we approximate $\|\overrightarrow{HD}\|$ by $f(t)$ to speed up the computation. This makes the bounding cone-sphere a little bigger than the exact one, but the difference between them is quite tiny. Fig. 4 shows the comparison of the two bounding cone-spheres. The one drawn with solid red line is the bounding cone-sphere computed by $f(t)$ and the one drawn with dashed black line is the exact one computed by $\|\overrightarrow{HD}\|$. From the figure, we can hardly distinguish them. In this example, we set $r_1 = 3.0$ and $r_2 = 5.5$. Then the offset distance computed by $f(t)$ is 4.384 and the offset distance computed by $\|\overrightarrow{HD}\|$ is 4.365. Hence, using $f(t)$ for computing the offset distance yields a safe, accurate approximation.

A key assumption in the above argument is that the point L lies within $\overline{Q_1Q_2}$; for otherwise it is not guaranteed that canal surface is enclosed by the intended

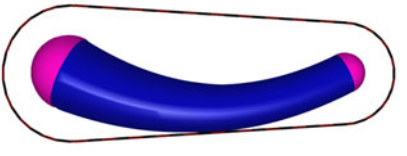


Fig. 4. The tiny difference between approximating and exact computed bounding cone-spheres

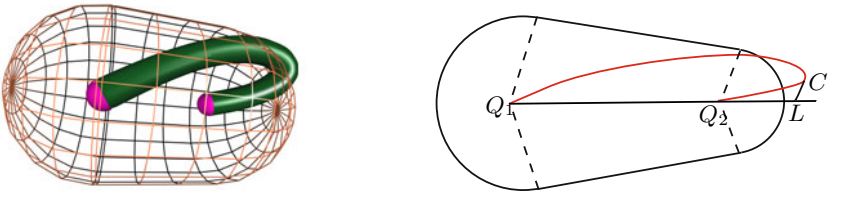


Fig. 5. A case that a bounding cone-sphere doesn't enclose the surface completely

bounding cone-sphere constructed above. See such an example in Fig. 5. To prevent such abnormal cases from happening, intuitively we should require that the center curve $P(t)$ do not vary widely between Q_1 and Q_2. Specifically, it can easily seen that such abnormal case does not occur if the angle between the tangent vector $P'(t)$ of $P(t)$ and $\overrightarrow{Q_1Q_2}$ is not greater than $\pi/2$ for any $t \in [0, 1]$. We will refer to this condition as the *monotonic condition* for the center curve $P(t)$.

4 Computing the Distance between Two Canal Surfaces

In this section, we will present the complete algorithm for computing distance between two canal surfaces $\mathcal{S}_1$ and $\mathcal{S}_2$. For each of these two canal surfaces, we perform initial subdivision that cuts its center curve into some segments, and consequently, each canal surfaces is segmented into a sequence of canal surfaces, called *sections*. For each section of $\mathcal{S}_1$ and $\mathcal{S}_2$, we compute its *bounding cone-sphere*, following the discussion in Section 3. Hence we obtain two sets of bounding cone-spheres, bounding $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively. Then we compute the distances between all possible pairs of these bounding cone-spheres and deduce which pairs may contain canal surface sections that realize the distance between the original canal surfaces $\mathcal{S}_1$ and $\mathcal{S}_2$; the other pairs of cone-spheres, which are called *irrelevant pairs*, will be discarded without further processing.

To improve the accuracy of distance computation for $\mathcal{S}_1$ and $\mathcal{S}_2$, those remaining canal surface sections can be further subdivided into even smaller subsections, and then their bounding cone-spheres are computed and analyzed. These subdivisions are performed recursively until a pre-specified threshold is reached. In the following we will discuss in detail the criteria for guiding the subdivision and explain how to prune most of irrelevant pairs at each level of subdivision for efficient implementation.

4.1 Segmentation of a Canal Surface

Let $\mathcal{S}$ be a canal surface with the center curve $P(t)$ and radius $r(t)$. We segment $\mathcal{S}$ by subdividing the curve $P(t)$ into a series of curve segments. Each curve segment corresponds a surface section. There are two stages of segmentation in our algorithm. At the beginning, a possibly long and winded canal surface needs to be segmented into a number of sufficient short sections so each can be properly bounded by a cone-sphere; this is called *initial segmentation*. The second stage is called *recursive subdivision* where a canal surface section is cut into two smaller sections to improve the error of approximation.

For initial segmentation, we propose the following two criteria:

- *(1) Radius monotonicity*: We require the radius function to change monotonically on each canal surface section, since such a section would be more compatible with a cone-sphere in shape, whose radius function is always monotonic, therefore tending to be bounded more tightly. So we solve the equation $r'(t) = 0$ to find all points on the center curve $P(t)$ where the radius

attains local minima or maxima and use them as cutting points to yield the initial segmentation.

– *(2) Bending control*: It is clear that a bounding cone-sphere will be quite loose if the canal surface to be bounded bends too much. The bending of the surface can be characterized by the *deviation angle* of the center curve $P(t)$, $t \in [0,1]$. Let $Q_1 = P(0)$ and $Q_2 = P(1)$. The *deviation angle* of $\mathcal{S}$ is defined to be the angle between the tangent $P'(t)$ of the center curve and line segment $\overline{Q_1Q_2}$. Then another consideration in initial segmentation is the control of the deviation angle.

  As discussed in Section 3, to ensure the validness of our computation of the bounding cone-sphere of $\mathcal{S}$, we require that the deviation angle of $\mathcal{S}$ be less $\pi/2$. In fact, it can be shown that if the deviation angle of $\mathcal{S}$ is less than $\pi/4$, then the deviation of any subsection of $\mathcal{S}$ that may result from the subsequent recursive subdivision will have its deviation angle less than $\pi/2$, thus ensuring that its bounding sphere can properly be computed by the method in Section 3. Due to space limitation, we will skip the proof of this result.

  Based on the above analysis, for initial segmentation we will enforce *bending control* by subdividing a canal surface section in half if its deviation angle is greater $\pi/4$.

After initial segmentation, two input canal surfaces, represented as two sets of canal surface sections together with their bounding cone-spheres, will be subjected to analysis for distance computation as will be described in the next section. During that process, if the error of computation needs to be reduced, further recursive subdivision of the canal surface sections has to be performed. When a canal surface section needs to be subdivided at this stage, to optimally reduce the approximation error, we choose the cutting point to be where the moving sphere of the canal surface touches the boundary of the existing bounding cone-sphere. (See Fig. 3).

4.2 Pruning Irrelevant Pairs

At any stage of subdivision, let $\mathcal{K}_1$ and $\mathcal{K}_2$ denote the two sets of bounding cone-spheres for the two canal surfaces $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively. Then we examine all pairs of bounding cone-spheres (C_1, C_2), $C_1 \in \mathcal{K}_1$ and $C_2 \in \mathcal{K}_2$, to deduce about the distance between $\mathcal{S}_1$ and $\mathcal{S}_2$. Typically, the distance between $\mathcal{S}_1$ and $\mathcal{S}_2$ is only realized by a smaller number of the pairs of bounding cone-spheres, and these pairs need to be subdivided further to improve the approximation error. The others pairs, called *irrelevant pairs*, cannot contribute the distance computation and thus should be discarded. Hence, we need an effective scheme to eliminate as many irrelevant pairs as possible.

The distance of a pair of bounding cone-spheres is only a lower bound of the distance between the canal surfaces contained therein, so it does not provide sufficient information for deciding which pair should be discarded as irrelevant pairs. Consider the example in Fig. 6. Here, The canal surface below is divided

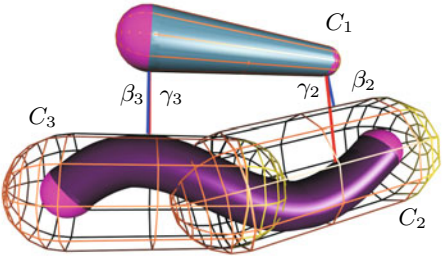


Fig. 6. Distance interval $[\beta, \gamma]$

into two surface sections bounded by two bounding cone-spheres C_2 and C_3. The other canal surface is just one section, bounded by a bounding cone-sphere C_1. So we have two pairs (C_1, C_2) and (C_1, C_3) to compare. Although the distance of (C_1, C_2) (β_2 in Fig. 6) is smaller than that of (C_1, C_3) (β_3), we cannot conclude (C_1, C_3) is irrelevant, because the distance between the canals surfaces in (C_1, C_2) is bigger than the distance between the canals surfaces in (C_1, C_3). That is, discarding (C_1, C_3) would lead to an erroneous result.

We resolve this issue by assigning a *distance interval* to each pair of bounding cone-spheres (C_1, C_2), and eliminate irrelevant pairs based on comparison of all distance intervals. For a pair of bounding cone-spheres, let β be the distance between the two cone-spheres and let γ be the distance between any two spheres respectively from the two family of moving spheres generating the two canal surface bounded by C_1 and C_2. Then the $[\beta, \gamma]$ is called a *distance interval* associated with (C_1, C_2). Let d be the distance between the two canal surfaces bounded by C_1 and C_2. Then, clearly, $\beta \leq d \leq \gamma$. Fig. 6 shows the distance interval $[\beta, \gamma]$ for the two pairs of cone-spheres (C_1, C_2) and (C_1, C_3).

In our implementation, from each pair of bounding cone-spheres, we choose the spheres in the definition of γ as follows. When we compute the closest distance between the two cone-spheres, we easily find the parameters $(u_0, v_0) \in [0, 1]^2$ that give the minimum of the distance function $d(u, v)$ of the two cone-spheres, given in Eqn. (5). Then we substitute u_0 and v_0 into the moving spheres defining the canal surfaces $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively, to get two spheres, and designate the distance between these two spheres to be γ. This selection is motivated by the need to make the upper bound γ as little as possible so to help eliminate more irrelevant pairs of bounding cone-spheres.

Distance intervals can be used to eliminate irrelevant pairs as follows. Consider two pairs of bounding cone-spheres, denoted (C_1, C_2) and (C_1', C_2'), with C_1 and C_1' bounding sections from a canal surface $\mathcal{S}_1$ and C_2 and C_2' bounding sections from a canal surface $\mathcal{S}_2$. Let $[\beta, \gamma]$ be a distance interval of (C_1, C_2) and $[\beta', \gamma']$ a distance interval of (C_1', C_2'). If $\gamma < \beta'$, then the distance between the two canal surface sections bounded by (C_1', C_2') will not be the eventual distance between $\mathcal{S}_1$ and $\mathcal{S}_2$, thus (C_1', C_2') should be pruned. Similarly, if $\gamma' < \beta$, then (C_1, C_2) be pruned. When $[\beta, \gamma] \bigcap [\beta', \gamma'] \neq \emptyset$, either of (C_1, C_2) and (C_1', C_2') may contain

canal surface sections that might realize the distance between $\mathcal{S}_1$ and $\mathcal{S}_2$, so both need to be retained for further processing.

4.3 Algorithm

Our complete algorithm is described below.

Algorithm 1 (Computing the distance between two canal surfaces)
INPUT: *Two canal surfaces $\mathcal{S}_1$ and $\mathcal{S}_2$ with radii $r_i(t)$, and centers $P_i(t)$, $i = 1..2$.*
OUTPUT: *The distance γ_min between $\mathcal{S}_1$ and $\mathcal{S}_2$.*

- *Step 1: Initialization.*
  - *Segment $\mathcal{S}_1$ and $\mathcal{S}_2$ into consecutive surface sections respectively. Compute corresponding bounding cone-spheres of these sections, we obtained two sets of bounding cone-spheres for $\mathcal{S}_1$ and $\mathcal{S}_2$. Denote them $\mathcal{K}_1$ and $\mathcal{K}_2$ respectively.*
  - *Combine two bounding cone-spheres, each from $\mathcal{K}_1$ and $\mathcal{K}_2$ respectively to form a pair of bounding cone-spheres. Enumerating all possible combinations of the members in $\mathcal{K}_1$ and $\mathcal{K}_2$. For each pair of the bounding cone-spheres, compute their distance interval $[\beta, \gamma]$. Taking each combination of two bounding cone-spheres as a unity, and define it by u.*
  - *Define a set of u as U to record the candidates of the most closed pair that forms the distance between two input canal surfaces. Add all the pairs of the bounding cone-spheres computed above into U.*
  - *Randomly initialize γ_min to be the γ of the distance interval of a pair in U.*
- *Step 2: For each pair of bounding cone-spheres u in the set U. Suppose the distance interval is $[\beta, \gamma]$, do the followings on u.*
  - *If the $\beta > \gamma\_min$, then remove u from U;*
  - *Otherwise if $\gamma < \gamma\_min$, assign γ to γ_min, check the other pairs whose $\beta > \gamma\_min$, remove them from U;*
- *Step 3: If the set U is empty, return the value γ_min as the distance between the two canal surface; otherwise, find the pair among all the pairs in U with the smallest value γ in its distance interval, again denote the selected pair by u, pick it out from U, go to step 4.*
- *Step 4: If both the two surface sections bounded by the cone-spheres in u are bounded within a pre-specified threshold ε, discard u and go back to step 3; otherwise, subdivide the surface section whichever is not bounded tightly by its bounding cone-sphere into two parts. Compute the bounding cone-sphere for every segment of the surface sections, and combine the new bounding cone-spheres with those of the other surface section one by one to form new pairs of bounding cone-spheres. Compute the distance intervals $[\beta, \gamma]$ of the new pairs, add them into U. Go to step 2.*

5 Experimental Results

In this section we present the experimental results for computing the distance between two objects modeled with canal surfaces. All experiments were run on the PC with 2.33 GHz Core(TM) 2 Duo CPU and 2 GB memory.

The Table 1 shows how the subdivision level is co-related to the change of the pre-specified approximation tolerance ε. The second column of Table 1 shows the depth of bounding tree, and the third one is the number of cone-sphere pairs which were tested. All the data in Table 1 are obtained by the example in Fig.7.(c). When we reduce the ε rapidly, the depth of the bounding tree increases slowly. It means that our bounding cone-spheres converge rapidly. Table 1 shows that the time and number of testing cone-sphere pairs also increase slowly with rapidly reducing ε.

In all the following examples, we let the pre-specified threshold ε be 10^{-5}.

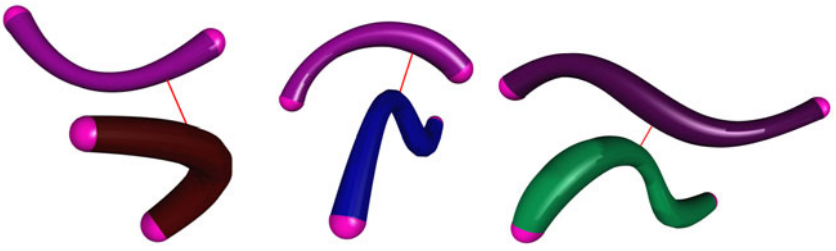
We use a group of tests to compare the run time performances and the accuracies of our method and that of the method by Lee et al [13]. In these tests the two methods are used to compute the distances between two canal surfaces with center curves $P(t)$ and radius functions $r(t)$ of different degrees. The timing results are shown in Table 2. Some of these examples are shown in Fig. 7. In the first column in Table 2, (i, j) stands for the degree of the center curve $P(t)$ and radii function $r(t)$, respectively. From the comparisons, we see that our method is faster, while achieving comparable accuracy.

Table 1. The effect of approximation tolerance ε

ε	Depth	Testing cone-spheres pairs	Time (s)	Distance
10^{-2}	4	72	0.017	5.8463058
10^{-3}	6	96	0.025	5.8455473
10^{-4}	7	124	0.031	5.8454567
10^{-5}	9	144	0.039	5.8454345
10^{-6}	10	168	0.043	5.8454344

Table 2. Runtime and distance comparisons of our method and the method in [13] for computing distances between simple canal surfaces

Benchmarks	**Method in [13]**		**Our method**	
	Time (s)	Distance	Time (s)	Distance
(2,2)	0.009	65.5312705	0.005	65.5312705
(2,3)	0.032	14.3156072	0.018	14.3156072
(2,4)	0.057	14.7398007	0.020	14.7398016
(3,2)	0.047	52.3472386	0.021	52.3472398
(3,3)	0.058	50.7140234	0.038	50.7140243
(3,4)	0.061	50.7528231	0.039	50.7528237
(4,2)	0.077	8.3783939	0.020	8.3783939
(4,3)	0.082	7.4441963	0.022	7.4441961
(4,4)	0.084	5.8454344	0.039	5.8454345



(a) Degrees (2,2) (b) Degrees (3,3) (c) Degrees (4,4)

Fig. 7. Some of the test examples in Table 1

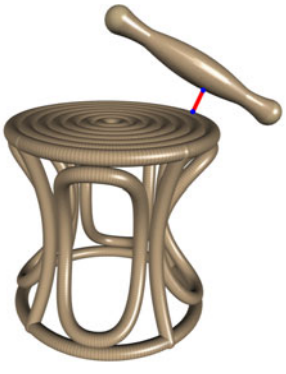


Fig. 8. Distance between a stool and a stick

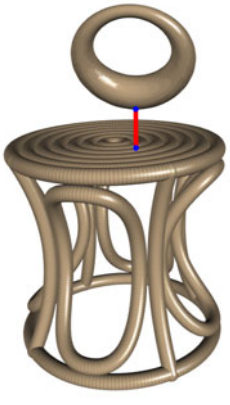


Fig. 9. Distance between a ring and a stool

Table 3. Runtime and distance comparisons of our method and the method in [13] for complex objects

Benchmarks	Method in [13]		Our method	
	Time (s)	Distance	Time (s)	Distance
stick and stool	2.236	6.3297552	0.024	6.3297552
ring and stool	2.923	8.9816040	0.047	8.9816041

Fig. 8 and Fig. 9 present the comparisons of the method by Lee et al [13] and ours for computing the distance between two complex objects consisting of multiple canal surfaces. The stool in the two figures comprises 28 canal surfaces. The stick in Fig. 8 comprises 3 canal surfaces and the ring in Fig. 9 comprises 2 canal surfaces. The timing results are shown in Table 3. We see that the efficiency improvement of our method over the method by Lee et al [13] is more significant for complex objects than for simple objects.

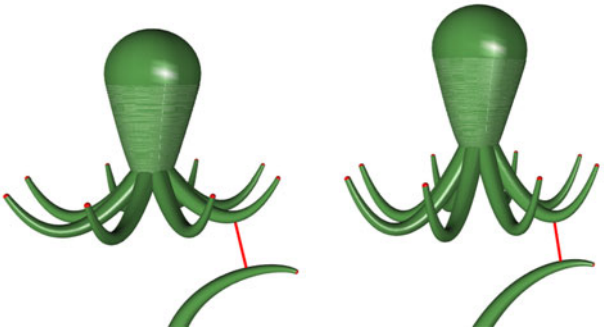


Fig. 10. Distance between octopus and floating grass

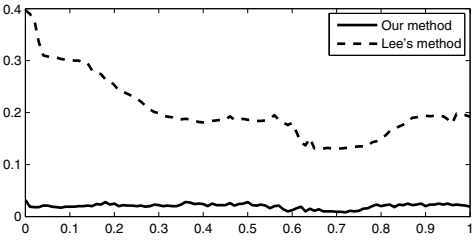


Fig. 11. Runtime comparisons of our method and the method in [13] when the inputs are deformable complex objects

Fig. 10 shows an example when the input are two deformable objects modeled with canal surfaces. The complex object in the figure is an octopus, made up of 9 deformable canal surfaces, and the simple object is a single canal surface. Fig. 10 shows the distances between the two objects at two different moments.

Fig. 11 shows the runtime comparisons of our method and the method in [13] for computing the distance between these two deformable objects. The horizontal axis is the time line of motion/deformation, and the vertical axis represents the computational time cost. The dashed curve represents our method's runtime, and the solid line for that of the method in [13]. We see that our method is again more efficient.

6 Conclusion

We have present a new method for computing the distance between two canal surfaces. We used cone-spheres as bounding volumes to speed up the computation. We design an efficient method for computing a bounding cone-sphere of one section of a canal surface, and present a novel method for pruning the irrelevant pairs for efficient distance computation. Our tests show that this method is much faster than the method by Lee et al [13]. The main reason for this superior

performance is that we just need to solve equations with one-variables due to the use of the cone-spheres bounding volumes, while Lee et al's method needs to solve a system of equations with two variables.

Acknowledgment

The authors would like to thank the reviewers for their invaluable comments. The work of Changhe Tu is partially supported by the Natural Science Foundation of China project (60970046) and the Natural Science Foundation of Shandong Province project (ZR2009GZ002). Wenping Wang is partially supported by NSFC project (60933008) and National 863 High-Tech Program of China (2009AA01Z304).

References

1. Cameron, S.: A comparison of two fast algorithm for computing the distance between convex polyhedra. IEEE Trans. on Robotics and Automation 13(6), 915–920 (1997)
2. Chen, X.–D., Yong, J.–H., Zheng, G.–Q., Paul, J.–C., Sun, J.–G.: Computing minimum distance between two implicit algebraic surfaces. Computer-Aided Design 38, 1053–1061 (2006)
3. Cho, H.C., Choi, H.I., Kwon, S.–H., Lee, D.S., Wee, N.–S.: Clifford algebra, Lorentzian geometry,and rational parametrization of canal surfaces. Computer Aided Geometric Design 21, 327–339 (2004)
4. Choi, H.I., Kwon, S.-H., Wee, N.-S.: Almost rotation-minimizing rational parametrization of canal surfaces. Computer Aided Geometric Design 21, 859–881 (2004)
5. Jia, J., Joneja, A., Tang, K.: Robustly Computing Intersection Curves of Two Canal Surfaces with Quadric Decomposition. In: Alexandrov, V.N., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2006, Part II. LNCS, vol. 3992, pp. 342–349. Springer, Heidelberg (2006)
6. Johnson, D.E., Cohen, E.: A framework for efficient minimum distance computations. In: Proceedings of the IEEE conference on robotics and automation, pp. 3678–3684 (1998)
7. Kawachi, K., Suzuki, H.: Distance computation between non-convex polyhedra at short range based on discrete Voronoi regions. In: Proc. of Geometric Modeling and Processing, Hong Kong, pp. 123–128 (2000)
8. Kazakeviciute, M., Krasauskas, R.: Blending cylinders and cones using canal surfaces. In: Nonlinear Analysis: Modelling and Control, Vilnius, IMI, vol. 5, pp. 77–89 (2000)
9. Kim, K.-J.: Minimum distance between a canal surface and simple surface. Computer–Aided Design 35, 871–879 (2003)
10. Kim, K.-J., Lee, I.-K.: The Perspective Silhouette of a Canal Surface. Computer Graphics forum 22, 15–22 (2003)
11. Klosowski, J.T., Held, M., Mitchell, J.S.B., Sowizral, H., Zikan, K.: Efficient collision detection using bounding volume hierarchies of k-DOPs. IEEE Trans. on Visualization and Computer Graphics 4(1), 21–36 (1998)

12. Larsen, E., Gottschalk, S., Lin, M.C., Manocha, D.: Fast Proximity Queries with Swept Sphere Volumes. In: Proceedings of IEEE Conference on Robotics and Automation (2000)
13. Lee, K., Seong, J.–K., Kim, K.–J., Hong, S.J.: Minimum distance between two sphere–swept surfaces. Computer-Aided Design 39, 452–459 (2007)
14. Lee, I.-K., Kim, K.-J.: Shrinking: Another Method for Surface Reconstruction. In: Proceedings of the Geometric Modeling and Processing 2004 (GMP 2004), pp. 7695–2078 (2004)
15. Lin, M.C., Canny, J.F.: A fast algorithm for incremental distance calculation. In: Proc. of IEEE Int'l Conference on Robotics and Automation, Sacramento, Califonia, pp. 1008–1014 (1991)
16. Max, N.: Cone-Spheres. Computer Graphics 24, 59–62 (1990)
17. Nishita, T., Johan, H.: A scan line algorithm for rendering curved tubular objects. In: Proc. of Pacific Graphics 1999, pp. 92–101 (1999)
18. Quinlan, S.: Efficient distance computation between non-convex objects. In: Proceedings of the IEEE Conference on Robotics and Automation, pp. 3324–3329 (1994)
19. Snyder, J., Woodbury, A., Fleischer, K., Currin, B., Barr, A.: Interval methods for multi-point collisions between time-dependent curved surfaces. In: Proc. of ACM SIGGRAPH 1993, pp. 321–334 (1993)
20. Kyung–ah, S., Juttler, B., Myung–soo, K., Wang, W.: Computing Distances Between Surfaces Using Line Geometry. In: Proceedings of the 10th Pacific Conference on Computer Graphics and Applications (PG 2002) (2002), ISBN: 0-7695-1784-6
21. van Wijk, J.J.: Ray tracing of objects defined by sweeping a sphere. In: Computer Graphics Forum (Eurographics 1984), pp. 73–82 (1984)

A Subdivision Approach to Planar Semi-algebraic Sets

Angelos Mantzaflaris and Bernard Mourrain

GALAAD, INRIA Méditerranée
BP 93, 06902 Sophia-Antipolis, France
FirstName.LastName@inria.fr
http://www-sop.inria.fr/galaad

Abstract. Semi-algebraic sets occur naturally when dealing with implicit models and boolean operations between them. In this work we present an algorithm to efficiently and in a *certified* way compute the connected components of semi-algebraic sets given by intersection or union of conjunctions of bi-variate equalities and inequalities. For any given precision, this algorithm can also provide a polygonal and isotopic approximation of the exact set. The idea is to localize the *boundary curves* by subdividing the space and then deduce their shape within small enough cells using only boundary information. Then a systematic traversal of the boundary curve graph yields polygonal regions *isotopic* to the connected components of the semi-algebraic set. Space subdivision is supported by a kd-tree structure and localization is done using Bernstein representation. We conclude by demonstrating our C++ implementation in the CAS Mathemagix.

Keywords: subdivision algorithm, semi-algebraic set, connected component, algebraic curve, topology computation.

1 Introduction

Planar semi-algebraic sets are unions of subsets $\mathcal{S}$ of $\mathbb{R}^2$ that satisfy a set of bivariate polynomial equalities and inequalities. These sets appear naturally when polynomial constraints are used for instance to describe regions of validity for a physical problem. Piecewise algebraic representation of shapes is commonly used in Computer Aided Geometric Design, for instance in B-spline parametric representation of curves, or even surfaces of volumes, that also belong to the class of real semi-algebraic sets. Constructive Solid Geometry models also used in CAGD are semi-algebraic sets if the involved solid primitives are algebraic. In domains such as optimization, an important problem is the computation of global optimum of (polynomial) functions under (polynomial) constraints. These constraints define a semi-algebraic set as the solution space, in which the optimal points will be searched [15], [12]. In other words, semi-algebraic sets provide a general framework to handle many shape representations that are commonly used in Shape Modeling.

B. Mourrain, S. Schaefer, and G. Xu (Eds.): GMP 2010, LNCS 6130, pp. 104–123, 2010.

In the present paper we present a new technique to handle semi-algebraic sets in the plane. We note that our method can be extended to dimension three, without theoretical obstacles. Indeed, the implementation is done in a generic programming framework that allows extension to dimension three without relatively little additional effort, since abstract types and templated data structures are heavily used.

The study of real semi-algebraic sets has a long historical background [18], with important theoretical contributions for instance on their triangulation [11], [13]. More algorithmic questions have also been tackled, essentially using the well-known Cylindrical Algebraic Decomposition [6]. This approach is based on performing successive projections of semi-algebraic sets onto subspaces of dimension one less and then lifting back to the projected set. It yields a decomposition of a semi-algebraic set S into (connected) components, defined by sign conditions deduced from some "subresultant" polynomial sequences [5], [8], [3].

One of the bottlenecks for practical applications of C.A.D.-based approaches, even in small dimension, is its double exponential complexity behavior. This is due mainly to computations with algebraic numbers of possibly high degree. Other obstacles include the lack of extension to approximate computation, required by applications in CAGD and the problem of robust description of the components. Our approach refrains from costly algebraic manipulations, hence avoids the high complexity of exact computation. It is based on real root isolation techniques, which are well suited for approximate yet certified computations. Moreover, it gives an answer to the problem of representing the semi-algebraic set in a way that is both topologically correct and suitable for applications. This overcomes the inflexible description by sign conditions or other implicit descriptions, for instance the one in [2], where each connected component is described itself as a semi-algebraic set.

We propose a subdivision approach that concentrates on rectangular domains of $\mathbb{R}^2$ and computes a piecewise linear approximation of a semi-algebraic set in the domain, which is topologically equivalent to it. The defining equations of the set are transformed to tensor-Bernstein form. This gives a numerically stable way to subdivide this representation into sub-domains, until certain regularity conditions are fulfilled. During the subdivision process the cells that touch the boundary of the semi-algebraic set are identified and their adjacency structure is represented as a graph. When this process terminates, we follow this graph to recover contours that define the geometry of the set. A tolerance $\varepsilon > 0$, given in the input, controls the precision of the computed approximation. Nevertheless, the regularity conditions imply a topologically correct result. In this sense, the algorithm extends the approach in [1] on the topology of algebraic curves, by providing a efficient way to deal with semi-algebraic regions and to perform boolean operations on these regions.

We start by defining the family of sets that we are interested in.

Definition 1. *The family* $\mathfrak{S} \subseteq 2^{\mathbb{R}^2}$ *of* semi-algebraic sets *is the closure under union and intersection of subsets of* $\mathbb{R}^2$ *of the form*

$$\{(x,y) \in \mathbb{R}^2 \,:\, f(x,y) = 0\} \quad \textit{and} \quad \{(x,y) \in \mathbb{R}^2 \,:\, g(x,y) > 0\}$$

where $f,\ g \in \mathbb{R}[x,y]$.

We call the above sets *basic semi-algebraic sets*. These definitions extend naturally to higher dimension.

If $\mathcal{S} \in \mathfrak{S}$, its complement $\mathcal{S}^c = \mathbb{R}^2 \setminus \mathcal{S}$ is easily seen to belong to $\mathfrak{S}$. The family $\mathfrak{S}$ is thus stable by intersection, union and complementary. Another important property of semi-algebraic sets is that the projection of a semi-algebraic set is a semi-algebraic set [3].

Our algorithm has as input an *initial frame* $\mathcal{D}_0 = [a,b] \times [c,d]$ and a semi-algebraic set $\mathcal{S}$, given in *disjunctive normal form*, that is, in the form $\mathcal{S}_1 \cup \cdots \cup \mathcal{S}_k$ where each $\mathcal{S}_i$ is an intersection of basic semi-algebraic sets, hence defined as a subset $\{(x,y) \in \mathbb{R}^2 \,:\, g_1 = 0, \ldots, g_m = 0,\ f_1 > 0, \ldots, f_n > 0\}$. It outputs a boundary effective representation of the connected components of this semi-algebraic set.

Given a precision $\varepsilon > 0$, it can also output a polygonal approximation of the set inside the domain $\mathcal{D}$, within the precision ε, which moreover is isotopic to S in the following sense:

Definition 2. *Two semi-algebraic sets* $\mathcal{S}_1, \mathcal{S}_2$ *of* $\mathbb{R}^2$ *are* isotopic *if there exists a continuous application* $F : \mathbb{R}^2 \times [0,1] \mapsto \mathbb{R}^2$ *such that* $F|_{t=0}$ *is the identity map,* $F(\mathcal{S}_1, 1) = \mathcal{S}_2$ *and for all* $t \in [0,1]$, $F|_t : \mathbb{R}^2 \mapsto \mathrm{Im}F|_t$ *is a homeomorphism.*

We introduce some notation. Throughout the text $\mathcal{S}$ will refer to an input semi-algebraic set. By a slight abuse of notation we might denote by $\mathcal{S}$ both the semi-algebraic set and the set of underlying defining polynomials. The meaning will be clear from the context. Let f be a polynomial of the input. We refer to parts of the real algebraic curve $f = 0$ that belong to $\partial\mathcal{S}$, the boundary of $\mathcal{S}$, as *boundary curves*. Points where boundary curves intersect (or a single boundary branch, part of some $f = 0$ is self-intersecting), are called *crossing points*. Also, we will refer to a *branch* of a curve, defined by two endpoints p, q, as the part of the curve between these points, e.g. the image of a continuous parametrized curve $r : [0,1] \to \mathbb{R}^2$ s.t. $r(0) = p$, $r(1) = q$ and $f \circ r = 0$.

This paper is organized as follows: In Sect. 2 we provide details on the representation of the main objects in memory. Then in Sect. 3 we describe a subdivision process that computes a collection of cells covering $\partial\mathcal{S}$. This representation is used to compute the connected regions of $\mathcal{S}$, in Sect. 4. We specialize the main functions that appear in the algorithm first for the case of basic sets, in Sect. 5 and then for a general set of $\mathfrak{S}$ in Sect. 6. We conclude with examples and an overview of our implementation in Sect. 7.

2 Representation

We begin by describing the main objects in the algorithm, called hereafter cells, and how they are represented in memory.

A *cell* carries local information for $\mathcal{S}$ in a rectangular domain $\mathcal{D} = [a, b] \times [c, d]$. This information includes the Bernstein representation over $\mathcal{D}$ of the defining equations of $\mathcal{S}_i$, whenever $\mathcal{S}_i \cap \mathcal{C} \neq \varnothing$. It also carries the intersections of every branch of $\partial\mathcal{S}$ that crosses the cell with the cell frame $\partial\mathcal{C}$. The cells of interest are exactly the cells that contain branches of boundary curves, i.e. parts of $\partial\mathcal{S}$. These cells are identified during the subdivision process.

A local description of $\mathcal{S}$ in a cell is achieved using the tensor-Bernstein representation over $\mathcal{D}$ of every polynomial that defines $\mathcal{S}$. This representation is computed using DeCasteljau's algorithm. It yields for $f \in \mathbb{R}[x, y]$, an expansion

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

where d_x, d_y is the degree of f in x, y resp. and $B^i_{d_x}(x; a, b)$ the $i-$th Bernstein polynomial of degree d_x over the interval $[a, b]$, namely $B^i_{d_x}(x; a, b) = \binom{d_x}{i} (x-a)^i (b-x)^{d_x - i} \, (b-a)^{-d_x}$, $0 \leq i \leq d_x, b < a$. Consequently we store an $(d_x + 1) \times (d_y + 1)$ matrix in memory to represent f, i.e. a dense Bernstein representation. A number of properties of this basis, e.g. convexity, variation diminishing, positivity etc, make it suitable for stable approximate computations. See [10] for more information.

The first cell $\mathcal{C}$ that is computed as soon as the algorithm is launched is the one corresponding to the initial frame $\mathcal{D}_0$. This initial cell carries all the polynomials of the input. When a sub-cell is computed, if $\partial\mathcal{S}_i$ does not cross that cell, for some i, $\mathcal{S} = \mathcal{S}_1 \cup \cdots \cup \mathcal{S}_r$, then the polynomials of $\mathcal{S}_i$ are not kept in the representation of it.

Another object is the *region*, which is a linear approximation of a 2-dimensional connected component of the semi-algebraic set. It is described as a collection of *contours*, that are closed loops properly oriented to delimit the region: The outer contour, or *shell*, is oriented counter-clockwise (CCW for short) whereas any internal contours, or *holes* are clockwise (CW) oriented. See Fig. 4 for a region defined by three contours.

Every contour is essentially a simple polygon described as a list of vertices that lie on the boundary of the exact set.

We also employ graph structures to keep adjacency information between cells. These are internally saved in memory using adjacency-list representation [7].

More specifically, we compute an undirected graph $\mathcal{A}$, in which the points where $\partial\mathcal{S}$ intersects $\partial\mathcal{C}$ correspond to edges and subdivision cells $\mathcal{C}$ correspond to vertices. We shall compute the restriction of the semi-algebraic set in a given initial domain, thus the border of this domain is from a computational point of view a limit for the regions to compute. For this reason, we also keep a directed graph containing the cells where boundary curves touch the initial frame and the four corner cells of $\mathcal{D}_0$. This forms a CCW loop and is used to complete any open contours that touch the boundary.

The space subdivision is tracked using a $kd-$tree, rooted at $\mathcal{D}_0$. The leaves of this tree is a partition of $\mathcal{D}_0$ into cells. The inner nodes represent the sequence of subdivisions that took place.

Example. In Fig. 3(left), we have a partition of the domain into 8 regular cells. The semi-algebraic set is the grayed area, described by a single contour. Here the graph $\mathcal{A}$ is the closed path of cells 2,7,6,8,3,2. The border graph is the directed closed path 2,1,7,6,4,3,2.

3 Subdivision Process

The subdivision of the initial domain into regular cells is a main operation of the algorithm. It consists in splitting the initial domain into smaller cells until certain local properties are satisfied. These properties will allow in a later step the construction of a topologically correct approximation of the (boundary of the) set in each cell.

During this process we construct a graph $\mathcal{A}$ whose vertices are the cells that span $\partial\mathcal{S}$. Alg. 3.1 presents the general process. Here a cell is regarded as an abstract object that supports the following operations:

• **Regularity test**(IsRegular). A cell is considered regular if the topology of $\mathcal{S}$ inside the cell is known, i.e. it can be deduced using only discrete data stored in the cell, namely the points in $\partial\mathcal{C} \cap \partial\mathcal{S}$, or even the sign of some derivatives on them. Hence interesting cases are the cells that contain branches of boundary curves. Some characteristic examples of this are presented in Fig. 1. If there is more than one crossing point in the cell, that is, branches that intersect each other, then there is ambiguity on how the region behaves in the cell. Thus the regularity implies that we have at most one crossing point inside the cell and that the branches inside $\mathcal{C}$ have a monotone behavior. This behavior is connected to special points on the boundary curves, namely points with vertical or horizontal tangents.

• **Boundary curve intersection test**(OnBoundary). It is used to identify if a cell is intersecting $\partial\mathcal{S}$, i.e. $\mathcal{C} \cap \partial\mathcal{S} \neq \varnothing$. This can be done by inspecting the sign variations of Bernstein coefficients of the polynomials that define $\mathcal{S}$. *Descartes' Rule of Signs* implies that if there is a branch of $\partial\mathcal{S}$ in $\mathcal{C}$, then there will be sign variations on the coefficients of some boundary equation. On the other hand, by the positivity property of the Bernstein basis, if the coefficients over a cell $\mathcal{C}$ of a curve $f = 0$ have no sign variations, then there cannot be a branch of this curve in $\mathcal{C}$.

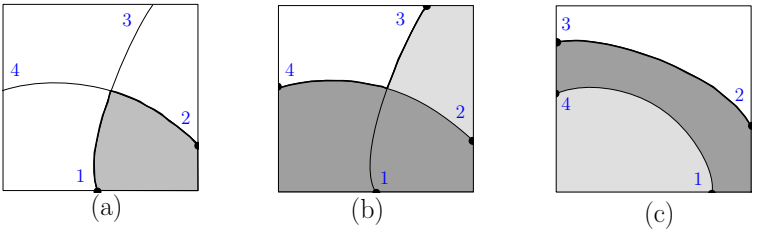


Fig. 1. Examples of cells that are regular and intersect $\mathcal{S}$: (a) intersection of two basic sets, (b) union of two basic sets with a crossing, (c) union of two sets

Algorithm 3.1. Subdivision algorithm

Input: A cell $\mathcal{C}_0$ corresponding to the initial domain $\mathcal{D}_0$.
Output: A partition of $\mathcal{C}_0$ into regular cells and a cell graph boxruled $\mathcal{A}$.
Initiate a $kd-$tree $\mathcal{K}$ and set its root to $\mathcal{C}_0$;
Initiate a graph $\mathcal{A}$ with a vertex $\mathcal{C}_0$;
for *all unvisited leaves* $\mathcal{C}$ *in* $\mathcal{K}$ **do**
  if OnBoundary($\mathcal{C}$) *and not* IsRegular($\mathcal{C}$) **then**
    subdivide $\mathcal{C}$ into two children $\mathcal{C}_L$ and $\mathcal{C}_R$;
    put an edge in $\mathcal{A}$ between $\mathcal{C}_L$ and $\mathcal{C}_R$;
    distribute the $\mathcal{A}-$neighbors of $\mathcal{C}$ to $\mathcal{C}_L$, $\mathcal{C}_R$;
    remove $\mathcal{C}$ from $\mathcal{A}$;
  else
    mark $\mathcal{C}$ as visited;
  end
  return $\mathcal{K}$, $\mathcal{A}$;
end

During the subdivision process the following information is computed:

- Space partition information in the kd-tree structure.
- Local information in the subdivided cells: the tensor-Bernstein representation over the cell, critical points contained in the cell, intersection points of $\partial\mathcal{S}$ with the cell frame.
- Adjacency information between the cells, in horizontal and vertical direction. The cells in which the boundary curves touch the border $\partial\mathcal{D}_0$ are also connected in a counter-clockwise loop, to serve the purpose of limiting the computation inside $\mathcal{D}_0$.

• **Space Partition.** The cells that derive from successive subdivisions are organized in a kd-tree structure [4], rooted at the initial domain $\mathcal{D}_0$. The nodes in this tree have pointers to their left and right children, as well as to the parent node. The coordinate in which the subdivision takes place at every level of the tree is not fixed; it is implied every time by the dimensions of the current cell, thus at the same level of the tree we may have cell subdivisions either in x or y coordinate.

This structured partition allows to perform fast point location queries. The reason we have chosen a kd-tree rather than a quad-tree is economy wrt the overall number of cell subdivisions as well as the modularity that it offers, for instance it's direct adaptation to three or more dimensions.

There are two basic tests to be defined, to guide the subdivision process. The first identifies that a cell is *regular*, i.e. the topology of the semi-algebraic set in the cell is known. In this case the subdivision stops at this branch of the kd-tree. The second test identifies if $\partial\mathcal{S}$ intersects the current cell. If not, then either $\mathcal{C} \subseteq \mathcal{S}$ or $\mathcal{C} \cap \mathcal{S} = \varnothing$, thus there is no need to subdivide it any further.

• **Cell subdivision.** Subdividing a cell $\mathcal{C}$ along some coordinate is essentially to compute, starting from the Bernstein representation over $\mathcal{C}$, representations over some sub-domains of $\mathcal{C}$. This operation is carried out by one call of DeCasteljau's algorithm [10].

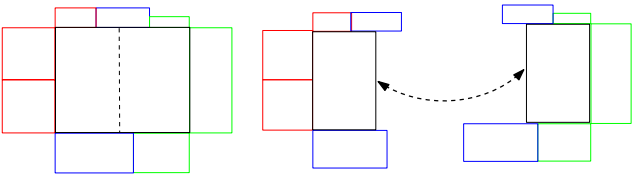


Fig. 2. Cell subdivision along $x-$direction. Neighbors of the parent (left) are distributed to the children(right). An edge is added between the latter.

Moreover, along the line where the splitting takes place, we solve a univariate Bernstein polynomial for every boundary curve that intersects the cell, in order to compute intersection with the new frame sides. The existing crossing points and frame intersection points are distributed to the resulting sub-cells, Fig. 2.

• **Adjacency graph update.** At each subdivision step, a former leaf of the kd-tree obtains two children. To update the cell graph, we disconnect this node and distribute it's neighbors to the new children, according to the direction of splitting. Finally, we introduce a new edge that joins the two children along the corresponding direction. These steps, demonstrated in Fig. 2, assure that at any point of the subdivision, the leaves of the kd-tree, which form a partition of $\mathcal{D}_0$, are connected to the neighboring cells in all four sides.

4 Region Recovery

In this section we explain how we pass from the cell description to a polygonal approximation of the (connected components of the) semi-algebraic set. We will demonstrate that as soon as the subdivision Alg. 3.1 terminates, we are able to recover the shape of the semi-algebraic set, and guarantee the correctness of the construction.

The output is a list of regions that correspond to connected components of the semi-algebraic set. The set of cells that intersect a region can readily provide a triangulation of the region, which can be outputted for use in rendering. Each region is represented as a set of closed oriented contours. The orientation of every contour reveals whether it is the exterior boundary, or *shell* of the region, or an internal gap, or a *hole*. There is a unique shell for every region of $\mathcal{S}$.

To compute the regions, it suffices to traverse the cell graph $\mathcal{A}$ in a suitable way and recover the shell and holes of every region in the set. The algorithm for region computation is summarized in Alg. 4.1.

The orientation check IsCCW depends only on the contour F. Every closed contour can be assigned an orientation; if one walks around the curve in such a way as to keep the bounded region on one's left at all times, the contour is said to be positively oriented. If the contour is traversed in the opposite direction, then it is said to be negatively oriented. Let $c = (p_1, p_2, \ldots, p_n)$ with $p_i = (x_i, y_i)$, $p_{n+1} = p_1$ be a list of points defining a closed polygonal contour. The sign of the

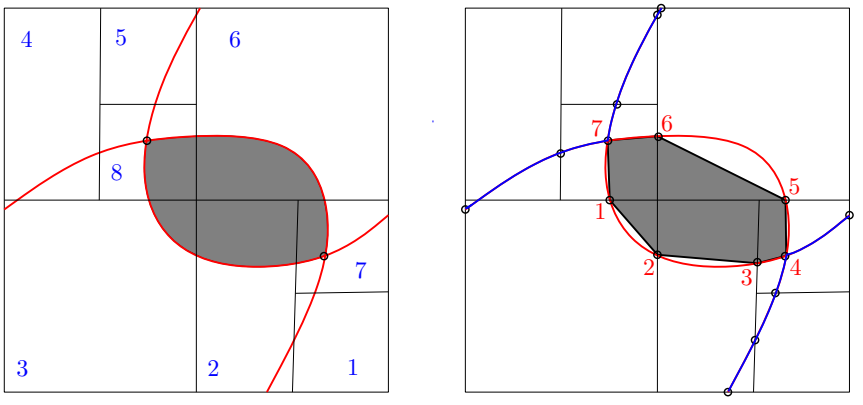


Fig. 3. Left: Subdivision process, with marked subdivided cells and intersections. Right: Computed polygonal region, marked with the oriented list of contour points.

Algorithm 4.1. Region computation

Input: A cell graph $\mathcal{A}$ covering the semi-algebraic set $\mathcal{S}$.
Output: A list L of polygonal regions, one for every connected component of $\mathcal{S}$.
$L \leftarrow \varnothing$;
for *all boundary cells* $\mathcal{C}$ *in* $\mathcal{A}$ **do**
  if $\mathcal{C}$ *is not visited* **then**
    $F \leftarrow$ DiscoverContour($\mathcal{C}$);
    if F IsCCW **then**
      Initialize region R with F;
      push R to L;
    else
      attach hole F to it's containing shell
    end
  end
end
return L;

quantity $\sum_{i=1}^{s}(x_i y_{i+1} - x_{i+1} y_i)$ determines weather c is positively or negatively oriented. This sum is twice the (signed) area of the contour.

The function DiscoverContour, presented in Alg. 4.2, returns a contour that crosses the cell $\mathcal{C}$ and is oriented CCW wrt the region it delimits. For instance, both the holes and the shell of the region in Fig. 4 are CCW oriented wrt the grayed region. It is required that the cell argument is *regular*, so that the global shape of the contour can be determined by following the known local topology in the cell. This is ensured by the subdivision process of Sect. 3.

Apart from the cells containing branches of the boundary contours, there are special cells that are needed in order to constrain the computation in the initial frame $\mathcal{D}$. These are the boundary cells that touch the frame as well as the four corners of $\mathcal{D}$. They are connected in a CCW loop during the subdivision process

that is used to complete the contours that escape $\mathcal{D}$ and would not be closed otherwise.

4.1 Following the Boundary Curves around a Region

The main function in Alg. 4.1 is DiscoverContour, which is in turn based on two routines, Pair and StartingPoint.

• **Pair.** If a cell intersects both a region and the region's boundary, then for every intersection point p there is a unique point q that is connected to p via a segment of $\partial\mathcal{S}$ that lies inside the cell. If the cell in question is also regular, q can be computed using sign conditions along $\partial\mathcal{C}$. We define this point q to be the result of Pair$(\mathcal{C}, p)$. If this point q is different from p, then evidently it is connected to p via a branch of some boundary contour of the region. Alg. 4.3 presents a general strategy to compute q.

Example. In Fig. 1 the result of Pair is: (a)$1 \to 2$, (b)$4 \to 3$, (c)$2 \to 3$. Note that in case (c), the branch $1 \to 4$ will not occur in the computation, since it does not belong to $\partial\mathcal{S}$.

• **Starting Point.** A contour has to be traversed with the correct orientation, otherwise we would not be able to distinguish between shells and holes of a region. For this, it suffices to provide the first two points in the point list of the contour with the correct orientation. This is the task of the StartingPoint$(\mathcal{C})$ routine. It returns a point p on $\partial\mathcal{C}$ s.t. the oriented branch with endpoints p, Pair(p) has on it's left side the region to be computed. This is a special case of the Pair computation described in the next paragraph.

Example. For Fig. 1 the result of StartingPoint is: (a)2, (b)3, (c)2. Indeed, the respective branches (a)$2 \to 1$, (b)$3 \to 4$ and (c)$2 \to 3$, are CCW-oriented wrt $\mathcal{S}$.

Looking at the graph $\mathcal{A}$ induced by Alg. 3.1, we distinguish two kinds of regular cells:

- Cells that contain non-crossing branches of $\partial\mathcal{S}$.
- Cells that contain branches that intersect at one crossing point.

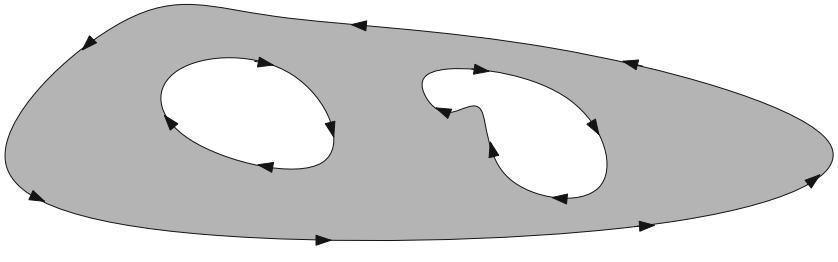


Fig. 4. A region defined by it's oriented border. All the contours are CCW-oriented wrt the grayed region. This leaves the holes CW oriented with respect to the bounded domain they define.

Recall that the outcome of PAIR($\mathcal{C}$) is the point connected to p via a branch which lies inside $\mathcal{C}$. The general algorithm is presented in Alg. 4.3. The essential tool for this computation is an efficient way to check if a given point on $\partial\mathcal{C}$ is contained in $\mathcal{S}$. This is done using the sign of the Bernstein coefficients. For every polynomial f of $\mathcal{C}$, there are four *extreme* coefficients that are equal to it's value on the four corners of $\partial\mathcal{C}$. Now taking into account that the sign of f along $\partial\mathcal{C}$ alternates every time we pass a boundary intersection point, we can determine the sign on any point of $\partial\mathcal{C}$ by starting from an extreme coefficient and counting points along $\partial\mathcal{C}$, up to the desired point.

If there is one crossing point in $\mathcal{C}$ the topology of $\partial\mathcal{S}\cap\mathcal{C}$ is conic (Fig. 1(a,b)). To choose the correct pair of a given point on $\partial\mathcal{C}\cap\partial\mathcal{S}$, we check whether a $\partial\mathcal{C}-$neighborhood on the left of p belongs to $\mathcal{S}$ or on the right of p. We output accordingly the point on the side where the test was positive.

If there is no crossing point, (Fig. 1(c)) it suffices to return the other end of the branch that starts from p. We shall see in the sequel how this information is recovered on regular cells.

Example. In the case of Fig. 3(left) we execute Alg. 4.2. Starting from cell 2, we obtain a first point of the contour, using STARTINGPOINT routine. Succesive calls of the pair function give the sequence of points shown in Fig. 3(right). The process stops when we reach the cell 2 again, thus completing the contour.

Algorithm 4.2. DISCOVERCONTOUR($\mathcal{C}$)

Input: A *regular* cell $\mathcal{C}$ of $\mathcal{A}$.
Output: A list F of points in the plane that define a closed contour.
$p \leftarrow$ STARTINGPOINT($\mathcal{C}$);
Initialize a contour F and push p to it;
$\mathcal{C}_0 \leftarrow \mathcal{C}$;
repeat
  mark $\mathcal{C}$ as visited;
  $p \leftarrow$ PAIR($\mathcal{C}, p$);
  push p to contour F;
  $\mathcal{C} \leftarrow$ the $\mathcal{A}-$neighbor of $\mathcal{C}$ that contains p;
until $\mathcal{C} = \mathcal{C}_0$;
return F;

Algorithm 4.3. PAIR

Input: A regular cell $\mathcal{C}$ and an intersection point p on $\partial\mathcal{C}$.
Output: The intersection point q such that $\{p, q\}$ lie on a branch of $\partial\mathcal{S}$.
if *there is a crossing in* $\mathcal{C}$ **then**
  Let l, r be the CCW previous and next point, resp., of p, in $\partial\mathcal{C}\cap\{f=0\}$;
  Based on which of the segments $\overline{lp}$ or $\overline{pr}$ lies in $\mathcal{S}$, return either l or r ;
else
  return the other end of the $\mathcal{C}-$branch starting from p;
end

It remains to specialize these functions. We continue by doing so, first in the case of basic algebraic sets and then in the case of intersection and union.

5 The Case of Basic Semi-algebraic Sets

A basic semi-algebraic set is defined by one polynomial, $\mathcal{S} = \{(x, y) : f > 0\}$, or $\mathcal{S} = \{(x, y) : f = 0\}$. In both cases the treatment is quite the same, and depends on the boundary curve $f = 0$, hence we shall suppose $\mathcal{S} = \{(x, y) : f > 0\}$. In the case of equality it is only the contour lines that will be outputted rather than two-dimensional regions. After fully treating this case, we shall generalize by extending the operations to the cases of intersection and union.

This case is closely related to the topology computation of an implicit real algebraic curve. The latter is the partition of space into points, edges and faces defined by the curve $f = 0$. See Figure 5 for an example. Note that recovering the topology of the real algebraic curve $f = 0$ is a special case of our algorithm. Indeed, it suffices to execute the subdivision algorithm on $\mathcal{S} = \{f = 0\}$ and then run the region recovery twice, once with $\mathcal{S} = \{f > 0\}$ and once with $\mathcal{S} = \{-f > 0\}$. The union of these two outputs is exactly the set of faces defined by the curve $f = 0$.

5.1 Regularity Test

We describe the regularity criteria that are used for the boundary curve of the set. We shall provide a brief overview and refer the reader to [1, Sect. 4] for an extended presentation.

The regularity depends on special points on the curve, that reveal the local shape of the curve in a neighborhood around them. These are:

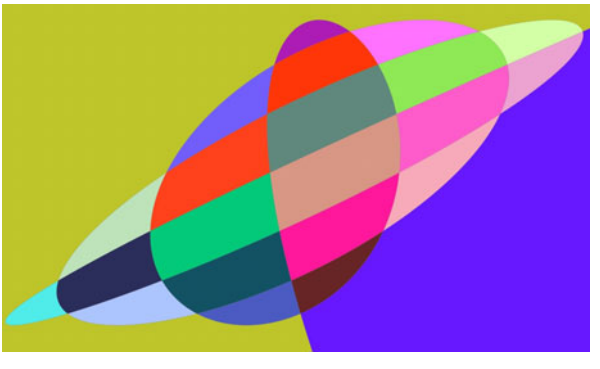


Fig. 5. The 23 faces in the topology of the degree 8 curve $f = 2 + 7x - 7y - 14x^3 + 7x^5 - x^7 - 16y^2 + 14y^3 + 20y^4 - 7y^5 - 8y^6 + y^7 + y^8 - 42y^2x - 70y^3x^2 + 35xy^4 + 70y^2x^3 + 42yx^2 - 35x^3y^4 + 7x^6y - 21x^5y^2 - 35x^4y + 21x^2y^5 + 35y^3x^4 - 7xy^6$ computed by running our algorithm on $\mathcal{S} = \{f > 0\}$, $\mathcal{S} = \{f < 0\}$ and $\mathcal{D} = [-4, 4] \times [-3, 3]$

Definition 3. *The set of* extremal points *of* $f \in \mathbb{R}[x, y]$ *is the solutions of the system* $\partial_x f(x, y) = \partial_y f(x, y) = 0$.
The set of singular points *of* f *is the subset of extremal points that also satisfy the equation* $f(x, y) = 0$.
The set of x-critical *(*y-critical*) points of* f *is the solution set of* $\partial_x f(x, y) = f(x, y) = 0\}$ *(*$\partial_y f(x, y) = f(x, y) = 0\}$*).*

Computing these points, approximately but also efficiently, is a vital ingredient of the algorithm. In [16], an algorithm is presented that acts on polynomials in Bernstein form. It uses domain subdivision as well as enveloping and preconditioning techniques to provide a robust polynomial solver. We rely on this solver to obtain good approximations of the points in Def. 3. These points are precomputed and during the subdivision process they are isolated between the cells, i.e. we do not allow more than one of them in a single cell. As a result, after the subdivision process terminates, we obtain a partition of $\mathcal{D}_0$ into regular cells of the following type:

- *x-regular* cells, those that contain no x-critical points (similarly for $y-$regular).
- *simply singular* cells, that contain a single singular point and all branches of $\partial\mathcal{S} \cap \mathcal{C}$ intersect it.

• **Regular cells.** If a cell is $x-$regular, it contains a number of $x-$monotone branches. In short, the direction of the tangential gradient vector $(\partial_y f, -\partial_x f)$ evaluated at the points in $\partial\mathcal{C} \cap \partial\mathcal{S}$ yields the connection of the branches inside $\mathcal{C}$. The Bernstein representation of the derivatives themselves are easily computed, since they are given as differences of Bernstein coefficients of f. A sufficient condition for f to be x-regular is that the Bernstein coefficients of $\partial_x f$ maintains a constant sign. By Descartes' law, this statement implies that the sign variations in $x-$direction should be at most one.

Note that in special cases where the critical point is on $\partial\mathcal{C}$ two branches may share a starting or ending point.

• **Simply singular cells.** If there is a single singular point in a cell $\mathcal{C}$, and no additional extremal points, one must test whether all the branches inside $\mathcal{C}$ cross this point. This would imply that the topology inside $\mathcal{C}$ is a cone starting from the singular point. The test is based on computing the topological degree, or Gauss map [17] of the vector field $\nabla f = (\partial_y f, \partial_x f)$ around the closed curve $\partial\mathcal{C}$. This breaks down to isolating the real roots of $\partial_x f$ and $\partial_y f$ along $\partial\mathcal{C}$. Khimshiashvili's theorem [14] relates the number of branches that reach the singular point to the topological degree $\deg(\nabla f, \mathcal{C})$; it states that the number of branches is exactly $2\,(1 - \deg(\nabla f, \mathcal{C}))$. If this number coincides with the cardinality of $\partial\mathcal{C} \cap \partial\mathcal{S}$ then we can treat this cell, otherwise there are additional branches in $\mathcal{C}$ and the subdivision will continue until they are isolated from the singular point.

6 The General Case

To treat semi-algebraic sets with more than one defining equation, it suffices to extend the main operations in this case. Our aim is to have a covering of

the boundary curves of $\partial\mathcal{S}$ by regular cells. The main difference is that crossing branches in a cell can correspond to two basic sets in a union, or two basic sets in an intersection. Treating correctly these cases will extend our algorithm to the whole family of semi-algebraic sets. Again, we assume that the basic sets are defined by inequalities, since restricting to (in the case of intersection) or attaching (in the case of union) a curve segment to the output is not essentially different from treating boundary curves of two dimensional components. In particular, the cell graph $\mathcal{A}$ that we obtain from the subdivision Alg. 3.1 will span any components of lower dimensions.

Let $\mathcal{S} = \mathcal{S}_1 \cup \cdots \cup \mathcal{S}_k$. Recall that a cell $\mathcal{C}$ carries the polynomials of $\mathcal{S}_i$ if $\partial\mathcal{S}_i \cap \partial\mathcal{C} \neq \varnothing$. For all the other parts $\mathcal{S}_j$, it is either $\mathcal{S}_j \cap \mathcal{C} = \varnothing$ or $\mathcal{C} \subseteq \mathcal{S}_j$, hence $\mathcal{C}$ does not interfere with the boundary curves of these components.

We define a regular cell to be a cell in which every attached polynomial is regular (in the sense of Sect. 5.1) and conforms to any of the following properties:

1. There is only one set $\mathcal{S}_i$ in $\mathcal{C}$ and at most one (self-)intersection.
2. There are two sets $\mathcal{S}_i$ and $\mathcal{S}_j$ and one intersection between a branch of $f \in \mathcal{S}_i$ and $g \in \mathcal{S}_j$.

These intersection points are also computed using the Bernstein solver [16] and are isolated among the cells during the subdivision process.

Deciding if a region spans $\partial\mathcal{S}$ is done by checking whether it belongs to the boundary of every $\mathcal{S}_i$ that is carried by $\mathcal{C}$, and consists again in checking signs on the boundary.

To simplify the process, we rely on basic cells (cells that have branches of a single basic set contributing to $\mathcal{S}$) for determining the orientation of regions, i.e. applying StartingPoint. This is a mild assumption, since in any case, boundary curves away from crossings define basic semi-algebraic sets. This assumption also simplifies the way we deal with cells like Fig. 1(c), since we only need to know the connection inside the cell in order to traverse them and choose the correct branch (for instance, in Fig. 1(c), discard the locally redundant curve).

We describe how we compute Pair in the above two cases:

• **Case 1.** There is a set of branches in the cell that intersect in one point only, similar to 1(b). Since the corresponding basic sets are combined by intersection we search around $\partial\mathcal{C}$ for a part that attains positive sign on all involved polynomials, to decide the Pair routine.

• **Case 2.** Two branches intersect, corresponding to basic sets combined by union, for instance 1(c). We propagate the search to points around parts of $\partial\mathcal{C}$ that satisfy any of the sign conditions implied by $\mathcal{S}_i$ or $\mathcal{S}_j$. When we reach a part that is outside $\mathcal{S}$, we return the last point found.

7 Implementation and Demonstration

Our implementation is generic, working on abstract classes of cells, that define internally a small number of predicates. We chose to use the open-source project Mathemagix[1], for the fast data structures it provides for polynomials and it's

[1] http://www.mathemagix.org

Fig. 6. $\mathcal{S} = \{(x, y) : f_1 > 0, f_2 > 0\}$ with $f_1 = x^4 + 2x^2y^2 + y^4 + 3x^2y - y^3$, $f_2 = -105y^2x^4 - 80y^3 + 140x^3y^3 - 140y^3x + 35y^4 - 105y^4x^2 + 48y^5 + 42xy^5 - 42x^2 + 35x^4 - 7x^6 + 32y + 84xy - 140x^3y + 42x^5y + 210x^2y^2 - 42y^2 - 7y^6 - 8y^7 + 7$ over the box $[-1, 1]^2$

Fig. 7. Left: defining curves and cell graph. Right: boundary contours of the underlying set.

support to certified arithmetic primitives. Our code is written in the frame of the `shape` module, which is the part of MATHEMAGIX providing a variety of geometric operations in two or three dimensions.

Solution of univariate and bi-variate systems of polynomial is performed using the algorithm in [16], which is hosted in the module `realroot`. This module also provides algebraic operations, Bernstein dense representation and a variety of zero-dimensional system solvers. Hardware accelerated rendering of output has been made possible using AXEL[2] platform.

[2] `http://axel.inria.fr`

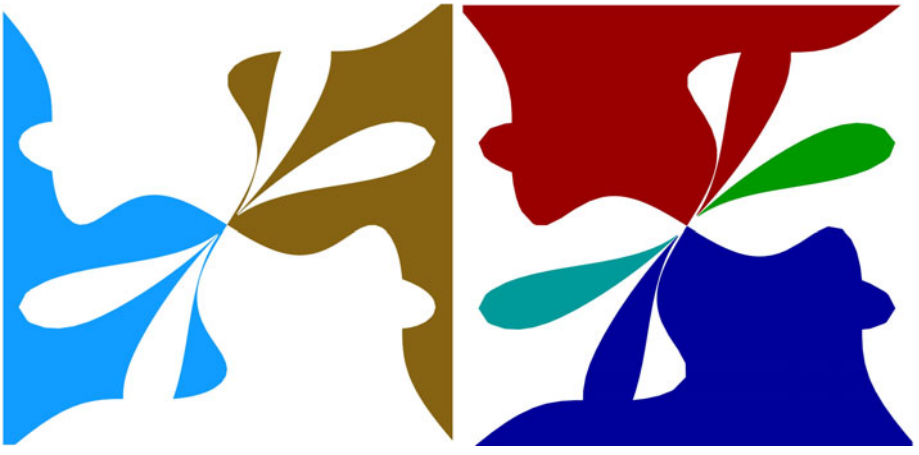


Fig. 8. Left: Two connected components of a semi-algebraic set, each containing a hole. Right: regions of the complementary set.

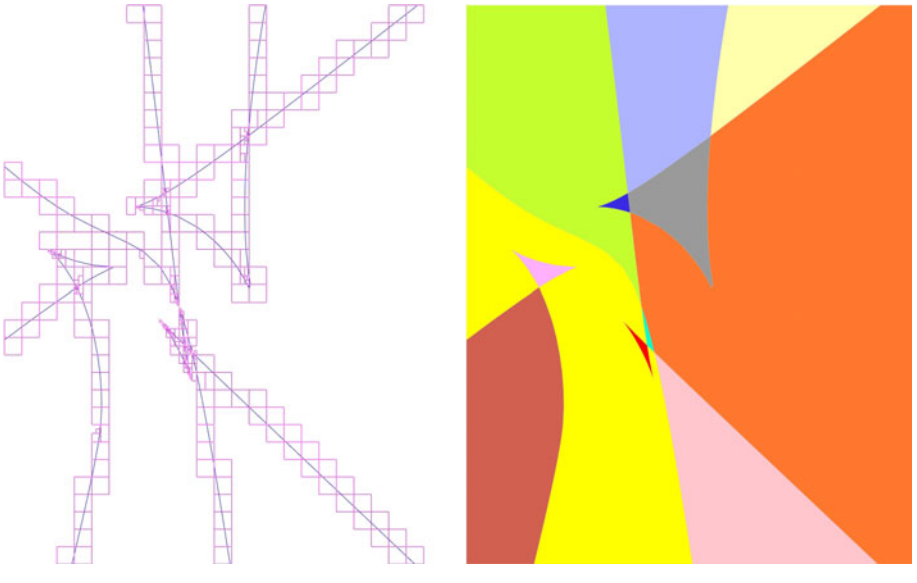


Fig. 9. Computing the topology of a degree 28 algebraic curve with cusps

A first example is given in Fig. 6, where we can see the cells deduced by the subdivision process together with the defining curves (left), and the computed regions (right) based on this cell graph. The boxes span only the actual boundary curves of $\partial\mathcal{S}$, but we also draw the full defining curves to give an idea of the situation.

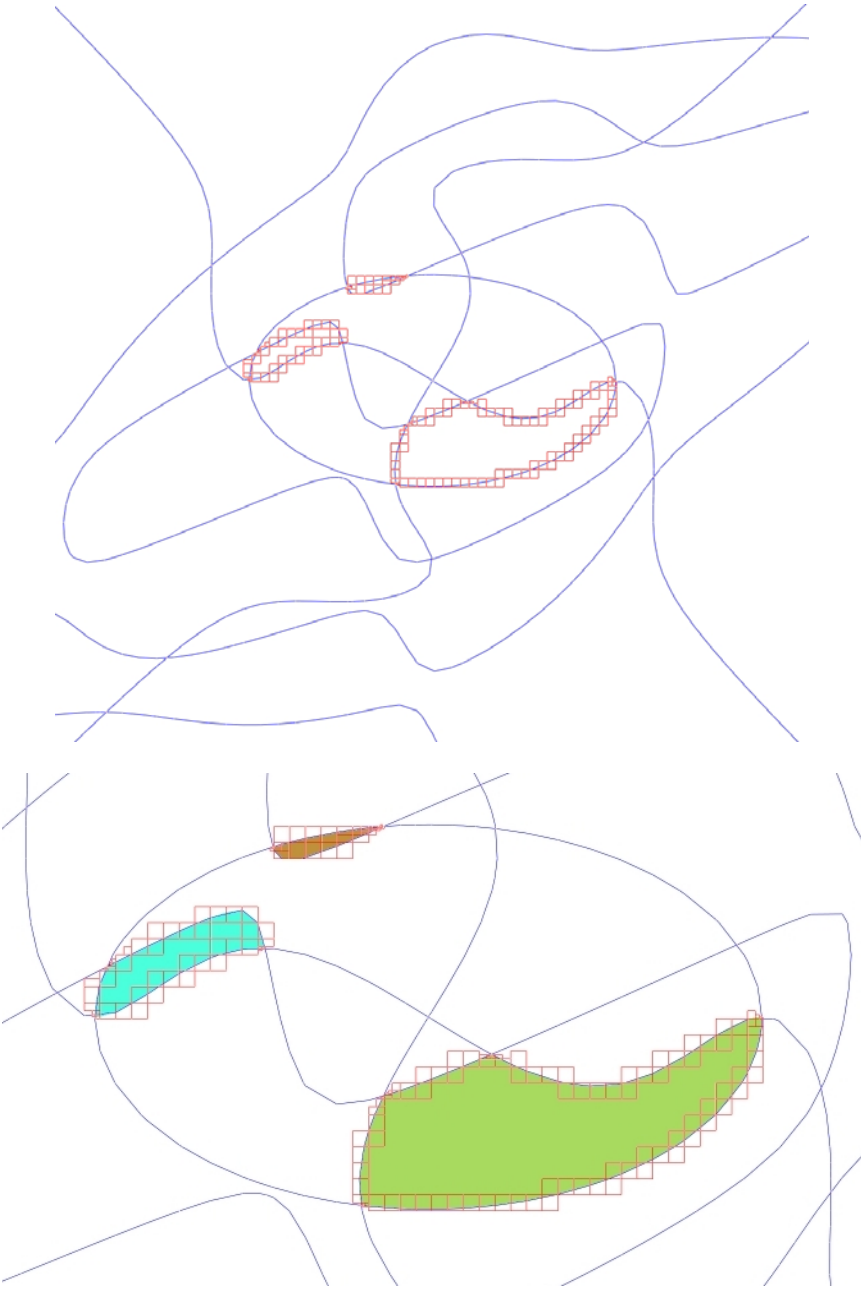


Fig. 10. Semi-algebraic set defined by: $f_1 = -105y^2x^4 - 80y^3 + 140x^3y^3 - 140y^3x + 35y^4 - 105y^4x^2 + 48y^5 + 42xy^5 - 42x^2 + 35x^4 - 7x^6 + 32y + 84xy - 140x^3y + 42x^5y + 210x^2y^2 - 42y^2 - 7y^6 - 8y^7 + 7$, $f_2 = x^2 + 3y^2 - 1$, $f_3 = x^6 + y^2x^4 - y^4x^2 - 2x^4 - y^6 + 2y^4 + x^2 - y^2 + xy$ in domain $[-3, 3]^2$

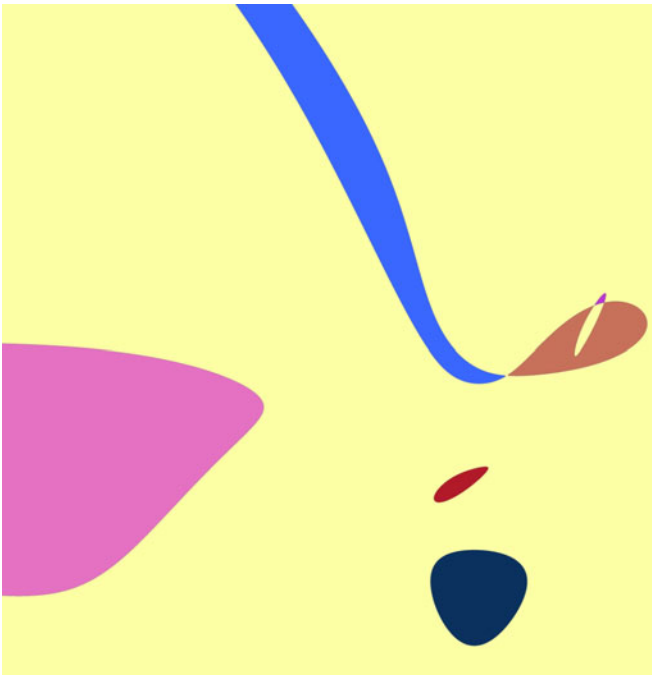


Fig. 11. Topology of a degree 76 curve coming from the self-intersection locus of a 3D surface

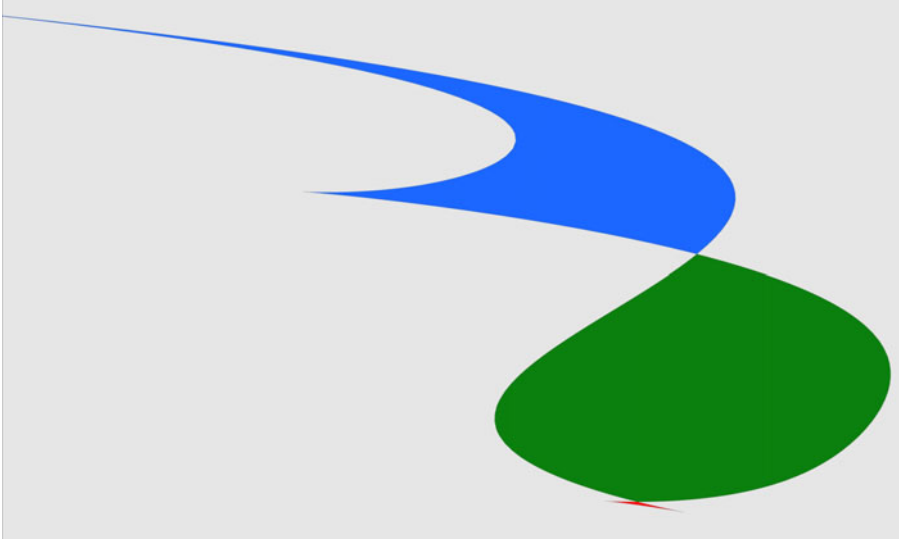


Fig. 12. A (degree 12) apparent contour of 3D surface with cusps

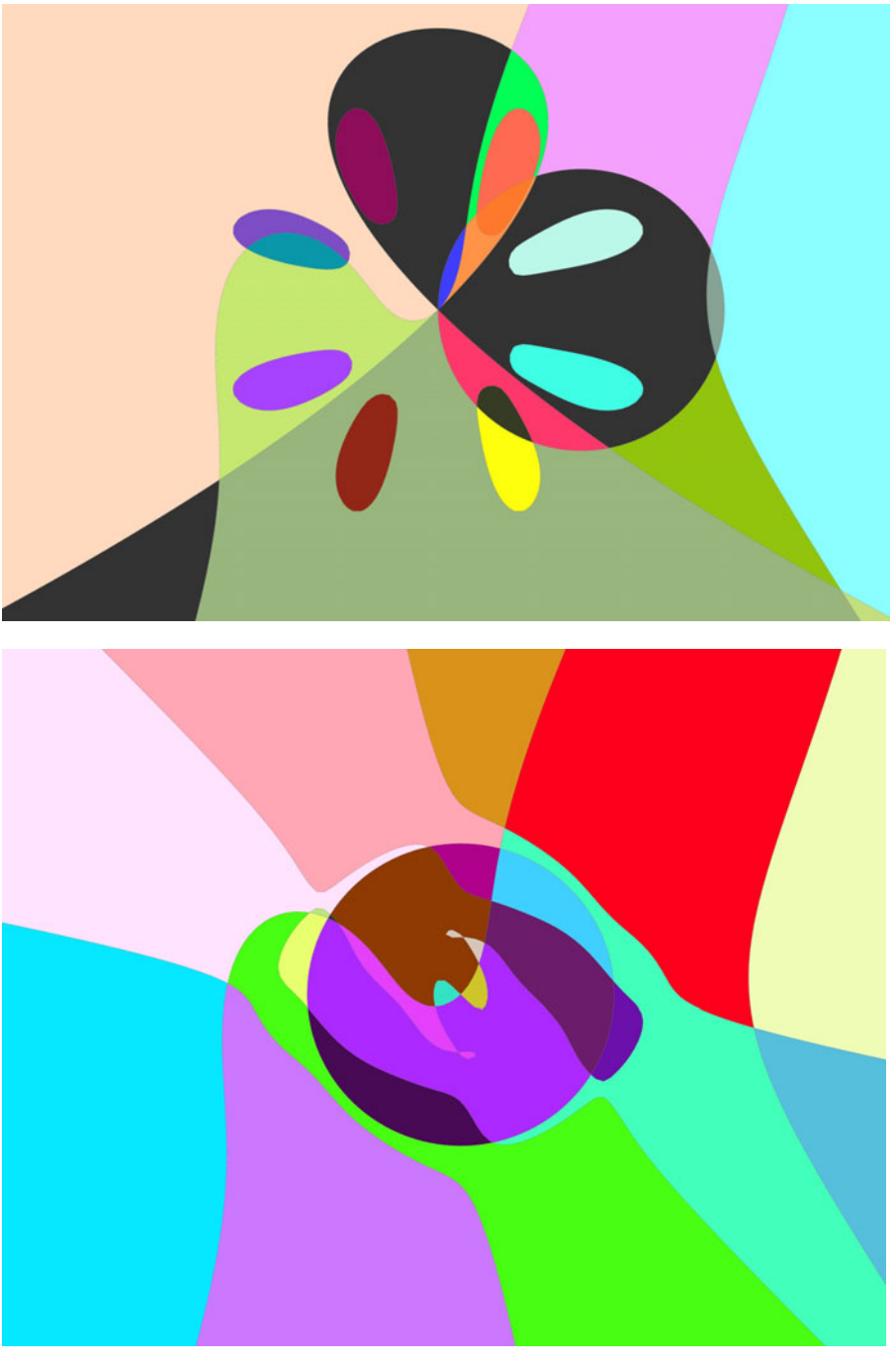


Fig. 13. Regions in the arrangement of three curves, of resp. degrees 32,4,4(top), 32,4,13 (bottom) computed using our algorithm on the underlying semi-algebraic domains.

A precision of $\varepsilon = 0.05$ is used, that is, the cells are subdivided down to this size, to obtain a smooth visual result. Note the two branches that are almost tangent near the bottom left corner. They cause the subdivision to continue further around this area until the branches are properly separated.

In Fig. 7 we compute a set $\mathcal{S} = \{(x, y) : f_1 > 0, f_2 > 0\}$ defined by a degree 6 and a degree 32 polynomial. The domain of computation is $[-1.5, 1.5]^2$ and precision set as before, $\varepsilon = 0.05$. The running time for this example is less than one second. Our implementation is able to handle polynomials of quite higher degree, up to 100 or more. Here the resulting regions contain holes, which are correctly recognized. Finally, Fig. 7 presents the complementary set, $\mathcal{S}^c = \{(x, y) : -f_1 > 0\} \cup \{(x, y) : -f_2 > 0\}$ given by 4 connected components.

The purpose of the third example is to demonstrate how our implementation can handle degenerate cases, namely cusps. We treat a single curve of degree 28, having several cusps. This curve is taken from a real application in non-linear computational geometry, namely the computation of the Voronoi diagram of ellipses, see recent paper [9]. We compute all regions defined by the curve, in the domain $[-7, 3]^2$ and set precision to $\varepsilon = 0.5$. Detailed output is shown in Fig. 9.

Acknowledgments. The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013], Marie Curie ITN SAGA, grant agreement n^o [PITN-GA-2008-214584].

References

1. Alberti, L., Mourrain, B., Wintz, J.: Topology and Arrangement Computation of Semi-algebraic Planar Curves. Comput. Aided Geom. Des. 25(8), 631–651 (2008)
2. Basu, S., Pollack, R., Roy, M.-F.: Complexity of computing semi-algebraic descriptions of the connected components of a semi-algebraic set. In: ISSAC 1998: Proceedings of the 1998 international symposium on Symbolic and algebraic computation, pp. 25–29. ACM, New York (1998)
3. Basu, S., Pollack, R., Roy, M.-F.: Algorithms in Real Algebraic Geometry. Springer, Berlin (2003)
4. Bentley, J.L.: Multidimensional divide-and-conquer. Commun. ACM 23(4), 214–229 (1980)
5. Bochnak, J., Coste, M., Roy, M.-F.: Géométrie Algébrique Réelle. Springer, Heidelberg (1987)
6. Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In: Brakhage, H. (ed.) GI-Fachtagung 1975. LNCS, vol. 33, pp. 134–183. Springer, Heidelberg (1975)
7. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein., C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (2001)
8. Coste, M.: An introduction to semi-algebraic geometry. RAAG network school (2002)
9. Emiris, I.Z., Tsigaridas, E.P., Tzoumas, G.M.: Exact delaunay graph of smooth convex pseudo-circles: general predicates, and implementation for ellipses. In: SPM 2009: 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling, pp. 211–222. ACM, New York (2009)

10. Farin, G.: Curves and Surfaces for CAGD: A Practical Guide. Morgan Kaufmann Publishers Inc., San Francisco (2002)
11. Hardt, R.M.: Triangulation of Subanalytic Sets and proper light subanalytic maps. Invent. Math. 38(3), 207–217 (1976/1977)
12. Henrion, D., Lasserre, J.-B., Lofberg, J.: GloptiPoly 3: Moments, Optimization and Semidefinite Programming. Optimization Methods and Software 24(4-5), 761–779 (2009)
13. Hironaka, H.: Triangulations of algebraic sets. In: Algebraic geometry, Proc. Sympos. Pure Math., Humboldt State Univ., Arcata, Calif., 1974, vol. 29, pp. 165–185. Amer. Math. Soc., Providence (1975)
14. Khimšiašvili, G.N.: The local degree of a smooth mapping. Sakharth. SSR Mecn. Akad. Moambe 85(2), 309–312 (1977)
15. Lasserre, J.B.: Moments, Positive Polynomials and their Applications. Optimization Series, vol. 1. Imperial College Press, London (2009)
16. Mourrain, B., Pavone, J.P.: Subdivision methods for solving polynomial equations. J. Symb. Comput. 44(3), 292–306 (2009)
17. Stenger, F.: Computing the topological degree of a mapping in $\mathbb{R}^n$. Numer. Math. 25(1), 23–38 (1975)
18. Tarski., A.: A decision method for elementary algebra and geometry. Univ. of California Press, Berkeley (1951)

Non-manifold Medial Surface Reconstruction from Volumetric Data

Takashi Michikawa and Hiromasa Suzuki

The University of Tokyo, Tokyo, Japan
{michi,suzuki}@den.rcast.u-tokyo.ac.jp

Abstract. We present a method for medial surface reconstruction from volumetric data of thin-plate objects including junctions. Given medial voxels and distance fields computed from binarized volumes, we polygonize medial voxels by covering them with spherical supports and connecting the center points of the supports. These spherical supports are constructed by distributing spheres depending on the topological type of the voxels so that junction and boundary voxels are distributed first. Triangular meshes are built from Voronoi diagrams on medial voxels. This improvement builds correct junctions, whereas conventional voxel-based methods tend to result in small cavities around them. This paper also demonstrates several results computed from CT-scanned engineering objects.

1 Introduction

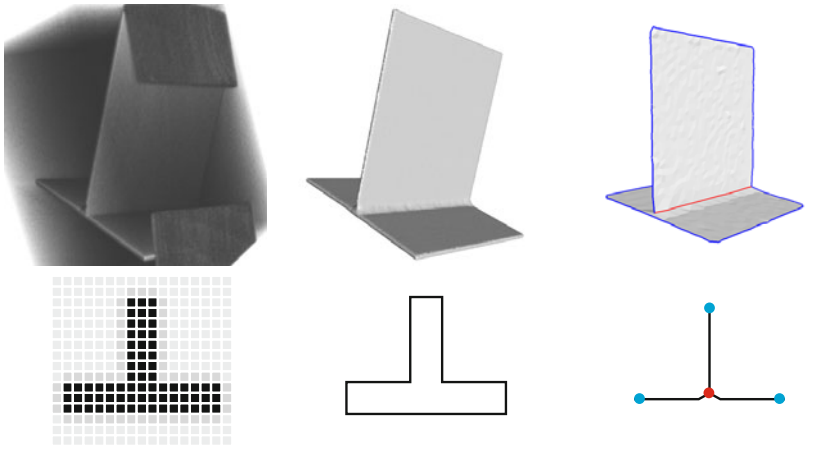
This paper outlines a method for converting volumetric data of thin-plate objects to polygonal meshes for use in digital engineering applications. Thin-plate engineering objects are made of metal plates, and are formed by stamping and welding; they are often found in cars and home electronics. Industrial companies measure such objects as volumetric images using X-ray CT scanners or non-destructive scanning devices (Fig. 1 (a)). Since each voxel has a CT value that identifies its material, we can extract volumetric data with certain threshold values from CT images. Companies need polygonal meshes of these CT images for use in accelerating engineering processes such as comparison with CAD models and CAE simulation.

It is well known that polygonal meshes can be obtained by isosurface contouring of volumetric data [1,2]. This is effective for solid objects, but can create closed surfaces with thin-plate objects (Fig. 1 (b)). The center-line surfaces (Medial surfaces) of the objects shown in Fig. 1 (c) are preferable because common CAD systems represent thin-plate objects as open surfaces [3].

Thin-plate objects are usually welded to other objects for reinforcement. The medial surfaces of such objects become non-manifold, and their welded parts become junctions. Since our focus here is on engineering application, these forms must be reconstructed correctly. One of the challenges involved is capturing non-manifold junctions from CT images correctly as shown in Fig. 1 (c).

The computingation of medial surfaces from volumetric data has been extensively discussed in previous studies [3,4,5]. The basic approach first creates

B. Mourrain, S. Schaefer, and G. Xu (Eds.): GMP 2010, LNCS 6130, pp. 124–136, 2010.



(a) CT data (b) Isosurface of (a) (c) Medial surface of (a)

Fig. 1. CT images of welded aluminum and its polygonal meshes. The resolution of volumetric data (a) is 350x330x220 and its isosurface (b) and medial surface have 130,000 and 3,000 faces (Computation Time: about 1 min.) respectively. Note that the junction edges (red) of the medial surfaces created using our method (c) are correctly reconstructed. Lower row shows corresponding illustrations.

medial voxels from input volumetric data, and polygonal meshes are then created formed based on the resulting voxel connectivity. However, this approach creates small cavities around junctions (Fig. 3 (b)) even though no such cavities exist in the input data. This is due to the ambiguity of triangulation from voxel connectivity. Prohaska and Hege [4] introduced a look-up table for non-manifold topology polygonization. However, local voxels are still checked due to the existence of bad cases of cavities.

We present a method of medial surface reconstruction from volumetric data of thin-plate objects including junctions. Given medial voxels with distance, we first apply sub-sampling to these voxels so that junction and boundary voxels can be preferentially selected. This computation is performed by covering the medial voxels with spherical supports [6]. We then perform Voronoi diagram computation for medial voxels using the sampled points as Voronoi sites to build vertex connectivity, from which triangular meshes are then created.

This approach offers various advantages in medial surface reconstruction from volumetric data. In particular, spherical supports in the sub-sampling phase remove unnecessary voxels around junctions. This contributes to the absorption of various types of artifacts, including branches and small cavities. Indeed, we confirmed this by applying the method to CT-scanned engineering objects.

This paper consists of five sections. We review related work in Section 2, the proposed algorithm is introduced in Section 3, we outline the experimental results and discuss them in Section 4, and Section 5 concludes the paper.

2 Related Work

Our issue is related to a medial axis whose formulation was first introduced in [7]; that is, the set of center points of a collection of maximal inscribed spheres. Many approaches involving MATs (medial axis transforms) have been studied over a period of four decades in the image processing and geometric modeling community. This section reviews several approaches related to our objective.

Volume-based methods. Volume-based approaches compute medial voxels from binarized volumetric data. This produces a volumetric version of the medial axis. A typical strategy is topological thinning, or voxel pruning, which leaves the topology of the input data unchanged (such voxels are called "simple voxels") and preserves the ends of surface voxels. This process is iteratively applied until no more voxels can be removed.

Sequential thinning removes simple voxels one by one. This approach preserves the topology of the original data, but the order of pruning may preserve excess simple voxels and cause many branches or bumps. Although several solutions to this have been discussed (e.g., sub-cycles [8], distance fields [9] and relaxed conditions [10]), the issue has not yet been completely resolved.

Parallel thinning removes all simple voxels on the boundary of an object simultaneously. Tsao and Fu [11] formulated conditions for building smooth surfaces, but the topology of the input model may be lost with this technique. The resulting surfaces are relatively smooth, but it is difficult to preserve topology because many voxels are removed at once. There are also topology-preserving methods for parallel thinning [12,13], but these cause a deterioration in voxel quality. Prohaska and Hege [4] used the geodesic distance between two closest boundary voxels to determine pruning priority. This global metric is more stable for noisy data, and hardly any unnecessary branches are generated. Accordingly, it is used in a number of engineering applications [3,5,14].

Polygonal meshes can be computed from medial voxels. For instance, we can cast this issue to surface reconstruction problems (e.g., [15]). However, non-manifold surfaces cannot be handled. Another approach builds triangular meshes using voxel connectivity, but this method produces ambiguous cases of triangulation from such connectivity, and small cavities may be created. Prohaska and Hege [4] introduced a look-up table for the polygonization of non-manifold topology. However, local voxels are still checked due to the existence of bad cases of cavities.

Polygon-based methods. Alternative approaches involve computing medial axes from polygonal meshes. Although an exact solution can be computed by solving algebraic equations, this is not practical for large models, and most polygon-based methods deal with approximate solutions of MAT. One popular approach is the use of Voronoi diagrams[16,17,18,19].

The structures of medial axes computed using the above methods are usually complex, and may have small branch surfaces. For simplified medial axes, these are usually removed using certain criteria. One popular criterion is the separation

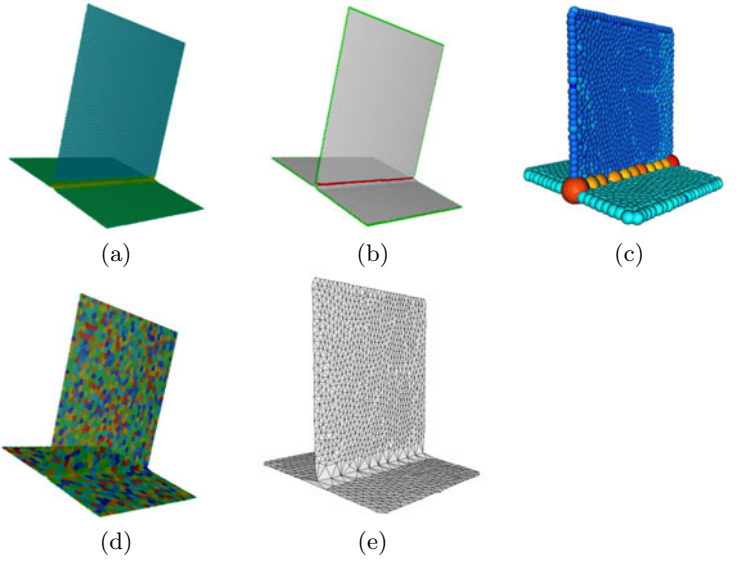


Fig. 2. An overview of our approach. (a) Medial voxels. Each voxel includes a distance to the closest boundary point (indicated by color mapping). (b) Topology classification of (a). Junction and boundary voxels are shown in red and green, respectively. (c) Spherical supports produced by sub-sampling medial voxels based on (b). (d) Voronoi diagram on medial voxels. (e) Result of meshing.

angle of sheets [16,20]. Dey and Zhao [19] used additional conditions to improve enhancement the of quality of medial surfaces. Sud et al. [21] also improved the medial axis simplification method [20] so that to preserve the homotopy of the input data is preserved.

When these techniques are applied to our issue, polygonal meshes must be computed using isosurface extraction methods. This is straightforward, but the resulting meshes have several problems including numerical issues for complex models and artifacts caused by noise and blur in CT images. Accordingly, polygon-based approaches are not practical in achieving our objective.

3 Medial Surface Reconstruction from Medial Voxels

The goal of this algorithm is to compute medial surfaces with junctions as polygonal meshes from volumetric data. Fig. 2 shows an overview of our method.

In our method, the input is a set of binary voxels generated by thresholding a volumetric model obtained by scanning a thin-plate structure. Note that the medial voxels $M = \{v_i\}$ and distance field $D = \{d_i\}$ of the input volumetric models are pre-computed using conventional methods (Fig. 2(a)), where d_i is the distance value of v_i from its nearest voxel in the volumetric data on the boundary computed using distance transforms.

Our method consists of three steps:

Sub-sampling - distributes spherical supports on voxels iteratively so that they completely cover the input medial voxels (Fig. 2(c)).

Voronoi diagram - computes a Voronoi diagram on voxels to build mesh edges (Fig. 2(d)).

Non-manifold meshing - computes polygonal meshes by finding three vertices connected to each other and applies topological cleaning to remove excess non-manifold edges (Fig. 2(e)).

3.1 Sub-sampling of Medial Voxels

The first step of the algorithm involves medial voxel sub-sampling. Given a set of medial voxels M, we select one voxel v_i from M and set it to vertex list V. Then, we remove neighboring voxels v_j such that $||\mathbf{v}_i - \mathbf{v}_j|| < \alpha d_i$ from M, where $\mathbf{v}_i$ denotes the coordinates of v_i, and α denotes the radius scaling factor. Note that a larger value for α makes robust triangulation at very thin parts of the object because the radii of such parts are usually small and may cause bad sampling of medial voxels. Accordingly, a value of $\alpha = 2$ is set for all examples in this paper. Iteration is performed until M is empty.

An important point here is how to find these v_i voxels from M, because their distribution affects mesh topology directly. Fig. 3 illustrates a 2D example. Random distribution may not involve placement at junctions, and sampling points around junctions may be connected to each other, thereby creating small cavities as shown in Fig. 3 (a). On the other hand, in Fig. 3 (b), spheres are distributed at junction and boundary voxels first, and correct polygonization is obtained. We therefore introduce topology-dependent distribution as described below.

Topology criterion control sampling is implemented so that points are well distributed on junctions and boundary edges. This contributes to fastening low-dimensional topology features such as junctions and boundaries, as shown in Fig. 3 (b). We use it here as the primary priority. A similar idea can be found in the bubble mesh algorithm [22] for polygonal meshes with explicit topology, and

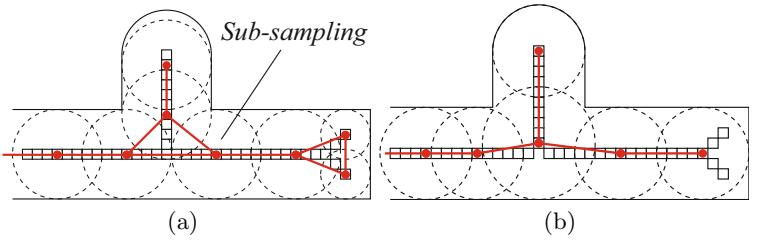


Fig. 3. A 2D comparison of meshing results with random distribution (a) and topology-dependent distribution (b). In each figure, the voxels shown in red denote sampled points, and dotted circles represent the spherical supports of voxels. The red lines show medial surfaces connected by a Voronoi diagram on medial voxels.

we apply similar approach to volumetric data. Classification of voxel topology is described in Appendix A.

We also consider the distance value of voxels as a secondary criterion. This is because medial voxels have relatively larger distances than neighboring voxels, and it is natural that such voxels should be selected first.

Finally, the topology-dependent sub-sampling algorithm is summarized as below.

1. Find the v_i with the top priority in M and copy it to V.
2. Remove v_j from M such that $||\mathbf{v}_i - \mathbf{v}_j|| < \alpha d_i$.
3. Iterate 1 and 2 until M becomes empty.

3.2 Building a Voronoi Diagram on Medial Voxels

We next build mesh edges from the set of sampled vertices V. In our targeting of objects of thin-plate structures, we can assume that medial voxels can be decomposed into a collection of subsets, each of which is an analog of a two-manifold surface (See below for details). Therefore This subset therefore inherits the properties of a two-manifold surface. In particular, it allows us to define Voronoi diagrams on this subset. Fig. 4 illustrates the formulation of Voronoi diagrams on medial voxels. Given sites $V = \{v_i\} \subseteq M$, we define the region $R(v_i)$ as follows :

$$R(v_i) = \{x | d(v_i, x) \leq d(v_j, x), \forall j \neq i\}, \tag{1}$$

where $d(v_i, x)$ denotes the shortest path length between v_i and x in the voxel space. In practice, we compute it using the wavefront propagation of distance from the sites. Fig. 2(d) shows the result of the diagram for medial voxels.We define an edge (v_i, v_j) if there is a neighboring voxel pair (x, y) such that $x \in R(v_i)$ and $y \in R(v_j)$ in 26-adjacent.

Validity of polygonization. Medial voxels in our method can be divided into surface, boundary and junction types by topology classification. They can be

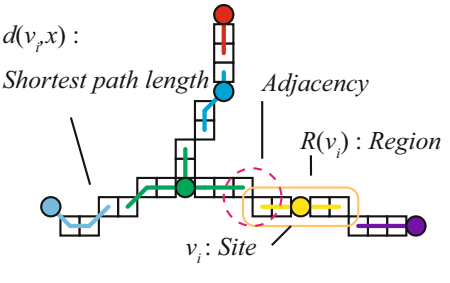


Fig. 4. Voronoi diagram definition on medial voxels. Identically colored voxels belong to the same sites.

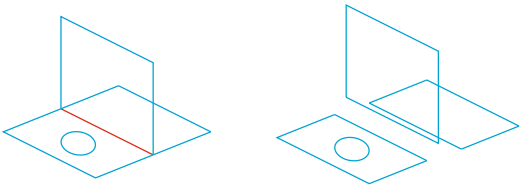


Fig. 5. Medial voxels can be separated into three simple sheets consisting of boundary and surface voxels

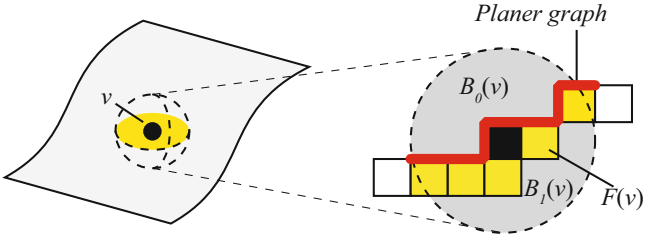


Fig. 6. 2D illustration of neighboring voxels in surface voxels. Note that the planer graph can be defined between $F(v)$ and $B_0(v)$ (indicated by the bold red line).

separated into a set of sheets consisting of boundary and surface voxels by junction voxels (Fig. 5 (b)). We can also consider that the Voronoi diagram is defined on a surface composed of the faces of foreground voxels shared by background voxels. It should be noted here that a voxel is regarded as a cubic volume.

Based on the definition of topological classification, each surface voxel v in a sheet separates its 26-neighboring voxels $N(v)$ into two background components $B_i(v)(i = 0, 1)$ using one foreground component $F(v)$ (Fig. 6). When we focus on $B_0(v)$ and $F(v)$, the boundary surface between their voxels is topologically equivalent to a disk, because $B_0(v)$ is 6-connected and its surface must be two-manifold. This indicates that every surface piece at a voxel on the sheet is topologically equivalent to a disk and thus the total surface of the sheet is topologically equivalent to a two-manifold surface. The Voronoi diagram is therefore defined on surface voxels and we can generate a polygonal mesh by taking the dual of the Voronoi diagram.

3.3 Non-manifold Meshing from Vertex Connectivity

In the final step, polygonal meshes are constructed from the vertices and edges computed in the previous steps. This computation is performed by finding triples (v_i, v_j, v_k) that are connected to each other (Fig. 7).

This approach creates non-manifold meshes with no small cavities caused by invalid sampling, but may also create overlapping triangles on surface voxels

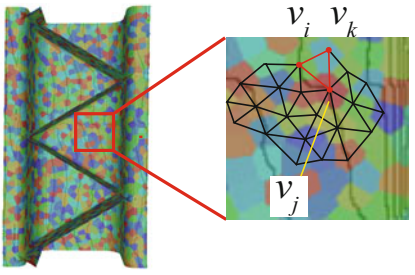
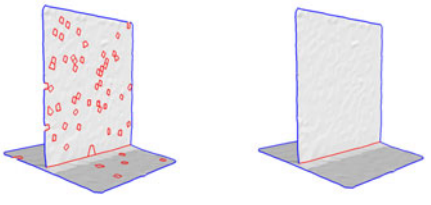


Fig. 7. Polygonal mesh generation from a Voronoi diagram



(a)The result of meshing (b) After cleaning

Fig. 8. Topological cleaning for non-manifold meshes. The red and blue lines show junctions and boundaries, respectively.

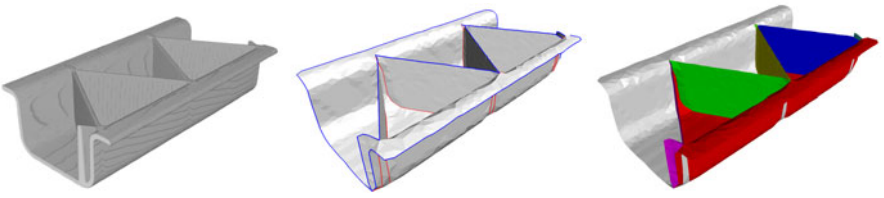
(Fig. 8 (a)) for similar reasons related to the degeneration of Delaunay triangulation. Such triangles must be eliminated so that necessary ones around junctions are not removed.

Based on medial voxel topology classification, we decompose all vertices into manifold vertices (on surfaces) and non-manifold vertices (on boundaries or junctions), and then apply manifold cleaning [6] to the manifold vertices only.

Simply speaking, this method performs iteration to find a planar graph from the one-ring neighborhood of each vertex such that the number of vertices in the graph is maximized. Triangles excluded from the planar graph are removed from the mesh. Fig. 8 (b) shows the results of cleaning. Note that some overlapping triangles may still be left if all vertices belong to junctions. Such issues are generally seen when junctions are very close.

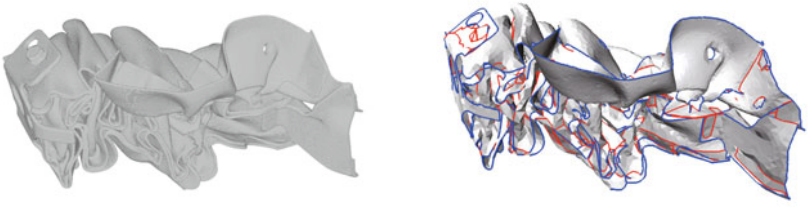
4 Result and Discussion

The experimental results of meshing are shown in Figs. 1, 9 and 10. The results of decomposition into a set of manifold polygonal surfaces are also given in Fig. 9 (c). All examples were tested on a 3.16 GHz computer, and the computation times are provided in the caption of each figure. There is room for improvement here, as our prototype is not yet optimized. The current bottleneck lies in Voronoi diagram creation and mesh cleaning. Since Voronoi



(a) Input volumes (b) Medial surfaces (c) Segmentation

Fig. 9. Experimental results for CT-scanned shock absorber data (400 x 400 x 576 voxels; computation time: 5 min.))



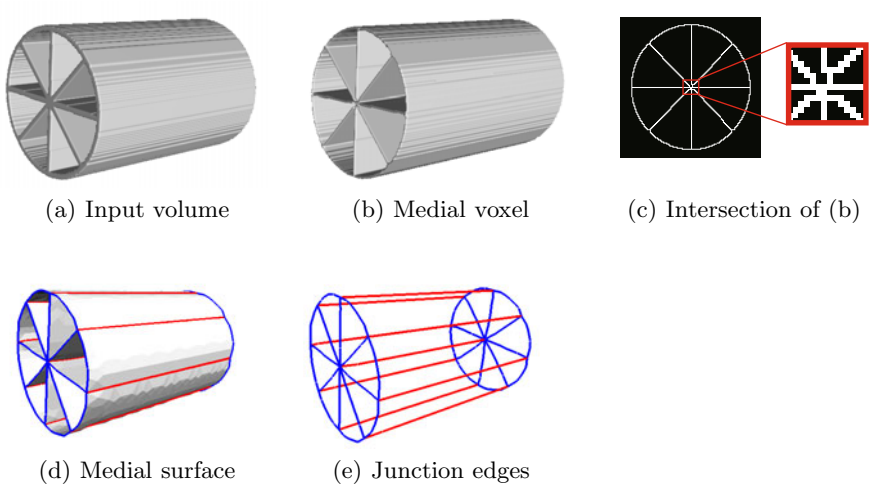
(a) Input volume (b) Result

Fig. 10. Experimental results for a crushed automobile frame (708 x 965 x 813 voxels; computation time: over 8 hours)

diagrams compute using a multi-source Dijkstra's algorithm, the computation cost is $O(n \log n)$. Note that although connecting spherical supports [6] may create polygons fast, the technique truncates topology information hidden in medial voxels, whereas the Voronoi-based method considers such topological information. The non-manifold mesh data structure used in cleaning is based on an IndexedFaceSet (a.k.a. Wavefront OBJ format), and takes a long time to search neighboring voxels. An advanced data structure for non-manifold meshes (e.g., [23]) is expected to improve the level of performance.

Using topology-dependent sampling, medial surfaces with junctions were reconstructed from CT-scanned engineering objects. In addition, sub-sampling with spherical supports contributes to the recovery of non-manifold junctions.

Fig. 11 shows an efficient example with junctions connecting to multiple sheets. In voxel space, it is hard to represent such complex junctions exactly, so they are represented by a set of simple junctions as an approximation (Figs. 11 (b) and (c)). Since conventional methods create polygonal meshes from them, the meshing result also includes a set of simple junctions. On the other hand, our method removes neighboring junction voxels during the sub-sampling phase using spherical supports. This is because correct junction voxels have larger distance values, and unnecessary ones are included in the spherical supports of correct junction points. Fig. 11 (d) shows the outcome, and non-manifold edges are also shown in Fig. 11 (e). Note that the center axis of the object is represented by one line sequence.



(a) Input volume (b) Medial voxel (c) Intersection of (b)

(d) Medial surface (e) Junction edges

Fig. 11. Medial surface reconstruction from an artificial volume (256 x 256 x 256) with high-valence junctions. Note that the center axis can represent only one edge sequence in (d) and (e).

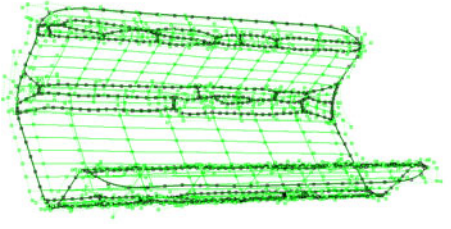


Fig. 12. CAD surface reconstruction from the bottom part of shock absorber model

Our method contributes to the acceleration of digital engineering processes. One possible application is in conversion to parametric surfaces (a.k.a. reverse engineering). Fig. 12 shows the results of converting NURBS surfaces using reverse engineering software. Once CAD models are converted, they can be used for a number of advanced applications such as CAE and quality assurance.

One limitation of our method is that the output surfaces represent only thin-plate objects even though solid objects are given. It is necessary to combine isosurface meshing techniques with our method to handle thin-plate and solid objects simultaneously (see [10] for details). Another limitation is its use with thin objects. We use medial voxels sampled on a uniform grid, which causes singularity of dual meshes when the thickness value is small. This can be eliminated by taking larger spherical supports or larger values of α. However, with larger α values, close junctions may be merged.

5 Conclusion and Future Work

This paper has proposed a method for non-manifold medial surface reconstruction using volumetric data of thin-plate objects. The technique applies sub-sampling on medial voxels with spherical supports depending on digital topology so that junctions and boundary voxels are preferentially sampled. Triangular meshes are computed by connecting neighboring triple vertices. Since sampling points are placed on singular voxels, junction edges are well reconstructed. Indeed, we applied this method to several CT images of thin-plate engineering objects.

A number of directions need to be followed in future work. The first is the simultaneous meshing of solid, surface and wire objects [10]. We believe our approach has the potential to handle this issue by controlling the distribution of sub-sampling. Another direction is to extend the method to unrecognized points extracted by laser scanners. If junctions and boundary points can be estimated in a similar way to digital topology, then our method can be applied to point sets of non-manifold models, whereas conventional methods assume manifolds.

Acknowledgment

The authors would like to thank Koichi Matsuzaki for converting polygonal meshes to CAD data using his reverse engineering software. Crashed side frame CT images are courtesy of Honda. This work is partially supported by Grant-in-Aid for Young Scientists(B) (No.20760096) and Grant-in-Aid for Scientific Research(B) (No.19360070).

References

1. Lorensen, W.E., Cline, H.E.: Marching cubes: A high resolution 3d surface construction algorithm. In: SIGGRAPH 1987: Proceedings of the 14th annual conference on Computer graphics and interactive techniques, pp. 163–169. ACM, New York (1987)
2. Ju, T., Losasso, F., Schaefer, S., Warren, J.: Dual contouring of hermite data. In: SIGGRAPH 2002: Proceedings of the 29th annual conference on Computer graphics and interactive techniques, pp. 339–346. ACM, New York (2002)
3. Fujimori, T., Suzuki, H., Kobayashi, Y., Kase, K.: Contouring medial surface of thin plate structure using local marching cubes. In: International Conference on Shape Modeling and Applications, pp. 297–306 (2004)
4. Prohaska, S., Hege, H.C.: Fast visualization of plane-like structures in voxel data. In: Proceedings of the conference on Visualization 2002, Washington, DC, USA, pp. 29–36. IEEE Computer Society, Los Alamitos (2002)
5. Suzuki, H., Fujimori, T., Michikawa, T., Miwata, Y., Sadaoka, N.: Skeleton surface generation from volumetric models of thin plate structures for industrial applications. In: Martin, R., Sabin, M.A., Winkler, J.R. (eds.) Mathematics of Surfaces 2007. LNCS, vol. 4647, pp. 442–464. Springer, Heidelberg (2007)
6. Ohtake, Y., Belyaev, A., Seidel, H.P.: An integrating approach to meshing scattered point data. In: SPM 2005: Proceedings of the 2005 ACM symposium on Solid and physical modeling, pp. 61–69. ACM, New York (2005)

7. Blum, H.: A Transformation for Extracting New Descriptors of Shape. In: Wathen-Dunn, W. (ed.) Models for the Perception of Speech and Visual Form, pp. 362–380. MIT Press, Cambridge (1967)
8. Palágyi, K., Kuba, A.: A parallel 12-subiteration 3d thinning algorithm to extract medial lines. In: Sommer, G., Daniilidis, K., Pauli, J. (eds.) CAIP 1997. LNCS, vol. 1296, pp. 400–407. Springer, Heidelberg (1997)
9. Toriwaki, J., Mori, K.: Distance transformation and skeletonization of 3d pictures and their applications to medical images. In: Bertrand, G., Imiya, A., Klette, R. (eds.) Digital and Image Geometry. LNCS, vol. 2243, pp. 412–428. Springer, Heidelberg (2002)
10. Ju, T., Baker, M.L., Chiu, W.: Computing a family of skeletons of volumetric models for shape description. Computer Aided Design 39(5), 352–360 (2007)
11. Tsao, Y.F., Fu, K.S.: A parallel thinning algorithm for 3-d pictures. Computer Graphics and Image Processing 17(4), 315–331 (1981)
12. Manzanera, A., Bernard, T., Preteux, F., Longuet, B.: Medial faces from a concise 3d thinning algorithm. In: IEEE International Conference on Computer Vision, vol. 1, p. 337 (1999)
13. Borgefors, G., Nystrom, I., Baja, G.S.D.: Computing skeletons in three dimensions. Pattern Recognition 32(7), 1225–1236 (1999)
14. Michikawa, T., Nakazaki, S., Suzuki, H.: Efficiend medial voxel extraction from large volumetric models. In: Proceeding of WSCG 2009, pp. 169–176 (2009)
15. Dey, T.K., Li, K., Ramos, E.A., Wenger, R.: Isotopic reconstruction of surfaces with boundaries. Computer Graphics Forum 28(5), 1371–1382 (2009)
16. Attali, D., Montanvert, A.: Computing and simplifying 2d and 3d continuous skeletons. Computer Vision and Image Understanding 67(3), 261–273 (1997)
17. Etzion, M., Rappoport, A.: Computing the voronoi diagram of a 3-d polyhedron by separate computation of its symbolic and geometric parts. In: SMA 1999: Proceedings of the fifth ACM symposium on Solid modeling and applications, pp. 167–178. ACM, New York (1999)
18. Amenta, N., Choi, S., Kolluri, R.K.: The power crust. In: SMA 2001: Proceedings of the sixth ACM symposium on Solid modeling and applications, pp. 249–266. ACM, New York (2001)
19. Dey, T.K., Zhao, W.: Approximate medial axis as a voronoi subcomplex. In: SMA 2002: Proceedings of the seventh ACM symposium on Solid modeling and applications, pp. 356–366. ACM, New York (2002)
20. Foskey, M., Lin, M.C., Manocha, D.: Efficient computation of a simplified medial axis. In: SM 2003: Proceedings of the eighth ACM symposium on Solid modeling and applications, pp. 96–107. ACM, New York (2003)
21. Sud, A., Foskey, M., Manocha, D.: Homotopy-preserving medial axis simplification. In: SPM 2005: Proceedings of the 2005 ACM symposium on Solid and physical modeling, pp. 39–50. ACM, New York (2005)
22. Shimada, K., Gossard, D.C.: Bubble mesh: automated triangular meshing of non-manifold geometry by sphere packing. In: SMA 1995: Proceedings of the third ACM symposium on Solid modeling and applications, pp. 409–419. ACM, New York (1995)
23. Masuda, H.: Topological operators and boolean operations for complex-based non-manifold geometric models. Computer-Aided Disign 25(2), 119–129 (1993)
24. Malandain, G., Bertrand, G., Ayache, N.: Topological segmentation of discrete surfaces. International Journal of Computer Vision 10(2), 183–197 (1993)

A Topology Classification of Voxels

The topology of each voxel can be estimated by counting the number of foreground and background components in neighboring voxels [24]. In this method, the following two entities are used:

- C^* : The number of 26-connected foreground voxel components in 26-neighbor voxels of v that are 26-adjacent to v.
- $\bar{C}$: The number of 6-connected background voxel components in 18-neighbor voxels of v that are 6-adjacent to v with the exception of v itself.

Using these entities, junction, boundary and surface voxels can be classified as follows :

- Junction voxel : $C^* = 1\ \bar{C} > 2$
- Boundary voxel : $C^* = 1\ \bar{C} = 1$
- Surface voxel : $C^* = 1\ \bar{C} = 2$

Fig. 13 shows examples of classification.

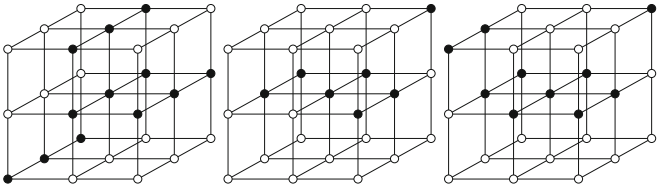


Fig. 13. Examples of topology classification. Foreground and background voxels are indicated in black and white, respectively.

Decomposing Scanned Assembly Meshes Based on Periodicity Recognition and Its Application to Kinematic Simulation Modeling

Tomohiro Mizoguchi[1] and Satoshi Kanai[2]

[1] Department of Computer Science, College of Engineering, Nihon University,
1 Nakagawara, Tokusada, Tamuramachi, Koriyama, Fukushima, 963-8642, Japan
mizo@cs.ce.nihon-u.ac.jp
[2] Graduate School of Information Science and Technology, Hokkaido University,
Kita-14, Nishi-9, Kita-ku, Sapporo, 060-0814, Japan
kanai@ssi.ist.hokudai.ac.jp

Abstract. Along with the rapid growth of industrial X-ray CT scanning systems, it is now possible to non-destructively acquire the entire meshes of assemblies consisting of a set of parts. For the advanced inspections of the assemblies, such as estimation of their assembling errors or examinations of their behaviors in the motions, based on their CT scanned meshes, it is necessary to accurately decompose the mesh and to extract a set of partial meshes each of which correspond to a part. Moreover it is required to create models which can be used for the real-product based simulations. In this paper, we focus on CT scanned meshes of gear assemblies as examples and propose beneficial methods for establishing such advance inspections of the assemblies. We first propose a method that accurately decomposes the mesh into partial meshes each of which corresponds to a gear based on periodicity recognitions. The key idea is first to accurately recognize the periodicity of each gear and then to extract the partial meshes as sets of topologically connected mesh elements where periodicities are valid. Our method can robustly and accurately recognize periodicities from noisy scanned meshes. In contrast to previous methods, our method can deal with single-material CT scanned meshes and can estimate the correct boundaries of neighboring parts with no previous knowledge. Moreover it can efficiently extract the partial meshes from large scanned meshes containing about one million triangles in a few minutes. We also propose a method for creating simulation models which can be used for a gear teeth contact evaluation using extracted partial meshes and their periodicities. Such an evaluation of teeth contacts is one of the most important functions in kinematic simulations of gear assemblies for predicting the power transmission efficiency, noise and vibration. We demonstrate the effectiveness of our method on a variety of artificial and CT scanned meshes.

1 Introduction

Mechanical products, especially the ones used in power transmission machineries, such as gear trains, bearings, ball screws and chain sprockets, are composed as assemblies consisting of a set of parts. Since such assemblies are typically covered by housings, it

B. Mourrain, S. Schaefer, and G. Xu (Eds.): GMP 2010, LNCS 6130, pp. 137–160, 2010.

is difficult to observe the assembling errors of each part or the dynamical behavior in their motions from outside. If one could non-destructively capture the source of the assembling errors or the behaviors in the motions of the assemblies, it would enable real product-based inspections. Such advanced inspections would be a big contribution for the performance improvements of mechanical products.

On the other hand, industrial X-ray CT scanning systems have been rapidly developed and it is now possible to non-destructively capture the "static" conditions of the entire products inside the housings. However such CT scanned static data cannot be directly used for inspections of assemblies. Specifically, for investigating the source of the assembling errors, the spatial position and the posture of each part should be identified. And for examining the behavior in the motions, the model of each part and their contact relations should be identified, the kinematic or multi-body dynamic simulation models need to be created, and then real-product based simulations must be performed. Hence, to achieve such advanced inspections of the assemblies, it is necessary to extract a set of separated data each of which corresponds to a part from their CT scanned static data and to create models which can be used for simulating the assembling conditions and the motions.

The problems become much difficult when we deal with single material assemblies. Although the entire surface meshes of the assemblies can be acquired by the CT scannings, the boundary information between parts cannot be clearly captured in the meshes. Therefore, using the current methods, we cannot decompose the assembly mesh and generate a set of models each of which corresponds to a single part.

In this paper, we focus on the periodicities on the surfaces of assembly components and propose beneficial methods that contribute to the advanced inspections of mechanical products based on periodicity recognitions. We first propose an automatic method that decomposes a CT scanned surface mesh of parts assembly into separated partial meshes each of which corresponds to a part based on periodicity recognition. Our method recognizes a set of rotational periodicities of parts from a CT scanned assembly mesh. Then, using the periodicities, the parts boundaries which cannot be captured by the CT scanning are estimated in the proper way from the engineering view points and the mesh is decomposed into separated parts along the boundaries. Our proposed method enables the accurate decomposition of single material scanned meshes which previous methods failed. And it also enables the generation of partial meshes whose boundaries are properly interpolated. In addition, as an application of our mesh decomposition, we propose a new method for advanced inspections, estimating teeth contacts, of gear trains which are the most frequently used assemblies in power transmission machineries. Our method generates models for estimating assembling conditions and for performing real-product based kinematic simulations.

We deal with only gear trains in this paper, but our method is not limited to them. It can also be used for the assemblies where rotational periodicities exist in the contact area between parts, i.e., power transmissions such as bearings and chain sprockets.

2 Related Works

2.1 Periodicity Recognition

Many algorithms have been proposed for recognizing periodicities in 2D images. Lin et al. [6] proposed an algorithm that recognizes a translational periodicity in a 2D

texture based on the generalized Hough transform. Liu et al. [7] proposed an algorithm that recognizes a variety of periodicities, including translations, rotations, and reflections, based on the crystallographic theory. Müller et al. [8] proposed an algorithm for extracting a translational periodicity from a 2D façade image of a building by subdividing the image and evaluating the mutual information between the subdivided images. However such algorithms cannot be easily extended to 3D meshes.

Periodicity recognitions have strong relations with symmetry detections in the sense that they both find pairs of local shapes that can be matched each other under certain transformations. Recently symmetry detections have gained much attention and many algorithms have been proposed in computer graphics field [9,10,11]. However these algorithms aim at detecting pairs of congruent regions and their transformations for the matching, and they cannot extract the parameters defining the periodicities.

Liu et al. [12] proposed an algorithm that extracts a single region from the periodically displaced ones in a 3D mesh of a relief with user interactions. However, this algorithm cannot extract a set of periodically-displaced regions and the parameters defining their periodicity. Pauly et al. [13] proposed an algorithm that discovers a class of periodicities defined by a combination of translations, rotations, and uniform scalings. The algorithm is based on computations of transformations under which pairs of local coordinate systems around the points can be matched and their voting to a transformation space, grid fittings in transformation space for sets of voted points, and the simultaneous registration in 3D space. However the algorithm needs to evaluate a large number of pairs of points for stably and robustly discovering periodicities on large and noisy scanned data, therefore, it needs long computational time for such data.

Li et al. [21] proposed an algorithm for detecting a wide class of approximate symmetries from B-rep CAD models by extracting characteristic points and analyzing their connectivity. The authors then used detected symmetries for estimating design intents from B-rep models [22]. However, the symmetry detection algorithm proposed in [21, 22] may be unable to work or to accurately detect the symmetries in case of noisy CT scanned meshes. Moreover the method cannot explicitly compute the parameters defining the periodicities such as rotational axes or the basis angles in case of the rotational periodicity.

2.2 Decomposition of 3D Models into Parts

Many methods have been proposed for decomposing 3D models into parts in the computer graphics field [2,3]. These methods aim at segmenting character models such as animals into parts, i.e., heads, bodies, and legs. However these methods finds only the authentic-looking boundaries between neighboring parts by analyzing concave area of the model based on optimizations and thus cannot uniquely define the correct boundaries in meshes of parts assemblies.

Alternative methods have been proposed in the digital engineering field. Shammaa et al. [4] proposed a method that registers the original CAD data corresponding to each part with CT volumetric data of parts assemblies and then accurately decomposes the data into parts each of which can be closely registered with a CAD data. However this method cannot be used when the original CAD data are not provided,

for example, a case of old products. Shammaa et al. [5] also proposed a method that decomposes the multi-material CT volumetric data into parts by analyzing the CT values which differ per material based on graph cut. However this method cannot be used for single-material data such as gear trains where CT values are almost constant.

3 An Overview of Our Method

From this section, the details of our method are described by taking a gear train as an example. In CAD systems, an entire model of a single gear can be designed first by creating a partial model including one tooth and then by rotating it around the axis by the integral multiple of the basis angle. Therefore, as shown in Figure 1, for recognizing such a rotational periodicity from a scanned mesh, it is required to extract a basis region R_0 and a set of parameters defining the periodicity, such as an axis directional vector $\mathbf{d}$, a point on an axis $\mathbf{p}$, and a basis angle θ.

In this paper, we propose a new method that recognizes rotational periodicities on a CT scanned surface mesh of an assembly and then decomposes the mesh into partial meshes each of which corresponds to a part based on the periodicities. In addition, we propose a method that evaluates gear teeth contact, which is one of the most important functions in kinematic simulations, using the results of periodicity recognition and the partial meshes. As shown in Figure 2, our method consists of following four steps.

(a) Gear (b) Scanned mesh (c) Rotational grid

Fig. 1. Periodicity recognition

Step1: Segmentation and classification of regions (Section 4)
Given a CT scanned surface mesh of a gear train generated from its CT volumetric data, principal curvatures are estimated at each vertex based on local polynomial fittings and high curvature vertices are detected by thresholding the estimated curvatures [14,15] (Section 4.1). Then a mesh is segmented into separated regions, which are sets of topologically connected vertices, so that each region can be bounded by high curvature vertices (Section 4.1). Next, the pairwise ICP matching [16] is performed for finding a set of congruent regions mostly corresponding to gear teeth from the segmented regions under the fact that all teeth in a single gear represent the same geometry (Section 4.2). Finally the congruent regions are classified into groups each of which corresponds to a gear based on the RANSAC algorithm (Section 4.3).

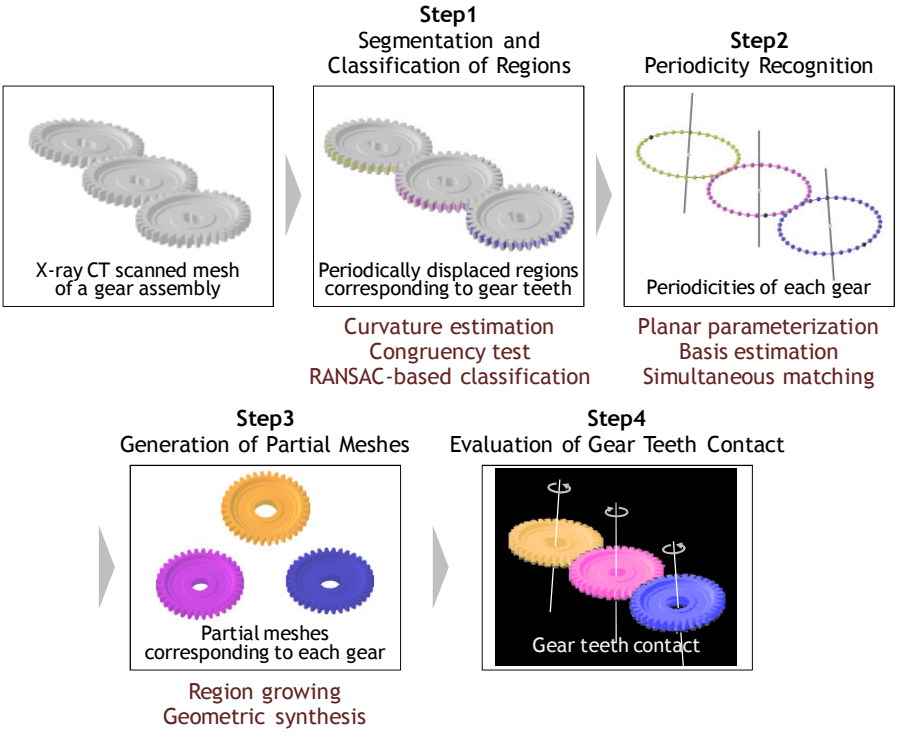


Fig. 2. An overview of our method

Since the top, bottom, and sides of teeth in almost all gears include minute fillets or chamfers and the estimated curvatures in such regions are high compared with those in teeth regions, our simple segmentation can extract a set of regions corresponding to gear teeth separately. Moreover our method uses only boundary vertices of regions for congruency tests based on the ICP matching, and it much reduces the computational cost without decreasing the matching accuracy.

Step2: Periodicity recognition (Section 5)
For each group of regions, a least-squares plane is fitted to a set of barycenters of the regions. Then the barycenters are projected onto the plane and their 2D parameters are calculated (Section 5.1). Then the regions in a group are classified into sub groups in the case of gears whose tooth are originally designed by multiple faces in CAD system (Section 5.2). Next, for each group or each sub group, an initial rotational axis and a basis angle are extracted based on our basis estimation method [20] which is the modification of Lin's [6] and then an index that specifies the multiple of the basis angle is assigned to each region (Section 5.3). Finally optimal parameters defining the periodicity are extracted based on our simultaneous matching algorithm [20] (Section 5.4).

Our basis estimation and simultaneous matching methods enable accurate extraction of parameters defining the periodicities from noisy scanned meshes. Our method

also uses only boundary vertices of regions for the simultaneous matching and it much reduces the computational cost without decreasing the extraction accuracy.

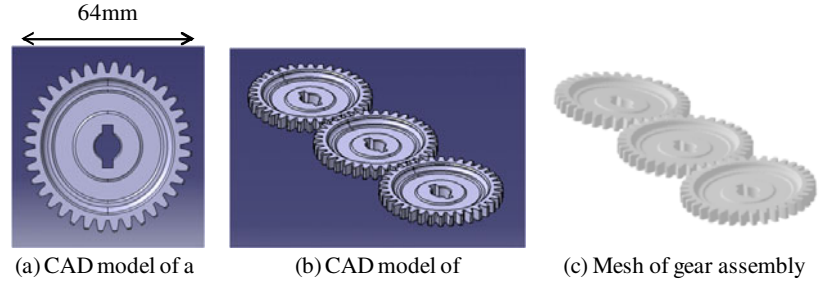
Step3: Generation of partial meshes (Section 6)
Extracted regions in step 1 are then simultaneously enlarged by our region growing according to the periodicities (Section 6.1). Next, points are interpolated by our geometric synthesis algorithm in the area where the regions are missing due to the segmentation and classification failure or where the surface meshes are not generated in the contact area of neighboring parts. This algorithm enables to generate all-round points for each gear (Section 6.2). Finally the partial meshes each of which corresponds to a gear can be generated by triangulating the points.

Our region growing enables the extraction of as large regions as possible where periodicities are valid under the user specified tolerance. And our geometric synthesis enables to accurately estimate the boundaries of neighboring gears and to extract the partial meshes each of which corresponds to a single gear.

Step4: Evaluation of gear teeth contact (Section 7)
Then pairs of partial gear meshes which are contact each other are sequentially rotated around the axis by the constant pitch angle and their teeth contacts are estimated at every angle. In our method, estimations of teeth contacts are reduced to the distance computations between surfaces approximating the regions corresponding to the teeth.

Such surface approximations reduce the effect of the scanning noise for the evaluations of gear teeth contacts. Moreover our method enables to estimate the behaviors in the motions of real-world gear assemblies based on their CT scanned surface meshes.

From the next section, the details of our method are explained using the mesh in Figure 3. To generate this mesh, we first copied the CAD model of the single gear with thirty six teeth in Figure 3(a) and then created the gear train model consisting of three gears in Figure 3(b). Then we tilted the rotational axes of the two gears by five degrees with reference to the one and generated the gear train mesh with 447,840 triangles in Figure 3(c) by triangulating the CAD model using the CAE meshing software. We also added the artificial noise to the mesh by displacing each vertex along its normal direction by a Gaussian distributed random distance with the standard deviation 5% proportional to the averaged mesh edge length.



(a) CAD model of a single gear (#teeth=36) (b) CAD model of gear assembly (c) Mesh of gear assembly (#tri: 447,840)

Fig. 3. An artificial mesh of a gear assembly

4 Segmentation and Classification of Regions

4.1 Curvature Estimation and Segmentation

In the beginning of our method, principal curvatures are estimated at each vertex based on the local quadratic polynomial surface fitting [14,15] as shown in Figure 4(a). Then, if the estimated maximum principle curvature $\kappa_{\max}$ at each vertex satisfy $1/|\kappa_{\max}| < th_{high}\, l_{avg}$, it is classified as high curvature vertex. As shown in Figure 4(b), most of high curvature vertices are in the vicinity of minute fillets or chamfers. Here th_{high} is the threshold which must be set in trial-and-error manner depending on the geometry and the resolutions of the mesh and l_{avg} is the averaged mesh edge length. Then the mesh is segmented into separated regions each of which is a set of topologically connected vertices and is bounded by high curvature vertices as shown in Figure 4(c). Since the top, bottom, and sides of teeth in almost all gears include minute fillets or chamfers and the estimated curvatures in such regions are high compared with those in teeth regions, our simple segmentation can extract a set of regions corresponding to gear teeth separately.

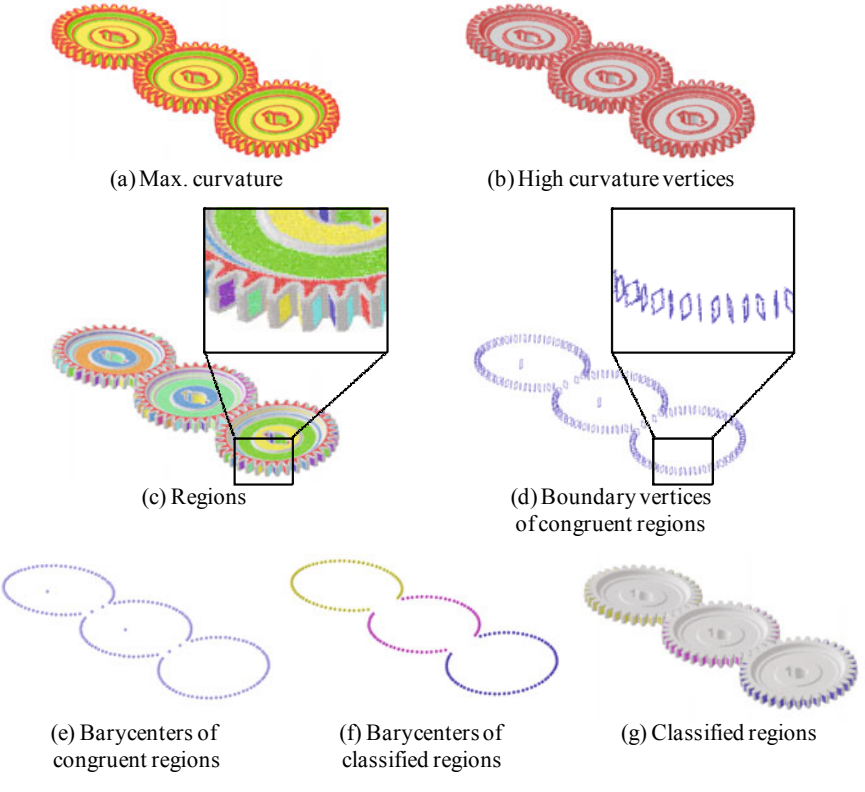


Fig. 4. Segmentation and classification of regions

4.2 Selection of Congruent Regions

Next, under the fact that all gear teeth in a single gear represent the same geometry, the congruency tests are performed for the segmented regions $\{R_\alpha | 1 \le \alpha \le n_{seg}\}$ based on the pairwise ICP algorithm [16] for selecting a set of congruent regions mostly corresponding to gear teeth. Here we assume that the most of $\{R_\alpha\}$ are congruent ones. The ICP algorithm matches pairs of regions so that the sum of the distances between corresponding points can be minimized. The congruent regions represent the same geometry and the pair of them can be matched by the ICP algorithm so that their averaged distance is minimized. However, when non congruent regions are paired, the distance will be larger. Based on this fact, our method applies the following procedures for selecting congruent regions.

Our method first randomly select the user specified number of regions $\{Q_\beta\}$ among $\{R_\alpha\}$, and for each Q_β, create a set of pairs $\{\langle Q_\beta, R_\alpha\rangle | 1 \le \alpha \le n_{seg}\}$.

1. For each pair in $\{\langle Q_\beta, R_\alpha\rangle\}$, apply the ICP algorithm to match them and compute the averaged distances $\{e_{\alpha\beta}\}$ between the corresponding points.
2. For each Q_β, count the number of false pairs $n_{\alpha\beta}$ whose $e_{\alpha\beta}$ is more than the threshold th_{cong}. We set th_{cong} so that it becomes proportional to the averaged mesh edge length l_{avg}, such that $th_{cong} = \tau_{cong} l_{avg}$. We set $\tau_{cong} = 1.0$.
3. If $n_{\alpha\beta} > w_{cong} n_{seg}$, verify that Q_β is not a congruent region and then do not apply the following procedure to any $\langle Q_\beta, R_\alpha\rangle$. Otherwise, select Q_β as a congruent region and then evaluate $e_{\alpha\beta}$ for each pair $\langle Q_\beta, R_\alpha\rangle$. If $e_{\alpha\beta}$ is less than th_{cong}, select R_α as a congruent region. Here we set $w_{cong} = 0.5$.

As a result of this process, a set of congruent regions most of which correspond to gear teeth can be accurately selected. As shown in the Figure 4(d), our method uses only boundary vertices of regions for the ICP matching and it much reduces the computational cost without decreasing the matching accuracy. We note that several regions which do not correspond to gear teeth remain in the end of this test.

4.3 Classification of Regions

Then, under the assumption that the reference points of gear teeth of a single gear should be positioned on a circle, the barycenters of the congruent regions in Figure 4(e) are classified into groups each of which corresponds to a gear based on the RANSAC-based circle fittings as shown in Figure 4(f). If a set of barycenters lie on one circle within a certain tolerance, they are classified into the group. This process classifies the regions into groups $G_i = \{R_i^j\}$ as shown in Figure 4(g) and it also eliminates the remained false regions which do not correspond to gear teeth from any G_i.

5 Periodicity Recognition

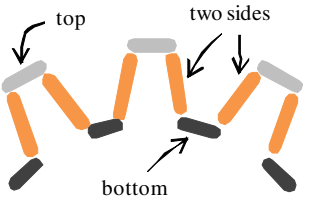
5.1 Planar Parameterization

Next, for each group G_i, a least-squares plane P_i is fitted for a set of barycenters $\{B_i^j\}$ of $\{R_i^j\}$. Each B_i^j is then projected onto the plane and their 2D parameters

(u_i^j, v_i^j) are calculated. The axes u and v on the plane are arbitrarily defined so that they can be orthogonal each another.

5.2 Sub Grouping

As shown in the inset figure, the gear shapes may be defined with several faces, e.g., top, bottom, and two side faces, in CAD system. We observed that the surface area of the top and the bottom faces are smaller than those of two side faces and that the mesh vertices corresponding to their faces are classified as high curvature vertices in step 1. Therefore the regions that correspond to top and bottom faces are not extracted by our segmentation and the only regions from two sides regions are extracted. Our current method of the periodicity recognition can deal with the case where one single region or two sides regions are extracted from a single gear tooth. An examples of single regions is shown in Figure 14 and those of two side regions are in Figure 4, 15, 17 and 18. When the two sides regions are extracted, our method separates them into sub groups, i.e., right side and left side sub groups, by the following procedures. The entire procedure is illustrated in Figure 5.



1. For each region R_i^j, compute the normal vector $\mathbf{N}_i^j$ as the averaged vector of the vertex normal vectors in R_i^j.
2. Arbitrarily select a region R_i^x among G_i and apply the following process for other regions $\{R_i^j\}$.
   - 2.1. Rotate the normal vector $\mathbf{N}_i^j$ by the angle between $\mathbf{B}_i^x$ and $\mathbf{B}_i^j$, and compute the angle $\theta_i^{x,j}$ between $\mathbf{N}_i^x$ and the rotated normal vector $\hat{\mathbf{N}}_i^j$.
   - 2.2. If $\theta_i^{j,x}$ is smaller than the threshold th_{sub}, classify R_i^j into the same sub group with R_i^x. We set $th_{sub} = 45.0\text{deg}$ for all the meshes in this paper.
3. Repeat the process 2 until there is no region which have not been classified into any sub groups.

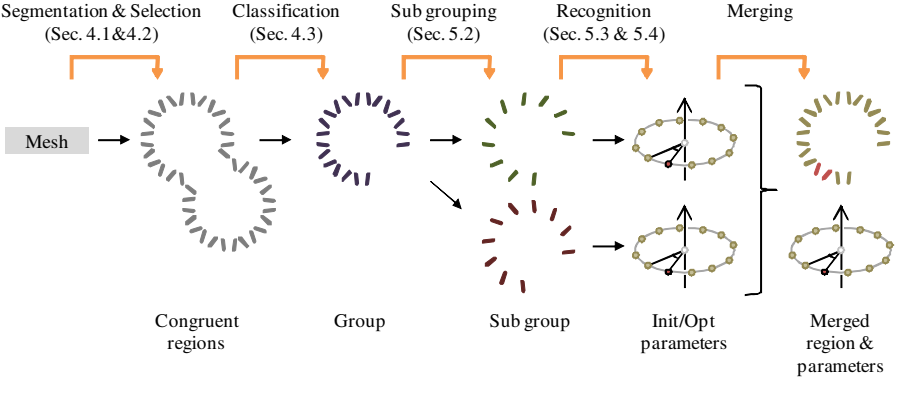
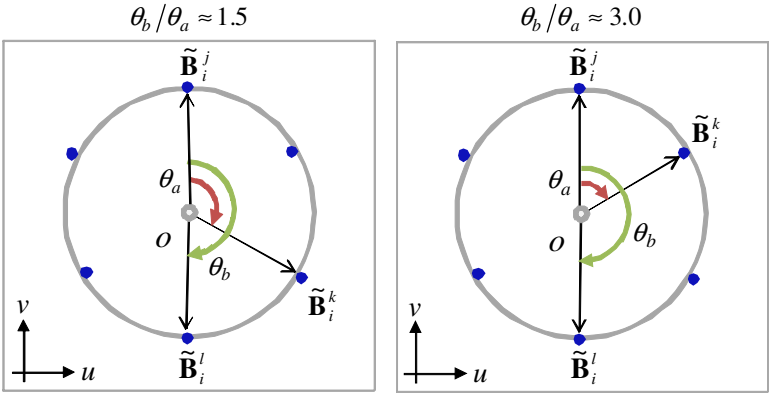


Fig. 5. Classification of regions



(a) Incorrect angle (b) Correct angle

Fig. 6. Initial estimation

In the following section, we describe our periodicity recognition method for each sub group. After periodicities are recognized by the methods describe in section 5.3 and 5.4, our method merges the sub groups into the group and obtains the merged regions and the common rotational axis and the basis angle of the group. The method is very simple and easy to implement, we skip to describe the details of it. From the next section, we refer the sub group as group and denote it as $G_i = \{R_i^j\}$ for simplicity.

5.3 Initial Estimation and Assignment of Indices

Then, for each group G_i, our method estimates the initial parameters defining the periodicity, including an axis directional vector **d**, a point on an axis **p**, and a basis angle θ. In rotational periodicities, a set of projected barycenters $\tilde{B}_i^j$ approximately forms a rotational grid on the plane as shown in Figure 6. The grid is spanned by a rotational basis angle around a center of rotation. A single region R_i^j can be simultaneously matched to others under a set of periodic rotations, and therefore a projected barycenter $\tilde{B}_i^j$ can also be approximately matched to others under the same transformations. Under this assumption, our method first estimates the initial parameters using our basis estimation method [20] which is the modification of Lin's [6]. The estimated parameters are then used to assign an index to each R_i^j.

Since we deal with CT scanned meshes, the projected barycenters have several perturbations which are caused by spatial distribution of vertex position and by scanning noise and they form an approximate rotational grid. The advantage of our method is the robustness for such distortions of the grid. Our method is a kind of voting scheme and it extracts the best parameters among several candidates.

Estimation of the initial axis:
The estimation of the axis is easy and intuitive. It is simply computed by performing the following procedures.

1. Calculate the initial directional vector $\mathbf{d}_i^{init}$ of the rotational axis as the normal vector of the projection plane P_i.
2. Fit a circle to $\{\tilde{B}_i^j\}$ and compute the initial point on the axis $\mathbf{p}_i^{init}$ as the center of the circle.

Estimation of the initial basis angle:
In the beginning of this step, we regard the projected barycenters as a set of vectors $\{\tilde{\mathbf{B}}_i^j = (u_i^j, v_i^j) \mid 1 \le j \le N_i\}$ originated by $\mathbf{p}_i^{init}$. We define the basis angle as the one whose absolute value is small and which other angles between pairs of vectors can be represented by its integral multiples as possible. Examples are shown in Figure 6. As for the example in Figure 6(a), if we select θ_a for the basis angle, θ_b cannot be represented by its integral multiple and therefore θ_a is incorrect as a basis angle. In contrast, as for the example in Figure 6(b), θ_b can be represented by the integral multiple of θ_a, and therefore θ_a is suitable for the basis angle. To find such a basis angle among all the angles between pairs of vectors, our method performs the following procedures.

1. Create a 2D accumulate array $S(j,k)$ and initialize entries of the array $S(j,k) = 0$ for $1 \le j,k \le N_i$.
2. For each pair of vectors $\tilde{\mathbf{B}}_i^j$ and $\tilde{\mathbf{B}}_i^k$, where $1 \le j < k \le N_i$, perform the following procedures:

   2-1. For each vector $\tilde{\mathbf{B}}_i^l$, where $2 \le l \le N_i$, compute the value a using $a = \theta_b / \theta_a$, where $\theta_a = angle(\tilde{\mathbf{B}}_i^j, \tilde{\mathbf{B}}_i^k)$ and $\theta_b = angle(\tilde{\mathbf{B}}_i^j, \tilde{\mathbf{B}}_i^l)$. Let a' be the round integer of a.

   2-2. Update the value of $S(j,k)$ using the scoring rule in Eq.(1).

$$S(j,k) \leftarrow S(j,k) + \frac{1 - 2|a - a'|}{\left\{angle\ (\tilde{\mathbf{B}}_i^j, \tilde{\mathbf{B}}_i^k)\right\}^{\gamma}} \tag{1}$$

3. Let the entry with the highest score in the array S locate at $(\hat{j}, \hat{k})$ and then set the initial basis angle θ_i^{init} as $angle(\tilde{\mathbf{B}}_i^{\hat{j}}, \tilde{\mathbf{B}}_i^{\hat{k}})$.

In our method, for each pair of vectors $\tilde{\mathbf{B}}_i^j$ and $\tilde{\mathbf{B}}_i^k$, if the vectors were located near the vertices of an accurate rotational grid and if their angle is small, their corresponding score becomes high. We set $\gamma = 0.5$ for all meshes in this paper and we found it provides satisfactory results from various experiments.

Assignment of Indices to the PDRs:
For each $\tilde{\mathbf{B}}_i^j$, compute the value c using $c = \theta_c / \theta^{init}$, where $\theta_c = angle(\tilde{\mathbf{B}}_i^j, \tilde{\mathbf{B}}_i^1)$. Let c' be the round integer of c. Then the index of $\tilde{\mathbf{B}}_i^j$ is computed as c'.

5.4 Optimal Estimation

In this section, we describe our simultaneous matching algorithm [20] which extracts optimal parameters $\langle \mathbf{d}^{opt}, \mathbf{p}^{opt}, \theta^{opt} \rangle$ defining the rotational periodicities. The initial parameters calculated in section 5.2 includes large estimation error of the rotational axis and the basis angle due to the perturbations of the projected barycenters, therefore, this step minimizes the error by an optimization and extracts more accurate parameters. Our simultaneous matching algorithm is the extension of ICP [16]. The ICP iterates computing a transformation and finding corresponding points for a single pair of regions, and it finally estimates an optimal transformation under which a pair of regions can be matched each other so that the sum of distances between corresponding points can be minimized. In contrast to the ICP, our simultaneous matching algorithm iterates computing a rotational axis and a basis angle and finding corresponding points for all the pairs of regions, and finally it estimates an optimal axis and an angle under which any single region can be matched to others by a periodic rotations using their indices so that their matching error can be minimized. More simply, our algorithm can be regarded as the bidirectional distance minimization between corresponding points in all pairs of regions.

Here we denote a region R_i^j in a group G_i as a set of points $R_i^j = \{\mathbf{x}_{i,k}^j \mid 1 \le j \le N_i, 1 \le k \le n_i^j\}$. The point $\mathbf{x}_{i,k}^j$ can be closely matched to its corresponding point $\mathbf{x}_{i,c(k)}^l$ in R_i^l by $\mathbf{T}^{itr}(\mathbf{x}_{i,k}^j)$, where $\mathbf{T}^{itr}(\mathbf{x}_{i,k}^j)$ is the rotation around the axis $\langle \mathbf{d}, \mathbf{p} \rangle$ with the angle $(id(R_i^j) - id(R_i^l))\theta$. Here $id(R_i^j)$ and $id(R_i^l)$ are the indices of R_i^j and R_i^l respectively. And $c(k)$ is the vertex ID of $\mathbf{x}_{i,c(k)}^l$ corresponding to $\mathbf{x}_{i,k}^j$. To estimate optimal parameters, our algorithm performs the following procedures.

1. **Initialize:** Set $\mathbf{d}_i^0 = \mathbf{d}_i^{init}$, $\mathbf{p}_i^0 = \mathbf{p}_i^{init}$, $\theta_i^0 = \theta_i^{init}$ and $itr = 0$.
2. **Find closest points:** For each R_i^j, where $1 \le j \le N_i$, perform the following process:

   For each point $\mathbf{T}^{itr}(\mathbf{x}_{i,k}^j)$ in the current position of R_i^j, find the set of closest point $\{\mathbf{x}_{i,c(k)}^l\}$ in each R_i^l. And for each point $\mathbf{x}_{i,m}^l$ in the initial position of R_i^l, find the set of closest point $\{\mathbf{T}^{itr}(\mathbf{x}_{i,c(m)}^j)\}$ in the current position of each R_i^j.
3. **Compute parameters:** Compute the parameters $\langle \mathbf{d}_i^{itr}, \mathbf{p}_i^{itr}, \theta_i^{itr} \rangle$ by minimizing the sum of squared distances between corresponding points in Eq.(2):

$$D^{itr} = \sum_{j=1}^{N_i} \sum_{l=j+1}^{N_i} D^{itr}(R_i^j \to R_i^l) + \sum_{j=1}^{N_i} \sum_{l=j+1}^{N_i} D^{itr}(R_i^l \to R_i^j) \tag{2}$$

   Here $D^{itr}(R_i^j \to R_i^l)$ and $D^{itr}(R_i^l \to R_i^j)$ are described in Eq.(3) and Eq.(4) respectively.

$$D^{itr}(R_i^j \to R_i^l) = \sum_{k}^{n_i^j} \left\| \mathbf{T}^{itr}(\mathbf{x}_{i,k}^j) - \mathbf{x}_{i,c(k)}^l \right\|^2 \tag{3}$$

$$D^{itr}(R_i^l \rightarrow R_i^j) = \sum_m^{n_i^l} \left\| \mathbf{x}_{i,m}^l - \mathbf{T}^{itr}(\mathbf{x}_{i,c^{-1}(m)}^j) \right\|^2 \tag{4}$$

This non-linear equation can be solved for $\mathbf{d}_i^{itr}$, $\mathbf{p}_i^{itr}$, and θ_i^{itr} using *Levenberg-Marquardt algorithm* [19].

4. **Update points and calculate distance:** Update the points such that $\mathbf{T}^{itr}(\mathbf{x}_{i,k}^j) \leftarrow \mathbf{x}_{i,k}^j$ using the current parameters $\mathbf{d}_i^{itr}$, $\mathbf{c}_i^{itr}$, and θ_i^{itr}. Then calculate the averaged distance from Eq.(2) as $E^{itr} = \sqrt{D^{itr}} \big/ N_i^{all}$, where N_i^{all} is the total number of pairs of corresponding points in group G_i.
5. **Termination condition:** If $E_i^{itr} - E_i^{itr+1} > \varepsilon$, update $itr \leftarrow itr + 1$ and repeat the process from Step 2. Otherwise, output the optimal parameters as $\mathbf{d}_i^{opt} = \mathbf{d}_i^{itr}$, $\mathbf{p}_i^{opt} = \mathbf{p}_i^{itr}$ and $\theta_i^{opt} = \theta_i^{itr}$, and then stop the process. (We set ε such that $\varepsilon = w_{opt} l_{avg}$, typically with $w_{opt} = 0.001$.)

6 Generation of Partial Meshes

For the purpose of inspecting the assembling errors of gear trains, the methods mentioned in the previous section are sufficient where the rotational axes are estimated. However, for performing kinematic simulations based on their CT scanned meshes, the meshes obtained from the periodicity recognition are still insufficient because several number of gear teeth portions are missing in the mesh at the gear-to-gear contact are. A set of complete meshes each of which expresses closed surface of an individual gear must be used in the simulation of gear teeth contacts. To generate such a complete mesh of each gear, our method first enlarges the gear teeth regions and extracts as large regions as possible where periodicities are valid by our region growing. Then it interpolates the points by our geometric synthesis for the false regions where the appropriate regions are not detected by our segmentation and classification (mentioned in section 4) and where surface meshes are not created in the area that the neighboring parts are contact in the CT scanning. These processes can extract the enlarged and the synthesized all-round point sets for each gear and the complete meshes of each part can be easily created by triangulating them.

6.1 Region Growing

For each group $G_i = \{R_i^j\}$, the regions extracted in step 1 are simultaneously enlarged by our region growing algorithm according to the estimated periodicity. Since we deal with scanned meshes, mesh vertices of each congruent regions do not lie in the equivalent position. Therefore our method uses a tolerance that allows the vertex perturbations. Our region growing enables the extraction of as large regions as possible where periodicities are valid under the user specified tolerance. As shown in Figure 7, our method performs the following procedures.

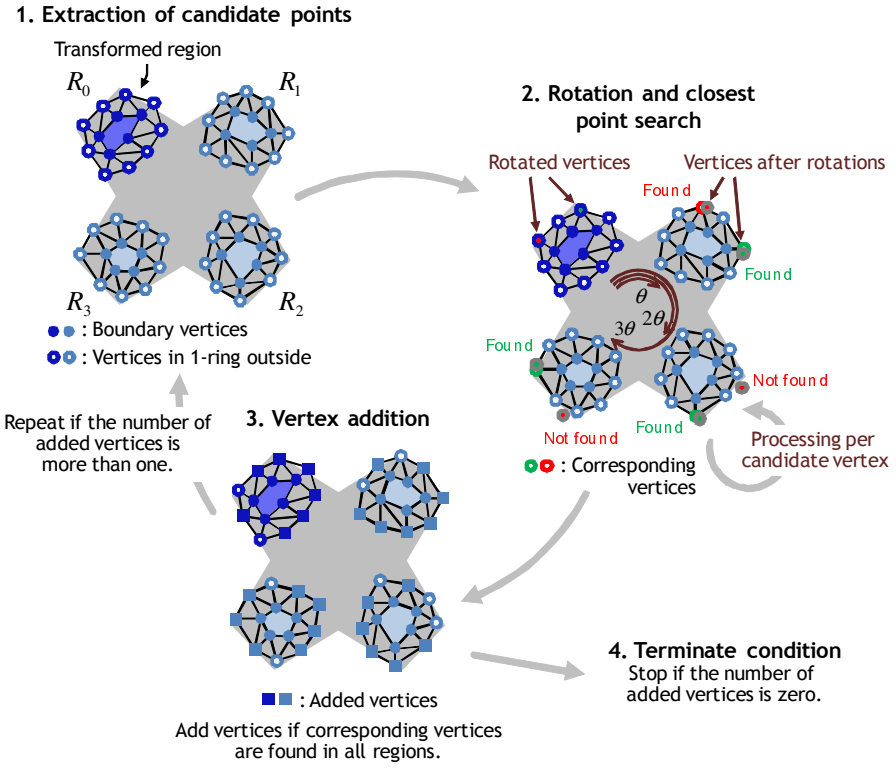


Fig. 7. An overview of our region growing

1. **Extraction of candidate points:** Extract a set of candidate points $C_i^j = \{\mathbf{x}_{i,k}^j\}$ for addition to the regions. Here C_i^j includes vertices which lie on the boundaries of the current region and vertices which lie outside of it among their 1-ring vertices.
2. **Rotation and closest point search:** For each vertex $\mathbf{x}_{i,k}^1$ in C_i^1, transform it to $\{\mathbf{x}_{i,k}^{1\to j}\}$ by rotating it by $(id(R_i^j) - id(R_i^1))\theta_i^{opt}$ so that C_i^1 can be matched to each C_i^j. Then, for each point $\mathbf{x}_{i,k}^{1\to j}$, search a set of closest points $\{\mathbf{x}_{i,c(k)}^j\}$ in each C_i^j and compute the distances $\{d_{i,k}^{1,j}\}$ between points.
3. **Vertex addition:** For each point $\mathbf{x}_{i,k}^{1\to j}$, if all the distances $\{d_{i,k}^{1,j}\}$ are less than threshold th_{add}, add $\mathbf{x}_{i,k}^1$ to R_i^1 and $\{\mathbf{x}_{i,c(k)}^j\}$ to $\{R_i^j\}$ respectively. We set $th_{add} = 1.0 l_{avg}$ for all the examples in this paper.
4. **Terminate condition:** If the number of points added to R_i^1 is more than one, continue the process from process 1. Otherwise, stop the process.

As a result of the process described above, regions can be enlarged as shown in Figure 10(a). In this figure, regions are colored according to their indices, i.e. light for odd and deep for even indices.

6.2 Geometric Synthesis

When pairs of teeth in gears are contact or close each other in the CT scanning process, these areas on meshes are connected and their boundaries cannot be measured as shown in Figure 8 due to the blur of CT values. Our segmentation and congruency tests in step 1 cannot detect regions corresponding to gear teeth separately from such areas. Moreover, even when the meshes are appropriately created, our segmentation may fail to extract regions due to the inadequate threshold setting especially when the number of vertices in the regions is small. In addition, even when our segmentation successes, our classification method may fail to appropriately classify regions into a group due to the perturbations of the regions' barycenters. In these cases, it results that there exists several false regions after region growing mentioned in section 6.1 as shown in Figure 10(a), 14(c,f), and 15(c). Therefore our geometric synthesis interpolates points in such areas and plausibly estimates the original geometry.

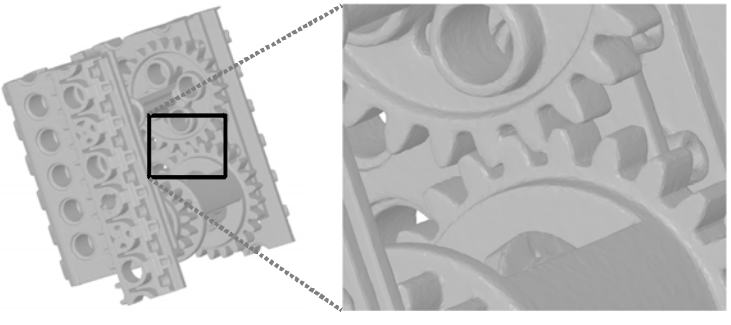


Fig. 8. Connected area on meshes due to the blur of CT values

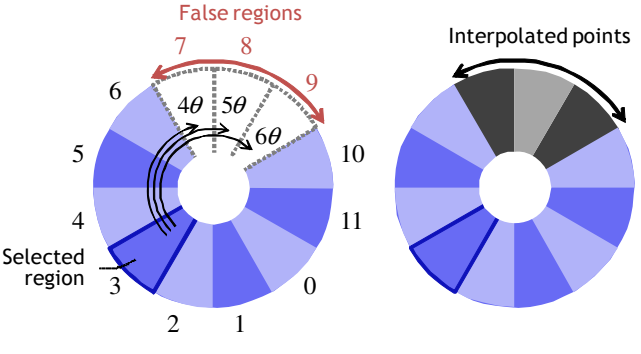


Fig. 9. Geometric synthesis

(a) Grown regions (b) Interpolated points (c) Partial meshes

Fig. 10. Generation of partial meshes

In our method, an arbitrarily selected region R_i^j is transformed to the false region $R_i^j = \{\}$ by rotating it by $(id(R_i^j) - id(R_i^l))\theta^{opt}$ and points on the boundaries are interpolated as shown in Figure 9 and 10(b). This enables to generate all-round points for each gear, and then partial meshes $\{M_i\}$ can be easily generated by triangulating them as shown in Figure 10(c). We are currently using commercial software Geomagic [18] but existing meshing algorithms such as marching cubes [19] can be used.

We note that our method cannot estimate the completely correct boundaries of the gears and can find only the boundaries where periodicities are valid. However such boundaries are sufficient for the kinematic simulation purposes.

7 Evaluation of Gear Teeth Contact

In this section, a method is described for evaluating gear teeth contacts using the extracted periodicities and the partial meshes. Such an evaluation is one of the most important functions in kinematic simulations of gear assemblies for predicting the power transmission efficiency, noise and vibration.. In our method, evaluations of gear teeth contacts are replaced by the distance computations between surfaces approximating the teeth regions. More precisely, our method rotates the pair of partial meshes biting each other around each axis by a specified pitch angle and computes the distances between the approximating surface of one tooth and the vertices projected onto the surface approximating the other one at each rotation. The surface approximations can decrease the effect of scanning noise for the contact evaluations. We use bicubic polynomial surfaces [15] for the approximations so that they can be closely fitted for a larger class of teeth faces. Our method performs the following procedures.

1. **Extraction of pairs of partial gear meshes:** Extract the pairs of partial gear meshes $\langle M_i, M_j \rangle$ biting each other by simply thresholding the shortest distance between pairs of approximating surfaces of the teeth.
2. **Computation of pitch angle ratio:** For each pair $\langle M_i, M_j \rangle$, compute a pitch angle $\langle \phi_i, \phi_j \rangle$ as $\phi_i = 360/L_i$ and $\phi_j = 360/L_j$ respectively, where L_i and L_j are the numbers of teeth in each gear. Then compute their ratio $r_{ij} = \phi_j / \phi_i$.

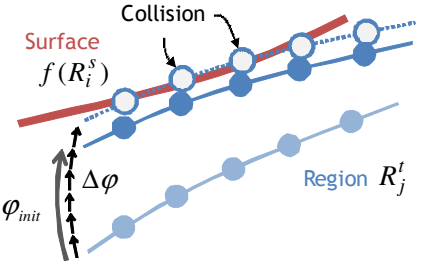


Fig. 11. Initial matching

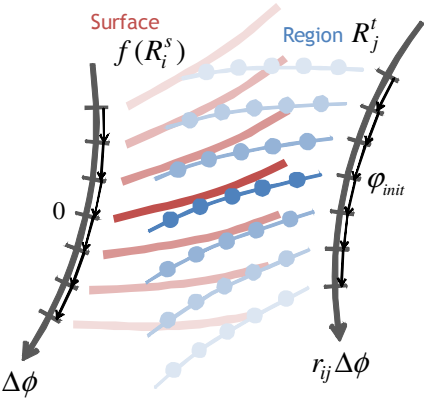


Fig. 12. Evaluation of gear teeth contacts

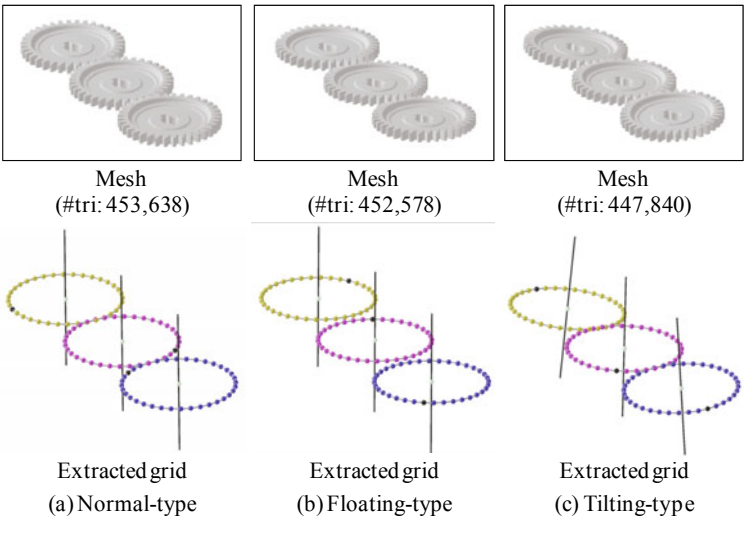


Fig. 13. Results for artificial meshes

Table 1. Accuracy evaluation of parameters extraction

	Normal-type			Floating-type			Tilting-type		
Noise level (a%)	5.0	10.0	15.0	5.0	10.0	15.0	5.0	10.0	15.0
Max. Direction error (deg)	0.19	0.05	0.45	0.15	0.02	0.21	0.05	0.02	0.29
Max. Basis angle error (deg)	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01

3. **Initial matching:** For each pair $\langle M_i, M_j \rangle$, find the pair of regions $\langle R_i^s, R_j^t \rangle$ corresponding to gear teeth which are almost contact. Then the initial matching can be performed so that the pair can be completely contact. First a bicubic polynomial surface $f(R_i^s)$ is fitted to R_i^s in a least-squares manner. Next, as shown in Figure 11, R_j^t is sequentially rotated by a specified pitch angle $\Delta\varphi$, the collision is tested between $f(R_i^s)$ and R_j^t, and the adjacent angle φ_{init} is detected before their collision. Then the initial matching can be performed by rotating M_j by the computed angle θ_{init} such that $M_j \to M_j'$. We set $\Delta\varphi = 0.01\text{deg}$ for the example in Figure 16. For more accurately performing the initial matching, we can up-sample the surface $f(R_i^t)$ and make $\Delta\varphi$ smaller.
4. **Evaluation of teeth contacts:** Rotate M_i and M_j' at the pitch angle $\Delta\phi$ and $r_{ij}\Delta\phi$ respectively, and evaluate the contacts as the distance between the surface $f(R_i^s)$ and the region $R_j'^t$ which are in contact at each rotation. We set $\Delta\phi = 0.1\text{deg}$ for the example in Figure 16.

8 Results

We applied the proposed method for various meshes and verified its effectiveness. All the experiments were run on the PC with Core2Duo 2.4GHz CPU and 2GB RAM.

Figure 13 shows the results of periodicity recognitions for the artificial meshes of gear trains. These three types of meshes are generated in the similar ways with the mesh in Figure 3. The first mesh is normal-type which are simply generated by triangulating CAD model of the gear assembly using CAE meshing software. The second mesh is floating-type which are generated first by moving two CAD models along the rotational axis by 3mm with reference to the one, which imitates positional offset error along the axis between gears, and then by triangulating them in the same way. The third mesh is tilting-type which are generated first by rotating two CAD models by five degrees with reference to the one, which imitates the axis misalignments between gears, and then by triangulating them in the same way. These three meshes contain about 450,000 triangles. We added the artificial noise to the meshes by displacing each vertex along its normal direction by Gaussian distributed random distances with the standard deviation α% proportional to the averaged mesh edge length. We set α=5.0, 10.0, 15.0, and evaluated the accuracy of parameters extraction at each setting. Our method extracted the periodicities described by the grids in Figure 13 from all meshes at any noise level settings. We evaluated the axis directional vector and the basis angle extracted by our method with the theoretical values. The results are shown in Table 1. The maximum estimation error of directional vectors among the three gears in a mesh was 0.45deg in the case of normal-type mesh with α=15.0. The errors of basis angles were less than 0.01deg in all cases. The averaged number of vertices in the interior of the regions corresponding to gear teeth was about 90 and the total running time for the periodicity recognitions were about 70 seconds.

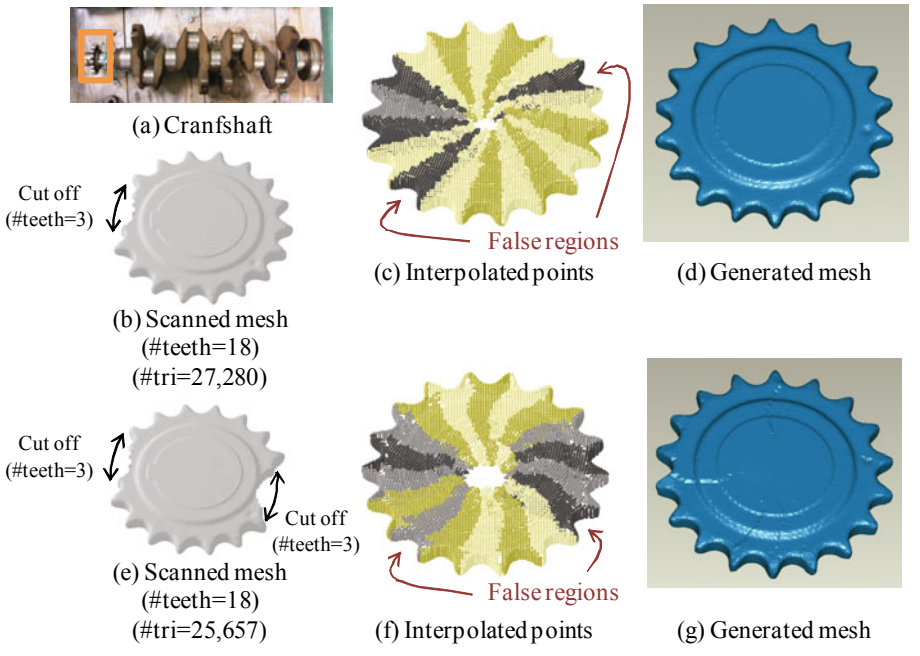


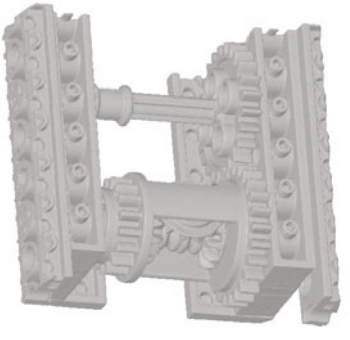
Fig. 14. Results for CT scanned meshes of the gear in crankshaft

Figure 14 shows the results of the generations of partial meshes from the CT scanned mesh of the single gear in the mechanical part in Figure 14(a). We interactively extracted the portions in the mesh corresponding to the gear shown in orange box. Assuming that the pairs of gear teeth are contact during the CT scanning process and that surface meshes cannot be properly created, we manually cut off the one and the two portions as shown in Figure 14(b) and (e). Estimation errors of the basis angles were less than 0.01deg with reference to the theoretical angle in both meshes. This result shows that our method can accurately recognize periodicities even when segmentation cannot be appropriately performed. Moreover it could generate all-round partial meshes from such incomplete meshes as shown in Figure 14(d) and (g). We set the threshold th_{high} for classifying high curvature vertices (mentioned in section 4.1) as $th_{high} = 3.0$ and the running time was less than 2 seconds for both meshes.

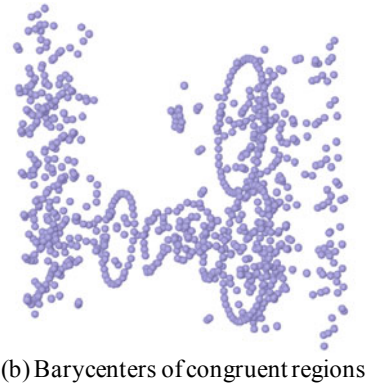
Figure 15 shows the results for the decomposition of the CT scanned mesh in Figure 15(a) of a LEGO gear assembly. The object includes six gears and our method generated three meshes among them. As for the detected three gears, our method could interpolate the points even when regions cannot be appropriately extracted in step 1 as shown in Figure 15(c) and generate partial meshes in Figure 15(d). The estimation errors of the basis angle were less than 0.01deg in all gears. The mesh includes 981,036 triangles. The running time was 261 seconds in step 1, 6 seconds in step 2, and 3 seconds in step 3 except for mesh generations from interpolated points. We note that most of the times in step 1 were spent for the congruency test based on pairwise ICP matching as in [13]. As shown in Figure 15(b), since many regions

which did not correspond to gear teeth still remained after congruency tests based on ICP matching, the following RANSAC algorithm could not classify the regions that correspond to gear teeth. We set $th_{high} = 10.0$ for this mesh.

Figure 16 show the results for the evaluation of gear teeth contacts for the pair of meshes M_1 and M_2 in Figure 15(d). In this figure, the contacts of the pair of teeth at several rotation starting from an angle φ_x are shown. Red color corresponds to larger distance and yellow to zero. In the case of spur gears as shown in this figure, if the rotational axes of the pair of gears are parallel, the teeth contact area should be a straight line parallel to the axes and the line move from the root to the head of the teeth. As shown in Figure 16, the behaviors of the real gear assemblies were observed on their CT scanned meshes by our method. We also observed the similar behaviors from other pairs. The total running time was about 10 seconds for this pair.

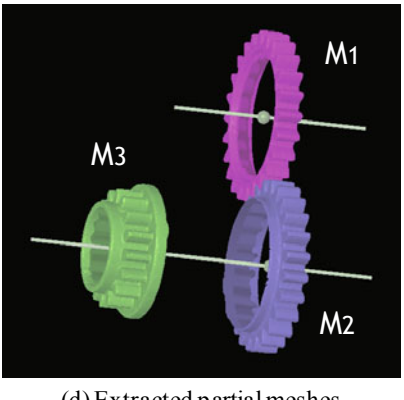


(a) Scanned mesh of a gear assembly of LEGO (#tri; 981,036)



(b) Barycenters of congruent regions

(c) Interpolated points



(d) Extracted partial meshes

Fig. 15. Results for a generation of partial meshes from a CT scanned mesh

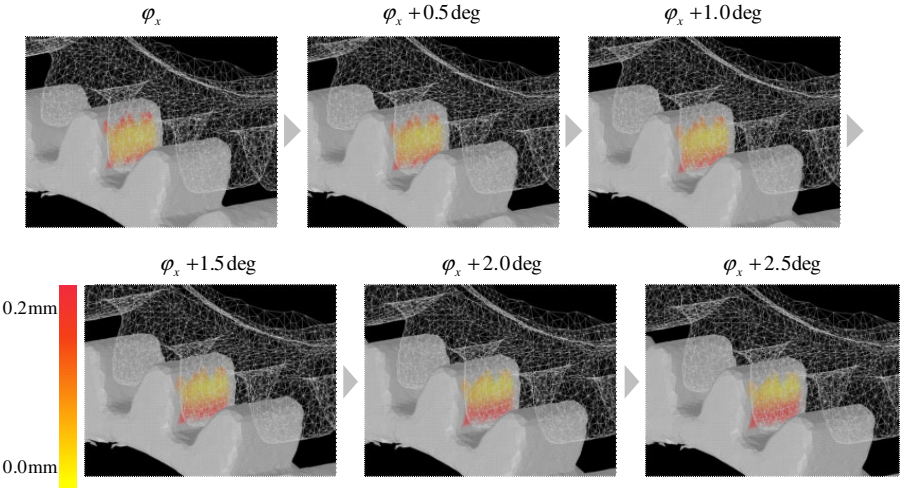


Fig. 16. Result for evaluating gear teeth contact

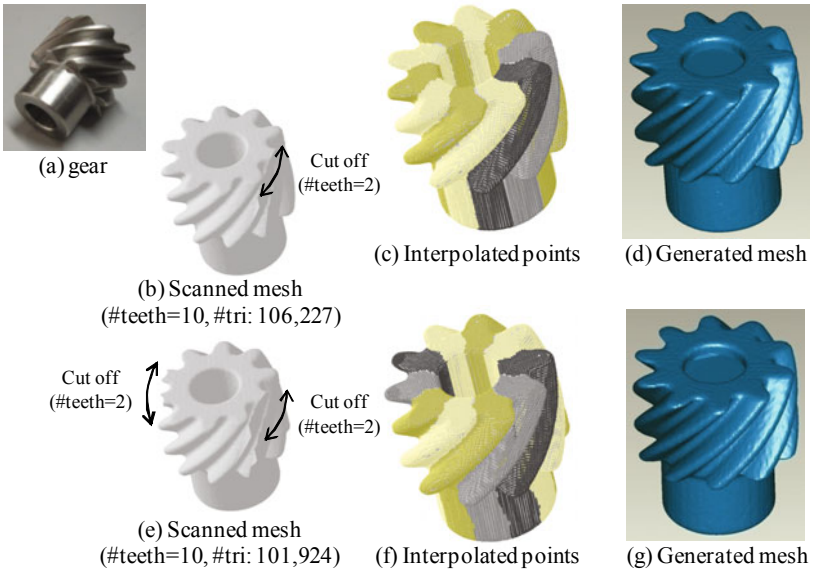


Fig. 17. Result for the all-round mesh generation from a CT scanned mesh of an gear

Figures 17 and 18 show the results for the single bevel gears. In both figures, the partial meshes are manually cut off in the same way as the meshes in Figure 14. These results show that our proposed processes can be applied for the various types of CT scanned meshes of real gears and that it can well generate the all-round surface meshes. Therefore, it is highly probable that our whole algorithm still works well for real CT data of real gear trains. We set $th_{high} = 10.0$ for all meshes in Figure 17 and 18.

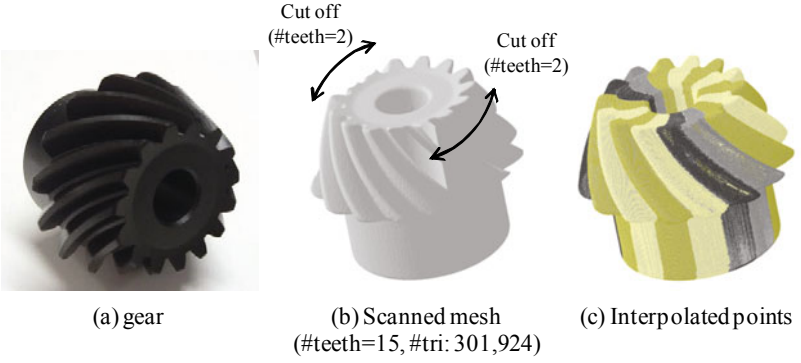


Fig. 18. Result for the all-round points generation from a CT scanned mesh of an engineering object

Discussion. Our segmentation algorithm mentioned in section 4 may fail to extract regions corresponding to gear teeth due to the scanning noise and to the inappropriate threshold setting for the classification of high curvature vertices. Moreover it cannot detect regions of gear teeth which are contact in the CT scanning process. In addition, our classification algorithm based on RANSAC based circle-fittings may fail to classify the regions into groups each of which corresponds to a single gear due to the perturbation of the barycenters of the regions. Even in such cases, our periodicity recognition and partial mesh generation algorithms can recover such false regions and reconstruct the complete meshes. Examples are shown in Figure 14 and 15. In these figures, although several regions are not correctly segmented or classified, our method could generate the complete meshes which can be usefully used for kinematic simulations. In our experiments, our algorithm can generate the complete meshes when about half of regions are correctly segmented and classified. We note that the extraction accuracy of parameters, such as rotational axes and basis angles, decreases when fewer regions are segmented and classified.

9 Conclusion and Future Works

In this paper, we proposed a new method for decomposing CT scanned surface meshes of parts assemblies based on periodicity recognition. We demonstrated the effectiveness of our proposed method from various experiments on the artificial and the CT scanned meshes. By taking the gear trains as examples, we verified that our method enabled to uniquely determine the correct boundaries between parts and to accurately decompose single material CT scanned surface meshes into partial meshes each of which corresponds to a gear without reference data such as original CAD data and CT values. Moreover we proposed a method for evaluating gear teeth contacts using the generated partial meshes and the recognized periodicities as an example of kinematic simulations. We found that our method could estimate the behaviors in the motions of real gear assemblies based on their CT scanned meshes.

Our proposed method can be used not only for gear trains as presented in this paper but also for a wide class of assemblies where rotational periodicities exist in the contact area between parts, e.g., bearings, chain sprocket. We will test the versatility of our method for such assemblies in future works.

Limitations. In some gears, the surface area of a tooth is very small and therefore the number of vertices in such a region in a scanned mesh is also small. Our segmentation cannot appropriately extract the regions corresponding to such teeth due to the difficulties of the threshold setting and thus cannot generate partial meshes of such gears.

As for the practical evaluations of gear teeth contacts, more precise evaluations are required, e.g., a few micron order, and the current resolution of CT scanning is not enough for the practical use of our method. However it can be estimated that the resolution of CT scanning will be higher in near future and we believe that our method can be practically used in such days.

Acknowledgements

We would like to thank anonymous reviewers for their helpful comments. The CT scanned meshes of the engineering parts in Figure 14, 17, and 18 were provided by Ichiro Nishigaki and Noriyuki Sadaoka in HITACHI Co., Ltd. The CT scanned mesh of the LEGO block in Figure 15 and 16 was provided by Hiroyuki Tanaka and Hideaki Aiyama in Hokkaido Industrial Research Institute. This work was financially supported by the Grant-in-Aid for Scientific Research under the project No.18004488-00.

References

[1] Suzuki, H.: Convergence engineering based on X-ray CT scanning technologies. In: Proc. JSME Digital Engineering Workshop, pp. 74–77 (2005)
[2] Katz, S., Tal, A.: Hierarchical mesh decomposition using fuzzy clustering and cuts. ACM Transactions on Graphics 22(3), 954–961 (2003)
[3] Katz, S., Leifman, G., Tal, A.: Mesh Segmentation using Feature Point and Core Extraction. The Visual Computer 21(8-10), 649–658 (2005)
[4] Shammaa, H.M., Suzuki, H., Michikawa, T.: Registration of CAD mesh models with CT volumetric model of assembly of machine parts. The Visual Computer 23(12), 965–974 (2007)
[5] Shammaa, H.M., Suzuki, H., Ohtake, Y.: Extraction of isosurfaces from multi-material CT volumetric data of mechanical parts. In: Proc. ACM symposium on Solid and Physical Modeling, pp. 213–220 (2008)
[6] Lin, H.C., Wang, L.L., Yang, S.N.: Extracting periodicity of a regular texture based on autocorrelation functions. Pattern Recognition Letters 18, 433–443 (1997)
[7] Liu, Y., Collins, T.T., Tsin, Y.: A computational model for pattern perception based on frieze and wallpaper groups. IEEE Transactions on Pattern Analysis and Machine Intelligence 26(3), 354–371 (2006)
[8] Müller, P., Zeng, G., Wonka, P., Gool, L.V.: Image-based procedural modeling of facades. ACM Transactions on Graphics 26(3), 85 (2007)

[9] Mitra, N.J., Guibas, L.J., Pauly, M.: Partial and approximate symmetry detection for 3D geometry. ACM Transactions on Graphics 25(3), 560–568 (2006)
[10] Podolak, J., Shilane, P., Golovinskiy, A., Rusinkiewicz, S., Funkhouser, T.: A planar-reflective symmetry transform for 3D shapes. ACM Transactions on Graphics 25(3), 549–559 (2006)
[11] Xu, K., Zhang, H., Tagliasacchi, A., Liu, L., Li, G., Meng, M., Xiong, Y.: Partial Intrinsic Reflectional Symmetry of 3D Shapes. In: ACM Transactions on Graphics (SIGGRAPH Asia 2009) (2009) (to appear)
[12] Liu, S., Martin, R.R., Langbein, F.C., Rosin, P.L.: Segmenting Periodic Reliefs on Triangle Meshes. In: Martin, R., Sabin, M.A., Winkler, J.R. (eds.) Mathematics of Surfaces 2007. LNCS, vol. 4647, pp. 290–306. Springer, Heidelberg (2007)
[13] Pauly, M., Mitra, N.J., Wallner, J., Pottmann, H., Guibas, L.: Discovering structural regularity in 3D geometry. ACM Transaction on Graphics 27(3), 43 (2008)
[14] Mizoguchi, T., Date, H., Kanai, S., Kishinami, T.: Quasi-optimal mesh segmentation via region growing/merging. In: Proc. ASME/DETC-35171 (2007)
[15] Vieira, M., Shimada, K.: Surface mesh segmentation and smooth surface extraction through region growing. Computer-Aided Geometric Design 22(8), 771–792 (2005)
[16] Besl, P., McKay, N.: A method for registration of 3-D shapes. IEEE Transactions on Pattern Analysis and Machine Intelligence 14(2), 239–256 (1992)
[17] Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes in C++: The Art of Scientific Computing, 2nd edn. Cambridge University Press, Cambridge (1992)
[18] Geomagic, `http://www.geomagic.com`
[19] Lorensen, W.E., Harvey, E.C.: Marching cubes: A high resolution 3D surface construction algorithm. ACM SIGGRAPH Computer Graphics 21(4), 163–169 (1987)
[20] Kondo, D., Mizoguchi, T., Kanai, S.: Recognizing Periodicities on 3D Scanned Meshes Based on Indexed-ICP Algorithm. In: Integrated Design and Manufacturing in Mechanical Engineering, Springer, Heidelberg (to appear, 2010)
[21] Li, M., Langbein, F.C., Martin, R.R.: Detecting approximate symmetries of discrete point subsets. Computer-Aided Design 40(1), 76–93 (2008)
[22] Li, M., Langbein, F.C., Martin, R.R.: Detecting design intent in approximate CAD models using symmetry. Computer-Aided Design 42(3), 183–201 (2010)

Automatic Generation of Riemann Surface Meshes

Matthias Nieser, Konstantin Poelke, and Konrad Polthier*

Freie Universität Berlin, Germany
{matthias.nieser,konstantin.poelke,konrad.polthier}@fu-berlin.de

Abstract. Riemann surfaces naturally appear in the analysis of complex functions that are branched over the complex plane. However, they usually possess a complicated topology and are thus hard to understand. We present an algorithm for constructing Riemann surfaces as meshes in $\mathbb{R}^3$ from explicitly given branch points with corresponding branch indices. The constructed surfaces cover the complex plane by the canonical projection onto $\mathbb{R}^2$ and can therefore be considered as multivalued graphs over the plane – hence they provide a comprehensible visualization of the topological structure.

Complex functions are elegantly visualized using domain coloring on a subset of $\mathbb{C}$. By applying domain coloring to the automatically constructed Riemann surface models, we generalize this approach to deal with functions which cannot be entirely visualized in the complex plane.

1 Introduction

Riemann surfaces are a fundamental concept in modern complex analysis, topology and algebraic geometry. Bernhard Riemann himself introduced them 1851 in his dissertation "Grundlagen für eine allgemeine Theorie der Functionen einer veränderlichen complexen Größe", but it was Felix Klein and Hermann Weyl who caused his idea to become known among the mathematicians in the beginning 20th century. Since then, among other things, Riemann surfaces serve as generalized domains for complex functions because multi-valued complex functions can be turned into single-valued functions when defined on such a surface instead of the complex plane. However, these surfaces might possess a complicated topological structure since multi-valued functions give rise to ramifications determined by characteristic points - the so-called branch points.

Riemann surfaces on manifolds are used for several methods in computer graphics. For example, [1] uses a universal covering for the computation of shortest cycles in each homotopy class of a surface. The surface parameterization method [2] computes a 4-sheeted covering in order to represent the (multivalued) parameter function on higher genus surfaces. The notion of covering spaces provides a nice theoretical foundation of this parameterization approach.

* Supported by DFG Research Center Matheon "Mathematics for key technologies".

B. Mourrain, S. Schaefer, and G. Xu (Eds.): GMP 2010, LNCS 6130, pp. 161–178, 2010.

1.1 Related Work

Surprisingly there is very little about the computer-aided visualization of Riemann surfaces. The most important contribution might be the work by Trott [3,4] for Wolfram Research. He uses the symbolic derivation and nonlinear equations solver provided by Mathematica and computes 3D plots based on an explicit function definition. To the best of our knowledge it is the only work that automatically generates visualizations of Riemann surfaces – most available images of Riemann surfaces usually show explicitly parametrized surfaces.

Important achievements concerning the technique of *domain coloring* are done by Farris [5], who also introduced this term. He uses simple color gradients without further features. The colorings of Pergler [6] and Lima da Silva [7] are of better quality and Lundmark [8] gives a detailed introduction and uses a color scheme revealing several important aspects of complex functions. Hlavacek [9] provides a gallery of complex function plots, using color schemes similar to Lundmark's and Pergler's. Further advancement and additional indicators are provided in [10].

1.2 Contribution

In this paper we present an algorithm for computing 3D models for Riemann surfaces based on given branch points and branch indices. Usually these surfaces possess a non-trivial topology and are thus hard to capture. The models we construct are a visualization technique for an easier understanding of the topology of these surfaces. This process consists of two parts:

1. Cutting the surface open, either by a user given cut graph or by computing the shortest cut graph as in [11], and computing multiple surface layers.
2. Computing a height function for each of the layers. The resulting surface is then interpreted as a graph over the complex plane. Since this embedding is used for visualization, there is possibly more than one height function that could be used in order to produce reasonable results. We use harmonic height functions as a natural choice as they produce almost smooth surfaces.

Whereas Trott's method [3] provides 3D plots of Riemann surfaces, we generate triangulated meshes which can be used as 3D models for further processing. As our meshes will be patched together from single sheets, we can easily extract parts of the model in order to focus on the important features of the surface.

We also apply domain coloring to these Riemann meshes – a method traditionally used for visualizing complex functions by a color map on the complex plane. However, complex functions usually define a covering of the complex plane, which can be explicitly constructed as a triangle mesh with our method. Using this triangle mesh as new domains for domain coloring, we can visualize analytical extensions of functions that are discontinuous when defined on the complex plane which helps for an overall understanding of those functions.

The paper is organized as follows: Sect. 2 introduces Riemann surfaces and gives the theoretical background and general setting of our approach. Sect. 3

explains the main concept of our visualization approach. The algorithm itself is explained in Sect. 4, and in Sect. 5 results are presented and discussed.

2 Riemann Surfaces and Complex Analysis

2.1 Problem Statement

The problem of visualizing covering surfaces has arisen to us in the analysis of complex functions. Given a holomorphic map $\eta : U \subset \mathbb{C} \to V \subset \mathbb{C}$ between two simply connected domains in the complex plane, its geometrical structure can be visualized by domain coloring as demonstrated in [10]. Domain coloring uses a reference image which is defined over V and transfers it to U via η. The resulting image in U yields many information about η.

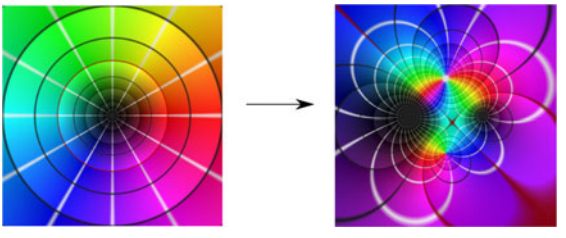


Fig. 1. Domain coloring (cf. section 5.2) is a technique that makes use of a color scheme to visualize complex valued functions. The left image shows a reference color scheme that represents the complex plane. The right image shows a typical color plot of a meromorphic function. The black spots denote zeros, here with multiplicity 2 (the bigger one on the left) and 1 (the one on the right). The white spot in the upper third is a double pole whereas the punctual spot at the bottom is a simple pole. This can also be seen from the multiplicity and order of the colors around these points – the double zeros and poles are surrounded by the complete color circle twice in contrast to the simple zeros and poles.

A problem arises if η is non-injective and one ones to color its inverse η^{-1} which is now a multivalued function. In this case, there are points in U which obtain more than one color and we would obtain sort of a *multivalued* image. We present a method for visualizing even those functions. The main idea is to use a covering surface X of V and a bijective map $f : U \to X$ that encodes the same geometrical information as η.

2.2 Theoretical Background

This section gives a short introduction into Riemann Surfaces and covering spaces. Good introductions to the general theory are e.g. [12,13,14,15].

Definition 1 (Riemann Surface). *A* Riemann Surface *is a Hausdorff space together with a holomorphic structure, i.e. with an atlas of charts* $\{(U_i, h_i)\}, h_i : U_i \to \mathbb{C}$ *whose transition maps* $h_j \circ h_i^{-1}$ *are* biholomorphic *complex functions.*

In our setting, we use the notion of coverings from complex analysis. They are special cases of topological coverings equipped with a complex structure and can be easily described using local winding maps:

Definition 2 (Winding Map, Covering, Branch Point, Branch Index). *A* winding map $\eta : U \to V$ *between two disks is a map which is isomorphic to the function* $z \mapsto z^n$ *on a unit disk in* $\mathbb{C}$. n *is called the winding number of* η.

A covering (X, η) *of a Riemann Surface* Y *is defined by a map* $\eta : X \to Y$, *such that each point* $y \in Y$ *has a neighbourhood* V *whose inverse image is a union of countably many disks (layers) and the restriction of* η *to each of these disks is a winding map (with winding number* $n(x)$, $\forall x \in \eta^{-1}(y)$*).*

If there is a point x *with* $n(x) \geq 2$, *the covering is* branched *and* $y = \eta(x)$ *is a* branch point *with* branch index $n(x)$. *If* $n(x) = 1$ *for all* $x \in X$, *the covering is* unbranched.

The preimage of a point $y \in Y$ is called the *fibre* of y. One can show, that if the fibre is finite then the sum $gr(\eta) := \sum_{x \in \eta^{-1}(y)} n(x)$ is independant of the choice of y and is therefore called the *grade* of η.

Theoretically, our approach can handle finite or infinite coverings. However, for simplicity we restrict to coverings with finite fibres. For instance, they arise naturally as coverings induced by implicit functions that are defined by algebraic equations. In particular, if one assumes η to be proper (i.e. preimages of compact sets are compact) and to have only finite fibres, η is called a *finite map* and we have the following theorem:

Theorem 1. *Every finite holomorphic map between Riemann surfaces defines a covering.*

The finite holomorphic maps $\mathbb{C} \to \mathbb{C}$ are exactly the non-constant polynomials, hence the powers $z \mapsto z^n$ and all finite concatenations of them are coverings with finite fibres. An example of an infinite covering is the exponential map $\exp : \mathbb{C} \to \mathbb{C}^*$.

For a given covering X, a *Deck map* is an automorphism on X which leaves all fibres invariant, i.e. $D(\eta) := \{g \in \mathrm{Aut}(X) : \eta \circ g = \eta\}$. A Deck map of a covering η is uniquely defined by a given point $a \in X$ and its image $\eta(a)$. The set of all Deck maps form a group and this *Deck group* uniquely defines the topology of the covering.

We consider coverings which are *normal* and *cyclic* meaning that the Deck group is isomorphic to the modulo group $(\mathbb{Z}/n\mathbb{Z}, +)$.

3 Approach

3.1 Visualization of Holomorphic Functions

We construct a topologically correct triangle mesh X for a given covering $\eta : U \to V$ on given simply connected sets $U, V \subset \mathbb{C}$. For simplicity, let η be finite (i.e. η has finitely many layers). The surface X covers V in the same way as U

does, i.e. there is a covering map $\pi : X \to V$ and an isomorphism $f : U \to X$, such that $\eta = f \circ \pi$. Thus, both coverings π and η are isomorphic (Fig. 2, left).

We realize the surface X as a triangle mesh, which admits the same combinatorics as V and whose vertices live in $\mathbb{C} \times \mathbb{R}$. The projection operator π is just the Euclidean projection $(z, r) \mapsto z$. The covering can therefore be seen as a (multivalued) graph over the complex plane. However, in general X cannot be embedded in $\mathbb{R}^3$ and we usually obtain self-intersections.

As an additional visualization, one can now apply the domain coloring technique to these Riemann surfaces (Fig. 2, right). Instead of V, the surface X gets colored by transfering the domain image via f onto X. This will produce a continuous pattern, since f is bijective (in contrast to η). Since π is just a projection along the real axis, all the metric information about the function is still contained in f and can intuitively be captured by the viewer.

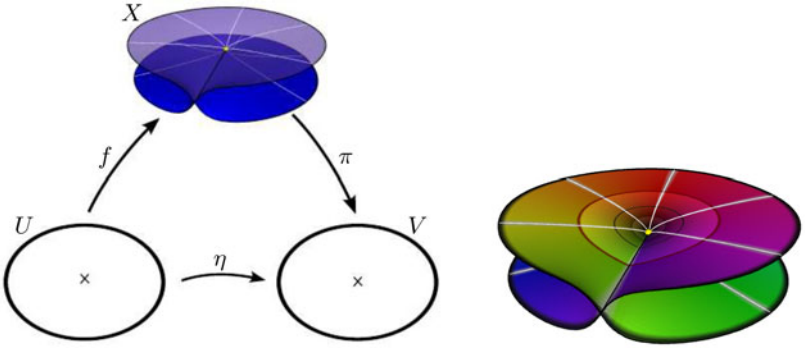


Fig. 2. Left: Given covering η will be visualized by X. Right: Domain coloring of $\eta(z) = z^2$.

This visualization helps to recognize the different types of extraordinary points. Branch points are characterized as center points of a helix, whereas singularities can be recognized as special points in the texture on the covering.

3.2 Branch Points and Branch Graph

Given the position of all branch points $B = \{b_0, \ldots, b_{d-1}\} \subset V$ with corresponding ramification indices $(r_0, \ldots, r_{d-1})$, this uniquely defines a covering over the complex plane which is normal and cyclic up to isomorphism.

A neighborhood of a branch point $p \in \eta^{-1}(B)$ on the covering looks like the union of one or more helices with r_i layers. Away from branch points, $V \setminus B$ is covered by an unbranched surface – the fibre of every open disc is isomorphic to just $N \in \mathbb{N}$ copies of the disk. For a globally consistent covering, we need at least $N := \mathrm{lcm}(r_0, \ldots, r_{d-1})$ many layers (the covering is of degree N). Thus, the fibre over each branch point b_i consists of N/r_i many helices with r_i layers each.

For the construction we need to enumerate the different layers of X. In general, there is no globally consistent enumeration since the covering is a connected surface and the layers exchange their role in different regions.

Given an arbitrary point $v_0 \in V \setminus B$ as root point, its fibre consists of N disjoint points which will be enumerated by $x_0, \ldots, x_{d-1}$. For each pair of points (x_0, x_i), $i \in \{0, \ldots, d-1\}$, there is a unique Deck map (defined on the whole covering) which maps x_0 to x_i. This Deck map transfers the cyclic order of the layers to any other fibre in V (the Deck map is a permutation in each fibre). Thus, all fibres in the covering are global consistently ordered in the same cyclic manner.

We can therefore enumerate the d layers as follows: Let G be a *cut graph* of $V \setminus B$, i.e. the union of paths $\{\gamma_k\}$, which cut the surface open into a simply connected disk (Fig. 3). Each path γ_k must start and end either at a branch point or at the boundary of V. The covering $\eta^{-1}(V \setminus G)$ of this slotted surface then decomposes into d disjoint connected components X_i.

The preimage $\eta^{-1}(\gamma_k)$ of a cut path on the covering consists of d paths in X. Each of these (lifted) paths separate two layers: X_i on the left side and X_j on the right side of the directed path. Because of the cyclic order, the difference $\mathrm{sh}(\gamma_k) = j - i$ is the same for all these lifted paths and is called the *shift* of γ_k. The paths γ_k together with their shifts $\mathrm{sh}(\gamma_k)$ form a *branch graph* of V.

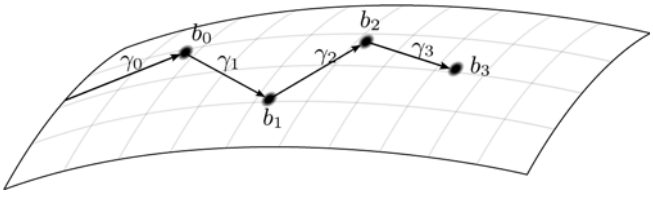


Fig. 3. A branch graph on a surface

The branch graph just connects different layers X_i to a closed covering surface. When a given point $p \in X_i$ in an arbitrary layer of the covering is continuously moved around, it still stays on X_i until a cut path is crossed. In this case, the layer changes to $X_{(i \pm \mathrm{sh}(\gamma_k)) \bmod d}$ (with a + sign if it crosses the path from left to right).

The labeling X_i therefore depends on the choice of the cut graph. However, the covering itself is independant from this choice. Our algorithm will construct an arbitrary branch graph from given branch points and uses it for the generation of the covering. The topology of the covering surface will not depend on G, but only on the position and indices of branch points.

3.3 Shifts

The shifts of a cut path define, how the different layers on the left and right side of the path are connected. Similarly, we introduce the shift at a branch point:

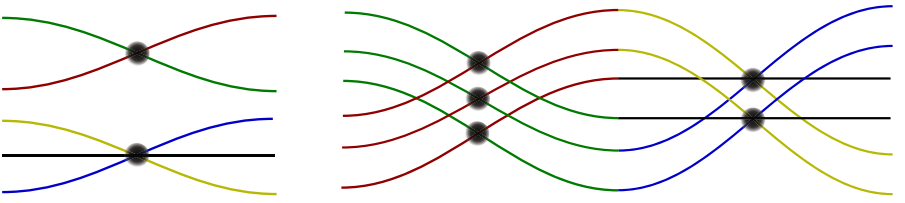


Fig. 4. 2D cut through the layers of different coverings. *Top left*: Branch point with $r_i = 2$. *Bottom left*: branch point with $r_i = 3$. *Right*: Covering with $N = 6$ sheets and two branch points. The left one has $r_i = 2$ and a shift of 3, the right one has $r_i = 3$ and a shift of 2.

Definition 3 (Lifting, Layer Shift). *Given a point $p \in V$ and an infinitesimal small loop $\delta : [0,1] \to V$ around p (counterclockwise). Let δ' be a* lifting *of δ, i.e. a (not necessarily closed) path in X with $\pi(\delta') = \delta$. Denote X_i, X_j the layers of X, such that $\delta'(0) \in X_i$ and $\delta'(1) \in X_j$. Then $sh(p) := (j - i) \bmod N$ is called the* layer shift *of the point p.*

The layer shift just measures how many layers are being crossed when walking around p once. The shift is 0 for all regular points and $\neq 0$ at branch points. The shifts of branch points and the shifts of cut paths are related by the following equation:

Let b_i be a branch point and γ_k the set of paths starting or ending in b_i. Then the *shift* of b_i is the sum

$$\mathrm{sh}(b_i) = \sum_{\text{starting } \gamma_k} \mathrm{sh}(\gamma_k) - \sum_{\text{ending } \gamma_k} \mathrm{sh}(\gamma_k). \tag{1}$$

This defines a linear relation between shifts of the d branch points and shifts of the d cut paths (since V has a boundary, which is also connected by the branch graph G). The sum $-\sum_{b_i} \mathrm{sh}(b_i) \bmod N$ is called the *shift* of the boundary. If it is 0, then the boundary of V lifts to a closed loop on X, otherwise walking along the boundary once will end on another layer on the covering.

The shift at a branch point depends on the ramification index as follows: The neighborhood of each branch point consists of N/r_i helices whose layers are entwinded. Thus, the shift is an arbitrary number $s_i N/r_i$, $s_i \in \mathbb{N}$, but s_i must be coprime to r_i in order to produce the correct number of helices, e.g. set $\mathrm{sh}(b_i) := N/r_i$.

The algorithm only needs the branch graph and the shifts of all cut paths. If only the branch points and their ramification indices are given, then their shifts can be chosen as described above and the shifts of the cut paths uniquely follow from Eqn. 1 (assuming that the shift of the boundary is also prescribed, e.g. to 0).

For the application of domain coloring, it is necessary to explicitly prescribe the shifts in addition to the ramification indices (see Sect. 5.2). This would be an optional input for the algorithm.

4 Algorithmic Generation of Riemann Surface Models

The algorithm for generating the surface models can be separated into several parts. As an input it takes a set of branch points $\{b_0, \ldots, b_{d-1}\}$ which are located on vertices on a simply connected planar geometry M_h, i.e. a triangulated planar mesh.

Additionally, the number of layers N of the covering and the shift of the branch points $\mathrm{sh}(b_i)$ are given. One could alternatively prescribe the local ramification indices r_i of all branch points and then set $N := \mathrm{lcm}(r_0, \ldots, r_{d-1})$ and $\mathrm{sh}(b_i) := s_i N/r_i$ for any arbitrary integer s_i which is coprime to r_i. The shift of the boundary is set to $-\sum_{b_i} \mathrm{sh}(b_i)$.

The following subsections describe the individual steps of the algorithm:

Algorithm 1. Compute Riemann Surface

Input: Triangulated domain M_h, Branch points b_i, Shifts $\mathrm{sh}(b_i)$

1 Generate a branch graph (Sect. 4.1)
2 Cut domain geometry along the branch graph (Sect. 4.2)
3 Compute height function on branch graph in all sheets (Sect. 4.3)
4 Extend height function smoothly to the inner (Sect. 4.4)

4.1 Building the Cut Graph

The cut graph of the surface M_h consists of paths γ_k along edges of M_h, whose union cut the surface open into a topological disk. Branch points are thereby considered as infinitesimal holes in the surface, thus they must be connected by the branch graph.

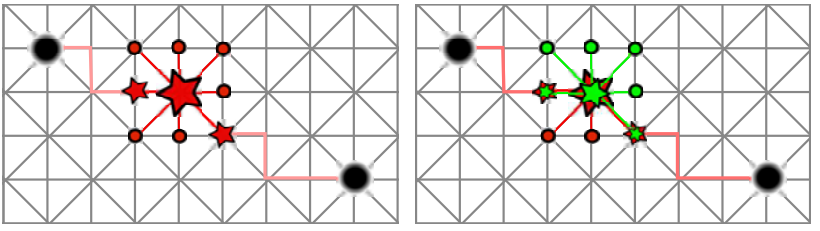
The choice of a special cut graph to given singularities is not unique. There may be more than one graph and these paths might be topologically different and the choice of another cut graph alters the embedding of our constructed surface into $\mathbb{C} \times \mathbb{R}$. However the topology of the embedding does only depend on the branch points and not on the cut graph.

Hence, we allow any user given cut graph as an input. If such a graph is not specified explicitly, it can be automatically generated – a canonical choice is the shortest cut graph of the surface. [11] describes an algorithm on computing this shortest cut graph for surfaces without boundary. A generalization to surfaces with boundary can be found in [16].

Given the cut paths γ_i, their shifts are computed as explained in Sect. 3.3. They are completely determined by the shifts of the branch points.

4.2 Cutting the Base Geometry

The next step is cutting the plane M_h along all cut paths. Each cut path γ_k is given as a path on edges of the planar domain geometry, i.e. it can be described by a list of vertices $v_1, \ldots, v_k$. Cutting M_h along γ_k means that every vertex



(a) Every vertex v (red star) has neighbors (red dots) which are its adjacent vertices in M_h and neighbors on the branch cut γ_k (small stars).

(b) Duplicating v yields a vertex v' (green star). Update the neighbor information such that v (resp. v')is connected with its adjacent vertices lying on the left hand side (resp. right hand side) of γ_k. Note that by cutting the plane, every path becomes part of the boundary.

Fig. 5. Cutting M_h along γ_k

v_j, $j \in \{2, \ldots, k-1\}$ has to be duplicated and its neighborhood has to be updated. The original vertex v_j is connected with vertices on the left hand side of γ_k, whereas the copy v'_j is connected with vertices on the right hand side only. Figure 5 demonstrates this procedure.

The resulting geometry then represents one sheet of the future covering surface. Since all sheets are of the same topology, the slotted domain surface is copied $N-1$ times and we obtain a total of N geometries X_i all cut in exactly the same manner.

4.3 Boundary Constraints for the Height Function

After having cut the domain, we now have N triangle meshes X_i representing the different sheets of the covering, which still live in the complex plane. This section and Sect. 4.4 deals on lifting them into $\mathbb{C} \times \mathbb{R}$ by computing a height function. We start by prescribing height values on the boundary of X_i, i.e. on the cut path and the outer surface boundary.

The sheets in the resulting geometry should be stacked according to their cyclic order, i.e. X_0 is at the bottom and X_{N-1} is the top most layer. The height difference between two consecutive sheets is defined by a constant $\Delta > 0$, thus layer X_i is assigned a height value of $i\Delta$.

Given a sheet X_i and a cut path γ_k between two branch points b_m and b_n, we will now describe, how the height values for the vertices on the left side of γ_k are computed. The vertices on the right side are then handled in the same manner, but using the inverted path with a negative shift value of $-\mathrm{sh}(\gamma_k)$.

The vertices in X_i on γ_k (on the left side) should be connected to vertices of the layer $X_{(i+\mathrm{sh}(\gamma_k)) \bmod N}$. The height function must therefore change smoothly from $i\Delta$ (in X_i) to $(i + \mathrm{sh}(\gamma_k)) \bmod N \cdot \Delta$ (in the neighboring sheet). The path

γ_k is somewhere between these two layers, thus let is give it a height of the average $h_{\text{left}}(\gamma_k) := (i + (i + \text{sh}(\gamma_k)) \bmod N)/2$ for a moment.

If we just assigned this constant height value to all vertices on γ_k, it would cause problems in the vicinity of the start and end points of γ_k (branch points b_m, b_n), since several layers with different height values are glued together there. Hence we interpolate smoothly between these height values.

As described in Sect. 3.2, the branch point b_m occures exactly $N/r_m = \text{sh}(b_m)$ times in the covering. Denote them by $b'_{m,0}, \ldots, b'_{m,N/r_j-1}$. Each $b'_{m,i}$ connects all the sheets $X_i, X_{i+\text{sh}(b_i)}, \ldots, X_{N-\text{sh}(b_j)}$ (Fig. 6, left). We define the height of $b'_{m,i}$ as the average value $((N - \text{sh}(b_j))/2 + i)\Delta$.

We can now define the height function on γ_k with the following properties:

- Height in start point b_m is $((N - \text{sh}(b_m))/2 + (i \bmod \text{sh}(b_m)))\Delta$.
- Height in end point b_n is $((N - \text{sh}(b_n))/2 + (i \bmod \text{sh}(b_n)))\Delta$.
- Height of vertices between b_n and b_m is $(i + ((i + \text{sh}(\gamma_k)) \bmod N))/2$.

We let the height function be constant on the middle third of the branch cut and interpolate on the other parts with a clamped cubic spline, i.e. a polynomial of the form

$$s(x) = a + bx + cx^2 + dx^3 \tag{2}$$

with given boundary values and first derivatives at the end points (Fig. 6, right).

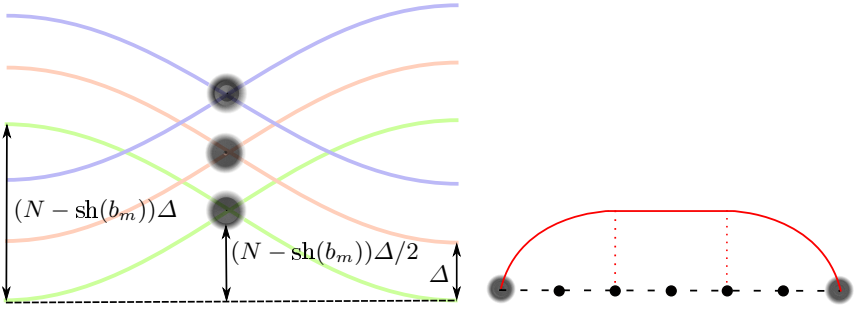


Fig. 6. Computing the height function. Left: Vicinity of a branch point b_m, where different layers are connected. Right: Height function along a cut path γ_k.

It remains to define a height function on the outer boundary of X_i. All our examples are constructed such that the shift of the outer boundary is 0 (i.e. the shifts of all branch points sum to 0). In this case, the height on the outer boundary is just the constant $i\Delta$. If the shift of the boundary is not 0, then the height function must interpolate smoothly between the different layers when walking around the boundary once. It does not matter how this interpolation is done as long as it defines a smooth height function along the boundary.

4.4 Computing the Height Function on the Inside

Having defined a height function on the boundary of each layer X_i, we now solve the Dirichlet problem with given boundary values:

$$\begin{aligned} \Delta f &= 0 \text{ in } X_i \\ f &= \text{prescribed height values on } \partial X_i \end{aligned} \tag{3}$$

More precisely, we solve the corresponding discrete variational formulation

$$Av = b \text{ with } v, b \in \mathbb{R}^{d\times 1}, A \in \mathbb{R}^{d\times d} \tag{4}$$

where d is the number of vertices of X_i, A is the Laplacian matrix, containing the well-known cotan weights of the underlying triangle mesh, and b is the right side, which is almost zero except for the boundary values f.

The solution is a harmonic function and produces a smooth surface in the interior of all sheets. However, along the cuts the surface is in general not C^1 continuous.

It should be mentioned that this height function is just one option and of course one is free to choose any arbitrary height function. Heuristically, harmonic functions often produce elegant surfaces and they are closely related to holomorphic functions (in fact, real and imaginary part of a holomorphic function are harmonic functions). The only thing one has to care about is that the values on the boundary fit together, such that the "gluing" is correctly reflected by the model.

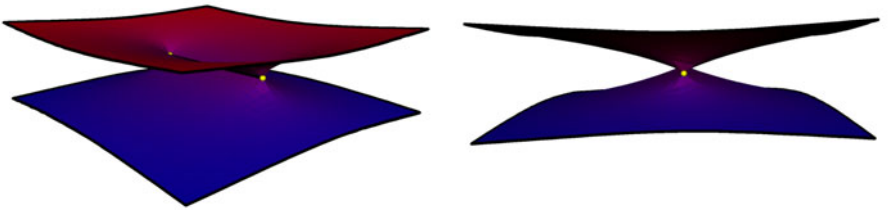
5 Results

5.1 Riemann Surface Models

The following models are generated by our algorithm. We applied further domain coloring techniques on some models, which are then shown in Sect. 5.2. Branch points are highlighted as small yellow balls. The transition between adjacent layers as well as the surface boundary is marked by black lines.

Figures 7 and 8 show a two-sheeted Riemann surface with two branch points of shift 1 (i.e. index is 2). We generated two embeddings using different cut graphs. The cut graph in Fig. 7 directly connects both branch points, whereas in Fig. 8, the two branch cuts emanate from a singularity and both meet at infinity. Thus, the situation is indeed the same and both surfaces are therefore topologically equivalent. Since we can only compute a compact subset of the infinite Riemann surface, we have to take care, that the height function respects the layer shift at the patch boundary, whenever the cut graph meets the boundary.

Figure 9 shows a Riemann surface with three sheets and two branch points of index 3, the shifts are 1 and -1. When crossing the branch cut between the two singularities from left to right one ends up the next lower sheet, or on the top most sheet when starting from the lowest one.



(a) In this case the branch cut coincides with the line of self intersection, which is in general not the case.

(b) A side view shows the similarity to the schematic pictures in Fig. 4

Fig. 7. A surface with two sheets and two branch points of index 2

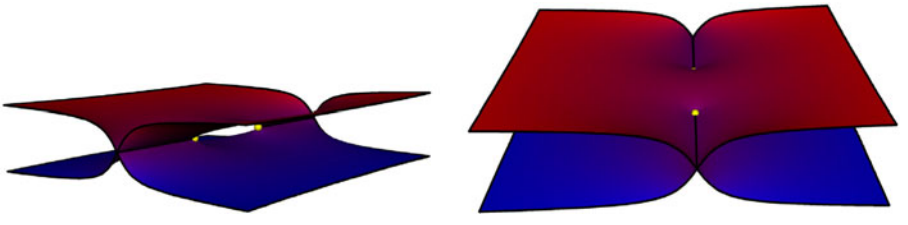
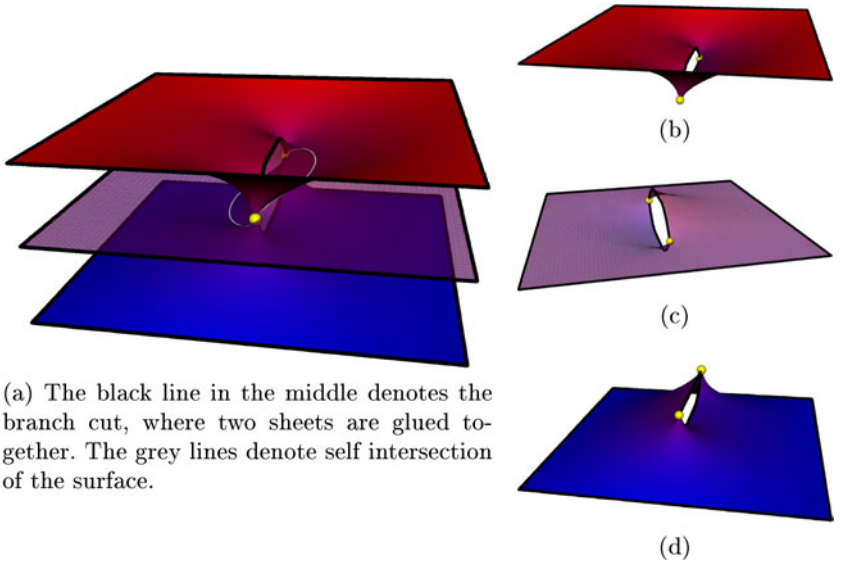


Fig. 8. Another embedding of the Riemann surface from Fig. 7



(a) The black line in the middle denotes the branch cut, where two sheets are glued together. The grey lines denote self intersection of the surface.

Fig. 9. A surface with three sheets and two branch points. (a) shows the whole surface. (b)-(d) show the three sheets separately.

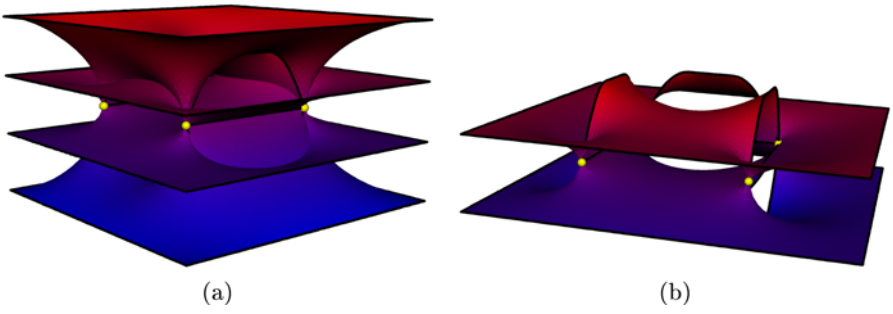


Fig. 10. Riemann surface with four layers and four branch points. (a) Whole surface. (b) Second and third sheet slightly rotated.

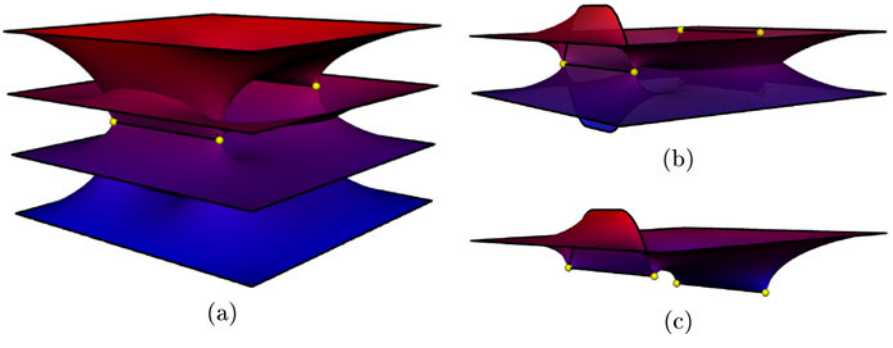


Fig. 11. Riemann surface with four sheets and four branch points. (a) Whole surface. (b) Second and third sheet. (c) Displaying only the third sheet reveals the hidden branch points.

A more complex model is shown in Fig. 10. It has four sheets and four branch points A, B, C, D of shift 1, 2, 2 and 3, respectively. The branch points form a square (the branch point on the corner in the back is hidden) and are connected by branch cuts between A and B, B and C and C and D. The two leftmost branch points (A and D) in (a) are not connected. When starting from the right on the upper sheet and crossing the branch cuts between C and D and A and B, one does not change the layer. This corresponds to the fact that the branch shifts along these cuts sum up to four which is zero modulo four.

Figure 11 again shows the same situation with the same branch points as in Fig. 10, but with a different branch graph. This time, A and C are connected as well as B and D, so the branch graph is disconnected. Note that this is also a valid configuration since the branch shifts along each cut sum up to zero modulo four. However, we obtain a different embedding in $\mathbb{R}^3$, although all branch shifts are the same as in the previous picture. In (b) one can clearly see the layer shift of one when crossing the cut in the front of the picture. Note that the second branch cut in the back connects two branch points of shift 2 and thus the model

locally decomposes into two connected components around this cut. That is why a second pair of branch points is introduced which can be slightly seen.

5.2 Domain Coloring on Riemann Surfaces

As mentioned in Sec. 2.2, Riemann surfaces are naturally induced by holomorphic covering maps. However the visualization of complex functions as graphs over a domain is not possible, since their graphs live in $\mathbb{C}^2 \cong \mathbb{R}^4$. That is why one often makes use of an elegant technique called *domain coloring* (see [5] or [10]), which encodes the range of a complex function as a color scheme that is plotted directly onto the domain. We use this technique to visualize complex functions whose proper domains are Riemann surfaces. In particular, for a given covering map η as in Fig. 2, we can define its inverse mapping as a function living on the Riemann surface. Those functions are naturally multivalued when defined over the complex plane and hence cannot be properly defined as holomorphic functions on $\mathbb{C}$ – in fact they are not even continuous and one has to cut the plane in order to define at least a so called *holomorphic branch of a function*. With the notation of Fig. 2, these branches can be considered as restrictions of f^{-1} to a sheet $X_i \subset X$ and $\pi\,|_{X_i}$ becomes an isomorphism between the cut complex plane and X_i.

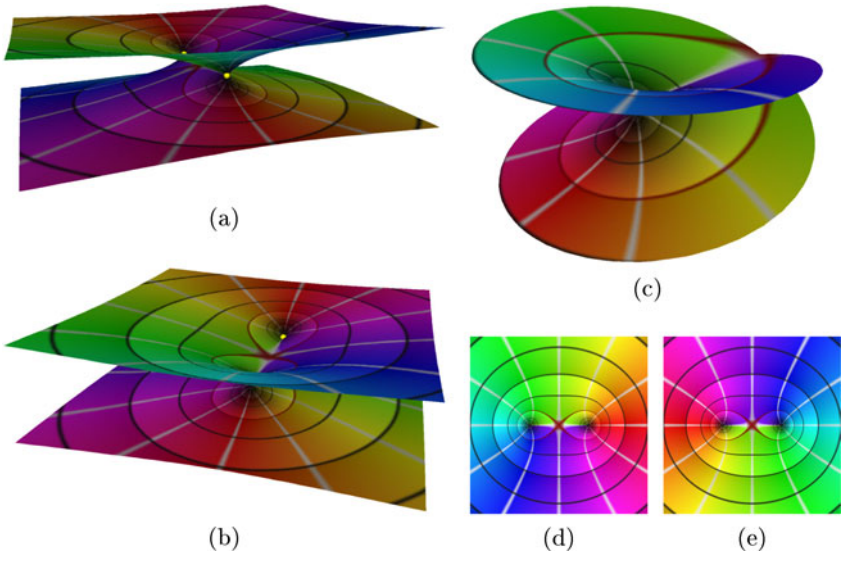


Fig. 12. A function having two branches and a branch cut between the branch points -1 and $+1$. (a) and (b) show the model from Fig. 7 with the coloring for $f^{-1}(z) := (z-1)^{1/2}(z+1)^{1/2}$. Whereas the restriction to a single sheet results in a color–discontinuous planar plot (see (d) and (e)), the coloring on the surface is continuous. (c) shows the local topology of our model close to a branch point. The typical two-sheeted helix is a picture often shown as an explicitly parametrized surface for the Riemann surface induced by $z \mapsto z^2$.

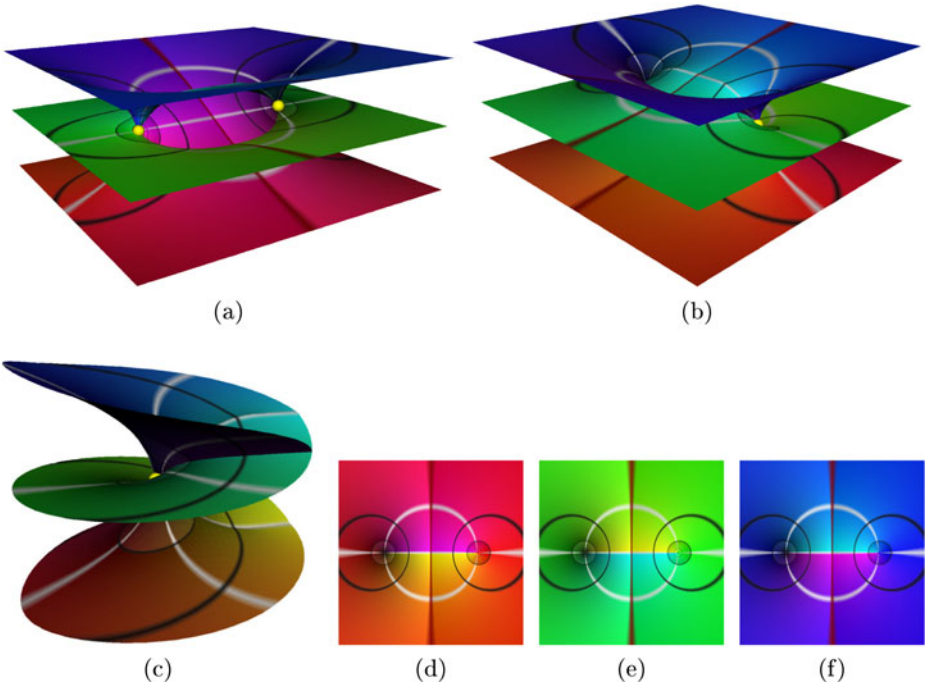


Fig. 13. Domain coloring of a function with three branches. (c) Local model in a neighborhood of a branch point The planar color plots (d)–(f) are discontinuous between the branch points which is resolved on the Riemann surface model.

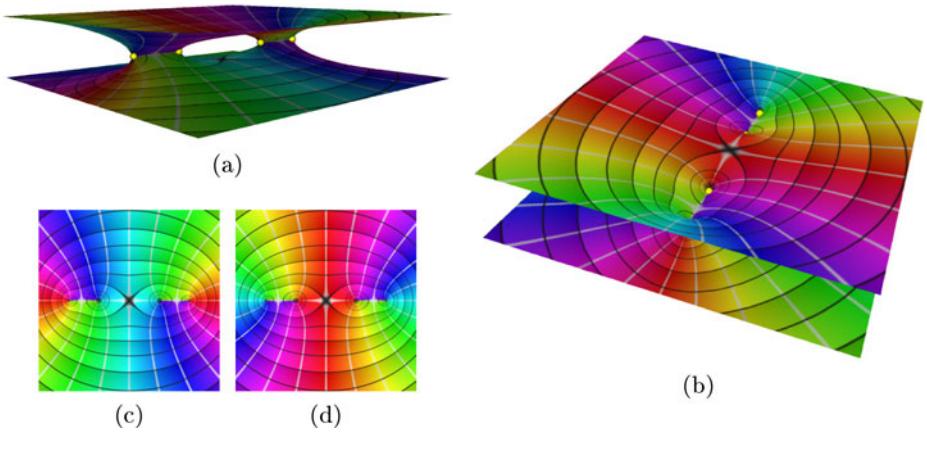


Fig. 14. (a) and (b) Domain colored Riemann surface with two disconnected branch cuts. (c) and (d) The planar domain coloring of the function is discontinuous at the branch cut.

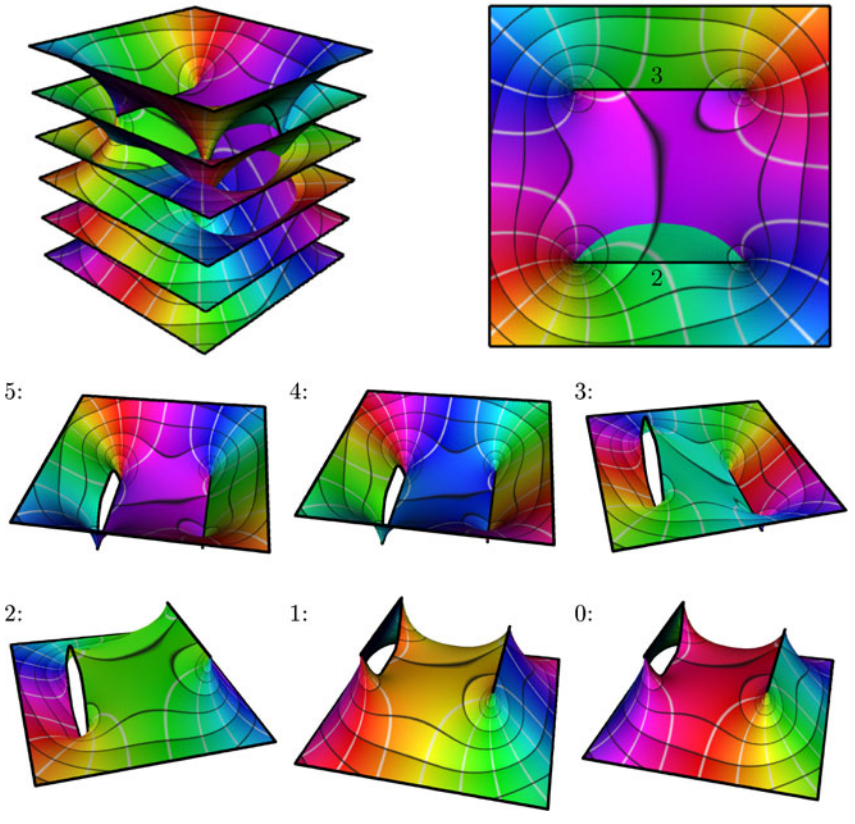


Fig. 15. A 6-sheeted covering of a function having two branch cuts with different shift and thus with branch points of different branch indices – two of index 3 and two of index 2. Top left: whole covering. Top right: 2d view on top most layer. The black lines are the cut paths with not vanishing shift (shift 2 and 3). The 6 pictures below show the layers separately.

We apply domain coloring to Riemann surfaces and obtain globally color-continuous plots corresponding to globally holomorphic mappings on the surfaces. These surfaces are computed by using our method with the branch points and the shifts of η which can be derived from the function definition. The coloring is induced by f^{-1} and the colored surfaces reveal the behaviour of a function as well as the topology of the induced Riemann surface.

Note that the composition $f^{-1} \circ (\pi \mid_{X_i})^{-1}$ maps from the cut complex plane into $\mathbb{C}$ and can be visualized using planar domain coloring, but the unrestricted map $(\pi \circ f)^{-1}$ is in general multivalued. Figures 9 – 12 show some colored surfaces together with planar domain colorings obtained by restriction to a single sheet. A two-sheeted Riemann surface with domain coloring is shown in Fig. 12.

Figure 13 shows the domain coloring of a function having three sheets and a two branch points at $+1$ and -1. The Riemann surface is the same one as

in Fig. 9, this time equipped with domain coloring of a holomorphic function. One can see the smooth transition between the first and the third sheet in (a) whereas (b) shows the transition between the second and third sheet.

Figure 14 shows a Riemann surface with two branch cuts which are not connected, or, equivalently, as being connected by a branch cut with shift 0, which does not cause any sheet transition. As in the example in Fig. 12, the branch cuts connect two sheets and the restriction to a single sheet results in a discontinuouities of the branched function, see (c), (d). Note that the function is continous on every sheet between the branch cuts.

6 Outlook

Although our algorithm is capable to deal with a large class of functions, there are some issues left to solve.

Having realized the generation of Riemann surface models in $\mathbb{R}^3$ it is tempting to switch to the projective setting and consider compact surfaces over the compactified complex plane $\mathbb{C} \cup \{\infty\}$, i.e. the Riemann sphere. A closely related advancement is the consideration of covering surfaces over arbitrary manifolds.

Another challenging task is the extension of domain coloring to maps between arbitrary Riemann surfaces, i.e. maps which are multivalued and non-injective when defined on the complex plane. This implies that neither the map itself nor its inverse can be continuously defined on $\mathbb{C}$.

References

1. Yin, X., Jin, M., Gu, X.: Computing shortest cycles using universal covering space. Vis. Comput. 23(12), 999–1004 (2007)
2. Kälberer, F., Nieser, M., Polthier, K.: Quadcover - surface parameterization using branched coverings. Comput. Graph. Forum. 26(3), 375–384 (2007)
3. Trott, M.: Visualization of Riemann surfaces (2009), http://library.wolfram.com/examples/riemannsurface/ (retrieved December 8, 2009)
4. Trott, M.: Visualization of Riemann surfaces of algebraic functions. Mathematica in Education and Research 6, 15–36 (1997)
5. Farris, F.A.: Visualizing complex-valued functions in the plane, http://www.maa.org/pubs/amm_complements/complex.html (retrieved December 8, 2009)
6. Pergler, M.: Newton's method, Julia and Mandelbrot sets, and complex coloring, http://users.arczip.com/pergler/mp/documents/ptr/ (retrieved December 8, 2009)
7. da Silva, E.L.: Reviews of functions of one complex variable graphical representation from software development for learning support, http://sorzal-df.fc.unesp.br/~edvaldo/en/index.htm (retrieved on December 8, 2009)
8. Lundmark, H.: Visualizing complex analytic functions using domain coloring (2004), http://www.mai.liu.se/~halun/complex/domain_coloring-unicode.html (retrieved on December 8, 2009)

9. Hlavacek, J.: Complex domain coloring, http://www6.svsu.edu/~jhlavace/Complex_Domain_Coloring/index.html (retrieved on December 8, 2009)
10. Poelke, K., Polthier, K.: Lifted domain coloring. Computer Graphics Forum 28(3), 735–742 (2009)
11. Erickson, J., Whittlesey, K.: Greedy optimal homotopy and homology generators. In: SODA, pp. 1038–1046 (2005)
12. Farkas, H.: Riemann Surfaces. Springer, New York (1980)
13. Lamotke, K.: Riemannsche Flächen. Springer, Berlin (2005)
14. Forster, O.: Lectures on Riemann Surfaces, 4th edn. Graduate Texts in Mathematics. Springer, Heidelberg (1999)
15. Needham, T.: Visual Complex Analysis. Oxford University Press, Oxford (2000)
16. Kälberer, F., Nieser, M., Polthier, K.: Stripe parameterization of tubular surfaces. In: Topology-Based Methods in Visualization III. Mathematics and Visualization. Springer, Heidelberg (to appear 2010)
17. Riemann, B.: Grundlagen für eine allgemeine theorie der functionen einer veränderlichen complexen größe (1851)

G^1 Bézier Surface Generation from Given Boundary Curve Network with T-Junction

Min-jae Oh[1], Sung Ha Park[1], and Tae-wan Kim[2]

[1] Department of Naval Architecture and Ocean Engineering, Seoul National University, Seoul 151-744, Republic of Korea
[2] Department of Naval Architecture and Ocean Engineering, and Research Institute of Marine Systems Engineering, Seoul National University, Seoul 151-744, Republic of Korea

Abstract. T-junctions usually appear in surface modeling processes that start with a given curve network. However, since T-shaped patches are not available in current CAD system so existing G^1 surface generation methods are restricted to n-sided patches. Therefore a designer must design a curve network without T-junctions, or subdivide it into n-sided patches, to avoid T-shaped topologies. We generate G^1 Bézier surfaces at a T-junction by combining the coplanar G^1 continuity condition with the de Casteljau algorithm to avoid the collision of twist points. Both T-junctions in the middle of boundary curves and at 3-valent vertices are considered. Our method requires no subdivision or triangulation of the surface, and the curve network is not changed.

1 Introduction

In modeling processes which start from a given curve network, continuity of the resulting surface is usually important. For example, a car body can be modeled by surfacing a curve network. The resulting surface has to meet criteria such as low air resistance and a smooth surface is also likely to be aesthetically pleasing. Similarly, the hull of a ship should be smooth to create as little drag as possible. Surface continuity is critical in modeling these types of structure. But the curve networks on which composite surfaces are constructed usually have some T-junctions. G^1 current CAD systems cannot generate smooth surfaces at T-junction, and the designer usually has to re-design the curve network, or it is preprocessed to reduce the T-junctions.

A great deal of work has been done on the interpolation[1] of a surface over a given curve network. Bézier[2] himself introduced a method of joining two Bézier patches with G^1 continuity across the common boundary curve, and Farin[3] and Sarraga[4] developed this condition in terms of the Bézier control points of adjacent patches. Several other authors[5, 6, 7, 8, 9] have dealt with the coplanar condition for G^1 continuity, employing patches and boundary curves of various degrees. Loop[10] devised another way to construct a C^2 boundary curve network of arbitrary topological type using the theory of circulant matrices, so that the network approximates the vertices of a triangular control mesh. In

B. Mourrain, S. Schaefer, and G. Xu (Eds.): GMP 2010, LNCS 6130, pp. 179–191, 2010.

an extension of Loop's scheme, Hanmann and Bonneau[11] suggested domain-splitting techniques in which a macro triangle consists of 4 quintic triangular Bézier patches, while the boundary curves are piecewise C^1 continuous cubics. Liu and Mann[12] also adopted Loop's scheme to interpolate parametric triangular surfaces to achieve approximate G^1 continuity. Recently, Cho et al.[13] have proposed a method of interpolation for ship hulls that cares G^1 Bézier surfaces over an irregular curve network; they analyzed the degree of the fill-in patches and the local singularities that can occur during construction[14]. However, several subdivision schemes are required to preprocess curves with T-junctions, which still represent topological restrictions on the curve network, although some improvements can be found in more recent work[15, 16].

We extend the work of Cho et al., and present a new method of generating a G^1 Bézier surface at a T-junction, which combines the coplanar G^1 continuity condition with the de Casteljau algorithm to satisfy the vertex enclosure constraint. The advantages of our method can be summarized as follows:

(1) We can generate a G^1 continuous Bézier surface from a given boundary curve which has a T-junction by classifying the T-junction into two types and applying a different G^1 surface generation method in each case.
(2) We do not need to subdivide or change the given curve network.
(3) Our method is applied locally to the vertex and edge conditions.

The rest of this paper is structured as follows: In Section 2, we introduce the two types of T-junction that we deal with. In Section 3, we describe the geometric conditions for G^1 continuity, and the algorithm to generate a smooth surface for both types of T-junction. In Section 4, we show the results of our algorithm on example surface configurations, showing the resulting continuity by drawing reflection lines. We conclude the paper in Section 5.

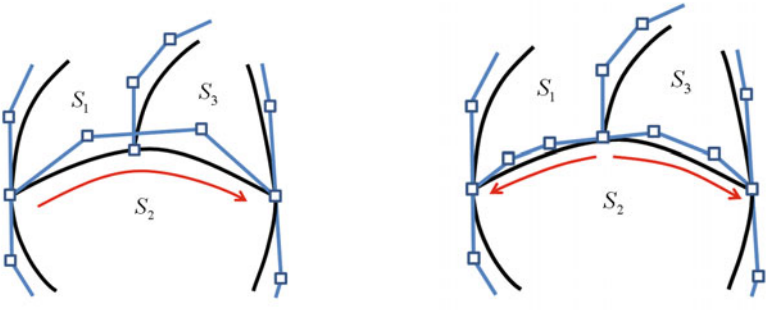
2 Two Types of T-Junction

In this paper we consider two types of T-junction; one occurs when an edge ends in the middle of a boundary curve(Fig. 1(a)); the other occurs when three edges, two of which are collinear, meet at a 3-valent vertex(Fig. 1(b)). The topology in Fig. 1(a) frequently occurs when modeling starts with a given curve network, while that in Fig. 1(b) is less frequent: nevertheless, we present a method for each type of T-junction. In this paper, we only consider about boundary curve with 'a' T-junction. If there are many T-junctions at boundary curves, it should be much more complicated problem. We will remain it as future work.

3 Generating a G^1 Bézier Surface at a T-Junction

3.1 T-Junction on a Boundary Curve

Consider three triangular patches S_1, S_2, and S_3, of degree 6 (see Fig. 2). Here, and throughout this paper, we will assume that boundary curves are of degree 3,



(a) T-junction on a boundary curve

(b) T-junction at a 3-valent vertex

Fig. 1. Two types of T-junction

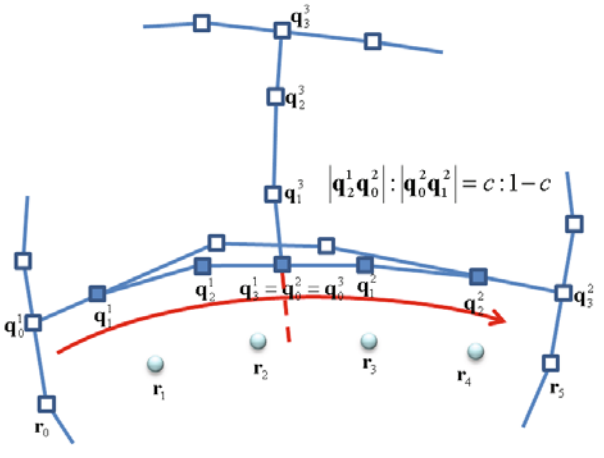


Fig. 2. The distribution of control points on the initial surfaces

with a linear weight function $\lambda(t)$ and a quartic weight function $\mu(t)$, as in previous work[14]. It is proved that the edge condition and the vertex condition can be solved separately when we use the triangular patches of degree 6(the degree will be 5 at rectangular case) in previous work. Farin[1] described the G^1 surface generation method using coplanar condition between two triangular patches using linear weight functions. Furthermore, he extended the result to the rectangular patches using degree-elevation method to the rectangular boundary. If we elevate the degree to the one boundary curve of the rectangle, the boundary curve and the cross boundary curve's topology will be same as triangle boundary. For example, if the rectangular patch's degree is 5, and elevate the one boundary curve for G^1 surface generation, then the degree is same as triangular patch with degree 6. Therefore, we suggest the G^1 surface generating algorithm among the triangular patches, and the algorithm can be extended to the rectangular patches by boundary degree-elevation.

G^1 Condition at the vertex. Cho et al.[14] suggest a method of solving the equations that determine both the vertex and the edge condition, so as to generate G^1 surface. We use this method to solve the T-junction problem. The T-junction is itself a vertex, but we cannot directly apply the same condition that we apply to other vertices because there is no information about the interior of the surface S_2 in Fig. 2. So we subdivide S_2 at parameter c and create auxiliary control points $\widetilde{\mathbf{r}}_i^1$, $\widetilde{\mathbf{r}}_i^2$ [1]. The parameter c can be calculated by given boundary curve and intersecting point($\mathbf{q}_0^2$). Keeping the requirement for planarity at the junction in mind, we define the control point $\widetilde{\mathbf{r}}_5^1$ to be collinear with $\mathbf{p}_5^1$ and $\mathbf{q}_0^2$. For simplicity, we choose the ratio of $\mathbf{p}_5^1\mathbf{q}_0^2$ and $\mathbf{q}_0^2\widetilde{\mathbf{r}}_5^1$ to be $1:1$. This ratio can be changed depending on the shape of the surface.

Fig. 3 shows the subdivided surface with the auxiliary control points. Now we can apply the same approach as Cho et al. to obtain equations for both the vertices and edges that correspond to the G^1 continuity condition between surface S_1 and S_2, which can then be expressed as follows:

$$
\begin{aligned}
&k=0: \left(\overline{\lambda_0^1}\mathbf{p}_0^1+\lambda_0^1\widetilde{\mathbf{r}_0^1}\right)=\left(\overline{\mu_0^1}\mathbf{q}_0^1+\mu_0^1\mathbf{q}_1^1\right)\\
&k=1:5\left(\overline{\lambda_0^1}\mathbf{p}_1^1+\lambda_0^1\widetilde{\mathbf{r}_1^1}\right)+\left(\overline{\lambda_1^1}\mathbf{p}_0^1+\lambda_1^1\widetilde{\mathbf{r}_0^1}\right)\\
&\qquad=2\left(\overline{\mu_0^1}\mathbf{q}_1^1+\mu_0^1\mathbf{q}_2^1\right)+4\left(\overline{\mu_1^1}\mathbf{q}_0^1+\mu_1^1\mathbf{q}_1^1\right)\\
&k=2:10\left(\overline{\lambda_0^1}\mathbf{p}_2^1+\lambda_0^1\widetilde{\mathbf{r}_2^1}\right)+5\left(\overline{\lambda_1^1}\mathbf{p}_1^1+\lambda_1^1\widetilde{\mathbf{r}_1^1}\right)\\
&\qquad=\left(\overline{\mu_0^1}\mathbf{q}_2^1+\mu_0^1\mathbf{q}_3^1\right)+8\left(\overline{\mu_1^1}\mathbf{q}_1^1+\mu_1^1\mathbf{q}_2^1\right)+6\left(\overline{\mu_2^1}\mathbf{q}_0^1+\mu_2^1\mathbf{q}_1^1\right)\\
&k=3:10\left(\overline{\lambda_0^1}\mathbf{p}_3^1+\lambda_0^1\widetilde{\mathbf{r}_3^1}\right)+10\left(\overline{\lambda_1^1}\mathbf{p}_2^1+\lambda_1^1\widetilde{\mathbf{r}_2^1}\right)\\
&\qquad=4\left(\overline{\mu_1^1}\mathbf{q}_2^1+\mu_1^1\mathbf{q}_3^1\right)+12\left(\overline{\mu_2^1}\mathbf{q}_1^1+\mu_2^1\mathbf{q}_2^1\right)+4\left(\overline{\mu_3^1}\mathbf{q}_0^1+\mu_3^1\mathbf{q}_1^1\right)\\
&k=4:5\left(\overline{\lambda_0^1}\mathbf{p}_4^1+\lambda_0^1\widetilde{\mathbf{r}_4^1}\right)+10\left(\overline{\lambda_1^1}\mathbf{p}_3^1+\lambda_1^1\widetilde{\mathbf{r}_3^1}\right)\\
&\qquad=6\left(\overline{\mu_2^1}\mathbf{q}_2^1+\mu_2^1\mathbf{q}_3^1\right)+8\left(\overline{\mu_3^1}\mathbf{q}_1^1+\mu_3^1\mathbf{q}_2^1\right)+\left(\overline{\mu_4^1}\mathbf{q}_0^1+\mu_4^1\mathbf{q}_1^1\right)\\
&k=5:\left(\overline{\lambda_0^1}\mathbf{p}_5^1+\lambda_0^1\widetilde{\mathbf{r}_5^1}\right)+5\left(\overline{\lambda_1^1}\mathbf{p}_4^1+\lambda_1^1\widetilde{\mathbf{r}_4^1}\right)\\
&\qquad=4\left(\overline{\mu_3^1}\mathbf{q}_2^1+\mu_3^1\mathbf{q}_3^1\right)+2\left(\overline{\mu_4^1}\mathbf{q}_1^1+\mu_4^1\mathbf{q}_2^1\right)\\
&k=6:\left(\overline{\lambda_1^1}\mathbf{p}_5^1+\lambda_1^1\widetilde{\mathbf{r}_5^1}\right)=\left(\overline{\mu_4^1}\mathbf{q}_2^1+\mu_4^1\mathbf{q}_3^1\right).
\end{aligned}
\tag{1}
$$

The continuity equations at the boundaries S_2S_3 and S_3S_1 are the same as there, except for the indices of the control points.

Then the equations for the vertices will be

$$
\begin{aligned}
5\left(\overline{\lambda_1^1}\mathbf{p}_4^1+\lambda_1^1\widetilde{\mathbf{r}_4^1}\right)&=-\left(\overline{\lambda_0^1}\mathbf{p}_5^1+\lambda_0^1\widetilde{\mathbf{r}_5^1}\right)+4\left(\overline{\mu_3^1}\mathbf{q}_2^1+\mu_3^1\mathbf{q}_3^1\right)+2\left(\overline{\mu_4^1}\mathbf{q}_1^1+\mu_4^1\mathbf{q}_2^1\right),\\
5\left(\overline{\lambda_0^2}\mathbf{p}_1^2+\lambda_0^2\widetilde{\mathbf{r}_1^2}\right)&=-\left(\overline{\lambda_1^2}\mathbf{p}_0^2+\lambda_1^2\widetilde{\mathbf{r}_0^2}\right)+2\left(\overline{\mu_0^2}\mathbf{q}_1^2+\mu_0^2\mathbf{q}_2^2\right)+4\left(\overline{\mu_1^2}\mathbf{q}_0^2+\mu_1^2\mathbf{q}_1^2\right),\\
5\left(\overline{\lambda_0^3}\mathbf{p}_1^3+\lambda_0^3\mathbf{r}_1^3\right)&=-\left(\overline{\lambda_1^3}\mathbf{p}_0^3+\lambda_1^3\mathbf{r}_0^3\right)+2\left(\overline{\mu_0^3}\mathbf{q}_1^3+\mu_0^3\mathbf{q}_2^3\right)+4\left(\overline{\mu_1^3}\mathbf{q}_0^3+\mu_1^3\mathbf{q}_1^3\right).
\end{aligned}
$$

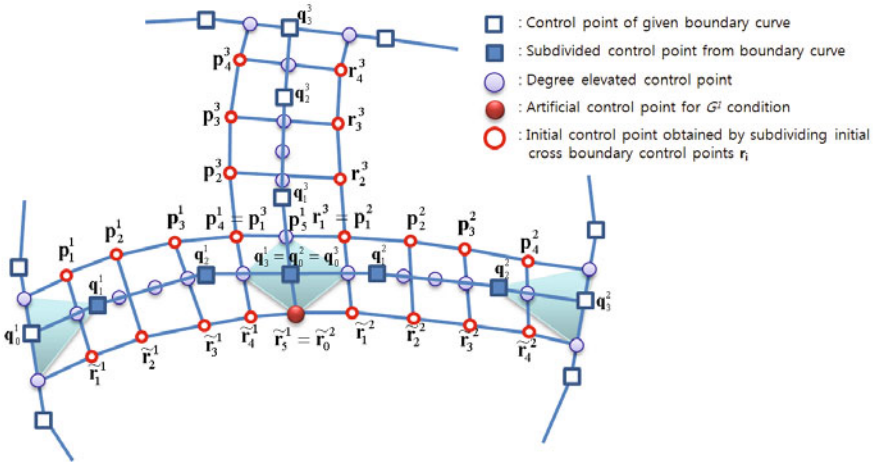


Fig. 3. The subdivided control points derived from the initial surfaces

We can see from Fig. 3 that $\mathbf{p}_1^3 = \mathbf{p}_4^1$ and $\mathbf{r}_1^3 = \mathbf{p}_1^2$, and hence we obtain

$$5 \begin{bmatrix} \overline{\lambda_1^1} & \lambda_1^1 & 0 & 0 \\ 0 & 0 & \overline{\lambda_0^2} & \lambda_0^2 \\ \overline{\lambda_0^3} & 0 & \lambda_0^3 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p}_4^1 \\ \widetilde{\mathbf{r}_4^1} \\ \mathbf{p}_1^2 \\ \widetilde{\mathbf{r}_1^2} \end{bmatrix} = \begin{bmatrix} rhs_1^T(\mu_3^1) \\ rhs_2^T(\mu_1^2) \\ rhs_3^T(\mu_1^3) \end{bmatrix},$$

where

$$\begin{aligned} rhs_1^T(\mu_3^1) &= -\left(\overline{\lambda_0^1}\mathbf{p}_5^1 + \lambda_0^1\widetilde{\mathbf{r}_5^1}\right) + 4\left(\overline{\mu_3^1}\mathbf{q}_2^1 + \mu_3^1\mathbf{q}_3^1\right) + 2\left(\overline{\mu_4^1}\mathbf{q}_1^1 + \mu_4^1\mathbf{q}_2^1\right), \\ rhs_2^T(\mu_1^2) &= -\left(\overline{\lambda_1^2}\mathbf{p}_0^2 + \lambda_1^2\widetilde{\mathbf{r}_0^2}\right) + 2\left(\overline{\mu_0^2}\mathbf{q}_1^2 + \mu_0^2\mathbf{q}_2^2\right) + 4\left(\overline{\mu_1^2}\mathbf{q}_0^2 + \mu_1^2\mathbf{q}_1^2\right), \\ rhs_3^T(\mu_1^3) &= -\left(\overline{\lambda_1^3}\mathbf{p}_0^3 + \lambda_1^3\mathbf{r}_0^3\right) + 2\left(\overline{\mu_0^3}\mathbf{q}_1^3 + \mu_0^3\mathbf{q}_2^3\right) + 4\left(\overline{\mu_1^3}\mathbf{q}_0^3 + \mu_1^3\mathbf{q}_1^3\right). \end{aligned}$$

This system is underdetermined because there are three equations and four unknowns. By Gaussian elimination we can put it into matrix form, as follows:

$$\left[\begin{array}{cccc:c} \overline{\lambda_1^1} & \lambda_1^1 & 0 & 0 & rhs_1^T \\ 0 & 0 & \overline{\lambda_0^2} & \lambda_0^2 & rhs_2^T \\ \overline{\lambda_0^3} & 0 & \lambda_0^3 & 0 & rhs_3^T \end{array}\right] \Rightarrow \left[\begin{array}{cccc:c} \overline{\lambda_1^1} & \lambda_1^1 & 0 & 0 & rhs_1^T \\ 0 & -\frac{\overline{\lambda_0^3}}{\overline{\lambda_1^1}}\lambda_1^1 & \lambda_0^3 & 0 & rhs_3^T - \frac{\overline{\lambda_0^3}}{\overline{\lambda_1^1}} rhs_1^T \\ 0 & 0 & \overline{\lambda_0^2} & \lambda_0^2 & rhs_2^T \end{array}\right].$$

Since $\lambda_1^1 \neq 0, \lambda_0^2 \neq 0$, and $\lambda_0^3 \neq 0$, the matrix has a full rank of 3, and the system can be solved iteratively by the least-squares method from an initial guess.

***G^1* Condition at the Edges.** We only need to consider the G^1 continuity condition the between the edges of patches S_1 and S_2, because the conditions at the edges S_2S_3 and S_3S_1 are the same as those for S_1S_2, except for the indices

of the control points. The conditions at the edge S_1S_2 can be written in matrix form as follows:

$$10\begin{bmatrix} \overline{\lambda_0^1} & \lambda_0^1 & 0 & 0 \\ \overline{\lambda_1^1} & \lambda_1^1 & \overline{\lambda_0^1} & \lambda_0^1 \\ 0 & 0 & \overline{\lambda_1^1} & \lambda_1^1 \end{bmatrix}\begin{bmatrix} \mathbf{p}_2^1 \\ \widetilde{\mathbf{r}_2^1} \\ \mathbf{p}_3^1 \\ \widetilde{\mathbf{r}_3^1} \end{bmatrix} = \begin{bmatrix} rhs_1^1(\mu_1^1, \mu_2^1) \\ rhs_2^1(\mu_1^1, \mu_2^1, \mu_3^1) \\ rhs_3^1(\mu_2^1, \mu_3^1) \end{bmatrix}, \tag{2}$$

where

$$\begin{aligned} rhs_1^1(\mu_1^1, \mu_2^1) &= -5\left(\overline{\lambda_1^1}\mathbf{p}_1^1 + \lambda_1^1\widetilde{\mathbf{r}_1^1}\right) + \left(\overline{\mu_0^1}\mathbf{q}_2^1 + \mu_0^1\mathbf{q}_3^1\right) \\ &\quad +8\left(\overline{\mu_1^1}\mathbf{q}_1^1 + \mu_1^1\mathbf{q}_2^1\right) + 6\left(\overline{\mu_2^1}\mathbf{q}_0^1 + \mu_2^1\mathbf{q}_1^1\right), \\ rhs_2^1(\mu_1^1, \mu_2^1, \mu_3^1) &= 4\left(\overline{\mu_1^1}\mathbf{q}_2^1 + \mu_1^1\mathbf{q}_3^1\right) + 12\left(\overline{\mu_2^1}\mathbf{q}_1^1 + \mu_2^1\mathbf{q}_2^1\right) \\ &\quad +4\left(\overline{\mu_3^1}\mathbf{q}_0^1 + \mu_3^1\mathbf{q}_1^1\right), \\ rhs_3^1(\mu_2^1, \mu_3^1) &= -5\left(\overline{\lambda_0^1}\mathbf{p}_4^1 + \lambda_0^1\widetilde{\mathbf{r}_4^1}\right) + 6\left(\overline{\mu_2^1}\mathbf{q}_2^1 + \mu_2^1\mathbf{q}_3^1\right) \\ &\quad +8\left(\overline{\mu_3^1}\mathbf{q}_1^1 + \mu_3^1\mathbf{q}_2^1\right) + \left(\overline{\mu_4^1}\mathbf{q}_0^1 + \mu_4^1\mathbf{q}_1^1\right). \end{aligned}$$

Since the parameters μ_1^1 and μ_3^1 are determined from the vertex condition, the right-hand side has only one parameter, μ_2^1.

System (2) is underdetermined, with three equations and four unknowns. By Gaussian elimination, we can find the augmented matrix

$$\begin{bmatrix} \overline{\lambda_0^1} & \lambda_0^1 & 0 & 0 & : rhs_1^1 \\ \overline{\lambda_1^1} & \lambda_1^1 & \overline{\lambda_0^1} & \lambda_0^1 & : rhs_2^1 \\ 0 & 0 & \overline{\lambda_1^1} & \lambda_1^1 & : rhs_3^1 \end{bmatrix} \Rightarrow \begin{bmatrix} \overline{\lambda_0^1} & \lambda_0^1 & 0 & 0 & : & rhs_1^1 \\ 0 & \lambda_1^1 - \frac{\overline{\lambda_1^1}}{\overline{\lambda_0^1}}\lambda_0^1 & \overline{\lambda_0^1} & \lambda_0^1 & : & rhs_2^1 - \frac{\overline{\lambda_1^1}}{\overline{\lambda_0^1}}rhs_1^1 \\ 0 & 0 & \overline{\lambda_1^1} & \lambda_1^1 & : & rhs_3^1 \end{bmatrix}.$$

Note that

$$\lambda_1^1 - \frac{\overline{\lambda_1^1}}{\overline{\lambda_0^1}}\lambda_0^1 = \frac{\lambda_1^1 - \lambda_0^1}{\overline{\lambda_0^1}}.$$

Thus, if $\lambda_1^1 \neq \lambda_0^1$, the matrix has a full rank of 3, and the system can be solved iteratively by the least-squares method. On the other hand, if $\lambda_1^1 = \lambda_0^1$, then the system will be

$$10\begin{bmatrix} \overline{\lambda_0^1} & \lambda_0^1 & 0 & 0 \\ 0 & 0 & \overline{\lambda_0^1} & \lambda_0^1 \\ 0 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} \mathbf{p}_2^1 \\ \widetilde{\mathbf{r}_2^1} \\ \mathbf{p}_3^1 \\ \widetilde{\mathbf{r}_3^1} \end{bmatrix} = \begin{bmatrix} rhs_1^1 \\ rhs_2^1 - rhs_1^1 \\ rhs_3^1 - rhs_2^1 + rhs_1^1 \end{bmatrix}.$$

For this system to have a solution, the right-hand side $rhs_3^1 - rhs_2^1 + rhs_1^1$ should be zero:

$$\begin{aligned}
& rhs_3^1 - rhs_2^1 + rhs_1^1 \\
= & -5\left(\overline{\lambda_0^1}\mathbf{p}_4^1 + \lambda_0^1\widetilde{\mathbf{r}_4^1}\right) + 6\left(\overline{\mu_2^1}\mathbf{q}_2^1 + \mu_2^1\mathbf{q}_3^1\right) + 8\left(\overline{\mu_3^1}\mathbf{q}_1^1 + \mu_3^1\mathbf{q}_2^1\right) \\
& + \left(\overline{\mu_4^1}\mathbf{q}_0^1 + \mu_4^1\mathbf{q}_1^1\right) \\
& -4\left(\overline{\mu_1^1}\mathbf{q}_2^1 + \mu_1^1\mathbf{q}_3^1\right) - 12\left(\overline{\mu_2^1}\mathbf{q}_1^1 + \mu_2^1\mathbf{q}_2^1\right) - 4\left(\overline{\mu_3^1}\mathbf{q}_0^1 + \mu_3^1\mathbf{q}_1^1\right) \\
& -5\left(\overline{\lambda_1^1}\mathbf{p}_1^1 + \lambda_1^1\widetilde{\mathbf{r}_1^1}\right) + \left(\overline{\mu_0^1}\mathbf{q}_2^1 + \mu_0^1\mathbf{q}_3^1\right) + 8\left(\overline{\mu_1^1}\mathbf{q}_1^1 + \mu_1^1\mathbf{q}_2^1\right) \\
& +6\left(\overline{\mu_2^1}\mathbf{q}_0^1 + \mu_2^1\mathbf{q}_1^1\right) \\
= & \mathbf{0}.
\end{aligned}$$

When $\lambda_1^1 = \lambda_0^1$, the formulae for $k = 0$ and 1 in Eq.(11) reduce to

$$\begin{aligned}
& 5\left(\overline{\lambda_1^1}\mathbf{p}_1^1 + \lambda_1^1\widetilde{\mathbf{r}_1^1}\right) \\
= & -\left(\overline{\mu_0^1}\mathbf{q}_0^1 + \mu_0^1\mathbf{q}_1^1\right) + 2\left(\overline{\mu_0^1}\mathbf{q}_1^1 + \mu_0^1\mathbf{q}_2^1\right) + 4\left(\overline{\mu_1^1}\mathbf{q}_0^1 + \mu_1^1\mathbf{q}_1^1\right),
\end{aligned}$$

and the formulae for $k = 5$ and 6 reduce to

$$\begin{aligned}
& 5\left(\overline{\lambda_0^1}\mathbf{p}_4^1 + \lambda_0^1\widetilde{\mathbf{r}_4^1}\right) \\
= & -\left(\overline{\mu_4^1}\mathbf{q}_2^1 + \mu_4^1\mathbf{q}_3^1\right) + 4\left(\overline{\mu_3^1}\mathbf{q}_2^1 + \mu_3^1\mathbf{q}_3^1\right) + 2\left(\overline{\mu_4^1}\mathbf{q}_1^1 + \mu_4^1\mathbf{q}_2^1\right).
\end{aligned}$$

Using these two relations, the right-hand side becomes

$$\begin{aligned}
& rhs_3^1 - rhs_2^1 + rhs_1^1 \\
= & \left(\overline{\mu_4^1}\mathbf{q}_2^1 + \mu_4^1\mathbf{q}_3^1\right) - 4\left(\overline{\mu_3^1}\mathbf{q}_2^1 + \mu_3^1\mathbf{q}_3^1\right) - 2\left(\overline{\mu_4^1}\mathbf{q}_1^1 + \mu_4^1\mathbf{q}_2^1\right) \\
& +6\left(\overline{\mu_2^1}\mathbf{q}_2^1 + \mu_2^1\mathbf{q}_3^1\right) + 8\left(\overline{\mu_3^1}\mathbf{q}_1^1 + \mu_3^1\mathbf{q}_2^1\right) + \left(\overline{\mu_4^1}\mathbf{q}_0^1 + \mu_4^1\mathbf{q}_1^1\right) \\
& -4\left(\overline{\mu_1^1}\mathbf{q}_2^1 + \mu_1^1\mathbf{q}_3^1\right) - 12\left(\overline{\mu_2^1}\mathbf{q}_1^1 + \mu_2^1\mathbf{q}_2^1\right) - 4\left(\overline{\mu_3^1}\mathbf{q}_0^1 + \mu_3^1\mathbf{q}_1^1\right) \\
& + \left(\overline{\mu_0^1}\mathbf{q}_0^1 + \mu_0^1\mathbf{q}_1^1\right) - 2\left(\overline{\mu_0^1}\mathbf{q}_1^1 + \mu_0^1\mathbf{q}_2^1\right) - 4\left(\overline{\mu_1^1}\mathbf{q}_0^1 + \mu_1^1\mathbf{q}_1^1\right) \\
& + \left(\overline{\mu_0^1}\mathbf{q}_2^1 + \mu_0^1\mathbf{q}_3^1\right) + 8\left(\overline{\mu_1^1}\mathbf{q}_1^1 + \mu_1^1\mathbf{q}_2^1\right) + 6\left(\overline{\mu_2^1}\mathbf{q}_0^1 + \mu_2^1\mathbf{q}_1^1\right) \\
= & \mathbf{q}_0^1\left(-\mu_0^1 + 4\mu_1^1 - 6\mu_2^1 + 4\mu_3^1 - \mu_4^1\right) \\
& +3\mathbf{q}_1^1\left(\mu_0^1 - 4\mu_1^1 + 6\mu_2^1 - 4\mu_3^1 + \mu_4^1\right) \\
& +3\mathbf{q}_2^1\left(-\mu_0^1 + 4\mu_1^1 - 6\mu_2^1 + 4\mu_3^1 - \mu_4^1\right) \\
& +\mathbf{q}_3^1\left(\mu_0^1 - 4\mu_1^1 + 6\mu_2^1 - 4\mu_3^1 + \mu_4^1\right) \\
= & \left(\mu_0^1 - 4\mu_1^1 + 6\mu_2^1 - 4\mu_3^1 + \mu_4^1\right)\left(-\mathbf{q}_0^1 + 3\mathbf{q}_1^1 - 3\mathbf{q}_2^1 + \mathbf{q}_3^1\right) \\
= & \mathbf{0}.
\end{aligned}$$

Therefore we have

$$\mu_0^1 - 4\mu_1^1 + 6\mu_2^1 - 4\mu_3^1 + \mu_4^1 = 0,$$

or

$$\mu_2^1 = \frac{1}{6}\left(-\mu_0^1 + 4\mu_1^1 + 4\mu_3^1 - \mu_4^1\right).$$

Using the value μ_2^1, we can solve the edge equation.

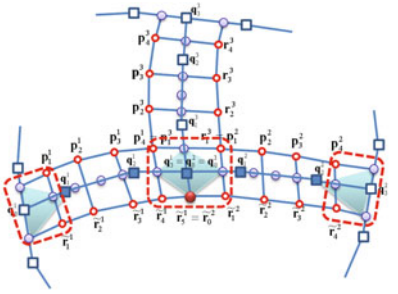
Merging the Subdivided Surfaces. We have now solved the G^1 continuity problem, but we still have the difficulty that S_2 should be a single surface, because we want only three surfaces to meet at the T-junction. So we need to merge the surfaces created by the subdivision. However it turns out to be impossible to merge the modified surfaces with G^1 continuity condition using the reverse de Casteljau algorithm. We will now suggest a series of steps that are capable of merging the subdivided surface into one patch.

Step 1: Solve the vertex condition (Fig. 4(a))
Step 2: Calculate $\mathbf{r}_1$ and $\mathbf{r}_4$ (Fig. 4(b))
Step 3: Calculate $\mathbf{r}_2$ and $\mathbf{r}_3$ using the relation (Fig. 4(c))
Step 4: Calculate the control points for the edge condition using the de Casteljau algorithm (Fig. 4(d))
Step 5: Solve the edge condition for the control points (Fig. 4(e))

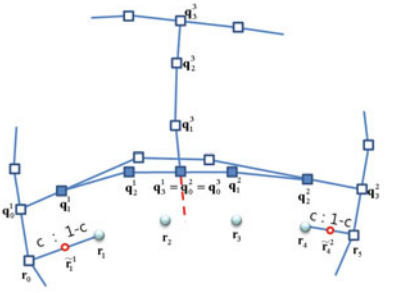
At first step, we can calculate the control points $\mathbf{p}_1^1$, $\widetilde{\mathbf{r}}_1^1$, $\mathbf{p}_4^1$, $\mathbf{p}_1^2$, $\widetilde{\mathbf{r}}_4^1$, $\widetilde{\mathbf{r}}_1^2$, $\mathbf{p}_4^2$, and $\widetilde{\mathbf{r}}_4^2$ from the vertex conditions. At step 2, $\mathbf{r}_1$ and $\mathbf{r}_4$ can be calculated by the de Casteljau algorithm using control points from step 1. At step 3, $\mathbf{r}_2$ and $\mathbf{r}_3$ can be obtained by the relation which is in the Fig. 4(c). The relation of the control points is also from the de Casteljau algorithm. These control points consist of cross boundary curve on S_2. Now we can get the $\widetilde{\mathbf{r}}$ values from cross boundary curves because $\widetilde{\mathbf{r}}$ values are from the subdivided cross boundary curve like step 4. Finally, we can get the $\mathbf{p}_2^1$, $\mathbf{p}_3^1$, $\mathbf{p}_2^2$, and $\mathbf{p}_3^2$ values by solving edge condition using fixed $\widetilde{\mathbf{r}}$ values. Since we have already defined the vertex and edge conditions, we can apply the above steps to the G^1 continuity equation, and thus resolve the G^1 continuity problem at a T-junction without requiring any subdivision process. The other interior control points which do not related with the cross boundary for G^1 continuity can be selected by coons' patch method. In this paper, we use the Coons' patch method to select the control points, but it is possible to use the other surface interpolating method from boundary curves because these points have no effect on the vertex and edge conditions.

3.2 T-Junction at a 3-Valent Vertex

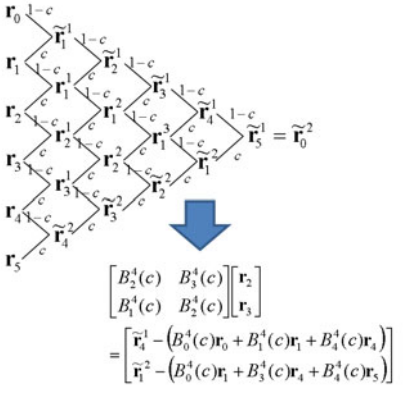
In this case, we also choose an auxiliary control point $\widehat{\mathbf{q}}_2^4$ (see Fig. 5) to allow the G^1 continuity condition to be met on the boundary curve at the T-junction. The $\widehat{\mathbf{q}}_i^4$ are control points on the auxiliary boundary curve with degree 6. We determine $\widehat{\mathbf{q}}_2^4$ in the same way that we selected $\widetilde{\mathbf{r}}_5^1$ in the previous section. In fact $\widehat{\mathbf{q}}_0^4$ and $\widehat{\mathbf{q}}_1^4$ are both the same as $\mathbf{q}_0^1$, because the two edges are collinear.



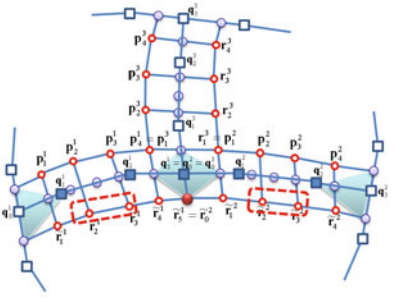
(a) Calculate the vertices values by solving vertex condition



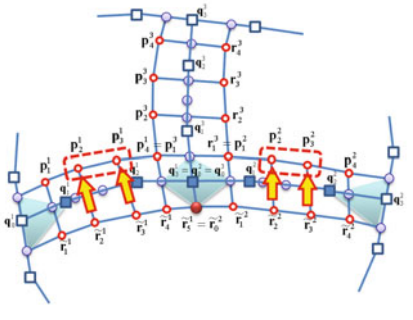
(b) Calculate the values of $\mathbf{r}_1$ and $\mathbf{r}_4$ using the de Casteljau algorithm



(c) Relation of $\mathbf{r}_2$ and $\mathbf{r}_3$ from the de Casteljau algorithm



(d) Calculate $\widetilde{\mathbf{r}}_2^1$, $\widetilde{\mathbf{r}}_3^1$, $\widetilde{\mathbf{r}}_2^2$, and $\widetilde{\mathbf{r}}_3^2$ using the de Casteljau algorithm



(e) Solve the edge condition for $\mathbf{p}_2^1$, $\mathbf{p}_3^1$, $\mathbf{p}_2^2$, and $\mathbf{p}_3^2$

Fig. 4. Sequence of solving the G^1 continuity equations

That means that there is no plane to be satisfied by the G^1 continuity condition because $\mathbf{q}_1^1$, $\mathbf{q}_0^1$, and $\mathbf{q}_1^2$ are collinear. So we cannot apply the G^1 continuity condition at the T-junction. Instead, we use the control point $\widehat{\mathbf{q}}_2^4$ to establish the G^1 continuity condition.

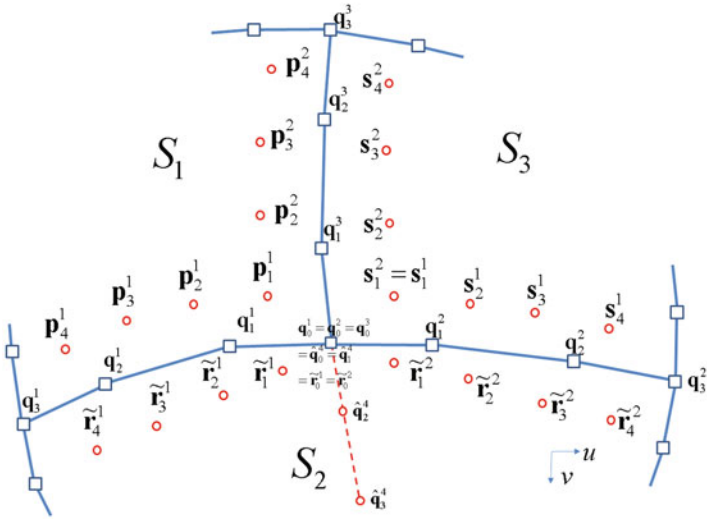


Fig. 5. The auxiliary control point for subdivision at a 3-valent T-junction

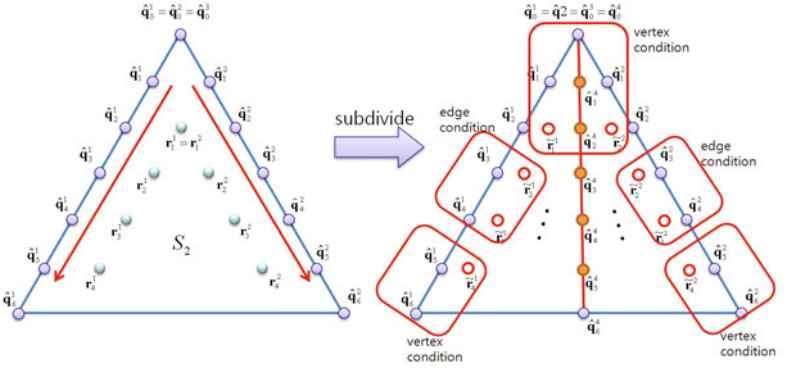


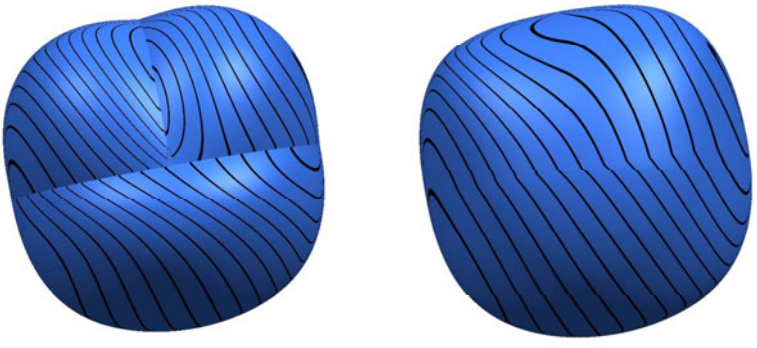
Fig. 6. Subdivision of triangle patch

Then we can apply the same method as before to a T-junction at a vertex. However, there are some differences in this case. We can apply the vertex condition and edge condition using the auxiliary control point like in Fig. 6. When we apply the vertex condition about $\widehat{\mathbf{q}}_0^4$, $\widehat{\mathbf{q}}_2^4$ is needed because $\widehat{\mathbf{q}}_0^4$ and $\widehat{\mathbf{q}}_1^4$ are same we have already mentioned. Also we should consider about the relation of the $\widehat{\mathbf{q}}_2^1$, $\widetilde{\mathbf{r}}_1^1$, $\widehat{\mathbf{q}}_2^4$, $\widetilde{\mathbf{r}}_1^2$, and $\widehat{\mathbf{q}}_2^2$. These control points should satisfy the de Casteljau algorithm with subdivision parameter c. And we do not need to merge the subdivided surface like T-junction on a boundary case, because $\widetilde{\mathbf{r}}_i^1$ $\widetilde{\mathbf{r}}_i^2$ are related with the near boundary control points. For instance, $\widetilde{\mathbf{r}}_2^1$ is a subdivision point of

$\widehat{\mathbf{q}}_3^1$ and $\mathbf{r}_2^1$ with parameter c. So we can solve the edge condition directly without fixing the $\widetilde{\mathbf{r}}_i^1$ and $\widetilde{\mathbf{r}}_i^2$ values. After we determined $\widetilde{\mathbf{r}}_i^1$ and $\widetilde{\mathbf{r}}_i^2$ using the edge condition, we can re-calculate the control points of the original surface using the de Casteljau algorithm.

4 Results

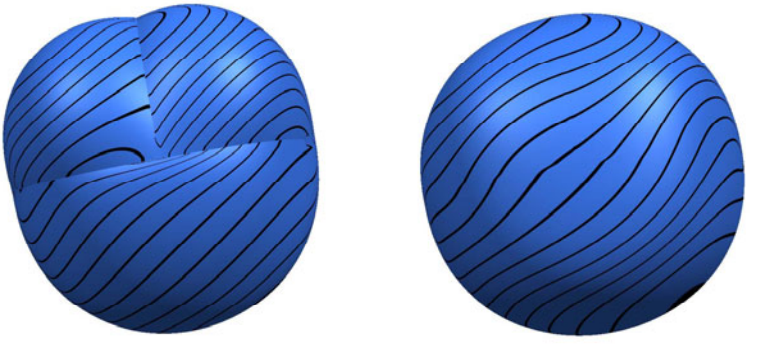
Figs. 7 and 8 show some results produced by our method. We can see the difference between the two examples when we look carefully at the lower surface: in



(a) Reflection lines on three initial surfaces constructed as G^0 Coons' patches and meeting at a T-junction

(b) Reflection lines of a G^1 surface constructed by applying our method to the initial surfaces

Fig. 7. G^1 surface at a T-junction on boundary curves



(a) Reflection lines on three initial surfaces constructed as G^0 Coons' patches and meeting at a 3-valent T-junction

(b) Reflection lines of a G^1 surface constructed by applying our method to the initial surfaces

Fig. 8. G^1 surface with a 3-valent T-junction

Fig. 7(a) it is flatter than it is the Fig. 8(a). We have checked the angles between the adjacent Bézier surfaces.The angles are same within numerical round-off error bounds. Also, we show the reflection lines about the Bézier surfaces. The reflection lines have one degree less continuity than the surfaces, and thus illustrate continuity: where a reflection line has G^0 continuity, the surface has G^1 continuity. Thus the figures show that all the surfaces have G^1 continuity.

5 Conclusions and Future Work

We presented a method of constructing a G^1 Bézier surface at a T-junction. We examined two different types of T-junction and suggested a way of constructing a G^1 surface in each case. We confirmed the functionality of our algorithm on example surfaces using reflection lines to clarify the continuity of the resulting surfaces. This is the first method of generating a G^1 surface at a T-junction that has appeared so far. So we expect it to have wide application in surface modeling processes that begin with a curve network. In future research we intend to look at the generation of surfaces with many T-junctions. We only consider about the surfaces with 'a' T-junction in this paper. We think that the problem can be possible when we extend our concept. Also, we will develop the generation algorithm for surfaces at T-junctions with near-G^2 continuity. We believe that this can be achieved by choosing the auxiliary control point more carefully. Furthermore, we hope that this method may be applicable to T-splines [17] and to polynomial hierarchical t-splines [18].

References

[1] Farin, G.: Curves and surfaces for CAGD: A Practical Guide, 5th edn. Morgan Kaufmann, San Francisco (2002)
[2] Bézier, P.: Essai de définition numérique des courbes et des surfaces expérimentales. PhD thesis, Université Pierre et Marie Curie, Paris (1977)
[3] Farin, G.: A construction for visual C^1 continuity of polynomial surface patches. Computer Graphics and Image Processing 20, 272–282 (1982)
[4] Sarraga, R.: G^1 interpolation of generally unrestricted cubic Bézier curves. Computer Aided Geometric Design 4, 23–40 (1987)
[5] Du, W.H., Schimitt, F.J.M.: On the G^1 continuity of piecewise Bézier surfaces: a review with new results. Computer-Aided Design 22, 556–573 (1990)
[6] Liu, Q., Sun, T.C.: G^1 interpolation of mesh curves. Computer-Aided Design 26(4), 259–267 (1994)
[7] Peters, J.: Local smooth surface interpolation: a classification. Computer Aided Geometric Design 7, 191–195 (1990)
[8] Piper, B.: Visually smooth interpolation with triangular Bézier patches. In: Farin, G. (ed.) Geometric Modeling: Algorithms and New Trends, pp. 221–233. SIAM, Philadelphia (1987)
[9] Shirman, L.A., Séquin, C.H.: Local surface interpolation with Bézier patches. Computer Aided Geometric Design 4, 279–295 (1987)
[10] Loop, C.: A G^1 triangular spline surface of arbitrary topological type. Computer Aided Geometric Design 11, 303–330 (1994)

[11] Hanmann, S., Bonneau, G.P.: Triangular G^1 interpolation by 4-splitting domain triangles. Computer Aided Geometric Design 17, 731–757 (2000)
[12] Liu, Y., Mann, S.: Parametric triangular Bézier surface interpolation with approximate continuity. In: Proceedings of the 2008 ACM Symposium on Solid and Physical Modeling, pp. 381–387 (2008)
[13] Cho, D.Y., Lee, K.Y., Kim, T.W.: Interpolating G^1 Bézier surfaces over irregular curve networks for ship hull design. Computer-Aided Design 38, 641–660 (2006)
[14] Cho, D.Y., Lee, K.Y., Kim, T.W.: Analysis and avoidance of singularities for local G^1 surface interpolation of Bézier curve network with 4-valent nodes. Computing 79, 261–279 (2007)
[15] Tong, W.H., Kim, T.W.: Local and singularity-free G^1 triangular spline surfaces using a minimum degree scheme. Computing 86(2-3), 235–255 (2009)
[16] Tong, W.H., Kim, T.W.: High-order approximation of implicit surfaces by G^1 triangular spline surfaces. Computer-Aided Design 41, 441–455 (2009)
[17] Sederberg, T.W., Zheng, J., Bakenov, A., Nasri, A.: T-spline simplilcation and local refinement. ACM Transactions on Graphics 22(3), 477–484 (2003)
[18] Deng, J., Chen, F., Li, X., Hu, C., Tong, W., Yang, Z., Feng, Y.: Polynomial splines over hierarchical T-meshes. Graphical Models 70(4), 76–86 (2008)

Efficient Point Projection to Freeform Curves and Surfaces

Young-Taek Oh[1], Yong-Joon Kim[1], Jieun Lee[2],
Myung-Soo Kim[1], and Gershon Elber[3]

[1] School of Computer Science and Eng., Seoul National Univ., Seoul 151-744, Korea
[2] School of Computer Science and Eng., Chosun Univ., Kwangju 501-759, Korea
[3] Computer Science Department, Technion, Haifa 32000, Israel

Abstract. We present an efficient algorithm for projecting a given point to its closest point on a family of freeform C^1-continuous curves and surfaces. The algorithm is based on an efficient culling technique that eliminates redundant curves and surfaces which obviously contain no projection from the given point. Based on this scheme, we can reduce the whole computation to considerably smaller subproblems, which are then solved using a numerical method. In several experimental results, we demonstrate the effectiveness of the proposed approach.

1 Introduction

The projection of a point to a set is the closest element of the set to the given point. The point projection problem plays a crucial role in many important geometric computations such as the Hausdorff distance computation, the minimum distance computation, simulation, haptic rendering, tolerance checking, freeform shape fitting and reconstruction, to mention only a few [1,2,3,4,5]. Because of its importance, many previous methods have been developed for the solution of the problem [5,6,7,8,9,10,11,12].

There are many variants of the point projection problem, depending on what to project, where to project, which distance-metric to use, etc. In this paper, we consider a basic type of the point projection problem, where we project a static point $\mathbf{p}$ to a finite family of freeform curves $C_i(t)$, $0 \le t \le 1$, or surfaces $S_i(u, v)$, $0 \le u, v \le 1$, for $i = 1, \ldots, n$. An immediate application of this problem can be found in an interactive selection of the nearest curve C_k to the cursor location $\mathbf{p}$ among all curves on the display screen. For a further manipulation of the selected curve C_k, we may also need to find the parameter t^* of the nearest curve point $C_k(t^*)$.

Translating the point $\mathbf{p}$ and the curves $C_i(t)$ or surfaces $S_i(u, v)$ by $-\mathbf{p}$, we may assume that the query point $\mathbf{p}$ is located at the origin. The problem is then reduced to finding an index k and a curve parameter t^* or surface parameters (u^*, v^*) such that

$$\|C_k(t^*)\| = \min_{1 \le i \le n} \min_{0 \le t \le 1} \|C_i(t)\|, \text{ or } \|S_k(u^*, v^*)\| = \min_{1 \le i \le n} \min_{0 \le u,v \le 1} \|S_i(u, v)\|.$$

B. Mourrain, S. Schaefer, and G. Xu (Eds.): GMP 2010, LNCS 6130, pp. 192–205, 2010.

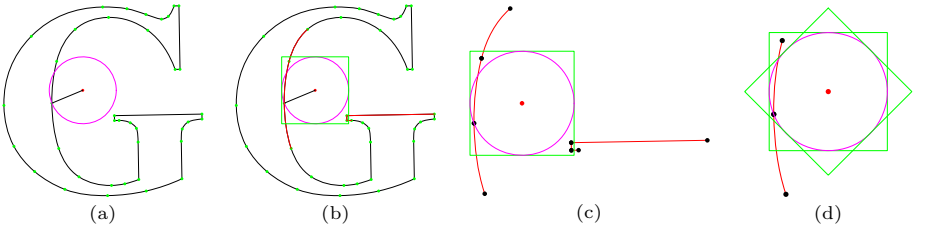


Fig. 1. Efficient culling via an axis-aligned bounding square for a clipping circle: (a) a clipping circle determined by the nearest sample point to the query point, (b) an axis-aligned bounding square for the clipping circle, (c) curves remaining after the culling stage, and (d) curves remaining after an additional culling with a rotated square

Conventional approaches attack this problem for curves by solving the footpoint constraint: $\langle C_i(t), C_i'(t)\rangle = 0$, and then comparing the distances to all solution curve points of this equation as well as to the curve end points. For surfaces, the footpoint constraint is a system of equations: $\left\langle S_i(u,v), \frac{\partial S_i}{\partial u}(u,v)\right\rangle = 0$ and $\left\langle S_i(u,v), \frac{\partial S_i}{\partial v}(u,v)\right\rangle = 0$. The distances to all solution surface points are then compared with those to the four boundary curves of the surface. (See Johnson [5] for an extensive literature survey on conventional methods.) Since the majority of these solutions turn out to be redundant, it is important to reduce the expensive equation solving as much as possible. Recent results propose efficient culling or clipping techniques for this purpose.

Chen et al. [6] proposed a circle/sphere clipping method that applies a certain Bézier clipping technique to the squared distance function of a NURBS curve. This is an improvement over Selimovic [8], which is based on a *Voronoi cell* test. The circle/sphere clipping demonstrates a better result. Nevertheless, as discussed in more details in Appendix A, it is quite expensive to set up the squared distance function which has degree almost twice higher than the original curve or surface degree(s). Moreover, it is also time-consuming to subdivide the squared distance function, in particular for the surface case.

In this paper, we propose an approach that is based on the clipping circle/sphere, but only conceptually; for efficiency reason, we instead use simple clipping lines/planes tangent to the circle/sphere. The simplest lines/planes for our purpose are those orthogonal to the coordinates axes. For example, given a clipping circle of radius r centered at the origin, we may consider the clipping lines $x = \pm r$ and $y = \pm r$. Figure 1 shows an example where many input curves can easily be eliminated from further consideration using a simple window clipping. When a Bézier curve has control points (x_i, y_i) with $x_i > r$ for all i (or $x_i < -r$ for all i), the distance to this curve must be larger than r and can safely be eliminated. Similarly, we can check the y-coordinates of the control points. We may then proceed to checking $x_i + y_i > \sqrt{2}\, r$ for all i, etc (Figure 1(d)).

In some sense, our approach employs the k-DOP structure that bounds the control points of the freeform curves or surfaces [13]. Though less tight than the convex hull of the control points, the k-DOP is considerably easier to construct; moreover, it is much tighter than the bounding circle/sphere of the control points. In our application, the k-DOP is constructed incrementally, which is essential for improving the overall performance of our algorithm. (According to our experiments, the first couple of separation tests usually cull away a large number of freeform curves and surfaces; thus the construction of a complete k-DOP is rarely needed.)

The real advantage of Chen et al. [6] is in checking the uniqueness of local minimum of the squared distance function, which can be tested by the same condition for the control coefficients of the function. Nevertheless, it is difficult to extend the uniqueness condition of Chen et al. [6] to the surface case. Consequently, Chen et al. [11] propose no property for testing the uniqueness of local minimum distance between two freeform curves. A simple remedy for this deficiency is to employ the condition of Elber and Kim [12] (Theorem 1) for testing the uniqueness of solution for a system of polynomial equations in a restricted domain.

Chen et al. [6] pointed out that there may be redundant solutions in the system of equations that Elber and Kim [12] solve. However, the chance of getting redundancy is extremely low in our approach. As the result of applying an efficient culling and clipping procedure, the remaining freeform curves and surfaces are usually defined on small domains. Consequently, the tests for the uniqueness of solution are mostly successful and no further expensive subdivisions are needed in the majority of case. Once the uniqueness is guaranteed, the equations can be solved efficiently using a numerical method.

The main contribution of this work can be summarized as follows:

- An efficient culling or clipping technique is proposed that can significantly reduce the problem size, i.e., the number of freeform curves and surfaces and also the size of their remaining subsegments and subpatches.
- Our technique is based on geometric concepts such as the separating axis and the k-DOP bounding volume [13], though they are only incrementally constructed on-the-fly as needed for freeform curves and surfaces; thus our basic approach can easily be extended to more general projection problems or combined with other geometric computations based on these useful concepts.
- For the point-to-surface projection problem, employing the condition of Elber and Kim [12] (Theorem 1), we can test the uniqueness of solution for a system of polynomial equations. This approach can also be extended to more general multivariate projection problems.

The rest of this paper is organized as follows. Section 2 presents a brief review on related previous work and also the basic idea of our approach. In Section 3, the details of our algorithm are presented. In Section 4, we demonstrate the effectiveness of our approach using several experimental results. Finally, in Section 5, we conclude this paper with discussions on future work.

2 Related Work and Our Basic Idea

In his PhD Thesis, Johnson [5] presented an extensive literature survey on the minimum distance computation and the associated applications in virtual prototyping and haptic rendering. He made an interesting observation that conventional methods for polygonal models mainly considered efficient pruning techniques; on the other hand, those for parametric surfaces developed numerical techniques for the minimum distance computation. Johnson [5] then proposed a pruning technique for freeform curves and surfaces.

The method of Johnson [5] starts with taking some samples on the freeform curves or surfaces and consider a circle/sphere centered at the query point and containing the nearest point among the samples. After that, the algorithm then prunes away curves and surfaces when the convex hulls of their control points have no overlap with the current clipping circle/sphere. To do the overlap test, the distance is computed from the query point to each convex hull, which is the main computational bottleneck of the algorithm.

In this paper, we instead use separation axis tests with simple directions such as the coordinate directions and other combinations of them. This approach is essentially the same as incrementally constructing a k-DOP bounding volume for the set of all control points.

There are some previous methods for testing a sufficient condition for the point-projection to a curve end point or to a surface corner point. We start with discussing the method of Ma and Hewitt [7]. (Selimovic [8] does the same thing but more efficiently; nevertheless, it is easier to visualize the condition of Ma and Hewitt [7].) Figure 2(a) shows a region A where each point $\mathbf{p}$ is guaranteed to be projected to the left end point $\mathbf{b}_0$ of a cubic Bézier curve. Consider the Voronoi cell decomposition for the convex hull of the Bézier control points. Then the region A is in fact the Voronoi cell for the end point $\mathbf{b}_0$ in the exterior of the convex hull; consequently, all points in the region A are closer to the end point $\mathbf{b}_0$ than to any other points in the convex hull including all the curve points. For higher degree curves, the convex hull computation can be somewhat intricate, in particular, for space curves. The same sufficient condition can be formulated for the surface case as well; however, the convex hull computation becomes even more cumbersome for the control points of a freeform surface.

Selimovic [8] tests the same sufficient condition more efficiently. When a point $\mathbf{p}$ is in the Voronoi cell, it is on an outward normal line of the convex hull from the end point $\mathbf{b}_0$. Now consider a line/plane (passing through the end point $\mathbf{b}_0$) which is orthogonal to the normal line and divides the whole space into two half-spaces. All the Bézier control points are then contained in a half-space opposite to that of the query point $\mathbf{p}$. Selimovic [8] checks this condition by testing $\langle \mathbf{b}_0 - \mathbf{p}, \mathbf{b}_i - \mathbf{b}_0 \rangle > 0$, for $i = 1, \ldots, d$. Since these sign tests are considerably easier than the construction of a convex hull and the Voronoi cell of a vertex, Selimovic [8] is more efficient than Ma and Hewitt [7].

Now the next question is whether the Voronoi cell concept is indeed a good idea for the point-projection problem. Figure 2(b) shows a region B which may immediately lead us to a negative answer to this question. In this specific example

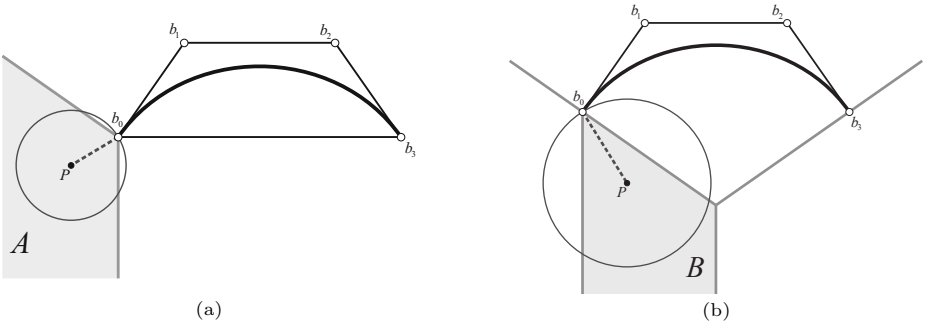


Fig. 2. (a) A region where each point is guaranteed to be projected to the left end point of the curve. (b) A similar region of points to be projected to $\mathbf{b}_0$, but which cannot be detected as such by the Voronoi cell test of the previous methods of Ma and Hewitt [7] or Selimovic [8].

of a symmetric cubic Bézier curve, all query points $\mathbf{p}$ in both regions A and B of Figures 2 will be projected to the curve end point $\mathbf{b}_0$. Thus we need a better condition than the Voronoi cell test [7,8].

Geometrically speaking, the circle/sphere clipping of Chen et al. [6] is optimal in the sense that, if a query point $\mathbf{p}$ has its projection to the end point $\mathbf{b}_0$, the circle with center $\mathbf{p}$ and radius $\|\mathbf{b}_0 - \mathbf{p}\|$ contains no other curve point in its interior. But the problem is how to test this condition efficiently. Chen et al. [6] employed the squared distance function; however, as discussed in Appendix A, it is quite expensive to set up the squared distance function, in particular, for freeform surfaces and rational curves of high degree. Thus we propose more efficient geometric tests than the circle/sphere test.

Figure 3(a) shows a clipping circle centered at $\mathbf{p}$ and a tangent line of the circle with normal $(1, 1)$. The Bézier clipping technique [14,15] applied to this line can remove some part of the curve as shown in Figure 3(b). The remaining

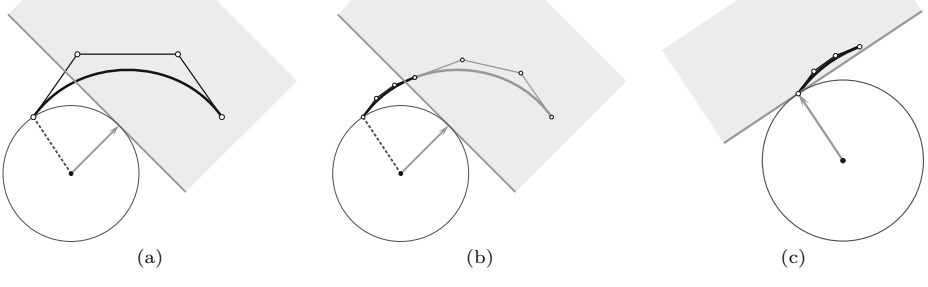


Fig. 3. Efficient curve clipping using tangent lines to the clipping circle: (a) a clipping line with normal $(1, 1)$, (b) a segment of the curve approximately clipped, and (c) the left end point is shown to be the closest point by the Voronoi cell condition of Selimovic [8]

curve segment can be tested by the Voronoi cell condition of Selimovic [8], which will guarantee the projection of $\mathbf{p}$ to $\mathbf{b}_0$ (Figure 3(c)).

The problem becomes more difficult when the point $\mathbf{p}$ is located in the concave area of a curve and the projection occurs in the curve interior. The squared distance function of Chen et al. [6] plays an important role here as more details to be discussed in the following section.

3 Our Approach

Our approach starts with sampling the given family of curves and surfaces, and making an initial guess of the minimum distance and the associated clipping circle/sphere. Simple tangent lines/planes are then considered for culling or clipping away redundant curves or surfaces. To the remaining curves and surfaces, geometric tests are applied to detect special cases of point-projection: (i) to a curve end point or a surface corner point, (ii) to a surface boundary curve, where the problem is reduced to a point-to-curve projection, or (iii) to a unique interior point of a curve segment or a surface patch, where a numerical method can be applied. Otherwise, the freeform curves and surfaces are further subdivided and the whole procedure is repeated recursively.

3.1 Clipping Circle/Sphere and Clipping Lines/Planes

Given a query point $\mathbf{p}$ and its nearest point $\mathbf{p}_k$ among all sample points $\mathbf{p}_i$, $i = 1, \ldots, n$, we consider the circle/sphere with center $\mathbf{p}$ and radius $r = \|\mathbf{p}_k - \mathbf{p}\|$ that will be used for culling or clipping redundancies from the given freeform curves or surfaces. To make the whole algorithm efficient, we should take only a suitable number of sample points that would be sufficient to give a good initial guess on the minimum distance. When there are many curve segments or surface patches to consider, we take only the curve end points and the surface corner points into account at the beginning. As we converge to a small number of freeform curves and surfaces, we may take more samples in their interior.

For the sake of simplicity of presentation, we consider the curve case in detail; however, a similar technique can be applied to surfaces as well. Given a curve $C(t)$, the clipping based on a squared distance: $\|C(t) - \mathbf{p}\|^2 > r^2$, is quite expensive as discussed in Appendix A. Thus we instead do an approximate clipping with a few simple tangent lines to the clipping circle: $ax + by + c > 0$, where $c^2 = (a^2 + b^2)r^2$ and (a, b) is an outward normal direction of the tangent line. When all the control points $\mathbf{b}_i = (x_i, y_i)$, $i = 0, \ldots, d$, satisfy the following condition:

$$ax_i + by_i + c > 0, \text{ for } i = 0, \ldots, d,$$

we can guarantee that the whole curve $C(t)$ is outside the clipping circle as well as the clipping line and thus contains no projection from the query point $\mathbf{p}$. However, this approach requires $2(d+1)$ multiplications, $2(d+1)$ additions, and $(d+1)$ sign tests. We can do better. Following the approach of k-DOP [13] , we consider simple normal directions such as $(\pm 1, 0), (0, \pm 1), (\pm 1, \pm 1)$, etc. Using

these direction vectors, many multiplications in the culling/clipping tests can be replaced by additions and subtractions of x_i's and y_i's. In the majority of case, only one or two of these directions would be needed to cull away redundancies.

3.2 Uniqueness of Solution

After the culling and clipping stage, we will end up with a relatively small number of freeform curve segments or surface patches. For each of these remaining curves and surfaces, we compute the nearest point from the query point $\mathbf{p}$ and dynamically update the minimum distance and the clipping circle/sphere when a closer projection point is found than the current one.

We consider a Bézier curve $C(t)$ of degree d defined by $(d+1)$ control points $\mathbf{b}_i$, for $i = 0, \ldots, d$. If $\mathbf{p}$ is closer to $\mathbf{b}_0$ than to $\mathbf{b}_d$, we test the projection of $\mathbf{p}$ to the end point $\mathbf{b}_0$ using the Voronoi cell condition of Selimovic [8]:

$$\langle \mathbf{b}_0 - \mathbf{p}, \mathbf{b}_i - \mathbf{b}_0 \rangle > 0, \text{ for } i = 1, \ldots, d.$$

If the query point $\mathbf{p}$ is closer to $\mathbf{b}_d$ than to $\mathbf{b}_0$, the projection of $\mathbf{p}$ to the other end point $\mathbf{b}_d$ is tested as follows:

$$\langle \mathbf{b}_d - \mathbf{p}, \mathbf{b}_i - \mathbf{b}_d \rangle > 0, \text{ for } i = 0, \ldots, d-1.$$

Even if the above Voronoi cell condition is not met, we cannot completely exclude the possibility of projection to $\mathbf{b}_0$ or $\mathbf{b}_d$ (as shown by the region B of Figure 2(b)); however, the chance is quite low since the region B is very small for a short curve segment we are dealing with after the culling and clipping stage. Thus we employ the squared distance function: $\|C(t) - \mathbf{p}\|^2$ of Chen et al. [6]. When the Bézier control coefficients f_i, $i = 0, \ldots, 2d$, of this function have only one local minimum, the function graph will have a U-shape with only one local minimum, which is thus the global minimum. (The exact curve location for the minimum distance can be computed by a numerical method.) Otherwise, the squared distance function is subdivided into two and each subproblem is tested recursively.

For a Bézier surface $S(u, v)$ of degree (d_1, d_2) defined by $(d_1+1)(d_2+1)$ control points $\mathbf{b}_{ij}$, for $i = 0, \ldots, d_1$ and $j = 0, \ldots, d_2$, we may assume that $\mathbf{p}$ is closer to $\mathbf{b}_{00}$ than to three other corner points. We can then test the projection of $\mathbf{p}$ to the surface corner point $\mathbf{b}_{00}$ using the condition of Selimovic [8]:

$$\langle \mathbf{b}_{00} - \mathbf{p}, \mathbf{b}_{ij} - \mathbf{b}_{00} \rangle > 0, \text{ for } i = 0, \ldots, d_1;\ j = 0, \ldots, d_2;\ i + j > 0.$$

Selimovic [8] also presents a condition for checking the projection of $\mathbf{p}$ to a boundary curve $S(u, 0)$:

$$\left\langle \mathbf{b}_{i0} - \mathbf{p}, \frac{\partial S}{\partial v}(u, v) \right\rangle > 0, \text{ for } i = 0, \ldots, d_1, \text{and } 0 \le u, v \le 1.$$

The projection to other boundary curves can be tested similarly.

However, the above condition is difficult to check since the set of all v-partial derivatives forms another freeform surface of degree $(d_1, d_2 - 1)$. From the relation: $\frac{\partial S}{\partial v}(u,v) = \sum_{l=0}^{d_1} \sum_{j=0}^{d_2-1} (\mathbf{b}_{l,j+1} - \mathbf{b}_{l,j}) B_l^{d_1}(u) B_j^{d_2-1}(v)$, we suggest a simpler sufficient condition for the projection of $\mathbf{p}$ to the boundary curve $S(u,0)$:

$$\langle \mathbf{b}_{i,0} - \mathbf{p}, \mathbf{b}_{l,j+1} - \mathbf{b}_{l,j} \rangle > 0, \quad \text{for all } i,l = 0,\ldots,d_1;\ j = 0,\ldots,d_2 - 1.$$

Now when the above conditions are not met, the minimum distance may occur in the surface interior (even though we cannot exclude some possibility of getting the minimum distance on the boundary curve or even at a corner point). Using the condition of Elber and Kim [12] (Theorem 1), we consider how to test the uniqueness of local minimum distance on the interior of the surface $S(u,v)$: $0 < u, v < 1$. At each local minimum distance, the solution point $S(u,v)$ must satisfy the following two bivariate equations:

$$F(u,v) = \left\langle S(u,v), \frac{\partial S}{\partial u}(u,v) \right\rangle = 0,\ G(u,v) = \left\langle S(u,v), \frac{\partial S}{\partial v}(u,v) \right\rangle = 0.$$

When there is a unique solution for this system of equations in the uv-domain: $0 < u, v < 1$, this solution may correspond to the minimum distance from the query point $\mathbf{p}$.

The above system of bivariate equations has at most one solution if the following condition is met [12]:

$$\{\alpha \nabla F(u,v) \mid \alpha \in R, 0 < u, v < 1\} \cap \{\beta \nabla G(u,v) \mid \beta \in R, 0 < u, v < 1\} = \{\mathbf{0}\}.$$

3.3 Numerical Improvement

For the curve case, we compute the minimum of the squared distance function $D^2(t) = \sum_{i=0}^{2d} f_i B_i^{2d}(t)$, by solving the unique solution of the following derivative equation:

$$\sum_{i=0}^{2d-1} (f_{i+1} - f_i) B_i^{2d-1}(t) = 0.$$

We employ Brent's method for the sake of robustness in solving the above equation [16]. Note that the uniqueness of solution is guaranteed by the condition of Chen et al. [6].

For the surface case, we employ a bivariate Newton-Raphson method as discussed in Elber and Kim [12] and implemented in the IRIT solid modeling system [17].

4 Experimental Results

We have implemented our point-projection algorithm in C on an Intel Pentium IV 2.4GHz PC with a 2GB main memory. To demonstrate the effectiveness of our approach, we have tested the algorithm for several freeform curves and surfaces.

Figure 4(a) shows a circle composed of four circular arcs of the same length. The circle itself is represented as a rational quadratic B-spline curve. The four end points of these component arcs are used as samples for estimating the initial guess on the minimum distance. To compare the performance of different algorithms, we randomly generate 100 query points within a box that is 50% larger in each dimension than the axis-aligned minimum bounding square of the circle. Table 1(a) shows the result of measuring the performance of different algorithms on these query points averaged over 100 independent tests.

As the first step of the experiment, we start with converting the B-spline representation of the circle to four rational quadratic Bézier curves, which takes approximately 11.6μs on average. The conversion time is not included in Table 1(a), since it is common in all four algorithms under comparison. For each random query point $\mathbf{p}$, the distances to the four curve end points are compared and the minimum is taken as the radius of an initial clipping circle centered at the query point $\mathbf{p}$, which takes approximately 0.9μs on average. This part is also common and thus not included in the performance measure.

Figure 4(b) shows a character font G that is designed with a non-uniform quadratic B-spline curve which can be converted to 38 quadratic Bézier curves. The conversion takes approximately 285μs. Figure 4(c) shows a uniform cubic B-spline space curve which can be converted to 9 cubic Bézier space curves, approximately in 37μs.

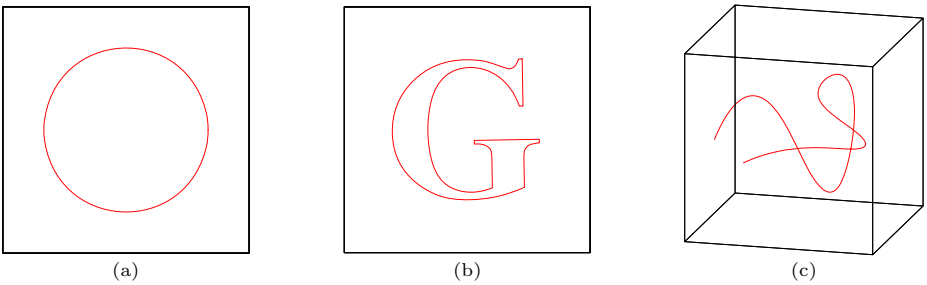


Fig. 4. (a) A rational quadratic B-spline circle, (b) a character G with a non-uniform quadratic B-spline curve, and (c) a uniform cubic B-spline space curve

In Table 1, the column under D^2 ONLY shows the result of Chen et al. [6] applied to the Bézier curves. The column under CIRCLE+D^2 corresponds to an algorithm that first culls away some redundant Bézier curves when their bounding circles have no overlap with the initial clipping circle. (The bounding circle of a Bézier curve has its center at the center of mass of the Bézier control points and its radius is taken to be the maximum distance from the center to the control points.) After that, the algorithm employs Chen et al. [6] for the remaining Bézier curves. The next column under KDOP+D^2 shows the result of applying our algorithm proposed in this paper; namely, we apply a k-DOP based culling to the Bézier curves and employ Chen et al. [6] for the remaining

Table 1. Results for (a) a rational B-spline circle (Fig. 4(a)), (b) a non-uniform B-spline character (Fig. 4(b)), and (c) a uniform B-spline space curve (Fig. 4(c))

	D^2 ONLY	CIRCLE+D^2	KDOP+D^2	K+SELIM+D^2
Cull/Clip	0.000	16.110	16.034	16.767
D^2+Subdiv	118.666	58.282	45.191	44.518
Numeric	9.482	5.834	5.106	5.104
Total (in μs)	128.148	80.227	66.331	66.389
# curve	4.000	2.170	1.690	1.650
# subdiv	0.000	0.000	0.000	0.000
# num iter	7.920	4.640	4.040	4.040

(a)

	D^2 ONLY	CIRCLE+D^2	KDOP+D^2	K+SELIM+D^2
Cull/Clip	0.000	111.623	112.147	112.607
D^2+Subdiv	553.575	37.542	27.580	23.069
Numeric	20.079	6.856	5.637	5.707
Total (in μs)	573.655	156.022	145.364	141.384
# curve	38.000	2.690	1.980	1.580
# subdiv	0.010	0.000	0.000	0.000
# num iter	20.480	5.260	4.140	4.140

(b)

	D^2 ONLY	SPHERE+D^2	KDOP+D^2	K+SELIM+D^2
Cull/Clip	0.000	30.687	30.913	31.324
D^2+Subdiv	145.569	36.451	23.657	22.148
Numeric	19.949	8.167	5.673	5.582
Total (in μs)	165.518	75.306	60.243	59.054
# curve	9.000	2.630	1.700	1.570
# subdiv	0.610	0.250	0.120	0.120
# num iter	24.400	9.030	6.070	5.920

(c)

Bézier curves. For the k-DOP based tests, we have employed only a subset of 8 directions: $(\pm 1, 0), (0, \pm 1), (\pm 1, \pm 1)$ for planar curves, and a subset of 14 directions: $(\pm 1, 0, 0), (0, \pm 1, 0), (0, 0, \pm 1), (\pm 1, \pm 1, \pm 1)$ for space curves and surfaces. The last column under KDOP+SELIM+D^2 reports the result from a variant of our algorithm where we apply Selimovic [8] immediately after the k-DOP based culling and right before we employ Chen et al. [6].

The first four rows in each of Table 1(a)–(c) show the computing time for each algorithm in each step of the computation, measured in micro seconds (μs). The first row reports the time taken in culling or clipping redundant Bézier curves. The second row shows the computing time for setting up the squared distance functions for all remaining Bézier curves and checking the uniqueness of local minimum for each function and recursively subdividing those with potentially multiple local minimums. The third row is for the stage of numerical improvement. The total computing time is shown in the fourth row.

The last three rows in each of Table 1(a)–(c) show the average number of Bézier curves remaining after the initial culling, the number of subdivisions taken in all the squared distance functions $D^2(t)$ until the uniqueness of local minimum is guaranteed for each curve segment, and the total number of Brent iterations for all subdivided curves in the numerical improvement stage. The numerical approximation is made within a precision of 10^{-6} in the Bézier curve parameter. All the data in each table show the average of results over 100 independent tests on randomly selected query points.

Note that, in the first column D^2 ONLY of Table 1(b), the number of Brent iterations is smaller than the number of remaining curve segments. This is because, for some curve segments, the control coefficients of the squared function $D^2(t)$ are all larger than the current minimum squared distance and thus no numerical iteration is needed for these redundant curves.

In Table 1, we can observe that the computing times for culling redundant curve segments are about the same in the two different methods: CIRCLE+D^2 and KDOP+D^2. However, the latter is more effective in the culling result itself, since the k-DOP bounding volume is usually tighter than the circle/sphere bounding volume. In Table 1(b)–(c), some performance improvement is achieved by employing the end-point projection condition of Selimovic [8] as reported in the last column under KDOP+SELIM+D^2, though it is not obvious in Table 1(a) because of the relatively larger size of the Bézier curve segments than those in other examples.

Figure 5(a) shows the spout of the Utah teapot represented as a non-uniform bicubic B-spline surface which can be converted to 4 bicubic Bézier surfaces. Figure 5(b) shows the B-spline surfaces for the whole Utah teapot which can be converted to 28 bicubic Bézier surfaces. In the surface case, after culling away redundant Bézier surfaces, we compute the local minimum distance to each of the remaining Bézier surfaces by employing the bivariate equation solver of Elber and Kim [12] as implemented in the IRIT solid modeling system [17].

In Table 2, the first row shows the conversion time from the B-spline representation to bicubic Bézier surfaces, the second row shows the culling time, and the third row shows the time taken in the solution procedure by the IRIT

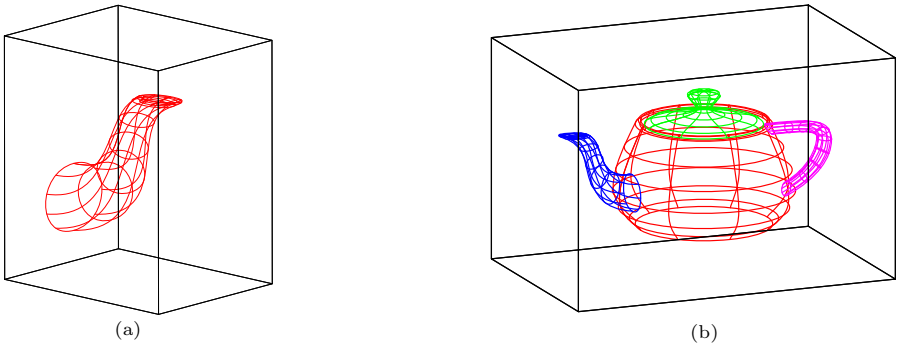


Fig. 5. The B-spline surfaces for: (a) the spout and (b) the Utah teapot

Table 2. Results for: (a) the spout (Fig. 5(a)) and (b) the Utah teapot (Fig. 5(b))

	SPHERE+D^2	KDOP+D^2
Conversion	43.023	43.064
Culling	9.897	7.516
IRIT	1186.613	1005.708
Total (in μs)	1196.510	1013.224
# surface	2.220	1.890

(a)

	SPHERE+D^2	KDOP+D^2
Conversion	438.731	434.625
Culling	35.287	25.906
IRIT	1690.218	1228.548
Total (in μs)	1725.505	1254.455
# surface	5.420	3.810

(b)

solver. Because of the large number of control points for the surface case, we can observe that the sphere method takes more culling time than the k-DOP method. Similarly to the curve case, for the culling result itself, the k-DOP is also more effective than the sphere.

5 Conclusions

We have presented an efficient point-to-curve/surface projection algorithm that computes the nearest point on a family of freeform curves and surfaces from a given query point. Effectively using only a small number (usually one or two) of separation axes among the k-DOP directions, we have developed a culling method as efficient as the circle/sphere method for the curve case and even more effective than the sphere for the surface case. Tighter than circles/spheres, our approach produces better culling results and consequently better performance for both curve and surface cases.

In future work, we plan to extend the current result to the case of projecting a dynamically moving point to freeform curves and surfaces. In this more general case, we may need an effective pre-processing of the freeform shapes so that we can fully utilize their geometric structure in the main computation of the projection problem. Furthermore, we hope our approach could be extended to the more general distance problems dealing with freeform shapes under continuous deformation.

Acknowledgment

This research was supported in part by the Israeli Ministry of Science Grant No. 3-4642, in part by the Israel Science Foundation (grant No. 346/07), in part by KICOS through the Korean-Israeli Binational Research Grant (K20717000006) provided by MEST in 2007, in part by the Korea Research Foundation under the Grant KRF-2008-313-D00923, and also in part by NRF Research Grant (No. 2009-0075116) provided by MEST in 2009. Y.-T. Oh was supported by the Seoul Fellowship.

References

1. Lin, M.C., Gottschalk, S.: Collision detection between geometric models: A survey. In: Proc. of IMA Conference on Mathematics of Surfaces, pp. 37–56 (1998)
2. Lin, M.C., Manocha, D.: Collision and proximity queries. In: Goodman, J.E., O'Rourke, J. (eds.) Handbook of Discrete and Computational Geometry, 2nd edn., pp. 787–807. Chapman & Hall/CRC (2004)
3. Gilbert, E., Johnson, D., Keerthi, S.: A fast procedure for computing the distance between complex objects in three-dimensional space. IEEE Trans. Robot. Automat. 4, 193–203 (1988)
4. Lin, M.C., Canny, J.: A fast algorithm for incremental distance calculation. In: IEEE Int. Conf. Robot. Automat., Sacramento, CA, April 1991, pp. 1008–1014 (1991)
5. Johnson, D.: Minimum distance queries for haptic rendering. PhD thesis, Computer Science Department, University of Utah (2005)
6. Chen, X.-D., Yong, J.-H., Wang, G., Paul, J.-C., Xu, G.: Computing minimum distance between a point and a NURBS curve. Computer-Aided Design 40(10-11), 1051–1054 (2008)
7. Ma, Y.L., Hewitt, W.: Point inversion and projection for NURBS curve and surface: control polygon approach. Computer Aided Geometric Design 20(2), 79–99 (2003)
8. Selimovic, I.: Improved algorithms for the projection of points on NURBS curves and surfaces. Computer Aided Geometric Design 23(5), 439–445 (2006)
9. Hu, S.-M., Wallner, J.: A second order algorithm for orthogonal projection onto curves and surfaces. Computer Aided Geometric Design 22(3), 251–260 (2005)
10. Liu, X.-M., Yang, L., Yong, J.-H., Gu, H.-J., Sun, J.-G.: A torus patch approximation approach for point projection on surfaces. Computer Aided Geometric Design 26(5), 593–598 (2009)
11. Chen, X.-D., Chen, L., Wang, Y., Xu, G., Yong, J.-H.: Computing the minimum distance between Bezier curves. Journal of Computational and Applied Mathematics 230(1), 294–310 (2009)
12. Elber, G., Kim, M.-S.: Geometric constraint solver using multivariate rational spline functions. In: Proc. of the Sixth ACM Symposium on Solid Modeling and Applications, pp. 1–10 (2001)
13. Klosowski, J., Held, M., Mitchell, J., Sowizral, H., Zikan, K.: Efficient collision detection using bounding volume hierarchies of k-dops. IEEE Trans. on Visualization and Computer Graphics 4(1), 21–37 (1998)
14. Sederberg, T.W., Nishita, T.: Curve intersection using Bézier clipping. Computer-Aided Design 22(9), 337–345 (1990)
15. Nishita, T., Sederberg, T.W., Kakimoto, M.: Ray tracing trimmed rational surface patches. Computer Graphics 24(4), 337–345 (1990)
16. Press, W., Teukolsky, S., Vetterling, W., Flannery, B.: Numerical Recipes: The Art of Scientific Computing, 3rd edn. Cambridge University Press, Cambridge (2007)
17. IRIT 9.5 User's Manual, Technion, http://www.cs.technion.ac.il/~irit

A Operation Counts for Squared Distance Functions

A.1 Squared Distance Functions for Cubic Bézier Curves

Given a cubic planar Bézier curve $C(t) = (x(t), y(t))$, $0 \leq t \leq 1$, with four control points $\mathbf{b}_i = (x_i, y_i)$, for $i = 0, 1, 2, 3$, the x-coordinate function is given

as $x(t) = (1-t)^3[x_0] + 3(1-t)^2t[x_1] + 3(1-t)t^2[x_2] + t^3[x_3]$, and its squared function is a Bézier function of degree 6:

$$\begin{aligned} x(t)^2 = {} & (1-t)^6[x_0^2] + 6(1-t)^5t[x_0x_1] + 15(1-t)^4t^2[0.4x_0x_2 + 0.6x_1^2] \\ & +20(1-t)^3t^3[0.1x_0x_3 + 0.9x_1x_2] + 15(1-t)^2t^4[0.4x_1x_3 + 0.6x_2^2] \\ & +6(1-t)t^5[x_2x_3] + t^6[x_3^2]. \end{aligned}$$

Note that the seven control coefficients can be computed using 16 multiplications and 3 additions. Similarly, the squared distance function for the curve is given as follows:

$$\begin{aligned} \|C(t)\|^2 &= x(t)^2 + y(t)^2 \\ &= (1-t)^6[x_0^2 + y_0^2] + 6(1-t)^5t[x_0x_1 + y_0y_1] \\ &\quad +15(1-t)^4t^2[0.4(x_0x_2 + y_0y_2) + 0.6(x_1^2 + y_1^2)] \\ &\quad +20(1-t)^3t^3[0.1(x_0x_3 + y_0y_3) + 0.9(x_1x_2 + y_1y_2)] \\ &\quad +15(1-t)^2t^4[0.4(x_1x_3 + y_1y_3) + 0.6(x_2^2 + y_2^2)] \\ &\quad +6(1-t)t^5[x_2x_3 + y_2y_3] + t^6[x_3^2 + y_3^2], \end{aligned}$$

which can be constructed as a Bézier polynomial function $D^2(t)$ using 26 multiplications and 13 additions.

Now the squared distance function for a cubic space Bézier curve can be computed using 36 multiplications and 23 additions. For a cubic rational planar Bézier curve $C(t) = (X(t), Y(t), W(t))$, represented in a homogeneous coordinate, its squared distance function $D^2(t) = (X(t)^2 + Y(t)^2)/W(t)^2$ is a rational Bézier function of degree 6, which can be constructed using 47 multiplications, 16 additions, and 7 divisions. (The additional 7 divisions are needed to get the control coefficients of $D^2(t)$ by dividing each control coefficient of $X(t)^2 + Y(t)^2$ by the corresponding coefficient of $W(t)^2$.) Similarly, for a cubic rational space Bézier curve, it requires 57 multiplications, 26 additions, and 7 divisions.

A.2 Differential of Squared Distance Functions

The local extremes for the squared distance function $D^2(t) = \|C(t)\|^2$ can be computed by solving the constraint equation: $< C(t), C'(t) >= 0$. At first, a direct multiplication of $C(t)$ and $C'(t)$ may look a reasonable approach to constructing the Bézier representation of $< C(t), C'(t) >$. However, because of the asymmetry of $C(t)$ and $C'(t)$, it is not the case.

For a cubic Bézier curve $C(t)$ with its squared distance function $D^2(t)$ with 7 control coefficient f_i, $i = 0, \ldots, 6$, the function $\frac{1}{3} < C(t), C'(t) >$ can be computed as a Bézier polynomial of degree 5 with 6 control coefficients $f_{i+1} - f_i$, for $i = 0, \ldots, 5$.

Construction of Minimal Catmull-Clark's Subdivision Surfaces with Given Boundaries

Qing Pan[1,⋆] and Guoliang Xu[2,⋆⋆]

[1] College of Mathematics and Computer Science,
Hunan Normal University, Changsha, 410081, China
panqing@lsec.cc.ac.cn
[2] LSEC, Institute of Computational Mathematics, Academy of Mathematics and System Sciences, Chinese Academy of Sciences, Beijing 100190, China
xuguo@lsec.cc.ac.cn

Abstract. Minimal surface is an important class of surfaces. They are widely used in the areas such as architecture, art and natural science etc.. On the other hand, subdivision technology has always been active in computer aided design since its invention. The flexibility and high quality of the subdivision surface makes them a powerful tool in geometry modeling and surface designing. In this paper, we combine these two ingredients together aiming at constructing minimal subdivision surfaces. We use the mean curvature flow, a second order geometric partial differential equation, to construct minimal Catmull-Clark's subdivision surfaces with specified B-spline boundary curves. The mean curvature flow is solved by a finite element method where the finite element space is spanned by the limit functions of the modified Catmull-Clark's subdivision scheme.

Keywords: Minimal Subdivision Surface, Catmull-Clark's Subdivision, Mean Curvature Flow.

MR (2000) Classification: 65D17

1 Introduction

Surfaces whose mean curvature H is zero everywhere are minimal surfaces. Minimal surfaces are often used as models in architecture because of having several desirable properties. Most important of all, minimal surfaces have the least surface area, which makes them almost indispensable in large scale and light roof constructions. Secondly, minimal surfaces are separable. Any sub-patch, no matter how small, sheared from a minimal surface still has the least area of all surface

⋆ Supported in part by NSFC grant 10701071 and Program for Excellent Talents in Hunan Normal University (No. ET10901).

⋆⋆ Supported in part by NSFC under the grant 60773165, NSFC key project under the grant 10990013. Corresponding author.

B. Mourrain, S. Schaefer, and G. Xu (Eds.): GMP 2010, LNCS 6130, pp. 206–218, 2010.

patches with the same boundary. Thirdly, minimal surfaces have the balanced surface tension in equilibrium at each point on the roof, as on a soap film, which stabilizes the whole construction. Finally, there are no umbilicus points on a minimal surface; hence no water can stay on the minimal surface roof. Architecture inspired from minimal surfaces embodies the unity of economy and beauty. The most representative buildings of that architectural style are the roofs of the Munich Olympic stadium, the former Kongreßhalle in Berlin. In art world we see plenty of ingenious sculpture works playing the ultimate of minimal surfaces. Scientists and engineers have anticipated the nanotechnology applications of minimal surfaces in the areas of molecular engineering and materials science.

Studies on minimal surfaces was traced back 250 years ago (1744) with Euler as the forerunner, whose research focused on the rotation surface with minimal area. Since then the research of minimal surfaces has been active for several hundred years. In 1760, Lagrange derived the equation minimal surfaces satisfy. The well-known Plateau (1855-90) problem is the existence problem of constructing a piece of surface that interpolates the given boundary curve and has minimal area. This problem, though raised by Lagrange in 1760, was named after Plateau, who created several special cases experimenting with soap films and wire frames. Various special forms of this problem were solved, but it was only in 1930 that general solutions were found independently by Douglas and Rado. The general solution of the equation $H = 0$ was given by Weierstrass (1855-90).

The construction of minimal surfaces have been a heat topic in the area of computer aided design. According to Consin and Monterde [4], there are certain conditions that the control points of Bézier surfaces must satisfy, and in the bicubical case all minimal surfaces are pieces of the Ennerper surfaces up to an affine transformation. Using the four-sided Bézier surface to approximate the minimal surface, Monterde (see [12]) solved Plateau-Bézier problem by replacing the area functional with the Dirichlet functional. Triangular Bézier surface based on a variational approach was constructed by Arnal et al.[1]. Much has been done (see [8], [10], [11]) on the use of minimal surfaces in geometry modeling and shape design. Discrete minimal surfaces were studied by Polthier in [15]. Minimal surfaces as the steady solution of the mean curvature flow (see [16]) were also produced, where they can be both continuous and discrete, usually Bézier surfaces, or B-spline surfaces for the former.

B-splines have been widely accepted as representation tools for curves and surfaces in the industrial design, however there is a serious limitation for designing minimal surfaces with any shaped boundaries using Bézier, B-spline and NURBS because they require the surface patch to be three- or four-sided. In 1974, Charkin first brought the concept of discrete subdivision into the area of computer graphics. Doo-Sabin (see [6]) and Catmull-Clark (see [3]) respectively proposed the subdivision schemes of biquadratic and bicubic B-spline for quadrilateral mesh in 1978. The quartic triangular B-splines was developed by Loop (see [9]) in 1987. Henceforth, subdivision surfaces have rapidly gained popularity in computer graphics and computer aided design. Subdivision algorithms have no limitation on the topology of the control mesh. They can efficiently generate

smooth surfaces from arbitrary initial meshes through a simple refinement algorithm, and they are flexible in creating the features of surface without difficulty.

It is obvious that these well-known subdivision algorithms suffer from serious problems when applied to a control mesh with a boundary because they are suitable for the interior control mesh. Boundary subdivision rules are very important: a plenty of surface designing work deals with the input mesh with boundaries, marked edges and vertices, and the specific treatment for the features of boundaries, such as concave corners, convex corners, sharp creases and smooth creases etc., is always necessary in order to satisfy the designing requirement. For many surface modeling problems, such as the construction of bodies of cars, aircrafts, machine parts and roofs, surfaces are usually piecewise constructed with fixed boundaries. The following are the related works. Subdivision rules of Doo-Sabin surfaces for the boundaries were discussed by Doo (see [5]) and Nasri (see [14]). Based on the work of Hoppe et al.(see [7]) and Nasri (see [13]), Biermann et al.(see [2]) extended the well-known subdivision schemes of Catmull-Clark and Loop. They solve some problems of the original ones, such as lack of smoothness at extraordinary boundary vertices and folds near concave corners, and improve control of the surface shapes with prescribed normals both on the boundary and in the interior.

In this paper, we construct minimal subdivision surfaces based on the modified Catmull-Clark's subdivision algorithms [2] which improves the subdivision scheme around boundaries, and it is preferable and acceptable to use B-spline to represent surface boundary. The well-known mean curvature flow with Dirichlet boundary condition is our evolution model. We adopt the finite element method, where the finite element space spanned by the limit functions of the modified Catmull-Clark's subdivision scheme, as the discretization tool. All the above frameworks contribute to our target, successful construction of desirable minimal subdivision surfaces.

The remainder of this paper is organized as follows: Section 2 is a brief review of the Catmull-Clark's subdivision scheme and its modification of the boundaries, as well as the evaluation of standard and nonstandard Catmull-Clark's subdivision surfaces. In Section 3 we provide the mean curvature flow used to construct the minimal surfaces, and the details of its discretization and numerical computation. Section 4 show several graphic examples and some error comparing results to illustrate the effects of our method. Section 5 is the conclusion.

2 Evaluation of Catmull-Clark's Subdivision Surfaces

Our goal is to construct Catmull-Clark's subdivision surface with specified boundary curves and minimal area. The subdivision surface is defined as the limit of an iterative refinement procedure starting from an initial control mesh where a sequence of increasing refined meshes can be achieved according to the subdivision scheme. The Catmull-Clark's subdivision scheme requires all faces of the initial control mesh must be quadrilaterals. The subsequent refined meshes consist of only quadrilaterals. The control vertices of the refined meshes are generated from

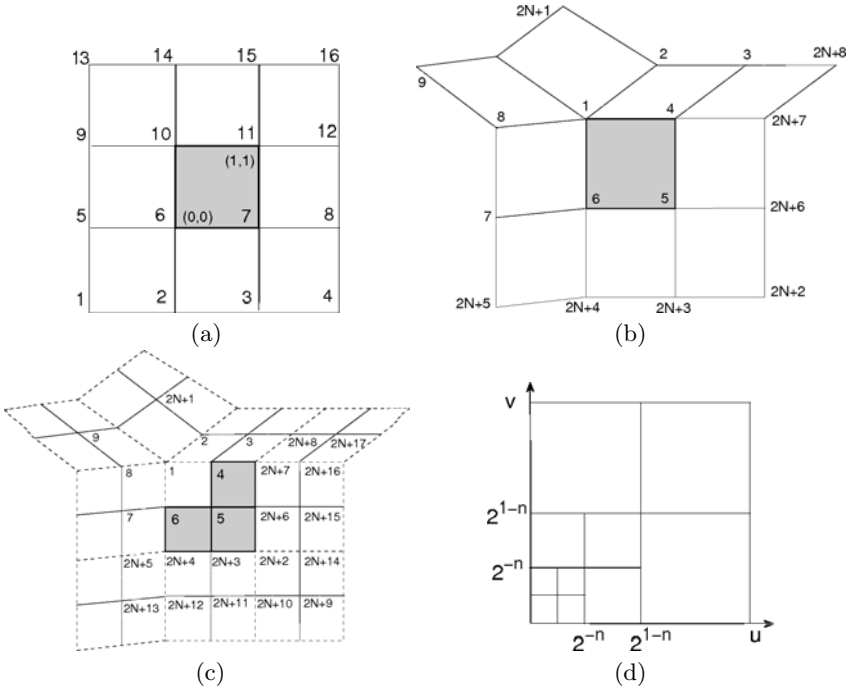


Fig. 1. (a): A regular patch over the shaded quadrilateral with its neighboring 16 control vertices. (b): An irregular patch over the shaded quadrilateral with an extraordinary vertex labeled '1' whose valence is 5. (c): Subdividing this irregular patch once generates 3 shaded sub-patches, and enough control vertices for evaluating them. (d): A unit square is subdivided into unlimited group of quadrilateral sub-domains.

the control vertices of the previous step by a portfolio of weight coefficients. Finally, this sequence of meshes converges to a limit surface composed of unlimited number of surface patches.

We can refer to [3] for the standard Catmull-Clark's subdivision scheme, and its modification proposed by Biermann et al. is described in [2]. we need classify the control mesh into two groups, i.e., standard mesh and nonstandard mesh. Nonstandard mesh includes boundary quadrilaterals and sub-boundary quadrilaterals. Standard mesh consists of only interior quadrilaterals. The quadrilaterals containing boundary vertices are named as boundary quadrilaterals, the ones adjacent to the boundary quadrilaterals are called sub-boundary quadrilaterals, and all others are called interior ones.

2.1 Evaluation of Standard Catmull-Clark's Subdivision Surface

In this section, we briefly describe the evaluation of the standard Catmull-Clark's subdivision surface whose control mesh consists of only interior quadrilaterals.

Each quadrilateral of the control mesh corresponds to one quadrilateral patch of the limit surface. The quadrilateral of the control mesh is regarded as the parameter domain of the surface patch. We choose a unit square

$$\Omega = \left\{(u, v) \in \mathbb{R}^2 : 0 \leq u \leq 1, 0 \leq v \leq 1\right\}$$

as the local parametrization for each quadrilateral t_α and (u, v) as its barycentric coordinates. A regular patch whose four control vertices have a valence of 4 can be represented by 16 basis functions and their corresponding 16 control vertices:

$$\mathbf{x}_\alpha(u, v) = \sum_{i=1}^{16} B_i(u, v)\mathbf{x}_i, \tag{1}$$

where the label i refers to the local sorting of the control vertices shown in Fig.1(a). The bicubic B-spline basis functions N_i are:

$$B_i(u, v) = N_{(i-1)\%4}(u)N_{(i-1)/4}(v), \quad i = 1, 2, \cdots, 16,$$

where "%" and "/" stand for the remainder and division respectively. The functions $N_i(t)$ are the cubic uniform B-spline basis functions:

$$\begin{aligned} N_0(t) &= (1 - 3t + 3t^2 - t^3)/6, \\ N_1(t) &= (4 - 6t^2 + 3t^3)/6, \\ N_2(t) &= (1 + 3t + 3t^2 - 3t^3)/6, \\ N_3(t) &= t^3/6. \end{aligned}$$

If a quadrilateral is irregular, i.e., at least one of its control vertices has a valence other than 4, the resulting patch is not a bicubic B-spline. Now we assume extraordinary vertices are isolated, i.e., there is no edge in the control mesh such that both of its vertices are extraordinary. This assumption can be fulfilled by subdividing the mesh once. Under this assumption, any irregular patch has only one extraordinary vertex. In order to evaluate the surface at any parametric value $(u, v) \in t_\alpha$, the mesh needs to be subdivided repeatedly until the parameter values of interest are interior to a regular patch. Each subdivision of an irregular patch produces three regular sub-patches and one irregular sub-patch (see Fig. 1(b) and (c)). Repeated subdivision of the irregular patch produces three groups of regular patches. This irregular surface patch can be piecewise parameterized as shown in Fig.1(d). The sub-domains $\Omega_j^n, n \geq 1, j = 1, 2, 3$, which can be evaluated, are given as:

$$\begin{aligned} \Omega_1^n &= \{(u, v) : u \in [2^{-n}, 2^{-n+1}], \quad v \in [0, 2^{-n}]\}, \\ \Omega_2^n &= \{(u, v) : u \in [2^{-n}, 2^{-n+1}], \quad v \in [2^{-n}, 2^{-n+1}]\}, \\ \Omega_3^n &= \{(u, v) : u \in [0, 2^{-n}], \quad v \in [2^{-n}, 2^{-k+1}]\}. \end{aligned} \tag{2}$$

They can be mapped onto the unit square Ω through the transform

$$\begin{aligned} t_{1,n}(u, v) &= (2^n u - 1, 2^n v), & (u, v) \in \Omega_1^n, \\ t_{2,n}(u, v) &= (2^n u - 1, 2^n v - 1), & (u, v) \in \Omega_2^n, \\ t_{3,n}(u, v) &= (2^n u, 2^n v - 1), & (u, v) \in \Omega_3^n. \end{aligned}$$

The surface patch $\mathbf{x}_\alpha(u,v)$ is then defined by its restriction to each quadrilateral

$$\mathbf{x}_\alpha(u,v)|_{\Omega_j^n} = \sum_{i=1}^{16} N_i(t_{j,n}(u,v))\mathbf{x}_i^{j,n}, \quad j=1,2,3; \quad n=1,2,\cdots, \tag{3}$$

where $\mathbf{x}_i^{n,j}$ are the properly chosen 16 control vertices around the irregular patch at the subdivision level $n = \text{floor}(\min(-\log2(u), -\log2(v)))$. Three sets of control vertices are (see Fig.1(c))

$$\begin{aligned}
\{\mathbf{x}_i^{1,n}\} = [\ & \mathbf{x}_8^n, \mathbf{x}_7^n, \mathbf{x}_{2N+5}^n, \mathbf{x}_{2N+13}^n, \mathbf{x}_1^n, \mathbf{x}_6^n, \mathbf{x}_{2N+4}^n, \mathbf{x}_{2N+12}^n, \mathbf{x}_4^n, \mathbf{x}_5^n, \mathbf{x}_{2N+3}^n, \mathbf{x}_{2N+11}^n, \\
& \mathbf{x}_{2N+7}^n, \mathbf{x}_{2N+6}^n, \mathbf{x}_{2N+2}^n, \mathbf{x}_{2N+10}^n\ ], \\
\{\mathbf{x}_i^{2,n}\} = [\ & \mathbf{x}_1^n, \mathbf{x}_6^n, \mathbf{x}_{2N+4}^n, \mathbf{x}_{2N+12}^n, \mathbf{x}_4^n, \mathbf{x}_5^n, \mathbf{x}_{2N+3}^n, \mathbf{x}_{2N+11}^n, \mathbf{x}_{2N+7}^n, \mathbf{x}_{2N+6}^n, \mathbf{x}_{2N+2}^n, \\
& \mathbf{x}_{2N+10}^n, \mathbf{x}_{2N+16}^n, \mathbf{x}_{2N+15}^n, \mathbf{x}_{2N+14}^n, \mathbf{x}_{2N+9}^n\ ], \\
\{\mathbf{x}_i^{3,n}\} = [\ & \mathbf{x}_2^n, \mathbf{x}_1^n, \mathbf{x}_6^n, \mathbf{x}_{2N+4}^n, \mathbf{x}_3^n, \mathbf{x}_4^n, \mathbf{x}_5^n, \mathbf{x}_{2N+3}^n, \mathbf{x}_{2N+8}^n, \mathbf{x}_{2N+7}^n, \mathbf{x}_{2N+6}^n, \mathbf{x}_{2N+2}^n, \\
& \mathbf{x}_{2N+17}^n, \mathbf{x}_{2N+16}^n, \mathbf{x}_{2N+15}^n, \mathbf{x}_{2N+14}^n\ ].
\end{aligned}$$

With the subdivision matrix A and the extended subdivision matrix $\bar{A}$, we can get these control vertices by

$$X^n = AX^{n-1} = \cdots = A^n X^0$$

and

$$\bar{X}^{n+1} = \bar{A}X^n = \bar{A}A^n X^0$$

where $X^n = [\mathbf{x}_1^n, \cdots, \mathbf{x}_{2N+8}^n]^T$ and $\bar{X}^n = [\mathbf{x}_1^n, \cdots, \mathbf{x}_{2N+17}^n]^T$.

2.2 Evaluation of Nonstandard Catmull-Clark's Subdivision Surface

As noted above, for the nonstandard Catmull-Clark's subdivision surface, whose control mesh includes boundary quadrilaterals and sub-boundary quadrilaterals, we adopt the modified Catmull-Clark's subdivision rules. Subdividing a sub-boundary quadrilateral once will result in four interior quadrilaterals, so it is easy to evaluate their corresponding patches using the evaluation method of the standard Catmull-Clark's subdivision surface.

The condition of boundary quadrilaterals is a little complicated, however we can repeatedly subdivide it till its sub-patches belong to the class of sub-boundary quadrilaterals. The patches for sub-boundary quadrilaterals can be evaluated using the method stated in the previous paragraph. The boundary quadrilaterals may need to be further subdivided if the parameter values, where the surface patch need to be evaluated, are in this domain. This process are carried through repeatedly till the parameter values to be evaluated are within a sub-boundary quadrilateral.

In the next section, we will introduce the evolution equation and its finite element method based on the modified Catmull-Clark's subdivision scheme.

3 Minimal Surface Construction

Let M_0 be a compact immersed orientable surface in $\mathbb{R}^3$ and $\mathbf{x} \in M_0$ be a general surface point. We intend to find a family $\{M(t) : t \geq 0\}$ of smooth orientable surfaces in $\mathbb{R}^3$ which evolve according to the mean curvature flow

$$\frac{\partial \mathbf{x}}{\partial t} = 2H\mathbf{n}, \quad M(0) = M_0, \tag{4}$$

where H and $\mathbf{n}$ are the mean curvature and the surface normal of M respectively. It is well known that the mean curvature flow is area reducing. The area reducing stops when $H = 0$. Since

$$\Delta_s \mathbf{x} = 2H\mathbf{n},$$

the steady solution of the following mean curvature flow

$$\frac{\partial \mathbf{x}}{\partial t} = \Delta_s \mathbf{x}, \quad M(0) = M_0, \tag{5}$$

is the minimal surface. We use a finite element method to obtain the numerical solution of (5), and our finite element basis functions are the limit form of the modified Catmull-Clark's subdivision scheme.

3.1 Finite Element Method for the Mean Curvature Flow

Let M be the limit surface of the modified Catmull-Clark's subdivision scheme for the control mesh M_d. We multiply a trial function ψ for (5) and apply the Green's formula, then we obtain the following weak form equation

$$\begin{cases} \text{Find} \quad \mathbf{x}(t) \in V^3_{M(t)}, \quad \text{such that} \\ \displaystyle\int_{M(t)} \left[\frac{\partial \mathbf{x}(t)}{\partial t} \psi + (\nabla_s \mathbf{x}(t))^T \nabla_s \psi \right] \mathrm{d}s = \mathbf{0}, \quad \forall \psi \in V_{M(t)} \cap C^1_0(M(t)), \\ M(0) = M_0, \quad \partial M(t) = \Gamma, \qquad\qquad\qquad\qquad \forall \mathbf{x} \in \Gamma, \end{cases} \tag{6}$$

where $V_{M(t)} \subset C^1(M(t))$ is a finite dimensional function space defined by the modified Catmull-Clark's subdivision scheme for the discrete function values on the vertices. $C^1(M(t))$ is the function space consisting of C^1 smooth functions on $M(t)$, and $C^1_0(M(t))$ consists of functions of $C^1(M(t))$ with compact support.

Let ϕ_i be a basis function of $V_{M(t)}$ corresponding to the control vertex $\mathbf{x}_i$ $(i = 1, \cdots, m)$ of the surface $M(t)$, where we assume $\{\mathbf{x}_i\}_{i=1}^{m_0}$ are the interior vertices, and the remaining $\{\mathbf{x}_i\}_{i=m_0+1}^{m}$ are the boundary vertices. Then $\mathbf{x}(t)$ can be represented as

$$\mathbf{x}(t) = \sum_{i=1}^{m_0} \mathbf{x}_i(t)\phi_i + \sum_{i=m_0+1}^{m} \mathbf{x}_i(t)\phi_i, \quad \mathbf{x}_i(t) \in \mathbb{R}^3.$$

Take trial function ψ to be $\phi_j (j = 1, \cdots, m_0)$, (6) can be rewritten as

$$\begin{cases} \sum_{i=1}^{m_0} \mathbf{x}_i'(t) \int_{M(t)} \phi_i \phi_j \mathrm{d}s + \sum_{i=1}^{m_0} \mathbf{x}_i(t) \int_{M(t)} (\nabla_s \phi_i)^T \nabla_s \phi_j \mathrm{d}s \\ \quad = - \sum_{i=m_0+1}^{m} \mathbf{x}_i(t) \int_{M(t)} (\nabla_s \phi_i)^T \nabla_s \phi_j \mathrm{d}s, \quad j = 1, \cdots, m_0, \\ \mathbf{x}_j(0) = \mathbf{x}_j, \quad j = 1, \cdots, m, \end{cases} \tag{7}$$

where $\mathbf{x}_j$ is the j-th control vertex of the initial surface $M(0)$. (7) is a set of nonlinear ordinary differential equations for the unknowns $\mathbf{x}_i(t)$, $i = 1, \cdots, m_0$. The system is nonlinear because the domain $M(t)$, over which the integrations are taken, is also unknown. We use forward Euler scheme to discretize $\mathbf{x}_i'(t)$ as $\frac{\mathbf{x}_i^{k+1} - \mathbf{x}_i^k}{\tau}$ for a given temporal step-size τ, and use a semi-implicit scheme to discretize the remaining terms. A linear system is obtained

$$\begin{cases} \sum_{i=1}^{m_0} \mathbf{x}_i^{k+1} \int_{M^k} \phi_i \phi_j \mathrm{d}s + \tau \sum_{i=1}^{m_0} \mathbf{x}_i^{k+1} \int_{M^k} (\nabla_s \phi_i)^T \nabla_s \phi_j \mathrm{d}s \\ = \sum_{i=1}^{m_0} \mathbf{x}_i^k \int_{M^k} \phi_i \phi_j \mathrm{d}s - \tau \sum_{i=m_0+1}^{m} \mathbf{x}_i^0 \int_{M^k} (\nabla_s \phi_i)^T \nabla_s \phi_j \mathrm{d}s, \ j = 1, \cdots, m_0, \\ \mathbf{x}_j^0 = \mathbf{x}_j, \quad j = 1, \cdots, m, \end{cases} \tag{8}$$

for the unknowns $\mathbf{x}_i^{k+1}$, where M^k is the limit surface of the control vertices $\mathbf{x}_i^k$. System (8) is iteratively solved for $k = 0, 1, \cdots$, using GREMS method till the termination condition

$$\max_i \|\mathbf{x}_i^{k+1} - \mathbf{x}_i^k\| \leq \epsilon$$

(ϵ is a given small value) is satisfied.

3.2 Definition of Basis Functions

As mentioned above, the basis functions of our finite element function space $V_{M(t)}$ is the bicubic B-spline. We use ϕ_i to represent the basis function associating with the control vertex $\mathbf{x}_i$ of the surface M, including its interior vertices, corner vertices and boundary vertices. The basis function ϕ_i is defined by the limit of the modified Catmull-Clark's subdivision scheme where its function value is one at this vertex $\mathbf{x}_i$, but zero at any other vertices. The support of ϕ_i is compact and it covers the 2-ring neighborhoods of vertex $\mathbf{x}_i$.

It needs to evaluate ϕ_i and its partial derivatives in forming the linear system (8), whose parameter values are chosen to be the Gaussian quadrature knots within a unit square. Therefore we only need a few subdivision steps so as to bring these Gaussian quadrature knots into the interior of a regular quadrilateral. Let e_j, $j = 1, \cdots, m_i$ be the 2-ring neighborhood elements of $\mathbf{x}_i$. If e_j is regular, the expression (1) exists for ϕ_i on e_j. If e_j is irregular, local subdivision, as described in §2.1 and §2.2, is needed around e_j until the parameter values of interest are interior to a regular patch.

3.3 Parametrization of Subdivision Surface and Functions on the Surface

In Riemannian geometry, differentiable functions are smooth and C^∞. However, our discretized version of the diffusion problem will be in the class C^1. As we mentioned earlier, the functions are defined by the limit form of the modified Catmull-Clark's subdivision. Such a function is C^2 smooth everywhere except at the extraordinary vertices, where it is C^1. The function is locally parameterized as the image of the unit square defined by

$$\Omega = \{(u, v) \in \mathbb{R}^2 : 0 \leq u \leq 1, 0 \leq v \leq 1\}.$$

That is, (u, v) is the barycentric coordinate of the quadrilateral. Using this parametrization, our discretized representation of M is

$$M = \bigcup_{\alpha=1}^{k} \mathcal{T}_\alpha, \quad \mathring{\mathcal{T}}_\alpha \cap \mathring{\mathcal{T}}_\beta = \emptyset \text{ for } \alpha \neq \beta,$$

where $\mathring{\mathcal{T}}_\alpha$ is the interior of the *quadrilateral function patch* $\mathcal{T}_\alpha$. Each quadrilateral surface patch is assumed to be parameterized locally as

$$\mathbf{x}_\alpha : \quad \Omega \to \mathcal{T}_\alpha; \quad (u, v) \ \mapsto \ \mathbf{x}_\alpha(u, v), \tag{9}$$

where $\mathbf{x}_\alpha(u, v)$ is defined by (1) and (3). Function itself on the surface and its partial derivatives, such as tangents and gradients, can be computed directly. The integration of a function on the surface M is calculated as

$$\int_M f dx := \sum_\alpha \int_\Omega f(\mathbf{x}_\alpha(u, v)) \sqrt{det(g_{ij})} \mathrm{d}u \ \mathrm{d}v, \tag{10}$$

where g_{ij} are the coefficients of the first fundamental form of the surface M. The integration on the square Ω is computed adaptively using Gaussian quadrature formulas (see [17]).

4 Experimental Results

In this section, we present several graphical and numerical results to show that the proposed method for constructing minimal subdivision surface is effective.

4.1 Graphical Examples

We firstly show three models of minimal surfaces with the analytic forms, Helicoid, Catenoid and Ennerper. In Fig. 2, we discretize these three analytic surfaces at a rough level and perturb their interior domain as shown in the first column, then we linearly refine them several times as the initial constructions of our equation evolution. The minimal subdivision surfaces as the steady solutions of (6) are presented in the forth column. We show their corresponding Catmull-Clark's

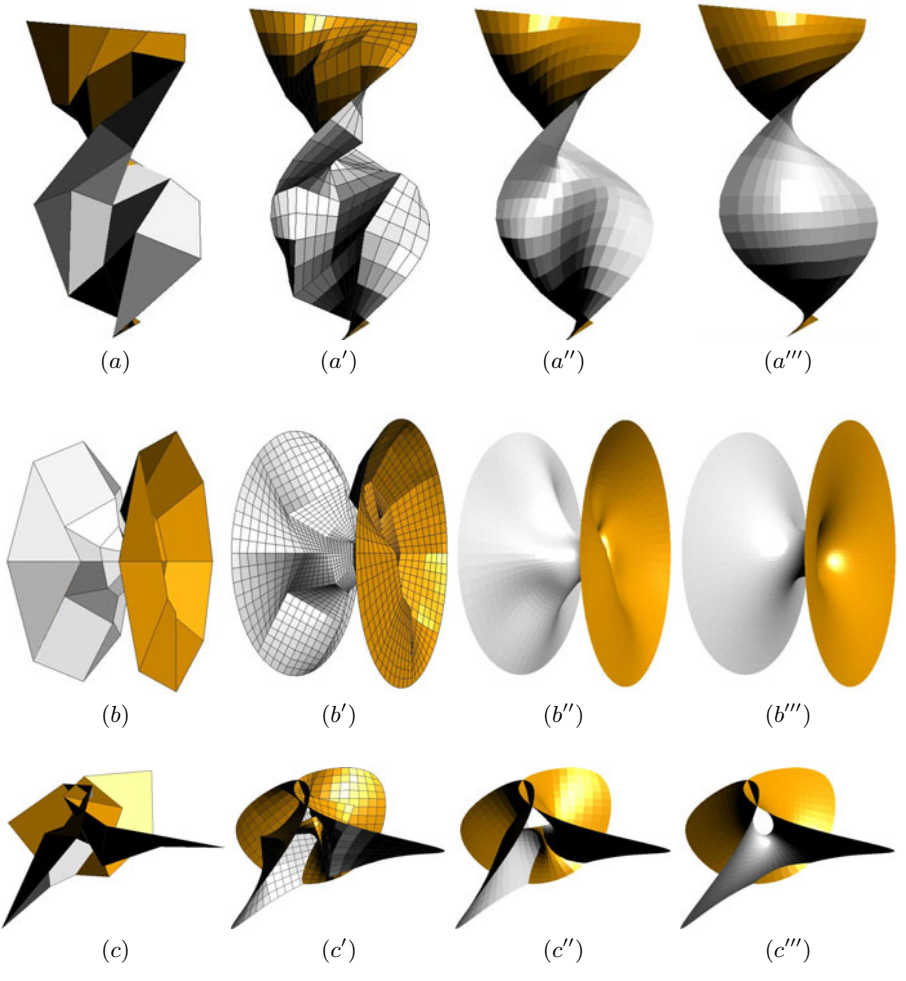


Fig. 2. The first row is the Helicoid surface model, the second row is the Catenoid model and the third row is Ennerper model. (a), (b) and (c) are their initial rough meshes. (a'), (b') and (c') are the initial constructions for the equation evolution by linearly refining the meshes in the first column. (a''), (b'') and (c'') are the Catmull-Clark's surface resulting from refining the rough constructions in the first column by the modified Catmull-Clark's subdivision scheme. On the base of the initial constructions in the second column, we show their corresponding minimal subdivision surfaces by our equation evolution in (a'''), (b''') and (c'''). The density of meshes in the third column and in the forth column is the same.

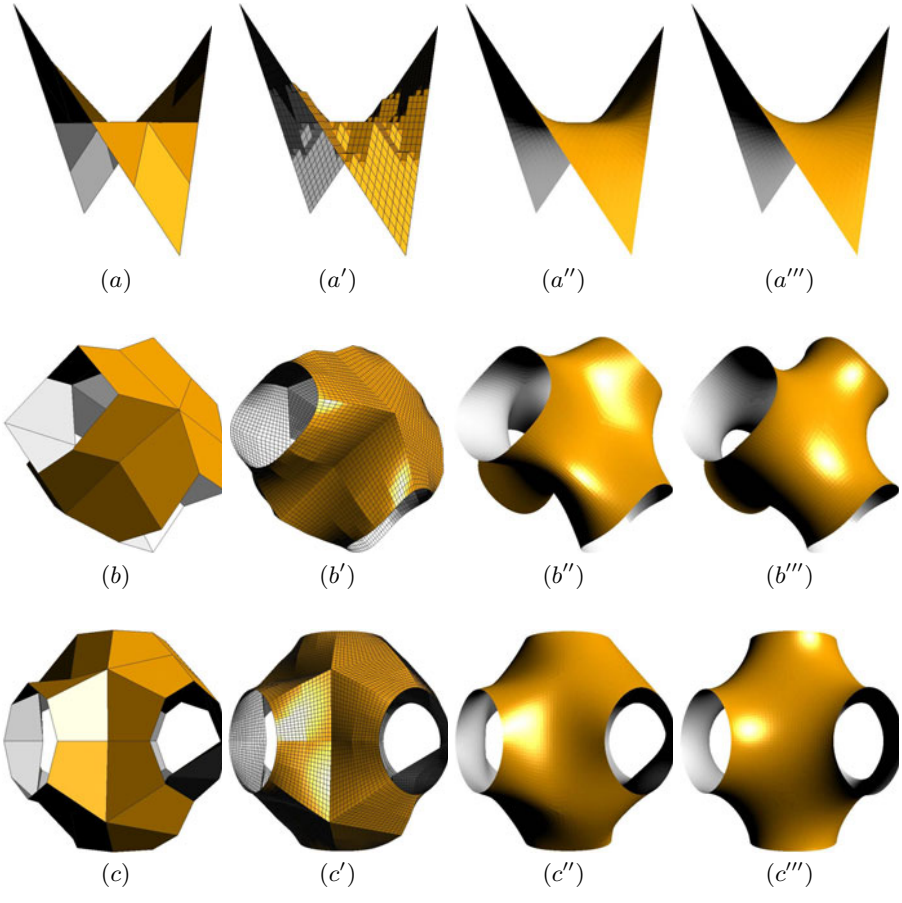


Fig. 3. (a), (b) and (c) are the roughest surface meshes of three models. (a'), (b') and (c') are their corresponding initial constructed surface meshes by linearly refining (a), (b) and (c) respectively. (a''), (b'') and (c'') are their subdivision surfaces through refining the meshes in the first column according to the modified Catmull-Clark's subdivision scheme. (a'''), (b''') and (c''') are their corresponding minimal subdivision surfaces constructed by use of our method based on the initial constructions in the second column.

surfaces in the third column which are obtained by refining the rough meshes in the first column according to the modified Catmull-Clark's subdivision scheme until they have the same density as the corresponding meshes do in the second column. It is clear to see that the Catmull-Clark's surfaces are very different from the final minimal subdivision surfaces.

Fig. 3 shows three examples with fixed boundaries and arbitrary genus. We construct their initial surfaces only from the boundary information at a rough level in the first column. We refine the initial meshes several times by linear

Table 1. k describes the subdivision times, where we subdivide the six models at 6 more and more dense levels respectively. The data from the second to the seven row are the maximum approximate errors of the mean curvature $|H|$ computed from the discrete solutions of the PDE evolution.

Asymptotic maximal values of $|H|$

Models	k	$k+1$	$k+2$	$k+3$	$k+4$	$k+5$
Fig 2(a)	3.783E-2	1.858E-2	1.009E-2	6.092E-3	4.236E-3	3.467E-3
Fig 2(b)	5.748E-2	2.972E-2	1.658E-2	1.077E-2	8.132E-3	6.787E-3
Fig 2(c)	4.638E-2	2.321E-2	1.418E-2	9.558E-3	7.628E-3	6.650E-3
Fig 3(a)	6.223E-2	3.015E-2	1.684E-2	9.713E-3	6.787E-3	5.595E-3
Fig 3(b)	1.073E-1	5.327E-2	2.916E-2	1.631E-2	1.061E-2	7.837E-3
Fig 3(c)	3.234E-1	1.628E-1	8.674E-2	5.607E-2	4.132E-2	3.314E-2

method and show the results in the second column which are the initial constructions of the equation evolution. The boundary curves can have discontinuity on its tangent direction, as shown in Fig. 3 (a'), and some model meshes have extraordinary vertices clearly presented in Fig. 3 (b') and (c') where the face valence of some control vertices is 6. Similarly we also compare the resulting minimal subdivision surfaces in the forth column with their corresponding Catmull-Clark's surfaces in the third column where they have the same density, but the difference of them is very clear.

4.2 Refinement and Convergence

In order to further show the proposed method is effective, we compute the maximum values of $|H|$ from the discrete solutions of our numerical method for the six models used above. We construct the initial surfaces of these models as the initial value of the PDE evolution by subdividing the six models at gradually more and more dense level according to the modified Catmull-Clark's subdivision scheme. The maximal asymptotic values of $|H|$ are presented in Table 1. From the numerical results, we can see that the maximal values of $|H|$ monotonously decline as the increasing of subdivision times k. Hence, our numerical method is convergent.

5 Conclusions

Extensive research work has been done about minimal surfaces. The fascinating characters of minimal surfaces make them widely used in shape designing and many other areas. Subdivision algorithm is a simple and efficient tool to describe free surfaces with any topology. In this paper we adopt the modification of the Catmull-Clark's subdivision scheme which improves the boundary subdivision rules for quadrilateral mesh. We successfully construct minimal Catmull-Clark's subdivision surfaces with given boundary curves using the mean curvature flow, and adopt the numerical method of the finite element based on the modified

Catmull-Clark's subdivision scheme. Our framework can uniformly and flexibly treat all kinds of boundary conditions. The asymptotic error data show our numerical method is also convergent.

References

1. Arnal, A., Lluch, A., Monterde, J.: Triangular Bézier Surfaces of Minimal Area. In: Kumar, V., Gavrilova, M.L., Tan, C.J.K., L'Ecuyer, P. (eds.) ICCSA 2003. LNCS, vol. 2669. Springer, Heidelberg (2003)
2. Biermann, H., Levin, A., Zorin, D.: Piecewise-smooth Subdivision Surfaces with Normal Control. In: SIGGRAPH, pp. 113–120 (2000)
3. Catmull, E., Clark, J.: Recursively generated B-spline surfaces on arbitrary topological meshes. Computer-Aided Design 10(6), 350–355 (1978)
4. Cosin, C., Monterde, J.: Bézier surfaces of minimal area. In: Sloot, P.M.A., Tan, C.J.K., Dongarra, J., Hoekstra, A.G. (eds.) ICCS-ComputSci 2002. LNCS, vol. 2330, pp. 72–81. Springer, Heidelberg (2002)
5. Doo, D.: A subdivison algorithm for smoothing down irregularly shaped polyhedrons. In: proceedings on Interactive Techniques in computer Aided Design, Bologna, pp. 157–165 (1978)
6. Doo, D., Sabin, M.: Behavious of recursive division surfaces near extraordinary points. Computer-Aided Design 10(6), 356–360 (1978)
7. Hoppe, H., DeRose, T., Duchamp, T., Halstend, M., Jin, H., McDonald, J., Schweitzer, J., Stuetzle, W.: Piecewise smooth surfaces reconstruction. In: Computer Graphics Proceedings, Annual Conference series, ACM SIGGRAPH 1994, pp. 295–302 (1994)
8. Jin, W., Wang, G.: Geometric Modeling Using Minimal Surfaces. Chinese Journal of Computers 22(12), 1276–1280 (1999)
9. Loop, C.: Smooth subdivision surfaces based on triangles. Master's thesis. Technical report, Department of Mathematices, University of Utah (1978)
10. Man, J., Wang, G.: Approximating to Nonparameterzied Minimal Surface with B-Spline Surface. Journal of Software 14(4), 824–829 (2003)
11. Man, J., Wang, G.: Minimal Surface Modeling Using Finite Element Method. Chinese Journal of Computers 26(4), 507–510 (2003)
12. Monterde, J.: Bézier surface of minimal area: The dirichlet approach. Computer Aided Geometric Design 21, 117–136 (2004)
13. Nasri, A.H.: Surface interpolation on irregular networks with normal conditions. Computer Aided Geometric Design 8, 89–96 (1991)
14. Nasri, A.H.: Polyhedral subdivision methods for free-form surfaces. ACM Transactions on Graphics 6(1), 29–73 (1987)
15. Polthier, K.: Computational aspects of discrete minimal surfaces. In: Hass, J., Hoffman, D., Jaffe, A., Rosenberg, H., Schoen, R., Wolf, M. (eds.) Proc. of the Clay Summer School on Global Theory of Minimal Surfaces (2002), citeseer.ist.psu.edu/polthier02computational.html
16. Xu, G.: Geometric Partial Differential Equation Methods in Computational Geometry. Science Press, Beijing (2008)
17. Xu, G., Shi, Y.: Progressive computation and numerical tables of generalized Gaussian quadrature formulas. Journal on Numerical Methods and the Computer Application 27(1), 9–23 (2006)

Parameterization of Star-Shaped Volumes Using Green's Functions

Jiazhi Xia[1], Ying He[1], Shuchu Han[1], Chi-Wing Fu[1], Feng Luo[2], and Xianfeng Gu[3]

[1] School of Computer Engineering, Nanyang Technological University, Singapore
{xiaj0002,yhe,schan,cwfu}@ntu.edu.sg
[2] Department of Mathematics, Rutgers University, USA
fluo@math.rutgers.edu
[3] Department of Computer Science, Stony Brook University, USA
gu@cs.sunysb.edu

Abstract. Parameterizations have a wide range of applications in computer graphics, geometric design and many other fields of science and engineering. Although surface parameterizations have been widely studied and are well developed, little research exists on the volumetric data due to the intrinsic difficulties in extending surface parameterization algorithms to volumetric domain. In this paper, we present a technique for parameterizing star-shaped volumes using the Green's functions. We first show that the Green's function on the star shape has a unique critical point. Then we prove that the Green's functions can induce a diffeomorphism between two star-shaped volumes. We develop algorithms to parameterize star shapes to simple domains such as balls and star-shaped polycubes, and also demonstrate the volume parameterization applications: volumetric morphing, anisotropic solid texture transfer and GPU-based volumetric computation.

1 Introduction

The recent decade has witnessed the great advancements of surface parameterizations, exemplified in a wide range of applications exhibited in science and engineering. Despite these successes, most real-world objects are in fact volumes rather than surfaces. It remains both unclear and challenging on how to generalize existing surface parameterization methods from surfaces to volumes. And with volume parameterization, we envision a large pool of applications that can benefit from the result, including solid texture mapping, volumetric tetrahedralization for simulation, and volumetric registration.

Due to the intrinsic difference between surfaces and volumes, many classical results on surface parameterization cannot be directly generalized to produce volume parameterization. For example, it is well-known that a harmonic map between a topological disk (a genus zero surface with a single boundary) and a planar convex domain is diffeomorphic (i.e., bijective and smooth), if the boundary map is homeomorphic (i.e., bijective and continuous). This result plays an important role in surface parameterization. Unfortunately, such an approach is not applicable to volumes, i.e., volumetric harmonic map is not guaranteed to be bijective even though the target domain is convex. In this paper, we aim at handling the challenges by proposing a theoretically sound algorithm that can produce a diffeomorphism between two star-shaped volumes.

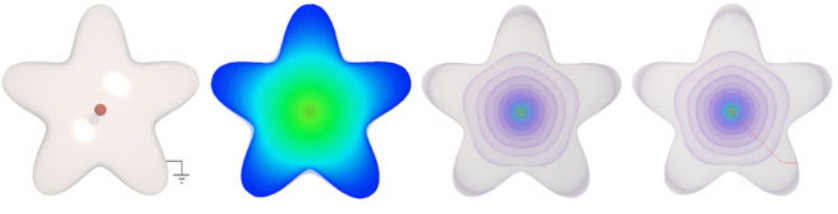
B. Mourrain, S. Schaefer, and G. Xu (Eds.): GMP 2010, LNCS 6130, pp. 219–235, 2010.

Our volume parameterization method is strongly motivated by the property of electric field. Given a closed genus-0 metal surface S, let M denote its interior volume, i.e., $\partial M = S$. We construct an electric field by putting a positive electric charge at a point c inside M, and connecting the boundary surface S to the ground. The *electric potential* inside M is a *Green's function*, $G : M \to \mathbb{R}$, such that

$$\begin{cases} \Delta G(x) = \delta(x-c) \\ G|_{\partial M} \equiv 0, \end{cases} \tag{1}$$

where $\delta(x-c)$ is the Dirac function. In general, the level set of G, $G^{-1}(r), r \in \mathbb{R}^+$ or an isopotential surface is a smooth surface in M. The gradient of the electric potential ∇G is the *electric field*. *Electric field lines* are the integration curves of the electric field, i.e., the tangent vectors of the electric field lines are parallel to the electric field. Different electric field lines only intersect at the points where we put the electric charge, or at the critical point of the potential. Electric field lines start from the electric charge and are orthogonal to the iso-potential surfaces everywhere, in particular to the boundary surface ∂M.

If M is a star-shaped volume, every ray cast from c intersect S only once, and there are no other critical points of the potential. Therefore, all the iso-potential surfaces can be topological spheres and all electric field lines intersect only at point c (see Fig. 1). Since each point inside M is now uniquely determined by a corresponding electric field line and an iso-potential surface, determining the map between two star-shaped volumes is equivalent to constructing the map between the corresponding iso-potential surfaces and the electric field lines. Therefore, we map the boundary surface to the unit sphere, thereby putting each iso-potential surface to a concentric sphere, the electric field lines to the radii, and the center $\mathbf{c}$ to the origin. In this way, the star-shaped volume can be parameterized to the unit solid ball. With the help of ball parameterization, the map between two star shapes can then be constructed by mapping each shape to the unit ball and constructing a bijective map between the two unit balls. Such a constructed volumetric map is guaranteed to be a diffeomorphism.



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

Fig. 1. Electric field on the star shape. Given a metal surface S, we put a positive charge at the center (the red point) and then connect S to the ground (shown in (a)). The electric field is a Green's function shown in (b). If the surface S is star-shaped, then the Green's function has a unique critical point. As a result, all iso-potential surfaces are topological spheres (shown in (c)). The electric field line (red curve in (d)) is perpendicular to all iso-potential surfaces.

Our contributions include

- First, we show that the Green's function on star shapes has a unique critical point and all level sets inside the star shape are topological spheres. Then we prove that the Green's function can induce a diffeomorphism between two star shapes. To our knowledge, this is the first constructive proof of the existence of a diffeomorphism between two non-trivial shapes.
- Secondly, based on our theoretical results, we develop algorithms to parameterize star shapes to star-shaped domains, such as solid balls and star-shaped polycubes.
- Thirdly, we showcase a variety of applications that benefit from our volume parameterization method, including volumetric morphing, anisotropic solid texture transfer and GPU-based volumetric computation.

The remaining of the paper is organized as follows. We first briefly discuss the previous work in Section 2. Next, we introduce the theoretic background in Section 3, and present our volume parameterization algorithm in Section 4. Experimental results are then reported in Section 5. We conclude this work in Section 6. The theoretic proofs are presented in the Appendix.

2 Previous Work

Extensive research has been done on surface parameterization due to its wide applications in computer graphics. The surveys of [1] [2] provide excellent reviews on various kinds of mesh parameterization techniques. In the following, we briefly review the related work on volumetric meshing and volumetric harmonic map.

Labelle and Shewchuk introduced the isosurface stuffing algorithm to generate tetrahedron meshes with bounded dihedral angles in [3]. The volumetric discrete Laplace-Beltrami operator used in this work generalizes the *cotan* formula in the surface case; the *cotan* value of dihedral angles is used to replace those of corner angles. The range of the dihedral angles affects the parameterization quality. A Delaunay-based variational approach to isotropic tetrahedral meshing is introduced by Alliez et al. in [4]; this method can produce well-shaped tetrahedra by energy minimization. Tandem algorithm is introduced to isosurfaces extraction and simplification in [5]. The volumetric harmonic map depends on volumetric Laplacian; Zhou et al. [6] applied volumetric graph Laplacian to large mesh deformation.

Harmonicity in volumes can be similarly defined via the vanishing Laplacian, which governs the smoothness of the mapping function. Wang et al. [7] studied the formula of harmonic energy defined on tetrahedral meshes and computed the discrete volumetric harmonic maps by a variational procedure. Volumetric parameterization using fundamental solution method is introduced in [8] and applied to volumetric deformation and morphing. Other than that, harmonic volumetric parameterization for cylinder volumes is applied for constructing tri-variate spline fitting in [9]. All the above approaches rely on volumetric harmonic maps. Unfortunately, as pointed out previously in Section 1, these volumetric harmonic maps cannot guarantee bijective mappings even though the target domain is convex.

Besides the volumetric harmonic map, another stream of research studies the mean value coordinates for closed triangular mesh [10,11]. Mean value coordinates are a powerful and flexible tool to define a map between two volumes. However, there is no guarantee that the computed map is a diffeomorphism.

Our approach differs intrinsically from these existing approaches in two-fold. First, we solve the Green's functions on star shapes and show that the resultant functions have unique critical points. As a result, the Green's function induced map is guaranteed to be a diffeomorphism. Second, we use fundamental solution method [12,8,13] rather than the conventional volumetric harmonic map [7], since the fundamental solution method is truly meshless, thus, it does not depend on the tetrahedral mesh. In sharp contrast, volumetric harmonic map heavily depends on the quality of the tetrahedral mesh. Irregular tetrahedralization may lead to numerical error and degeneracy of the volumetric harmonic map even on convex or star shapes.

Our work is also related to polycube map which can be used as the parametric domain for the volume parameterization. Tarini et al. pioneered a method to construct polycube map by projecting the vertices to the polycube domain [14]. Wang et al. presented an intrinsic approach to construct polycube map that is guaranteed to be a diffeomorphism [15]. Later, they developed a method that allows the users to freely specify the extraordinary points on the 3D models [16]. Lin et al. presented an automatic algorithm to construct polycube map with simple geometry and topology [17]. Using the divide-and-conquer strategy, He et al. developed a polycube map construction method that can process large 3D models [18].

3 Theoretic Foundation

This section briefly introduces the theoretic foundation of star shape parameterization; see the detailed proof in the Appendix section.

A volume M is called a *star shape* if there exists a point $\mathbf{c} \in M$ such that any ray cast from $\mathbf{c}$ intersects the boundary of M only once. The point $\mathbf{c}$ is called the center of M. In particular, any convex volume is a star shape, where any interior point can serve as the center. From the implementation point of view, computing the intersection of a ray with a surface is typically computationally expensive. Thus, we use an alternative approach to define a star shape:

Lemma 1. *A volume M is a star shape if and only if there exists a point $\mathbf{c} \in M$ such that for any boundary point $\mathbf{p} \in \partial M$,*

$$(\mathbf{c} - \mathbf{p}, \mathbf{n}(\mathbf{p})) \leq 0, \tag{2}$$

where $\mathbf{n}(\mathbf{p})$ is the normal vector at $\mathbf{p}$ and $(,)$ is the dot product.

The following lemmas reveal some nice properties of star shapes and lay a crucial role in our work.

Lemma 2 (Green's function on a star shape). *Suppose M is a star shape with a center $\mathbf{c} \in M$, G is the Green's function (see Eqn. (7)) with a pole at $\mathbf{c}$, then $\mathbf{c}$ is the only critical point of G.*

Lemma 3. *Suppose M is a star shape with a center $\mathbf{c} \in M$, G is the Green's function with a pole at $\mathbf{c}$. Then for any $r \in \mathbb{R}^+$, the level set $G^{-1}(r)$ is topologically equivalent to a sphere.*

Let γ_1 and γ_2 be two integration curves of the gradient field, ∇G. If γ_1 and γ_2 intersect at point $\mathbf{p}$, i.e., $\mathbf{p} \in \gamma_1 \cap \gamma_2$, then $\nabla G(\mathbf{p})$ must be zero. Namely, $\mathbf{p}$ must be a critical point of G. Since G has only one critical point $\mathbf{c}$, γ_1 and γ_2 only intersect at the center $\mathbf{c}$. Furthermore, each integration curve of ∇G intersects the boundary surface ∂M perpendicularly.

A map between two star-shaped volumes M and $\tilde{M}$ with centers $\mathbf{c}$ and $\tilde{\mathbf{c}}$, respectively, can be constructed in the following manner. First we compute a bijective map between their boundaries $\phi : \partial M \to \partial \tilde{M}$. Then we compute two Green's functions G and $\tilde{G}$ on M and $\tilde{M}$ with poles $\mathbf{c}$ and $\tilde{\mathbf{c}}$, respectively. Let $r \in \mathbb{R}^+$, then the level set $G^{-1}(r) \subset M$ matches the level set $\tilde{G}^{-1}(r) \subset \tilde{M}$. Let $\mathbf{p} \in \partial M$, the integration curve through $\mathbf{p}$ in M matches the integration curve through $\phi(\mathbf{p})$ in $\tilde{M}$. The centers of M and $\tilde{M}$ are mapped to each other. Each interior point (other than the origin) is the intersection of a unique level set and a unique integration curve, therefore, every point in M can be uniquely mapped to a point in $\tilde{M}$.

Therefore, we arrive at the following theorem, which lays down the theoretic foundation of our volumetric parameterization algorithm.

Theorem 1. *Suppose M and $\tilde{M}$ are star-shaped volumes with centers $\mathbf{c}$ and $\tilde{\mathbf{c}}$, G and $\tilde{G}$ are Green's functions with poles at $\mathbf{c}$ and $\tilde{\mathbf{c}}$, respectively. If the boundary map $\partial M \to \partial \tilde{M}$ is a diffeomorphism, then the map $f : M \to \tilde{M}$ induced by G and $\tilde{G}$ is also a diffeomorphism.*

Theorem 1 laid down the foundation of the proposed volume parameterization framework. We should point out that even though $\mathbf{c}$ and $\tilde{\mathbf{c}}$ are poles of the Green's functions G and $\tilde{G}$, the induced map $f : M \to \tilde{M}$ is smooth everywhere including the pole $\mathbf{c}$ since we define $f(\mathbf{c}) := \tilde{\mathbf{c}}$. This can be elucidated by the physical meaning of Green's function. Consider the phenomenon of a grounded conducting surface surrounding a charged body at the center $\mathbf{c}$. The electric potential inside the volume bounded by the surface is the Green's function. If the volume is a solid ball and the center is the origin, then parameterization induced by the Green function is equivalent to the polar coordinate. The center is the pole of the polar parameterization, but the mapping between two balls induced by the polar coordinates has no singularity [19].

Remark. Gergen showed that the gradient of a Green's function in a star-shaped three dimensional region never vanish[20]. This implies that there is no interior singularity of the Green function, therefore the level sets are topological spheres, the integration curves of the gradient field do not intersect either. This gives alternative proof for our main theoretic result.

4 Volume Parameterization Using Green's Functions

This section presents the algorithmic detail of parameterizing star-shaped volumes to simple domains, such as the unit ball and star-shaped polycubes.

4.1 Parameterizing a Star Shape to a Ball

Step 1. Star shape verification and center detection. The input of our algorithm is a closed genus-0 surface S which encloses a volume M, i.e., $S = \partial M$. S is represented by a triangular mesh with vertices $\{\mathbf{v}_i\}_{i=1}^n$. First, we need to verify whether M is star-shaped. If it is true, we determine the center of M. Note that for a given star shape, there could be infinite possible choices for the centers and the distribution of Green's function. A badly chosen center may introduce severe bias in the volume parameterization. Thus, we prefer a geometry-aware center, where a natural choice is a center that is close to the center of mass of M. This leads to the following linear constrained quadratic programming problem:

$$\min_{\mathbf{c}} \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{c}-\mathbf{v}_i\|^2$$
$$subject\ to\ (\mathbf{c}-\mathbf{v}_i,\mathbf{n}_i) \le 0 \quad i = 1,\cdots,n\ .$$

The objective function aims to minimize the distance between the center $\mathbf{c}$ and the center of mass, where the linear constraints precisely ensure the detected center $\mathbf{c}$ satisfies the star shape requirement (see Eqn. 3). If M is not star-shaped, then no valid solution will be found. In our implementation, we use the MOSEK optimization software [21] to solve this quadratic programming problem.

Input: S, the boundary mesh of a star-shaped volume M
Output: $f : M \to \mathbb{B}^3$ is diffeomorphism
1.1 *Find the center of M;*
1.2 *Compute the Green's function on M,* $G_M : M \to \mathbb{R}$*;*
1.3 *Map the center* $\mathbf{c}$ *to the center of* $\mathbb{B}^3$*,* $f(\mathbf{c}) = \mathbf{0}$*;*
1.4 *Parameterize the boundary points by constructing a conformal spherical mapping* $\phi : \partial M \to \partial\mathbb{B}^3$*;*
1.5 **for** *every interior vertex* $\mathbf{p} \in M$
1.6 *Trace the integration curve* γ *from* $\mathbf{p}$ *to the boundary point* $\mathbf{q} \in \partial M$*;*
1.7 *Set* $f(\mathbf{p}) = \frac{\phi(\mathbf{q})}{G_M(\mathbf{p})+1}$
1.8 **end for**

Algorithm 1. Ball parameterization of star shapes

Step 2. Computing Green's functions on M and $\mathbb{B}^3$. Next, we compute the Green's function on the star-shaped volume M using the method detailed in the fundamental solution [12,8]. Suppose we have an electric charge q_i at point $\mathbf{p_i}$, the electric potential caused by q_i at point $\mathbf{r}$ is

$$K(q_i,\mathbf{p}_i;\mathbf{r}) = \frac{1}{4\pi}\frac{q_i}{|\mathbf{p}_i-\mathbf{r}|}.$$

We need to put m electric charges $\{q_i\}$ at m points $\{\mathbf{p}_i\}$ on an offset surface above the boundary surface of ∂M, such that on the boundary ∂M, the total potential equals zero,

$$G_M(\mathbf{r}) = \sum_{i=1}^{m} K(q_i,\mathbf{p}_i;\mathbf{r}) + G(1,\mathbf{c};\mathbf{r}) = 0, \forall \mathbf{r} \in \partial M,$$

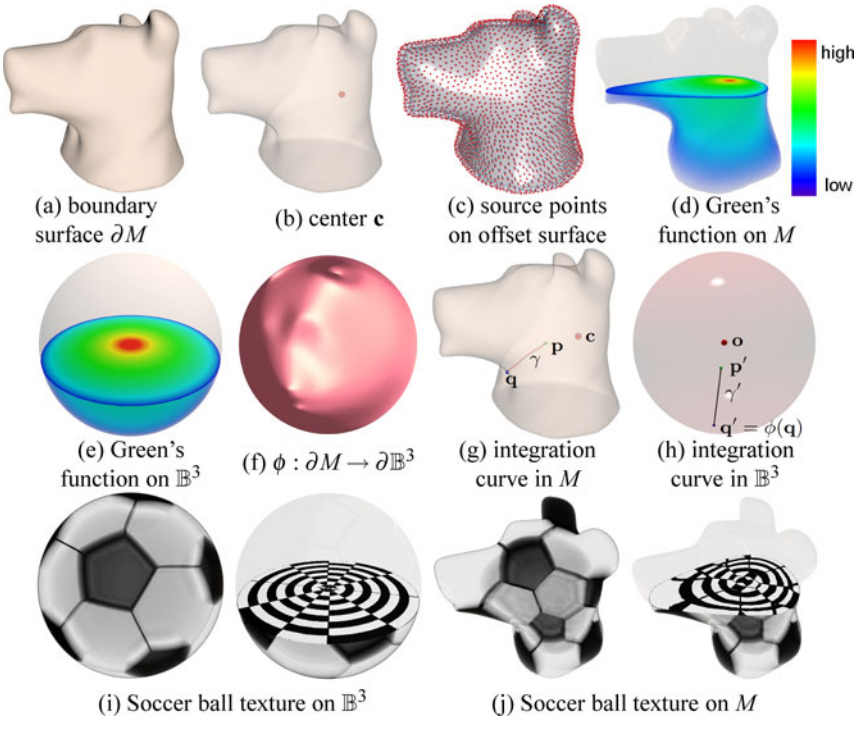


Fig. 2. Green's function induces a diffeomorphism between the star shape M and the unit ball $\mathbb{B}^3$. The input model is a triangular mesh (shown in (a)) which encloses a star-shaped volume. The red point in (b) shows the star shape center. Then we compute the Green's function on M using the fundamental solution method. (c) The source point placed on the offset surface of ∂M. (d) The Green's function on M. (e) The Green's function on $\mathbb{B}^3$ which is given by a closed form formula $\frac{1}{r} - 1$. (f) The boundary parameterization by constructing a conformal spherical map $\phi : \partial M \to \partial \mathbb{B}^3$. (g) We parameterize the interior point $\mathbf{p}$ by tracing the integration curve γ to the boundary point $\mathbf{q}$. Note that the integration curve is perpendicular to the iso-surfaces of G. (h) The image of $\mathbf{p}$ is given by $\frac{\phi(\mathbf{p})}{G_M(\mathbf{p})+1}$. (i) and (j) show the volume rendering of a soccer ball texture on M and $\mathbb{B}^3$, respectively.

where q_i's are unknowns. The equation is converted to a dense linear system, which can be solved using the singular value decomposition method provided in Matlab. As suggested in [8], we place $m = 0.6n$ source points on the offset surface with offset distance equals 0.05 times the main diagonal of M.

The Green's function on $\mathbb{B}^3$ (with the origin as the center) has a closed form, $G_B(p) = \frac{1}{r} - 1$, where r is the distance from p to the origin.

Step 3. Parameterizing the boundary points. Furthermore, we compute the boundary map $\phi : \partial M \to \mathbb{B}^3$. Since the boundary of the unit ball is the sphere $\mathbb{S}^2$, the conformal spherical mapping [22] is a diffeomorphism, and thus, can serve as the boundary map.

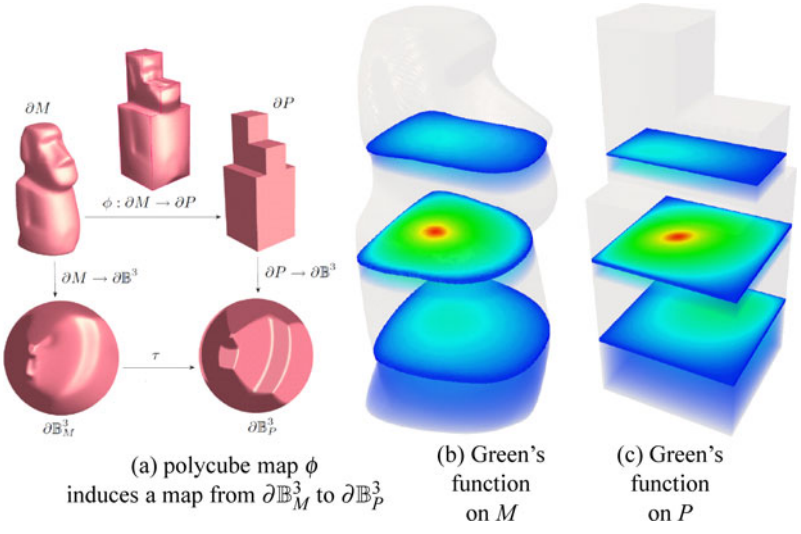


Fig. 3. Parameterizing a star shape M to a polycube P using the Green's function. We first construct the conformal polycube map $\phi : \partial M \to \partial P$. Then, we parameterize M and P to the ball using Algorithm 1. The conformal polycube map ϕ induces an identity map between $\partial \mathbb{B}^3_M$ and $\partial \mathbb{B}^3_P$. The volume parameterization is then given by $f = f_M \circ f_P^{-1}$.

Step 4. Parameterizing the interior points. The interior of the volume is represented by a tetrahedral mesh. We use Tetgen [23] to generate a tetrahedral mesh for a given surface mesh S to meet the boundary constraints. To improve the meshing quality, we employ the variational tetrahedral meshing techinque [4] which can significantly reduce the slivers and produce well-shaped tetrahedral meshes. The tetrahedral mesh M is represented by $M = (V, E, F, T)$ where V, E, F, and T are the vertex, edge, face, and tetrahedra sets, respectively. The Green's function G_M is represented as a piecewise linear function, $G_M : V \to \mathbb{R}$. The gradient of G can be computed as follows: suppose t_{ijkl} is a tetrahedron with vertices $\{v_i, v_j, v_k, v_l\}$, the face on the tetrahedron against vertex v_i is f_i; similarly v_j, v_k, and v_l are against f_j, f_k, and f_l, respectively. We define $\mathbf{s}_i$ to be the vector along the normal of f_i with length equal to 2 times the area of f_i, and so can $\mathbf{s}_j, \mathbf{s}_k, \mathbf{s}_l$ be defined. Then, the gradient of G_M in t_{ijkl} is a constant vector field

$$\nabla G_M = G_M(v_i)\mathbf{s}_i + G_M(v_j)\mathbf{s}_j + G_M(v_k)\mathbf{s}_k + G_M(v_l)\mathbf{s}_l.$$

We then define the vertex gradient as the average of the gradient vectors in the neighboring tetrahedra.

Finally, the parameterization from M to $\mathbb{B}^3$, $f : M \to \mathbb{B}^3$ is constructed as follows. We map the center $\mathbf{c}$ to the origin, i.e., the center of $\mathbb{B}^3$. Given an interior point $\mathbf{p} \in M$ (other than the center $\mathbf{c}$), we trace the integration curve γ of the gradient field from $\mathbf{p}$, γ intersects the boundary surface ∂M at $\mathbf{q}$, then γ corresponds to the radius of $\mathbb{B}^3$ through the point $\phi(\mathbf{q})$. Suppose the Green's function value at $\mathbf{p}$ is $G_M(\mathbf{p})$, then the image of $\mathbf{p}$ is defined by

$$f(\mathbf{p}) = \frac{\phi(\mathbf{q})}{G_M(\mathbf{p}) + 1}.$$

Figure 2 illustrates the pipeline of parameterizing the dog head to a solid ball. To visualize the parameterization, we design a soccer ball texture on $\mathbb{B}^3$ and then map it to the dog head. Note that the iso-parameter surfaces in M are curved, but the cut view is obtained by a cutting plane. Thus, the texture on the intersection plane in Fig. 2(1) may look irregular.

Input: boundary meshes of a star shape M and a star-shaped polycube P
Output: $f : M \to P$ is diffeomorphism
2.1 *Parameterize P to the unit ball $f_P : P \to \mathbb{B}^3_P$;*
2.2 *Parameterize M to the unit ball $f_M : M \to \mathbb{B}^3_M$;*
2.3 *Construct the polycube map $\phi : \partial M \to \partial P$;*
2.4 *Construct the map between two balls $\psi : \mathbb{B}^3_P \to \mathbb{B}^3_M$ induced by the polycube map ϕ;*
2.5 *Compute the composite map $f : M \to P$, $f = f_M \circ \psi \circ f_P^{-1}$.*

Algorithm 2. Polycube parameterization of star shapes

4.2 Parameterizing a Star Shape to a Polycube

Ball parameterization is useful for the star shapes which resemble the geometry of the sphere. However, a general star shape may be significantly different from a ball. Thus, ball parameterization may result in large distortions. For such cases, we propose to use the star-shaped polycube as the parametric domain since it resembles the input object better than the ball.

Given a star shape M and a polycube P, we want to find a bijective and smooth map $f : M \to P$. Rather than computing the map directly, we first individually parameterize M and P to the unit balls using Algorithm 1 (see Sec 4.1). Then we seek a smooth map between two balls $\psi : \mathbb{B}^3_M \to \mathbb{B}^3_P$. Finally, the polycube parameterization is given by the composite map $f = f_M \circ \psi \circ f_P^{-1}$.

The polycube parameterization can be illustrated clearly by the following commutative diagram:

$$\begin{array}{ccc} M & \xrightarrow{f:M\to P} & P \\ {\scriptstyle f_M:M\to\mathbb{B}^3_M}\downarrow & & \downarrow{\scriptstyle f_P:P\to\mathbb{B}^3_P} \\ \mathbb{B}^3_M & \xrightarrow[\psi:\mathbb{B}^3_M\to\mathbb{B}^3_P]{} & \mathbb{B}^3_P \end{array}$$

Note that there exists infinitely many smooth maps between two unit balls, but different ψ could result in different volumetric parameterization. To find a low-distortion volumetric parameterization, i.e., mapping the head of Moai (see Fig. 3(a)) to the top of the polycube, and so on, the polycube map can serve as a feasible boundary constraint. In our implementation, we choose the approach of conformal polycube map (for genus-0

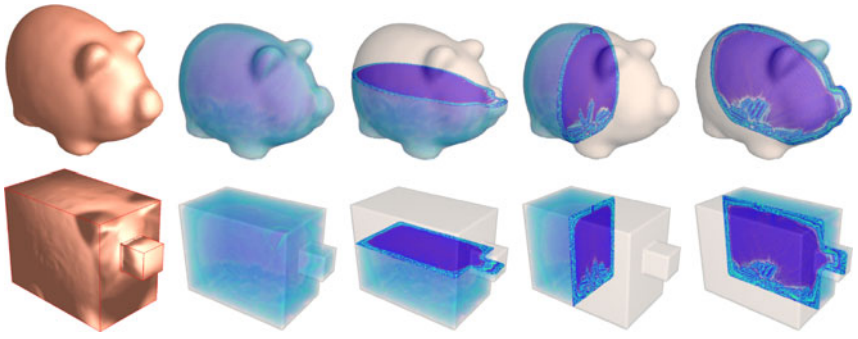


Fig. 4. Parameterizing the pig model to a polycube. Row 1 and 2 show the volume rendering of the volumetric data and polycube parameterization respectively.

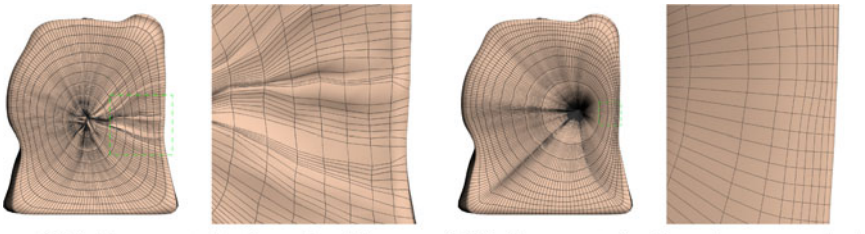


Fig. 5. Comparison. We map the dog head to unit ball using volumetric harmonic map [7] and our approach. As shown in the cut view of iso-parametric curves, our method is more robust and leads to hexahedral meshes with better quality. Our approach also guarantees that the iso-parametric curve which follow the direction of the gradient is orthogonal to the other two iso-parametric curves which span the iso-surface of the Green's function.

surfaces) [15]. Figure 3 illustrates the pipeline of parameterizing the star shapes to the polycube.

5 Experimental Results and Applications

This section showcases the experimental results and a variety of applications that can benefit from our star-shape parameterization method, from volumetric morphing to volume-based computation on the GPU.

Results. Figure 4 shows the parameterization of the pig-shaped coin box to a polycube. The volume rendering and the cut views reveal the quality of the parameterization.

We compared our method with the volumetric harmonic map method [7]. As mentioned above, the volumetric harmonic map is not guaranteed to be homeomorphic even though the domain is convex. In Figure 5, we parameterized the star model to the unit

ball. As shown in the cut view and iso-parametric curves, our method is robust and leads to hexahedral meshes with better quality.

Volumetric Morphing. To morph from one star-shape to another, we parameterize them to a common parametric domain, such as a ball and then determine a smooth map (e.g., identity map in our current implementation) between the balls. Figure 6 shows a running example from star to Venus head.

Anisotropic Solid Texture. Solid textures [24], or anisotropic solid textures [25], allow us to fill the interior of 3D models with spatially-varying and anisotropic texture patterns. Takayama et al. [25] proposed a lapped texture approach [26] to synthesize anisotropic solid textures by pasting solid texture examplars [24] repeatedly over the tetrahedron structure of 3D geometries. This approach can result in high-quality and large-scale solid textures with low computation cost; to create such a texture, the user, however, has to mark up volumetric tensor field and edit the texture in a geometry-dependent fashion.

Our star-shape volume parameterization method can further broaden the applicability of the lapped solid texture results to a larger pool of geometric models. As illustrated in Figure 7, we can first parameterize a given star shape that has been pre-synthesized with lapped solid texture to a solid ball using the Green's function; hence, we can transfer the synthesized texture information from the input geometric model to our star-shape model through the common parametric ground. Our approach allows the reuse of synthesized anisotropic solid textures without incurring additional texture synthesis. Furthermore, since our volume parameterization method is a diffeomorphism, we can guarantee the bijectivity and smoothness in the texture transfer process, as demonstrated in Figure 7.

GPU-based Volumetric Computation. Another advantage of having a smooth volume parameterization is the luxury of being able to perform computations throughout the volume by taking the computation process to the highly-structured parametric domain. Here we can parameterize the given star-shape model by a cube model (or polycube) so that the data inside the parametric domain can naturally be modeled by a 3D texture;

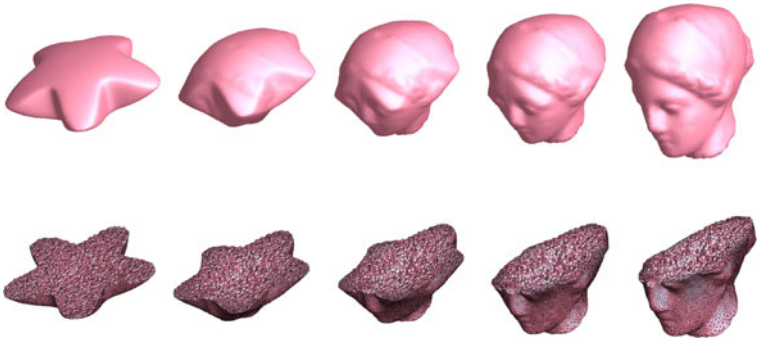


Fig. 6. Volumetric morphing between the star and the Venus head

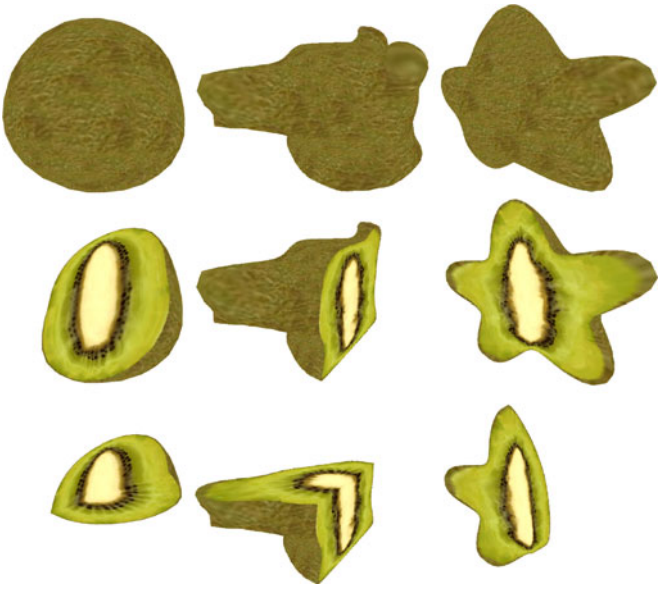


Fig. 7. Transferring the anisotropic solid texture from kiwi (column 1) to star shapes (column 2: dog head; column 3: star)

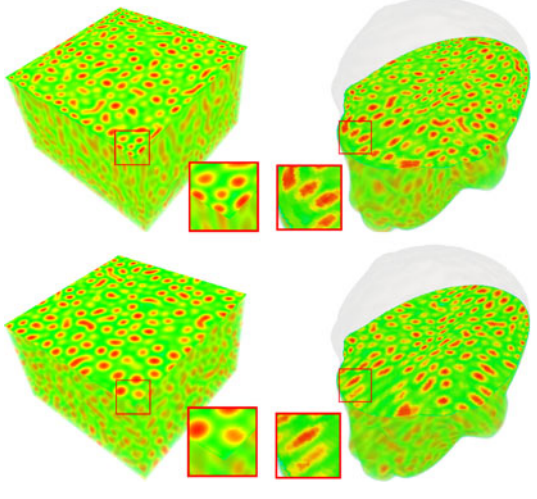


Fig. 8. Volumetric reaction-diffusion computation on the GPU. The first column shows the reaction-diffusion results in equilibrium state over the cube-based parametric domain, whereas the second column shows the reaction-diffusion results after mapped to the star-shaped model. Without distortion compensation by the metric matrix, we can see distortion in the lower right pattern, but such a distortion can be corrected (see upper row) if we take the metric matrix into account.

as a result, we can carry out the computation on the GPU and further accelerate the computation performance.

In detail, we first parameterize a given star-shape model by a cube shape so that the reaction-diffusion data (concentration values, etc.) can be naturally modeled as 3D textures stored in our GPU implementation. Here, we employ and extend Turing's reaction-diffusion model [27,28] to three-dimensional, and expectedly, 3D sphere-like spot patterns will be developed when the chemical concentrations reach a dynamic equilibrium state. Furthermore, we employ Witkin and Kass's method [29] to account for the distortion caused by the parameterization (since we compute the reaction-diffusion on the parameterization grid): Given the parameterization from star-shape to cube, we compute the local Jacobian per voxel element over the 3D parameterization grid; then, we can compute the metric tensor as a three-by-three matrix $M = J^T J$. Hence, we can adaptively and locally modify the rate of diffusion by the diagonal values in the metric matrix; this allows us to temper the reaction-diffusion pattern, thereby compensating the volumetric distortion in the parameterization. Figure 8 shows the reaction-diffusion results on the Venus head model.

6 Conclusion and Future Work

This paper presented a volume parameterization technique for star shapes. On the theoretical side, we showed that the Green's function in a star-shaped volume has a unique critical point and then give a constructive proof of the existence of a diffeomorphism between two star shapes. On the application side, we developed algorithms to parameterize star shapes to simple domains such as solid balls and star-shaped polycubes. We also applied the star shape parameterization to several applications, such as volumetric morphing, anisotropic solid texture transfer and GPU-based volumetric computing.

The proposed technique has several limitations that can lead to further investigations. First, the current framework only applies to star-shaped volumes. However, most real-world shapes are not star-shaped. One possible solution to parameterize volumes of arbitrary topology and geometry is to segment the shape into a set of disjoint star shapes, then parameterize each shape individually, and finally glue patches together with a certain order of continuity. As a future direction, we will develop automatic techniques to facilitate the segmentation and gluing procedures. Second, from the implementation point of view, we solve the Green's function using fundamental solution method, which requires solving a dense linear system. Thus, it is not efficient when the number of source points is too large.

Acknowledgements

This work was partially supported by AcRF RG69/07, AcRF RG13/08, NSF CCF-1081424, ONR N000140910228, CCF-0448399, and CCF-0830550. We would like to thank Kenshi Takayama for the anisotropic solid textures and the anonymous reviewers for their constructive comments. Special thanks go to one reviewer who pointed out the reference [20].

References

1. Floater, M.S., Hormann, K.: Surface parameterization: a tutorial and survey. In: Advances in Multiresolution for Geometric Modelling, pp. 157–186. Springer, Heidelberg (2005)
2. Sheffer, A., Praun, E., Rose, K.: Mesh parameterization methods and their applications. Foundations and Trends® in Computer Graphics and Vision 2(2) (2006)
3. Labelle, F., Shewchuk, J.R.: Isosurface stuffing: fast tetrahedral meshes with good dihedral angles. ACM Trans. Graph. 26(3), 57 (2007)
4. Alliez, P., Cohen-Steiner, D., Yvinec, M., Desbrun, M.: Variational tetrahedral meshing. ACM Trans. Graph. 24(3), 617–625 (2005)
5. Attali, D., Cohen-Steiner, D., Edelsbrunner, H.: Extraction and simplification of iso-surfaces in tandem. In: Symposium on Geometry Processing, pp. 139–148 (2005)
6. Zhou, K., Huang, J., Snyder, J., Liu, X., Bao, H., Guo, B., Shum, H.Y.: Large mesh deformation using the volumetric graph laplacian. ACM Trans. Graph. 24(3), 496–503 (2005)
7. Wang, Y., Gu, X., Thompson, P.M., Yau, S.T.: 3d harmonic mapping and tetrahedral meshing of brain imaging data. In: MICCAI (2004)
8. Li, X., Guo, X., Wang, H., He, Y., Gu, X., Qin, H.: Harmonic volumetric mapping for solid modeling applications. In: SPM, pp. 109–120 (2007)
9. Martin, T., Cohen, E., Kirby, M.: Volumetric parameterization and trivariate b-spline fitting using harmonic functions. In: Proceeding of Symposium on Solid and Physical Modeling (2008)
10. Ju, T., Schaefer, S., Warren, J.D.: Mean value coordinates for closed triangular meshes. ACM Trans. Graph. 24(3), 561–566 (2005)
11. Floater, M.S., Kós, G., Reimers, M.: Mean value coordinates in 3d. Computer Aided Geometric Design 22(7), 623–631 (2005)
12. Fairweather, G., Karageorghis, A.: The method of fundamental solution for elliptic boundary value problems. Advances in Computational Mathematics 9(1-2), 69–95 (1998)
13. Li, X., Guo, X., Wang, H., He, Y., Gu, X., Qin, H.: Meshless harmonic volumetric mapping using fundamental solution methods. IEEE Transactions on Automation Science and Engineering 6(3), 409–422 (2009)
14. Tarini, M., Hormann, K., Cignoni, P., Montani, C.: Polycube-maps. ACM Trans. Graph. 23(3), 853–860 (2004)
15. Wang, H., He, Y., Li, X., Gu, X., Qin, H.: Polycube splines. Computer-Aided Design 40(6), 721–733 (2008)
16. Wang, H., Jin, M., He, Y., Gu, X., Qin, H.: User-controllable polycube map for manifold spline construction. In: ACM Symposium on Solid and Physical Modeling (SPM 2008), pp. 397–404 (2008)
17. Lin, J., Jin, X., Fan, Z., Wang, C.C.L.: Automatic polycube-maps. In: Chen, F., Jüttler, B. (eds.) GMP 2008. LNCS, vol. 4975, pp. 3–16. Springer, Heidelberg (2008)
18. He, Y., Wang, H., Fu, C.W., Qin, H.: A divide-and-conquer approach for automatic polycube map construction. Comput. Graph. 33(3), 369–380 (2009)
19. Weber, H.J., Arfken, G.B.: Mathematical Methods For Physicists. Academic Press, London (2005)
20. Gergen, J.J.: Note on the green function of a star-shaped three dimensional region. American Journal of Mathematics 53(4), 746–752 (1931)
21. MOSEK, http://www.mosek.com/
22. Gu, X., Wang, Y., Chan, T.F., Thompson, P.M., Yau, S.T.: Genus zero surface conformal mapping and its application to brain surface mapping. TMI 23(8), 949–958 (2004)
23. Si, H.: Tetgen: A quality tetrahedral mesh generator and three-dimensional delaunay triangulator, http://tetgen.berlios.de/

24. Kopf, J., Fu, C.W., Cohen-Or, D., Deussen, O., Lischinski, D., Wong, T.T.: Solid texture synthesis from 2d exemplars. ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007) 26(3), 2:1–2:9 (2007)
25. Takayama, K., Okabe, M., Ijiri, T., Igarashi, T.: Lapped solid textures: filling a model with anisotropic textures. ACM Trans. Graph. 27(3), 1–9 (2008)
26. Praun, E., Finkelstein, A., Hoppe, H.: Lapped textures. In: SIGGRAPH, pp. 465–470 (2000)
27. Turing, A.: The chemical basis of morphogenesis. Royal Society of London Philosophical Transactions Series B 237, 37–72 (1952)
28. Turk, G.: Generating textures on arbitrary surfaces using reaction-diffusion. In: SIGGRAPH, pp. 289–298 (1991)
29. Witkin, A.P., Kass, M.: Reaction-diffusion textures. In: SIGGRAPH, pp. 299–308 (1991)

Appendix

We prove the main theoretical results in this appendix.

Lemma 1. *A volume M is a star shape if and only if there exists a point $\mathbf{c} \in M$ such that for any boundary point $\mathbf{p} \in \partial M$,*

$$(\mathbf{c}-\mathbf{p}, \mathbf{n}(\mathbf{p})) \leq 0, \tag{3}$$

where $\mathbf{n}(\mathbf{p})$ is the normal vector at the point $\mathbf{p}$.

Proof. Assume the boundary surface ∂M is represented by the zero level set of an implicit function $f : \mathbb{R}^3 \to \mathbb{R}$, i.e., $\partial M = f^{-1}(0)$ and the interior points $\mathbf{r} \in M$ satisfy $f(\mathbf{r}) < 0$.

($\Longrightarrow$ necessary condition) If M is a star shape, for any boundary point $\mathbf{p} \in M$, the ray $\mathbf{p}-\mathbf{c}$ intersects ∂M only once and the intersection point is $\mathbf{p}$. Thus, for any $\varepsilon \in [0,1]$, the point $\mathbf{q} = \mathbf{p} + \varepsilon\mathbf{c} - \mathbf{p} \in M$ is inside M. Then, for a small $\varepsilon > 0$,

$$f(\mathbf{q}) = f(\mathbf{p}) + \varepsilon \bigtriangledown f(\mathbf{p}) \cdot (\mathbf{c}-\mathbf{p}) + O(\varepsilon^2 \|\mathbf{c}-\mathbf{p}\|^2).$$

Note that $f(\mathbf{p}) = 0$ and $f(\mathbf{q}_\varepsilon) \leq 0$, thus, $\bigtriangledown f(\mathbf{p}) \cdot (\mathbf{q}_\varepsilon - \mathbf{p}) \leq 0$. Since $\bigtriangledown f(\mathbf{p})$ points to the normal direction $\mathbf{n}(\mathbf{p})$, and $\mathbf{c}-\mathbf{p}$ has the same direction as $\mathbf{q}_\varepsilon - \mathbf{p}$, then $(\mathbf{c}-\mathbf{p}, \mathbf{n}(\mathbf{p})) \leq 0$.

($\Longleftarrow$ sufficient condition) Given a point $\mathbf{c} \in M$, for every boundary point $\mathbf{p}$, $(\mathbf{c}-\mathbf{p}, \mathbf{n}(\mathbf{p})) \leq 0$ holds. Assume M is not a star shape, then there exists a ray from $\mathbf{c}$ which intersects ∂M at least twice. Without loss of generality, say $\mathbf{p}_1$ and $\mathbf{p}_2$ are the first two intersection points and $\mathbf{p}_1$ is closer to $\mathbf{c}$. Consider a point $\mathbf{q} = \varepsilon(\mathbf{p}_1 - \mathbf{p}_2) + \mathbf{p}_2$ with $\varepsilon > 0$. Clearly, $\mathbf{q}$ is on the segment $\mathbf{p}_1\mathbf{p}_2$ and out of M. Thus, $f(\mathbf{q}) > 0$. Using Taylor expansion,

$$f(\mathbf{q}) = f(\mathbf{p}_2) + \varepsilon \bigtriangledown f(\mathbf{p}_2) \cdot (\mathbf{p}_1 - \mathbf{p}_2) + O(\varepsilon^2 \|\mathbf{p}_1 - \mathbf{p}_2\|^2).$$

Note that $f(\mathbf{p}_2) = 0$ and $\bigtriangledown f(\mathbf{p}_2)$ points to the same direction as normal $\mathbf{n}(\mathbf{p}_2)$, $\mathbf{p}_1 - \mathbf{p}_2$ points to the same direction as $\mathbf{c} - \mathbf{p}_2$, thus, $\bigtriangledown f(\mathbf{p}_2) \cdot (\mathbf{p}_1 - \mathbf{p}_2) \leq 0$ and $f(\mathbf{q}) \leq 0$, contradiction! Q.E.D.

Lemma 2 [Green's function on a star shape]. *Suppose M is a star shape with a center $c \in M$, G is the Green's function with a pole at c, then c is the only critical point of G.*

Proof. Without loss of generality, we assume c is at the origin in $\mathbb{R}^3$. Let $B(c,\varepsilon)$ be a small ball centered at c with radius ε. Consider the following function, the inner product of the point $p=(x_1,x_2,x_3)$ and the gradient of G at p,

$$f(p)=(p,\nabla G)=x_1\frac{\partial G}{\partial x_1}+x_2\frac{\partial G}{\partial x_2}+x_3\frac{\partial G}{\partial x_3}.$$

By direct computation, it is easy to verify that

$$\Delta f=(\sum_k\frac{\partial^2}{\partial x_k^2})(\sum_i x_i\frac{\partial G}{\partial x_i})=0.$$

In details,

$$\frac{\partial^2}{\partial x_k^2}(\sum_i x_i\frac{\partial G}{\partial x_i})=2\frac{\partial^2 G}{\partial x_k^2}+\sum_i x_i\frac{\partial^3 G}{\partial x_k^2\partial x_i},$$

therefore

$$(\sum_k\frac{\partial^2}{\partial x_k^2})(\sum_i x_i\frac{\partial G}{\partial x_i})=2\Delta G+\sum_i x_i\Delta\frac{\partial G}{\partial x_i}.$$

Because G is harmonic, therefore, $\frac{\partial G}{\partial x_i}$ is also harmonic, and the above equation equals zero.

Therefore $f(p)$ is a harmonic function on $M/B(c,\varepsilon)$. According to the maximum principle of harmonic maps, f reaches its max and min values on the boundary surfaces ∂M and $\partial B(c,\varepsilon)$. Here by definition, and c is the pole of f, f is negative on $\partial B(c,\varepsilon)$. Because M is a star shape, on ∂M, $(\mathbf{n},p)>0$, where $\mathbf{n}$ is the normal on p to ∂M. ∇G is orthogonal to ∂M and is on the opposite direction of $\mathbf{n}$. Therefore, f is always negative in the whole volume $M/B(c,\varepsilon)$, ∇G is non-zero in $M/B(c,\varepsilon)$. Since ε is arbitrary, ∇G is non-zero for all points in $M/\{c\}$. We conclude that G has no critical points in M except c. Q.E.D.

Lemma 3. *Suppose M is a star shape with a center $c\in M$, G is the Green's function with a pole at c. Then for any $r\in\mathbb{R}^+$, the level set $G^{-1}(r)$ is a topological sphere.*

Proof. Let $r\in\mathbb{R}^+$, $G^{-1}(r)$ is the level set of G. $G^{-1}(0)$ is the boundary of M, ∂M, which is a topological sphere. By lemma 2, there is no critical points in $G^{-1}([0,r])$. According to Morse theory, $G^{-1}(r)$ and $G^{-1}(0)$ share the same topology. In fact, we can start from a point $p\in G^{-1}(r)$ and trace along the integration curve of the gradient of G and reach a unique point q on $G^{-1}(0)$, this gives us a diffeomorphism from $G^{-1}(r)$ to $G^{-1}(0)$. Q.E.D.

Theorem 1. *Suppose M and $\tilde{M}$ are star-shaped volumes with centers c and $\tilde{c}$, G and $\tilde{G}$ are Green's functions with the poles at c and $\tilde{c}$ respectively. The Green's functions induce foliations. If the boundary map $\partial M\to\partial\tilde{M}$ is a diffeomorphism, the map $M\to\tilde{M}$ constructed using the foliations is a diffeomorphism.*

Proof. We first introduce the concepts of *foliation* and *leaf*.

A dimension *m foliation* of an *n*-dimensional manifold M is a covering by charts U_i together with maps $\phi_i : U_i \to \mathbb{R}^n$, such that on the overlaps $U_i \cap U_j$, the transition functions $\phi_{ij} = \phi_j \circ \phi_i^{-1}$ take the form

$$\phi_{ij}(x,y) = (\phi_{ij}^1(x), \phi_{ij}^2(x,y))$$

where x denotes the first $n-m$ coordinates, y denotes the last m coordinates. In each chart U_i the $x = const$ stripes match up with the stripes on U_j. The stripes piece together from chart to chart to form maximal connected injectively immersed submanifolds called the *leaves*.

The we show the proof. Let F_1 be the foliation of M by topological spheres induced by the level sets of G, F_2 be the foliation of M induced by the gradient lines of G. We choose an open cover of $M/\{c\}$, $\{(U_\alpha, \phi_\alpha)\}$, U_α is the union for leaves in F_2, $U_\alpha = \cup f, f \in F_2$, such that $\phi_\alpha : U_\alpha \to \mathbb{R}^3$, leaves in F_1 are mapped to the planes $z = const$, leaves in F_2 are mapped to lines $(x,y) = const$. $\{(U_\alpha, \phi_\alpha)\}$ is a differential atlas. Similarly, we can construct a differential atlas of $\tilde{M}/\{\tilde{c}\}$, $\{\tilde{U}, \tilde{\phi}_\beta\}$, the level sets and the integration lines are mapped to canonical planes orthogonal to the z-axis and lines parallel to the z-axis.

The restriction of the map $f : M \to \tilde{M}$ on the local coordinate system

$$f_{\alpha\beta} = \tilde{\phi}_\beta \circ f \circ \phi_\alpha^{-1} : \phi_\alpha(U_\alpha) \to \tilde{\phi}_\beta(\tilde{U}_\beta)$$

has the following form

$$f_{\alpha\beta}(x,y,z) = (g(x,y), z),$$

where $g(x,y)$ is determined by the restriction of f on the boundary, $f|_{\partial M} : \partial M \to \partial \tilde{M}$. The restriction is a diffeomorphism, therefore $g(x,y)$ is a diffeomorphism, and $f_{\alpha\beta}$ is a diffeomorphism. Because U_α and $\tilde{U}_\beta$ is arbitrarily chosen, f itself is a diffeomorphism. Q.E.D.

Optimal Analysis-Aware Parameterization of Computational Domain in Isogeometric Analysis

Gang Xu[1], Bernard Mourrain[1], Régis Duvigneau[2], and André Galligo[3]

[1] GALAAD, INRIA Sophia-Antipolis, 2004 Route des Lucioles, 06902 Cedex, France
[2] OPALE, INRIA Sophia-Antipolis, 2004 Route des Lucioles, 06902 Cedex, France
Firstname.Lastname@sophia.inria.fr
[3] University of Nice Sophia-Antipolis, 06108 Nice Cedex 02, France
galligo@unice.fr

Abstract. In isogeometric analysis (IGA for short) framework, computational domain is exactly described using the same representation as that employed in the CAD process. For a CAD object, we can construct various computational domain with same shape but with different parameterization. One basic requirement is that the resulting parameterization should have no self-intersections. In this paper, a linear and easy-to-check sufficient condition for injectivity of planar B-spline parameterization is proposed. By an example of 2D thermal conduction problem, we show that different parameterization of computational domain has different impact on the simulation result and efficiency in IGA. For problems with exact solutions, we propose a shape optimization method to obtain optimal parameterization of computational domain. The proposed injective condition is used to check the injectivity of initial parameterization constructed by discrete Coons method. Several examples and comparisons are presented to show the effectiveness of the proposed method. Compared with the initial parameterization during refinement, the optimal parameterization can achieve the same accuracy but with less degrees of freedom.

Keywords: isogeometric analysis; analysis-aware parameterization of computational domain, injectivity, shape optimization, steepest descent method.

1 Introduction

CAGD software usually relies on splines or NURBS representations, but the analysis software for CAD object uses mesh-based geometric descriptions (structured or unstructured). Therefore, in conventional approaches, several information transfers occur during the design phase, yielding approximations and non-linear transformations that can significantly deteriorate the overall efficiency of the design optimization procedure.

The isogeometric approach proposed by Hughes et al. [19] is employed to overcome this difficulty by using CAD standards as unique representation for all disciplines. The isogeometric analysis consists in developing methods that use NURBS representations for all design and analysis tasks:

B. Mourrain, S. Schaefer, and G. Xu (Eds.): GMP 2010, LNCS 6130, pp. 236–254, 2010.

- the geometry is defined by NURBS curves or surfaces;
- the computation domain is defined by planar NURBS surfaces or NURBS volumes instead of discrete meshes;
- the solution fields are obtained by using a finite-element approach that uses NURBS basis functions instead of classical Lagrange polynomials;
- the optimizer controls directly NURBS control points.

This framework allows to compute the analysis solution on the exact geometry (not a discretized geometry), obtain a more accurate solution (high-order approximation), reduce spurious numerical sources of noise that deteriorate convergence, avoid data transfers between the design and analysis phases. Moreover, NURBS representation is naturally hierarchical and allows to perform refinement operations to improve the analysis result.

In finite element analysis (FEA), mesh generation, which generates discrete geometry as computational domain from given CAD object, is a key and the most time-consuming step. In IGA framework, parameterization of computational domain, which corresponds to the mesh generation in FEA, also has some impact on analysis result and efficiency. Moreover, in FEA, one can perform arbitrary refinements on the computational mesh, but in IGA using tensor product B-splines, the refinement is not arbitrary, we can only perform refinement operations in u direction and v direction by knot insertion or degree evaluation. Hence, parameterization of computational domain is more important in IGA.

The parameterization of a computational domain in IGA is determined by control points, knot vectors and the degrees of B-spline objects. For IGA problem of two dimension, the knot vectors and the degree of computational domain are determined by the given boundary curves. Hence, finding the optimal placement of inner control points for a specified physical problem, is a key issue in IGA. A basic requirement of resulting parameterization for IGA is that it doesn't have self-intersections. In this paper, we first propose a linear and easy-to-test sufficient condition for injectivity of planar B-spline parameterization. Then we show that different parameterizations of computational domain has different impact on the simulation results in IGA. For problems with exact solutions, a shape optimization method is proposed to obtain an optimal parameterization of computational domain. Some examples and comparisons are presented based on the heat conduction problem to show the effectiveness of the proposed method.

The remainder of the paper is organized as follows. Section 2 reviews the related work in isogeometric analysis. Section 3 proposes the linear sufficient conditions for injectivity of planar B-spline parameterization. Section 4 describes a test IGA model and shows the impact of different parameterizations of computational domain. Section 5 presents the shape optimization method to obtain an optimal parameterization of a computational domain. Some examples and comparisons are also presented in Section 5. Finally, we conclude this paper and outline future works in Section 6.

2 Related Work

In this section, we review related works in IGA and parameterization of computational domains.

The concept of IGA was firstly proposed by T.R Hughes et al. [19] in 2005 to achieve the seamless integration of CAD and FEA. Since then, many researchers in the fields of mechanical engineering and geometric modeling were involved in this topic. The current work on isogeometric analysis can be classified into three categories: (1) application of IGA to various simulation problems [2,5,6,10,14,18,20,27,28]; (2) application of various geometric modeling tools to IGA [7,12,24]; (3) accuracy and efficiency improvement of IGA framework by reparameterization and refinement operations [1,3,8,9,15,21,25].

The topic of this paper belongs to the third field. As far as we know, there are few works on the parametrizations of computational domains for IGA. T. Martin et al. [25] proposed a method to fit a genus-0 triangular mesh by B-spline volume parameterization, based on discrete volumetric harmonic functions; this can be used to build computational domains for 3D IGA problems. A variational approach for constructing NURBS parameterization of swept volumes is proposed by M. Aigner et al [1]. Many free-form shapes in CAD systems, such as blades of turbines and propellers, are covered by this kind of volumes. E. Cohen et.al. [8] proposed the concept of *analysis-aware modeling*, in which the parameters of CAD models should be selected to facilitate isogeometric analysis. They also demonstrated the influence of parameterization of computational domains by several examples. In this paper, a method for generating optimal analysis-aware parameterization of computational domain is proposed based on shape optimization method.

3 A Linear Sufficient Condition for Injectivity of Planar B-spline Parameterization

The main idea of the isogeometric approach is to use the same representation for the geometry and the physical solutions we are interested in. Schematically, the geometry Ω involved in the physical problem can be a surface or a volume in a three-dimensional space $\mathbb{R}^3$. Let us call $\mathbf{x} = (x, y, z)$ the coordinates associated to this space. In our case, this geometry will be represented by a parameterization σ for a domain $\mathcal{P}$ of the parameter space. Let us call $\mathbf{u}$ the coordinates of this parameter domain, which could be of dimension 2 for a surface or 3 for a volume. This parameterization will be given by B-spline functions with knots in $\mathcal{P}$ and control points in $\mathbb{R}^3$.

The concept of isogeometry consists in representing the physical quantities $\Phi \in \mathbb{R}^p$ on the geometry Ω using the same type of B-spline representation as for the geometry Ω. In other words, given a point $\mathbf{x} = \sigma(\mathbf{u}) \in \Omega$ with $\mathbf{u} \in \mathcal{P}$, we associate to it the physical quantities $\Phi(\mathbf{u})$ where $\Phi(\mathbf{u})$ is a B-spline function with nodes in $\mathcal{P}$ and control points in $\mathbb{R}^p$. This means that the map $\mathbf{x} \in \Omega \mapsto \Phi \in \mathbb{R}^p$ is defined *implicitly* as $\mathbf{x} \mapsto \Phi \circ \sigma^{-1}(\mathbf{x})$.

Consequently, the framework of isogeometry is thus valid when the parameterization σ of the geometry is *injective* (or bijective on its image). We are going to describe sufficient and easy-to-check conditions for the injectivity of σ. We will consider this problem in the context of finding a "good" parameterization of a domain when its boundary is given. In [23], a general sufficient condition is proposed for injective parameterization.

Proposition 1. *Suppose that σ is a C^1 parameterization from a compact domain $\mathcal{P} \subset \mathbb{R}^n$ with a connected boundary to a geometry $\Omega \subset \mathbb{R}^n$. If σ is injective on the boundary $\partial\mathcal{P}$ of $\mathcal{P}$ and its Jacobian J_σ does not vanish on $\mathcal{P}$, then σ is injective.*

For a parameterization σ from $[a,b] \times [c,d]$ to $\Omega \subset \mathbb{R}^2$, we define the boundary curves as the image of $\{a\} \times [c,d], \{b\} \times [c,d], [a,b] \times \{c\}, [a,b] \times \{b\}$ by σ. We say that σ defines a *regular boundary* if these curves do not intersect pairwise, except at their end points and if they have no self-intersection.

As a consequence of the previous proposition, we get the following injectivity test for standard B-spline tensor product parameterization of a planar domain.

Proposition 2. *Let σ be a C^1 parameterization from $[a,b] \times [c,d]$ to $\Omega \subset \mathbb{R}^2$ which defines a regular boundary. If its Jacobian J_σ does not vanish on $[a,b] \times [c,d]$, then σ is injective.*

These tests involve injectivity conditions on the boundary, which can be checked recursively using the same techniques, non-intersection tests for boundary curves and surfaces which are provided for instance by geometric (subdivision) algorithms and the local injectivity condition corresponding to the non-vanishing of the Jacobian. This last condition requires to test on all the domain Ω that the Jacobian does not vanish. Hereafter we propose a sufficient and easy-to-test condition to ensure the local injectivity condition.

We consider first the case of a planar parameterization

$$\sigma : \mathbf{u} \in \mathcal{P} := [a,b] \times [c,d] \mapsto \sigma(\mathbf{u}) := \sum_{0 \le i \le l_1, 0 \le j \le l_2} \mathbf{c}_{i,j} N_{i,j}(\mathbf{u}),$$

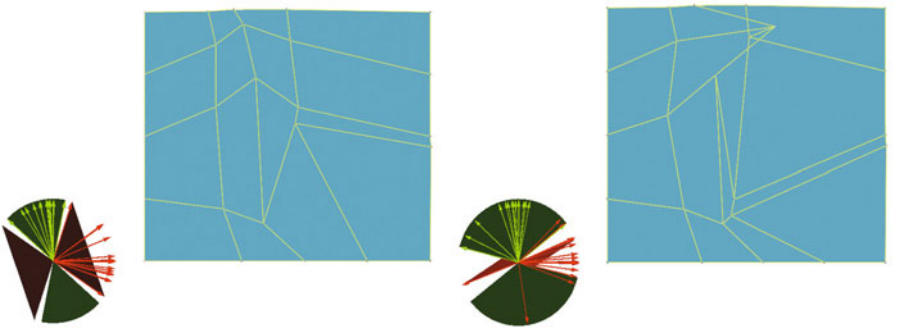
where $\mathbf{c}_{i,j} \in \mathbb{R}^2$ are the control points and $N_{i,j}(\mathbf{u})$ are the B-spline basis functions. The derivative of $\sigma(\mathbf{u})$ with respect to $\mathbf{u}_1$ can be expressed in terms of the differences $\Delta^1_{i,j} := \mathbf{c}_{i+1,j} - \mathbf{c}_{i,j}$:

$$\partial_{u_1}\sigma(\mathbf{u}) := \sum_{0 \le i \le l_1 - 1, 0 \le j \le l_2} \omega^1_{i,j} \Delta^1_{i,j} N^1_{i,j}(\mathbf{u}),$$

where $N^1_{i,j}$ is the B-spline basis function with one degree less in $\mathbf{u}_1$, $\omega^1_{i,j}$ is a positive factor. We denote by $\mathcal{C}_1(\mathbf{c})$ the convex cone of $\mathbb{R}^2$ generated by the half rays $\mathbb{R}_+ \cdot \Delta^1_{i,j}$.

Similarly, the derivative of $\sigma(\mathbf{u})$ with respect to $\mathbf{u}_2$ can be expressed in terms of the differences $\Delta^2_{i,j} := \mathbf{c}_{i,j+1} - \mathbf{c}_{i,j}$:

$$\partial_{u_2}\sigma(\mathbf{u}) := \sum_{0 \le i \le l_1 - 1, 0 \le j \le l_2 - 1} \omega^2_{i,j} \Delta^2_{i,j} N^2_{i,j}(\mathbf{u}),$$



(a) transverse cone (b) non-transverse cone

Fig. 1. Injectivity test by cones

where $N^2_{i,j}$ is the B-spline basis with one degree less in $\mathbf{u}_2$, $\omega^2_{i,j}$ is a positive factor. We denote by $\mathcal{C}_2(\mathbf{c})$ the convex cone of $\mathbb{R}^2$ generated by the half rays $\mathbb{R}_+ \cdot \Delta^2_{i,j}$. If there exist two opposite vectors, which are on a straight line, we define $\mathcal{C}_i(\mathbf{c})$ as a half-plane.

We say that two cones $\mathcal{C}_1, \mathcal{C}_2$ are *transverse* if $\mathbb{R} \cdot \mathcal{C}_1$ and $\mathbb{R} \cdot \mathcal{C}_2$ intersect only at $\{0\}$.

Proposition 3. *Let σ be a B-spline parametrisation, which is at least C^1 from $\mathcal{P} := [a, b] \times [c, d]$ to $\Omega \subset \mathbb{R}^2$ given by the control points $\mathbf{c}$. If the boundary curves do not intersect and have no self-intersection point and the cones $\mathcal{C}_1(\mathbf{c}), \mathcal{C}_2(\mathbf{c})$ are transverse, then σ is injective on $\mathcal{P}$.*

Proof. We check first that the transversality of the cones $\mathcal{C}_1(\mathbf{c}), \mathcal{C}_2(\mathbf{c})$ implies that the Jacobian of σ is not vanishing. This jacobian $J_\sigma(\mathbf{u})$ is obtained by taking the determinant $|\partial_{\mathbf{u}_1}\sigma, \partial_{\mathbf{u}_2}\sigma|$ which expands as

$$\sum_{0 \le i \le l_1 - 1, 0 \le j \le l_2} \sum_{0 \le i' \le l_1 - 1, 0 \le j' \le l_2 - 1} |\Delta^1_{i,j}, \Delta^2_{i',j'}| \omega^1_{i,j} \omega^2_{i',j'} N^1_{i,j}(\mathbf{u}) N^2_{i',j'}(\mathbf{u}).$$

Since the cone $\mathcal{C}_1(\mathbf{c})$ and $\mathcal{C}_2(\mathbf{c})$ are transverse, the determinants $|\Delta^1_{i,j}, \Delta^2_{i',j'}|$ have a constant sign for $\Delta^1_{i,j} \in \mathcal{C}_1(\mathbf{c})$, $\Delta^2_{i',j'} \in \mathcal{C}_2(\mathbf{c})$. As the basis functions and the factors are positive, the Jacobian $J_\sigma(\mathbf{u})$ cannot vanish at $\mathbf{u} \in \mathcal{G}$, except if all the $N^1_{i,j}(\mathbf{u})\, N^2_{i',j'}(\mathbf{u})$ vanish, which is not possible.

The map σ is locally injective on $\mathcal{P}$. By Proposition 2, we deduce that σ is globally injective on $\mathcal{P}$. □

Fig.1 shows two examples of the injectivity testing method. In Fig.1 (a), it satisfies the sufficient condition in our method, but it does not satisfy the sufficient condition of the method proposed in [17]. Hence, our method is an improved version of the method presented in [17].

Linear constraint for injectivity. This condition can be used to devise an algorithm which constructs an injective parameterization for given boundary control points. We first consider the planar case. Given four planar boundary curves described by the controls points $\mathbf{c}_{i,0}, \mathbf{c}_{i,l_2}, \mathbf{c}_{0,j}, \mathbf{c}_{l_1,j}$, with $0 \leq i \leq l_1, 0 \leq j \leq l_2$, we define the boundary cone $\mathcal{C}_1^0(\mathbf{c})$ (resp. $\mathcal{C}_2^0(\mathbf{c})$) as the cone generated by the vectors $\Delta_{i,0}^1(\mathbf{c})$, $\Delta_{i,l_2}^1(\mathbf{c})$ for $0 \leq i \leq l_1 - 1$ (resp. $\Delta_{0,j}^2(\mathbf{c})$, $\Delta_{l_2,j}^2(\mathbf{c})$ for $0 \leq j \leq l_2 - 1$). We assume that these boundary curves form a regular boundary and that the two boundary cones $\mathcal{C}_1^0(\mathbf{c})$, $\mathcal{C}_2^0(\mathbf{c})$ are transverse. $\mathbb{R} \cdot \mathcal{C}_1^0(\mathbf{c})$ is the cone defined by $F_1^+(\mathcal{C}_1^0(\mathbf{c})) \leq 0, F_1^-(\mathcal{C}_1^0(\mathbf{c})) \leq 0$, where F_1^+ and F_1^- are the linear equations defining the boundary of $\mathbb{R} \cdot \mathcal{C}_1^0(\mathbf{c})$. We defined similarly F_2^+, F_2^- for $\mathcal{C}_2^0(\mathbf{c})$.

To apply Proposition 3, the inner control points $\mathbf{c}_{i,j}$ should satisfy the following linear constraints for injective parameterization:

$$\begin{cases} F_1^+(\mathbf{c}_{i+1,j} - \mathbf{c}_{i,j}) \leq 0, F_1^-(\mathbf{c}_{i+1,j} - \mathbf{c}_{i,j}) \leq 0, 0 \leq i < l_1, 0 < j < l_2 \\ F_2^+(\mathbf{c}_{i,j+1} - \mathbf{c}_{i,j}) \leq 0, F_2^-(\mathbf{c}_{i,j+1} - \mathbf{c}_{i,j}) \leq 0, 0 < i < l_1, 0 \leq j < l_2. \end{cases} \tag{1}$$

The linear condition in (1) is a rather restrictive condition, and it is sufficient to require that the two cones constructed from the first derivative vectors are separated. Inspired from [22], the following constraints are proposed as alternative condition

$$\begin{cases} F_2^+(\mathbf{c}_{i+1,j} - \mathbf{c}_{i,j}) + F_1^-(\mathbf{c}_{i+1,j} - \mathbf{c}_{i,j}) \leq 0, F_2^-(\mathbf{c}_{i+1,j} - \mathbf{c}_{i,j}) + F_1^+(\mathbf{c}_{i+1,j} - \mathbf{c}_{i,j}) \geq 0, \\ F_2^+(\mathbf{c}_{i,j+1} - \mathbf{c}_{i,j}) + F_1^-(\mathbf{c}_{i,j+1} - \mathbf{c}_{i,j}) \geq 0, F_2^-(\mathbf{c}_{i,j+1} - \mathbf{c}_{i,j}) + F_1^+(\mathbf{c}_{i,j+1} - \mathbf{c}_{i,j}) \geq 0, \end{cases}$$

where $0 < i < l_1, 0 \leq j < l_2$.

Remarks 1. For 3D case, the 3D convex cones can be also constructed from the derivative vectors in three parametric directions. The difference is that the cross product condition should be considered in the injectivity condition as in [17].

These conditions provide an easy-to-check method for the injectivity of a parameterization. In Section 5, we will employ it to check the injectivity of initial parameterization.

4 Isogeometric Analysis and Parameterization of Computational Domain

In this section, we aim at presenting the reasons why solutions from IGA depend strongly on the choice of the parameterization. This will be illustrated by a heat conduction problem.

4.1 Test Model — Heat Conduction Problem

Given a domain Ω with $\Gamma = \partial\Omega_D \cup \partial\Omega_N$, we consider the following thermal conduction problem:

$$\begin{aligned} &\boldsymbol{\nabla}(\kappa(\mathbf{x})\boldsymbol{\nabla}T(\mathbf{x})) = f(\mathbf{x}) \quad \text{in } \Omega \\ &T(\mathbf{x}) = T_0(\mathbf{x}) \text{ on } \partial\Omega_D \\ &\kappa(\mathbf{x})\frac{\partial T}{\partial \mathbf{n}}(\mathbf{x}) = \Phi_0(\mathbf{x}) \text{ on } \partial\Omega_N, \end{aligned} \tag{2}$$

where $\mathbf{x}$ are the Cartesian coordinates, T represents the temperature field and κ the thermal conductivity. Dirichlet and Neumann boundary conditions are applied on $\partial\Omega_D$ and $\partial\Omega_N$ respectively, T_0 and Φ_0 being the imposed temperature and thermal flux ($\mathbf{n}$ unit vector normal to the boundary). f is a user-defined function that allows to generate problems with an analytical solution, by adding a source term to the classical heat conduction equation.

According to a classical variational approach, we seek for a solution $T \in H^1(\Omega)$, such as $T(\mathbf{x}) = T_0(\mathbf{x})$ on $\partial\Omega_D$ and:

$$\int_\Omega \boldsymbol{\nabla}(\kappa(\mathbf{x})\boldsymbol{\nabla}T(\mathbf{x}))\ \psi(\mathbf{x})\ d\Omega = \int_\Omega f(\mathbf{x})\ \psi(\mathbf{x})\ d\Omega \quad \forall\psi \in H^1_{\partial\Omega_D}(\Omega),$$

where $\psi(\mathbf{x})$ are test functions. After integrating by parts and using boundary conditions, we obtain:

$$-\int_\Omega \kappa(\mathbf{x})\boldsymbol{\nabla}T(\mathbf{x})\ \boldsymbol{\nabla}\psi(\mathbf{x})\ d\Omega + \int_{\partial\Omega_N} \Phi_0(\mathbf{x})\ \psi(\mathbf{x})\ d\Gamma = \int_\Omega f(\mathbf{x})\ \psi(\mathbf{x})\ d\Omega. \quad (3)$$

According to the IGA paradigm, the temperature field is represented using B-spline basis functions. For a 2D problem, we have:

$$T(\xi,\eta) = \sum_{i=1}^{n_i}\sum_{j=1}^{n_j} \hat{N}_i^{p_i}(\xi)\,\hat{N}_j^{p_j}(\eta)\,T_{ij},$$

where $\hat{N}_i$ functions are B-Spline basis functions and $\mathbf{u} = (\xi,\eta) \in \mathcal{P}$ are domain parameters. Then, we define the test functions $\psi(\mathbf{x})$ in the physical domain such as:

$$N_{ij}(\mathbf{x}) = N_{ij}(x,y) = N_{ij}(T(\xi,\eta)) = \hat{N}_{ij}(\xi,\eta) = \hat{N}_i^{p_i}(\xi)\,\hat{N}_j^{p_j}(\eta).$$

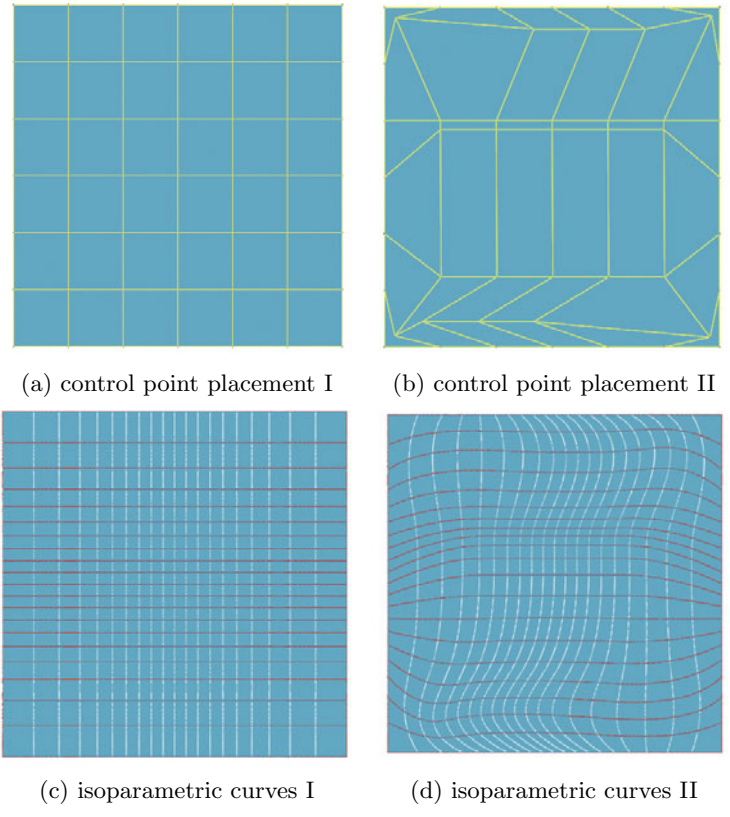
The weak formulation Eq. 3 reads:

$$\sum_{k=1}^{n_k}\sum_{l=1}^{n_l} T_{kl}\int_\Omega \kappa(\mathbf{x})\boldsymbol{\nabla}N_{kl}(\mathbf{x})\ \boldsymbol{\nabla}N_{ij}(\mathbf{x})\ d\Omega = \int_{\partial\Omega_N} \Phi_0(\mathbf{x})\ N_{ij}(\mathbf{x})\ d\Gamma + \int_\Omega f(\mathbf{x})\ N_{ij}(\mathbf{x})\ d\Omega.$$

Finally, we obtain a linear system similar to that resulting from the classical finite-element methods, with a matrix and a right-hand side defined as:

$$\begin{aligned} M_{ij,kl} &= \int_\Omega \kappa(\mathbf{x})\boldsymbol{\nabla}N_{kl}(\mathbf{x})\ \boldsymbol{\nabla}N_{ij}(\mathbf{x})\ d\Omega \\ &= \int_\mathcal{P} \kappa(T(\mathbf{u}))\boldsymbol{\nabla}_\mathbf{u}\tilde{N}_{kl}(\mathbf{u})B(\mathbf{u})^T B(\mathbf{u})\ \boldsymbol{\nabla}_\mathbf{u}\tilde{N}_{kl}(\mathbf{u})J(\mathbf{u})\ d\mathcal{P} \\ S_{ij} &= \int_{\partial\Omega_N} \Phi_0(\mathbf{x})\ N_{ij}(\mathbf{x})\ d\Gamma + \int_\Omega f(\mathbf{x})\ N_{ij}(\mathbf{x})\ d\Omega \\ &= \int_{\partial\mathcal{P}_N} \Phi_0(T(\mathbf{u}))\ \tilde{N}_{kl}(\mathbf{u})J(\mathbf{u})\ d\tilde{\Gamma} + \int_\mathcal{P} f(T(\mathbf{u}))\ \tilde{N}_{kl}(\mathbf{u})J(\mathbf{u})\ d\mathcal{P}. \end{aligned}$$

where J is the Jacobian of the transformation, B^K is the transposed of the inverse of the Jacobian matrix. The above integrations are performed in the parameter space using classical Gauss quadrature rules.



(a) control point placement I (b) control point placement II

(c) isoparametric curves I (d) isoparametric curves II

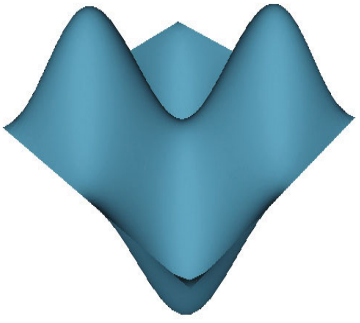
Fig. 2. Two different parameterizations of computational domains. (a),(b): two different placements of inner control points. (c), (d): isoparametric curves on the computational domain with respect to the control points placements in (a) and (b).

Starting from a planar B-spline surface as computational domain, a general framework of an isogeometric solver for thermal conduction problem (2) has been implemented as plugins in the AXEL[1] platform, yielding a B-spline surface as solution field. Gauss-Seidel algorithm is employed to solve the linear system. In order to improve the simulation results, refinement operation can be performed for two parametric directions. Additional details concerning the methods can be found in [13].

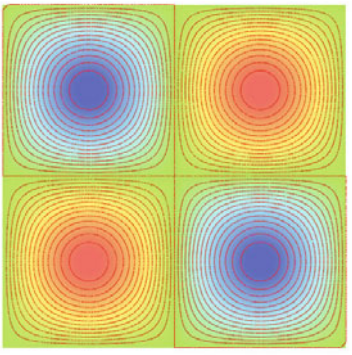
4.2 Isogeometric Analysis with Different Parameterization

As mentioned above, given four boundary planar B-spline curves, we can construct various planar B-spline surfaces with different parameterizations. For Example I in Fig. 2, we present two kinds of parameterization for a computational domain

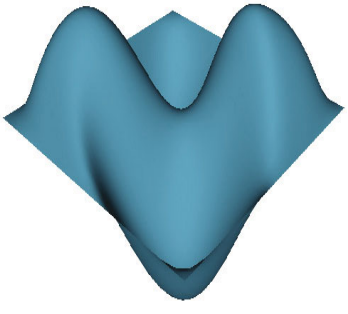
[1] http://axel.inria.fr/



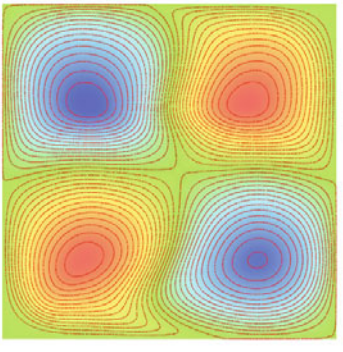
(a) solution surface of parameterization I



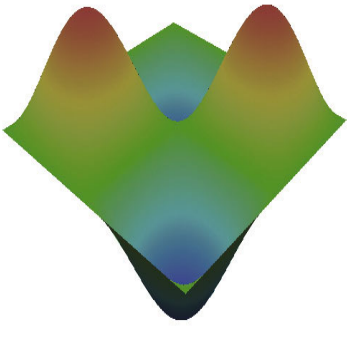
(b) colormap of solution surface based on parameterization I



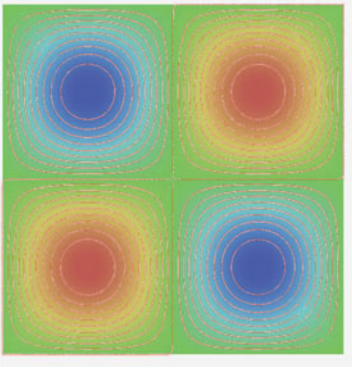
(c) solution surface of parameterization II



(d) colormap of solution surface based on parameterization II



(e) exact solution surface



(f) colormap of exact solution surface

Fig. 3. Simulation results and exact solution

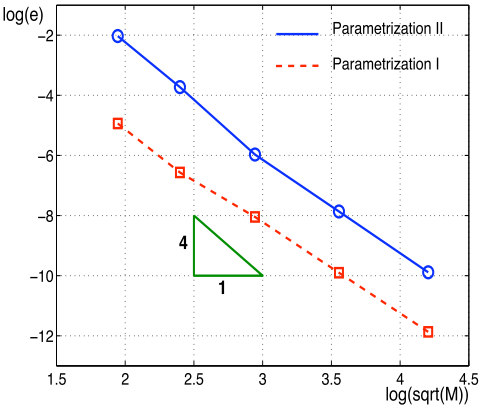


Fig. 4. Error analysis with the curve $(\log\sqrt{M}, \log e)$, where M is the number of control points in each refinement

$\Omega(x, y) = [0, 6] \times [0, 6]$ represented by cubic B-spline surfaces, where the knot vectors in u and v directions are both $\{0, 0, 0, 0, 1, 2, 3, 4, 4, 4, 4\}$. Fig. 2 (a) and Fig. 2(b) present two different placements of inner control points, Fig. 2 (c) and Fig. 2(d) show the isoparametric curves on the computational domain with respect to different placements of inner control points.

We test these two parameterizations on the heat conduction problem (2) with source term

$$\boldsymbol{f}(x, y) = -\frac{4}{9}\sin(\frac{\pi x}{3})\sin(\frac{\pi y}{3}). \tag{4}$$

For this problem with boundary condition $\boldsymbol{T}_0(\boldsymbol{x}) = 0$ and $\Phi_0(\boldsymbol{x}) = 0$, the exact solution over the computational domain $[0, 6] \times [0, 6]$ is

$$\boldsymbol{T}(x, y) = 2\sin(\frac{\pi x}{3})\sin(\frac{\pi y}{3}). \tag{5}$$

Fig.3 (a) and Fig.3 (b) show the approximate solution surface, color map and iso-temperature lines with respect to parameterization I; Fig.3 (c) and Fig.3 (d) show the approximate solution surface, color map and iso-temperature lines with respect to parameterization II. In Fig.3 (e) and Fig.3(f), the exact solution surface and its colormap are presented. Obviously, parameterization I is better than parameterization II for this specified heat conduction problem.

Refinement via knot insertion is an efficient operation to improve the result of isogeometric analysis. We compare the error history during refinement operation for these two different parameterization in Fig.4. The error is computed in relative L_2 norm as follows [24]

$$e = \sqrt{\frac{\int_\Omega (\boldsymbol{T} - \tilde{\boldsymbol{T}})^T (\boldsymbol{T} - \tilde{\boldsymbol{T}}) d\Omega}{\int_\Omega \boldsymbol{T}^T \boldsymbol{T} d\Omega}},$$

where $\boldsymbol{T}$ is the exact solution and $\tilde{\boldsymbol{T}}$ is the approximate solution. From Fig.4, we see that different parameterizations have different impact on the final result after refinement operation. Though the convergence rates of the two different parameterization are in good agreement with theoretical convergence (4 for cubic parameterization), for an error value about 5×10^{-5}, parameterization I requires 35×35 control points, and parameterization II requires 67×67 control points. One reasonable explanation is that with B-spline tensor product surfaces, we can only perform the refinement operations along the parametric directions in IGA, hence it is more restricted than the refinement of a mesh in FEA.

The above example and its analysis show that good parameterization of computational domain is a key issue for IGA. In the next section, we will propose a shape optimization method to construct optimal parameterization of a computational domain.

5 Optimization Method for Parametrization of Computational Domain

5.1 Problem Statement

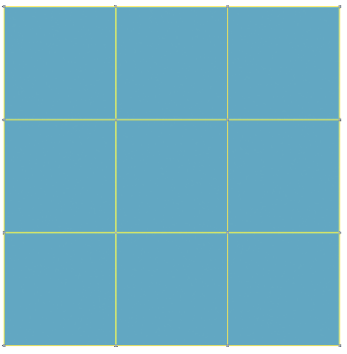
The problem studied in this section can be stated as follows: given four coplanar boundary B-spline curves, find the inner control points such that the parameterization of a computational domain is optimal for an IGA problem with known exact solution. The extension of the proposed method to isogeometric problems without known exact solution is one of our ongoing work.

5.2 Shape Optimization Method

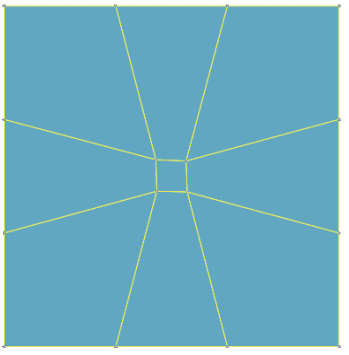
The shape optimization problem consists in finding the shape which is optimal in that it minimizes a certain cost function while satisfying given constraints. The purpose of shape optimization in CAE is to optimize the CAD object for some physical problem, and the design variables are the control points of the CAD object. For 2D isogeometric shape optimization problem, the design variables are the control points of boundary B-spline curves.

Inspired from the idea of shape optimization, in order to obtain optimal parameterization of computational domain, we should let the inner control points, rather than boundary control points, be the design variables for the shape optimization, and find the best placement of inner control points to make the value of a cost function as small as possible.

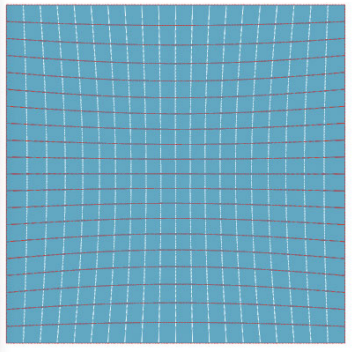
Initial construction of inner control points. As the shape optimization problem, we need to construct an initial placement of inner control points as starting point in the iteration process. We rely on the discrete Coons method presented in [16] to generate inner control points as initial value from boundary control points.



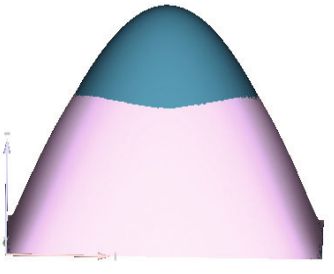
(a) initial control mesh and surface



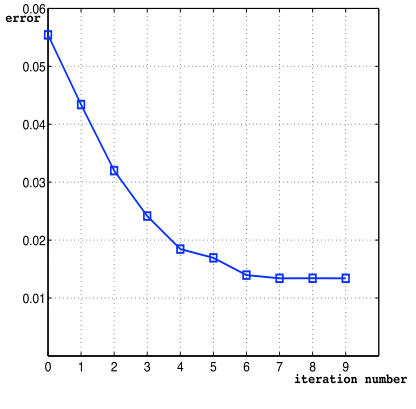
(b) final control mesh and surface



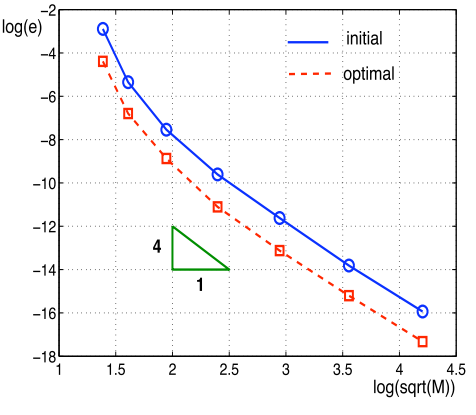
(c) final parametrization



(d) initial solution surface(red) and final solution surface(blue)



(e) error history during the optimization process



(f) error history during the refinement process

Fig. 5. Example II

Given the boundary control points $\boldsymbol{P}_{0,j}, \boldsymbol{P}_{n,j}, \boldsymbol{P}_{i,0}, \boldsymbol{P}_{i,m}$, $i = 0, \ldots, n, j = 0, \ldots, m$, the inner control points $\boldsymbol{P}_{i,j}$ $(i = 1, \ldots, n-1, j = 1, \ldots, m-1)$ can be constructed by the discrete Coons method as follows:

$$\begin{aligned}\boldsymbol{P}_{i,j} = (1 - \frac{i}{n})\boldsymbol{P}_{0,j} + \frac{i}{n}\boldsymbol{P}_{n,j} + (1 - \frac{j}{m})\boldsymbol{P}_{i,0} + \frac{j}{m}\boldsymbol{P}_{i,m} \\ -[1 - \frac{i}{n} \quad \frac{i}{n}] \begin{pmatrix} \boldsymbol{P}_{0,0} & \boldsymbol{P}_{0,m} \\ \boldsymbol{P}_{n,0} & \boldsymbol{P}_{n,m} \end{pmatrix} \begin{pmatrix} 1 - \frac{j}{m} \\ \frac{j}{m} \end{pmatrix}\end{aligned}$$

Remarks 2. Since the sum of the coefficients equals 1, the resulting inner control points lie in the convex hull of the boundary control points.

Remarks 3. For some given boundary curves, this construction may cause some self-intersections, and lead to an improper parameterization for IGA. We use the linear injectivity condition proposed in Section 3 to check the injectivity of initial parameterization. If it does not satisfy the condition, the linear programming method is used to produce another initial parameterization.

Optimization method. In the proposed approach, we minimize the error computed from the IGA solution and the exact solution, by moving inner control points of the computational domain. Therefore, we consider as optimization variables the coordinates of the inner control points and as cost function the error of the IGA solution. The optimization algorithm used for this study is a classical steepest-descent method in conjunction with a back-tracking line-search. For this exercise, the gradient of the cost function is approximated using a centered finite-differencing scheme.

Each iteration k of the optimization algorithm can be summarized as follows, starting from a point x_k in the variable space:

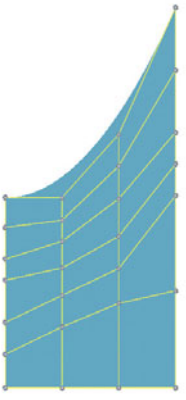
1. Evaluation of perturbed points $x_k + \epsilon e_k$
2. Estimation of the gradient $\nabla f(x_k)$ by finite-difference
3. Define search direction $d_k = -\nabla f(x_k)$
4. Line search : find ρ such as $f(x_k + \rho d_k) < f(x_k)$

These steps are carried out until a stopping criterion is satisfied.

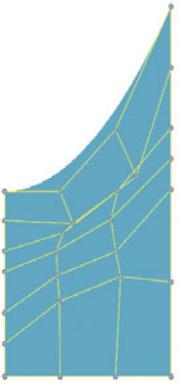
5.3 Examples and Comparison

In this section, we will present some parameterization results and compare them with the initial solution with respect to the heat conduction problem (2).

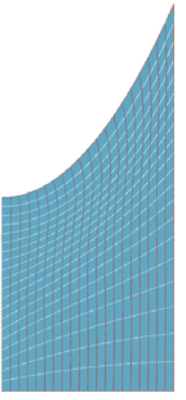
Example II . The second example is for the parameterization of the domain $\Omega = [0, 3] \times [0, 3]$ by cubic Bézier surfaces. The corresponding source term and exact solution is presented in (4) and (5). The parameterization result and comparison with initial parameterization are shown in Fig.5. The initial error is reduced by 24.52% as shown in Fig.5 (e). The final parameterization is clearly better than the initial parameterization during refinement operations as presented in Fig.5 (f).



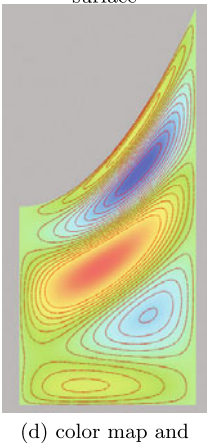
(a) initial control mesh and surface



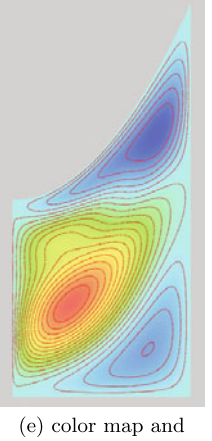
(b) final control mesh and surface



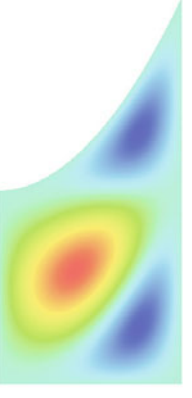
(c) final parametrization



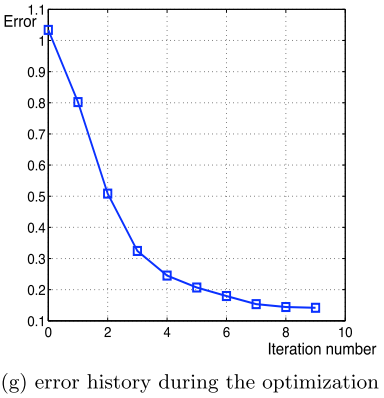
(d) color map and iso-temperature lines of initial solution surface



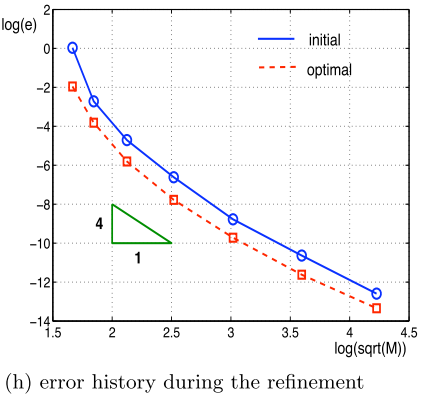
(e) color map and iso-temperature lines of final solution surface



(f) color map of exact solution surface

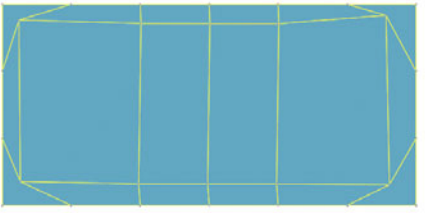


(g) error history during the optimization process

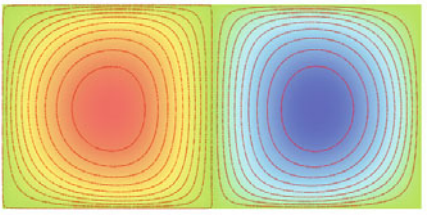


(h) error history during the refinement process

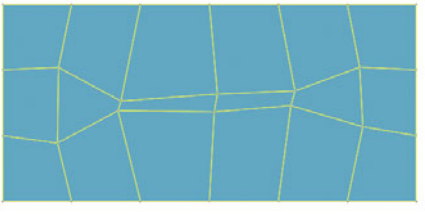
Fig. 6. Example III



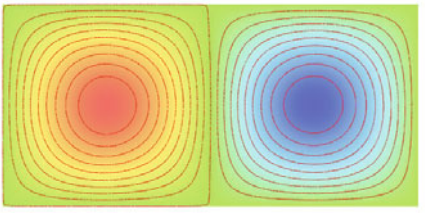
(a) initial control mesh and surface



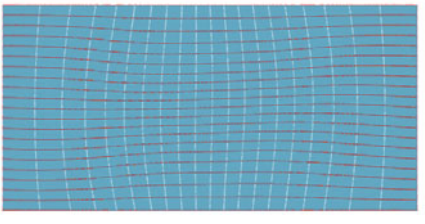
(b) color map of initial solution surface



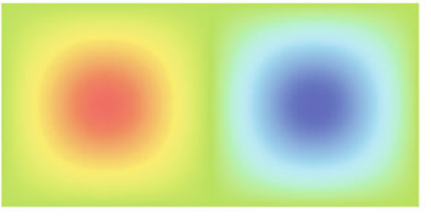
(c) final control mesh and surface



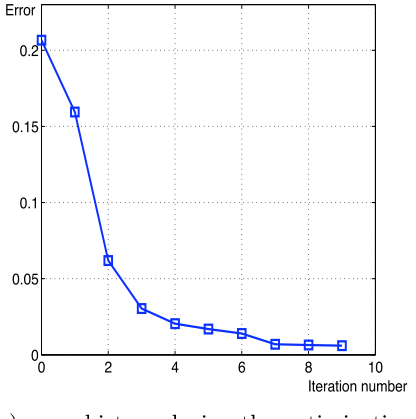
(d) color map and iso-temperature lines of final solution surface



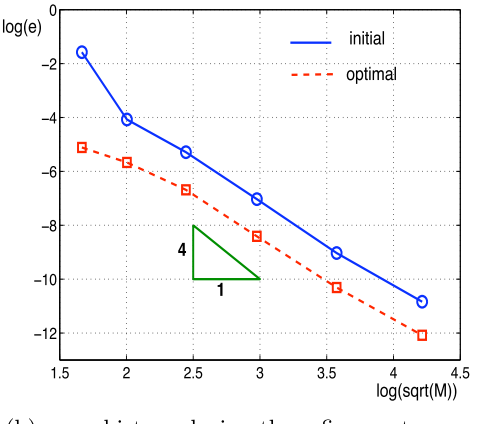
(e) final parametrization



(f) color map of exact solution surface



(g) error history during the optimization process



(h) error history during the refinement process

Fig. 7. Example IV

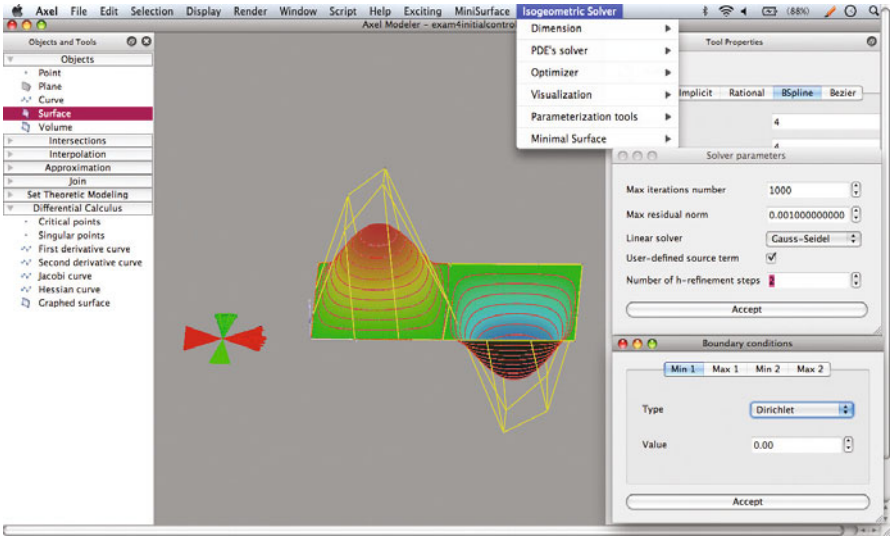


Fig. 8. Interface for isogeometric solver in AXEL

Example III . The next example is for the parameterization of the domain

$$\Omega(x, y) = \{(x, y)| - 1 \leq y \leq x^2, 0 \leq x \leq 1\}$$

by Bézier surface with degree 3×6. The parabola is represented by degenerate cubic Bézier curve. For the problem with boundary condition $\boldsymbol{T}_0(\boldsymbol{x}) = 0$ and $\Phi_0(\boldsymbol{x}) = 0$ in (2), we can construct an exact solution $\boldsymbol{T}(x, y)$ as follows

$$\boldsymbol{T}(x, y) = \sin(\pi(y - x^2)) \sin(\pi x) \sin(\pi y)$$

The initial placement of inner control points is produced by the discrete Coons method as shown in Fig.6 (a). The final parameterization results and some comparisons are also shown in Fig.6. We can find that there are some self-intersections on the control mesh in Fig.6 (b). However, there is no self-intersection on the final parameterization as shown in Fig.6 (c). During the optimization, the initial error is reduced by 14.65% as shown in Fig.6 (g). The error history during refinement operation is presented in Fig.6 (h).

Example IV . The final example is for the parameterization of the domain $\Omega = [0, 3] \times [0, 6]$ by cubic B-spline surface with knot vector $\{0, 0, 0, 0, 1, 2, 3, 4, 4, 4, 4\}$ in the u direction and knot vector $\{0, 0, 0, 0, 1, 1, 1, 1\}$ in the v direction. The corresponding source term and exact solution is presented in (4) and (5). The initial placement of inner control points is non-uniform as shown in Fig.7 (a), and the final parametrization result and some comparison are also shown in Fig.7. During the optimization, the initial error is reduced by 3.31% as shown in Fig.7 (g). The error history during refinement operation is presented in Fig.7 (h).

6 Conclusion and Future Work

Parameterization of computational domains is the first step in an IGA process. In this paper, we show that for different parameterizations of a computational domain, different simulation results can be obtained. Based on this observation and inspired by shape optimization, an approach for optimal parameterization of computational domain is proposed. We also proposed a linear and easy-to-check sufficient condition for injectivity of planar B-spline parameterization. Several examples are presented to illustrate the effectiveness of the proposed method. As shown in Fig.8, a user-friendly interface for isogeometric solver and optimizer is implemented as plugin in the AXEL platform.

The proposed method will be tested on more complex computational domain and generalized to 3D cases with exact solutions in the future. The construction of a proper parameterization of computational domain for general problem, in which the exact solution is unknown, is also a part of our ongoing work. One possible way is to find an accurate posteriori error estimation method for IGA, and perform the optimization based on this estimation. We will discuss this topic in another paper.

Acknowledgments. The authors would like to thank the reviewers for their constructive comments and suggestions. The authors are supported by the 7th Framework Program of the European Union, project SCP8-218536 "EXCITING".

References

1. Aigner, M., Heinrich, C., Jüttler, B., Pilgerstorfer, E., Simeon, B., Vuong, A.-V.: Swept volume parametrization for isogeometric analysis. In: Hancock, E.R., Martin, R.R., Sabin, M.A. (eds.) MOS XIII 2009. LNCS, vol. 5654, pp. 19–44. Springer, Heidelberg (2009)
2. Auricchio, F., da Veiga, L.B., Buffa, A., Lovadina, C., Reali, A., Sangalli, G.: A fully locking-free isogeometric approach for plane linear elasticity problems: A stream function formulation. Computer Methods in Applied Mechanics and Engineering 197, 160–172 (2007)
3. Bazilevs, Y., Beirao de Veiga, L., Cottrell, J.A., Hughes, T.J.R., Sangalli, G.: Isogeometric analysis: approximation, stability and error estimates for refined meshes. Mathematical Models and Methods in Applied Sciences 6, 1031–1090 (2006)
4. Bazilevs, Y., Calo, V.M., Hughes, T.J.R., Zhang, Y.: Isogeometric fluid structure interaction: Theory, algorithms, and computations. Computational Mechanics 43, 3–37 (2008)
5. Bazilevs, Y., Calo, V.M., Zhang, Y., Hughes, T.J.R.: Isogeometric fluid structure interaction analysis with applications to arterial blood flow. Computational Mechanics 38, 310–322 (2006)
6. Bazilevs, Y., Hughes, T.J.R.: NURBS-based isogeometric analysis for the computation of flows about rotating components. Computational Mechanics 43, 143–150 (2008)
7. Bazilevs, Y., Calo, V.M., Cottrell, J.A., Evans, J., Hughes, T.J.R., Lipton, S., Scott, M.A., Sederberg, T.W.: Isogeometric analysis using T-Splines. Computer Methods in Applied Mechanics and Engineering 199(5-8), 229–263 (2010)

8. Cohen, E., Martin, T., Kirby, R.M., Lyche, T., Riesenfeld, R.F.: Analysis-aware Modeling: Understanding Quality Considerations in Modeling for Isogeometric Analysis. Computer Methods in Applied Mechanics and Engineering 199(5-8), 334–356 (2010)
9. Cottrell, J.A., Hughes, T.J.R., Reali, A.: Studies of refinement and continuity in isogeometric analysis. Computer Methods in Applied Mechanics and Engineering 196, 4160–4183 (2007)
10. Cottrell, J.A., Reali, A., Bazilevs, Y., Hughes, T.J.R.: Isogeometric analysis of structural vibrations. Computer Methods in Applied Mechanics and Engineering 195, 5257–5296 (2006)
11. Dokken, T., Skytt, V., Haenisch, J., Bengtsson, K.: Isogeometric representation and analysis–bridging the gap between CAD and analysis. In: 47th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition, Orlando, Florida, January 5-8 (2009)
12. Dörfel, M., Jüttler, B., Simeon, B.: Adaptive isogeometric analysis by local h-refinement with T-splines. Computer Methods in Applied Mechanics and Engineering 199(5-8), 264–275 (2010)
13. Duvigneau, R.: An introduction to isogeometric analysis with application to thermal conduction. INRIA Research Report RR-6957 (June 2009)
14. Elguedj, T., Bazilevs, Y., Calo, V.M., Hughes, T.J.R.: $\bar{B}$ and $\bar{F}$ projection methods for nearly incompressible linear and non-linear elasticity and plasticity using higher-order NURBS elements. Computer methods in applied mechanics and engineering 197, 2732–2762 (2008)
15. Evans, J.A., Bazilevs, Y., Babuka, I., Hughes, T.J.R.: n-Widths, supinfs, and optimality ratios for the k-version of the isogeometric finite element method. Computer Methods in Applied Mechanics and Engineering 198, 1726–1741 (2009)
16. Farin, G., Hansford, D.: Discrete coons patches. Computer Aided Geometric Design 16(7), 691–700 (1999)
17. Gain, J.E., Dodgson, N.A.: Preventing self-Intersection under free-form deformation. IEEE Transactions on Visualization and Computer Graphics 7(4), 289–298 (2001)
18. Gomez, H., Calo, V.M., Bazilevs, Y., Hughes, T.J.R.: Isogeometric analysis of the Cahn-Hilliard phase-field model. Computer Methods in Applied Mechanics and Engineering 197, 4333–4352 (2008)
19. Hughes, T.J.R., Cottrell, J.A., Bazilevs, Y.: Isogeometric analysis: CAD, finite elements, NURBS, exact geometry, and mesh refinement. Computer Methods in Applied Mechanics and Engineering 194(39-41), 4135–4195 (2005)
20. Hughes, T.J.R., Realli, A., Sangalli, G.: Duality and unified analysis of discrete approximations in structural dynamics and wave propagation: Comparison of p-method finite elements with k-method NURBS. Computer methods in applied mechanics and engineering 197, 4104–4124 (2008)
21. Hughes, T.J.R., Realli, A., Sangalli, G.: Efficient quadrature for NURBS-based isogeometric analysis. Computer Methods in Applied Mechanics and Engineering 199(5-8), 301–313 (2010)
22. Jüttler, B.: Shape-preserving least-squares approximation by polynomial parametric spline curves. Computer Aided Geometric Design 14, 731–747 (1997)
23. Kestelman, H.: Mappings with non-vanishing Jacobian. Amer. Math. Monthly 78, 662–663 (1971)
24. Kim, H.J., Seo, Y.D., Youn, S.K.: Isogeometric analysis for trimmed CAD surfaces. Computer Methods in Applied Mechanics and Engineering 198(37-40), 2982–2995 (2009)

25. Martin, T., Cohen, E., Kirby, R.M.: Volumetric parameterization and trivariate B-spline fitting using harmonic functions. Computer Aided Geometric Design 26(6), 648–664 (2009)
26. Sevilla, R., Fernandes-Mendez, S., Huerta, A.: NURBS-enhanced finite element method for Euler equations. International Journal for Numerical Methods in Fluids 57, 1051–1069 (2008)
27. Wall, W.A., Frenzel, M.A., Cyron, C.: Isogeometric structural shape optimization. Computer Methods in Applied Mechanics and Engineering 197, 2976–2988 (2008)
28. Zhang, Y., Bazilevs, Y., Goswami, S., Bajaj, C., Hughes, T.J.R.: Patient-specific vascular NURBS modeling for isogeometric analysis of blood flow. Computer Methods in Applied Mechanics and Engineering 196, 2943–2959 (2007)

Construction of Subdivision Surfaces by Fourth-Order Geometric Flows with G^1 Boundary Conditions

Guoliang Xu[1,⋆] and Qing Pan[2,⋆⋆]

[1] LSEC, Institute of Computational Mathematics, Academy of Mathematics and System Sciences, Chinese Academy of Sciences, Beijing, 100190, China
[2] College of Mathematics and Computer Science, Hunan Normal University, Changsha, 410081, China
panqing@lsec.cc.ac.cn

Abstract. In this paper, we present a method for constructing Loop's subdivision surface patches with given G^1 boundary conditions and a given topology of control polygon, using several fourth-order geometric partial differential equations. These equations are solved by a mixed finite element method in a function space defined by the extended Loop's subdivision scheme. The method is flexible to the shape of the boundaries, and there is no limitation on the number of boundary curves and on the topology of the control polygon. Several properties for the basis functions of the finite element space are developed.

Keywords: Subdivision Surface, Geometric Partial Differential Equations, G^1 continuity.

1 Introduction

A surface satisfing a geometric partial differential equation (PDE) is referred to as geometric PDE surface in this paper. Geometric PDE surfaces, such as minimal surfaces (see [13]), constant mean-curvature surfaces (see [7,16]), Willmore surfaces (see [3,9,10,19])) and minimal mean-curvature variation surfaces (see [23]), are important and preferred in the shape designing and modeling because they share certain optimal properties. For instance, the minimal surfaces have minimal area, the Willmore surfaces have minimal total squared mean curvature and minimal mean-curvature variation surfaces have minimal total mean-curvature variation. Here the terminology *total* means the integration over the surfaces. Various type geometric PDE surfaces have been constructed in the literatures (see [21]). Most of them are discrete surfaces (triangular or quadrilateral

⋆ Supported in part by NSFC under the grant 60773165, NSFC key project under the grant 10990013, and National Key Basic Research Project of China (2004CB318000).
⋆⋆ Supported in part by NSFC grant 10701071 and Program for Excellent Talents in Hunan Normal University (No. ET10901). Corresponding author.

B. Mourrain, S. Schaefer, and G. Xu (Eds.): GMP 2010, LNCS 6130, pp. 255–268, 2010.

control polygons), a few of them are continuous surfaces. Usually, the representation of the continuous surfaces are Bézier (see [8]), rational Bézier, B-spline (see [11,12]) and NURB surfaces.

Obviously, Bézier surfaces, B-spline and NURB surfaces have to be three- or four-sided. This is a serious limitation for designing geometric PDE surfaces with arbitrary shaped boundaries. In this paper, our intension is to construct geometric PDE subdivision surfaces with piecewise B-spline curve boundaries and tangent conditions. There is no limitation on the number of spline curve pieces. B-spline representation for curves and surfaces have been widely accepted in the CAD and industrial design. Using B-spline to represent surface boundary is preferable and acceptable. To represent a surface patch with any topology, subdivision surfaces are the best candidates, since there is no limitation on the topology of the control polygon. However, subdivision surfaces, such as Loop's subdivision surfaces and Catmull-Clark subdivision surfaces are traditionally closed, which cannot be used directly for serving our purpose.

For many surface modeling problems, such as the construction of the bodies of cars and aircrafts, machine parts and roofs, surfaces are usually constructed in a piecewise manner with fixed boundaries for each of the pieces. In such a case Loop's subdivision scheme cannot be applied near the boundary of the control polygon. Therefore, an extension of the Loop's scheme to control polygon with boundaries is required. On this aspect, an excellent work has been done by Biermann et al [2] and that is just sufficient for achieving the goal of constructing piecewise smooth surface.

In this paper we construct geometric PDE subdivision surface patches with given G^1 boundary conditions and a given topology of the control polygon using several fourth-order geometric partial differential equations. These equations are solved by a mixed finite element method in a function space defined by the extended Loop's subdivision scheme. By the term *topology of the control polygon*, we mean the connection mode among the vertices of the control polygon.

Fourth-order geometric flows have been used to solve the problems of discrete surface blending, N-sided hole filling and the free-form surface fitting (see [4,17,18,22]). In [17,18], the surface diffusion flow has been used for fairing/smoothing meshes while satisfying the G^1 boundary conditions. The finite element method is used by Clarenz et al. [4] to solve the Willmore flow equation, based on a new variational formulation of this flow, for the discrete surface restoration.

Problem Description

Input: Given an initial open control polygon (a piece of triangular mesh) of a surface patch with fixed boundary control points, and some of the boundary control points are to be interpolated. The boundary curve is defined as piecewise cubic B-spline with the boundary control points as the B-spline control points and equal spaced knots for each piece. The interpolated boundary control points are served as the end-points of the B-spline curves. On each

of the boundary curves, we are also given a tangential vector (co-normal) curve, which is represented in the same form as the boundary curve.
Output: We want to construct a geometric PDE subdivision surface that interpolates the boundary curves and tangents, at the same time its control polygon has the same topology as the initial one.

2 Geometric PDEs and Their Weak-Form Formulations

To describe precisely the geometric partial differential equations used in this paper, we need introduce a few notations (see the details in [21]).

2.1 Notations

Let

$$\mathcal{S} := \left\{\mathbf{x}(u^1, u^2) \in \mathbb{R}^3 : (u^1, u^2) \in \mathscr{D} \subset \mathbb{R}^2\right\}$$

be a parametric surface which is sufficiently smooth and orientable. Let

$$g_{\alpha\beta} = \langle \mathbf{x}_{u^\alpha}, \mathbf{x}_{u^\beta} \rangle \quad \text{and} \quad \alpha\beta = \langle \mathbf{n}, \mathbf{x}_{u^\alpha u^\beta} \rangle$$

be the coefficients of the first and the second fundamental forms of $\mathcal{S}$ with

$$\mathbf{x}_{u^\alpha} = \frac{\partial \mathbf{x}}{\partial u^\alpha}, \ \mathbf{x}_{u^\alpha u^\beta} = \frac{\partial^2 \mathbf{x}}{\partial u^\alpha \partial u^\beta}, \ \alpha, \beta = 1, 2,$$
$$\mathbf{n} = (\mathbf{x}_u \times \mathbf{x}_v)/\|\mathbf{x}_u \times \mathbf{x}_v\|, \quad (u, v) := (u^1, u^2),$$

where $\langle \cdot, \cdot \rangle$, $\| \cdot \|$ and $\cdot \times \cdot$ stand for the usual inner product, Euclidean norm and cross product in $\mathbb{R}^3$, respectively.

Curvatures. To introduce the notions of the mean curvature and the Gaussian curvature, we use the concept of *Weingarten map* or *shape operator* (see [5]). It is a self-adjoint linear map on the tangent space

$$T_{\mathbf{x}}\mathcal{S} := \operatorname{span}\{\mathbf{x}_u, \mathbf{x}_v\}.$$

In matrix form, it can be represented by a 2×2 matrix

$$S = [\, b_{\alpha\beta} \,][\, g^{\alpha\beta} \,] \quad \text{with} \quad [\, g^{\alpha\beta} \,] = [\, g_{\alpha\beta} \,]^{-1}.$$

The eigenvalues k_1 and k_2 of S are the *principal curvatures* of $\mathcal{S}$ and their arithmetic average and product are the *mean curvature* H and the *Gaussian curvature* K, respectively. That is

$$H = \frac{k_1 + k_2}{2} = \frac{\operatorname{tr}(S)}{2},$$
$$K = k_1 k_2 = \det(S).$$

Let $\mathbf{H} = H\mathbf{n}$. It is referred to as the mean curvature normal.

Tangential gradient operator. Suppose $f \in C^1(\mathcal{S})$, where $C^1(\mathcal{S})$ stands for a function space consisting of C^1 smooth functions on $\mathcal{S}$, then the tangential gradient operator ∇_s acting on f is defined as

$$\nabla_s f = [\mathbf{x}_u,\ \mathbf{x}_v][\,g^{\alpha\beta}\,][f_u,\ f_v]^T \in \mathbb{R}^3. \tag{1}$$

For a vector-valued function

$$\mathbf{f} = [f_1, \cdots, f_k]^T \in C^1(\mathcal{S})^k,$$

its gradient is defined as

$$\nabla_s \mathbf{f} = [\nabla_s f_1, \cdots, \nabla_s f_k] \in \mathbb{R}^{3\times k}.$$

The third tangential operator. Let $f \in C^1(\mathcal{S})$. Then the third tangential operator $\oslash$ acting on f is defined as

$$\oslash f = [\mathbf{x}_u, \mathbf{x}_v][\,g^{\alpha\beta}\,]S[f_u, f_v]^{\mathrm{T}} \in \mathbb{R}^3.$$

Apart from these two tangential operators, there is another one, called the second tangential operator (see [21]). Since it is not involved in this work, we do not introduce it.

Divergence operator. Suppose $\mathbf{v}$ is a smooth vector field on surface $\mathcal{S}$, then the divergence operator div_s acting on $\mathbf{v}$ is defined as

$$\mathrm{div}_s(\mathbf{v}) = \frac{1}{\sqrt{g}}\left[\frac{\partial}{\partial u}, \frac{\partial}{\partial v}\right]\left[\,\sqrt{g}\,[\,g^{\alpha\beta}\,]\,[\mathbf{x}_u, \mathbf{x}_v]^T\mathbf{v}\,\right]. \tag{2}$$

Laplace-Beltrami operator. Let $f \in C^2(\mathcal{S})$. Then the Laplace-Beltrami operator (LBO) Δ_s acting on f is defined as (see [5], p. 83)

$$\Delta_s f = \mathrm{div}_s(\nabla_s f).$$

Theorem 1 (Green's formula for LBO). *Let $\mathcal{S}$ be an orientaable surface, Ω a subregion of $\mathcal{S}$ with a piecewise smooth boundary $\partial\Omega$. Let $\mathbf{n}_c \in T_{\mathbf{x}}\mathcal{S}$ $(\mathbf{x} \in \partial\Omega)$ be the outward unit normal (also named as co-normal) along the boundary $\partial\Omega$. Then for a given C^1 smooth vector field $\mathbf{v}$ on $\mathcal{S}$, we have*

$$\int_\Omega [\langle \mathbf{v}, \nabla_s f\rangle + f\,\mathrm{div}(\mathbf{v})]\mathrm{d}A = \int_{\partial\Omega} f\langle \mathbf{v}, \mathbf{n}_c\rangle \mathrm{d}s. \tag{3}$$

2.2 Used Geometric PDEs

For completeness, we describe briefly the equations used and their behaviors. More details on these equations can be found in [21].

Surface Diffusion Flow

$$\frac{\partial \mathbf{x}}{\partial t} = -2\Delta_s H\mathbf{n}. \tag{4}$$

This flow was introduced by Mullins in 1957 (see [14]), to describe the interface motion law of the growing crystal. It is well known that surface diffusion flow is volume preserving and area shrinking. The area shrinkage stops when H is a constant. Surfaces with constant mean curvature are the steady solution of (4).

Willmore flow

$$\frac{\partial \mathbf{x}}{\partial t} = -2\left[\Delta_s H + 2H(H^2 - K)\right]\mathbf{n}. \tag{5}$$

Willmore flow is derived from minimizing total squared mean-curvature $\int_{\mathcal{S}} H^2 \mathrm{d}A$. A factor 2 is added to the original Willmore flow for comparability with other equations used in this paper. There are sound published research papers that use this flow (see [3,9,10,19]). There is no volume/area preserving/shrinking property for this flow. However, if the initial surface is a sphere, Willmore flow keeps the spherical shape unchanged. Moreover, surfaces with zero mean curvature are the the steady solution of (5). A torus with $R/r = \sqrt{2}$ is a steady solution of (5), where the torus is defined by rotating a circle with radius r along another circle with radius R.

Quasi-surface diffusion flow

$$\frac{\partial \mathbf{x}}{\partial t} = -\Delta_s^2 \mathbf{x}. \tag{6}$$

This flow is introduced in [22], and used in discrete surface design. We remove the tangential movement of (6) because tangent motion of a surface does not alter the surface shape (see [6]), then obtain the following geometric flow

$$\frac{\partial \mathbf{x}}{\partial t} = -2\left[\Delta_s H - 2H(2H^2 - K)\right]\mathbf{n}. \tag{7}$$

Quasi-surface diffusion flow is area diminishing and the solution surfaces of (6) approach to the minimal surface.

These three flows share the same fourth-order term $-2\Delta_s H\mathbf{n}$ and only the second order terms are different, however, their behaviors are quite different.

2.3 Mixed Variational Formulations

Let

$$\mathbf{y}(\mathbf{x}) = H(\mathbf{x})\mathbf{n}(\mathbf{x}) \in \mathbb{R}^3$$

stand for the mean curvature normal. Then the mixed variational form of SDF (4) is: Find $(\mathbf{x}, \mathbf{y}) \in H^2(\mathcal{S})^3 \times H^1(\mathcal{S})^3$ such that

$$\begin{cases} \displaystyle\int_{\mathcal{S}} \frac{\partial \mathbf{x}}{\partial t}\phi \,\mathrm{d}A + 2\int_{\mathcal{S}} \left[\phi \oslash \mathbf{y} - \mathbf{n}(\nabla_s \phi)^{\mathrm{T}} \nabla_s \mathbf{y}\right]\mathbf{n} \,\mathrm{d}A = \mathbf{0}, & \forall \phi \in H_0^1(\mathcal{S}), \\ \displaystyle\int_{\mathcal{S}} \mathbf{y}\psi \,\mathrm{d}A + \frac{1}{2}\int_{\mathcal{S}} (\nabla_s \mathbf{x})^{\mathrm{T}} \nabla_s \psi \,\mathrm{d}A - \frac{1}{2}\int_{\partial \mathcal{S}} \mathbf{n}_c \psi \,\mathrm{d}s = \mathbf{0}, & \forall \psi \in H^1(\mathcal{S}), \\ \mathcal{S}(0) = \mathcal{S}_0, \quad \partial\mathcal{S}(t) = \Gamma, \quad \mathbf{n}_c(\mathbf{x}) = \mathbf{n}_c^{(\Gamma)}(\mathbf{x}), \ \forall \mathbf{x} \in \Gamma, \end{cases} \tag{8}$$

where $\mathbf{n}_c^{(\Gamma)}$ is the given co-normal on the boundary curve Γ. The mixed variational form of (5) and (7) are similar, and the differences occur only in the first equation. For WF (5), the first equation of the mixed variational form is

$$\begin{aligned}\int_{\mathcal{S}} \frac{\partial \mathbf{x}}{\partial t}\phi \, \mathrm{d}A + 2\int_{\mathcal{S}} \left[\phi \oslash \mathbf{y} - \mathbf{n}(\nabla_s \phi)^{\mathrm{T}} \nabla_s \mathbf{y}\right] \mathbf{n} \, \mathrm{d}A \\ + 4\int_{\mathcal{S}} \mathbf{n}(H^2 - K)\phi \mathbf{n}^{\mathrm{T}}\mathbf{y} \, \mathrm{d}A = \mathbf{0}, \qquad \forall \phi \in H_0^1(\mathcal{S}).\end{aligned} \tag{9}$$

For QSDF (7), the first equation of the mixed variational form is

$$\begin{aligned}\int_{\mathcal{S}} \frac{\partial \mathbf{x}}{\partial t}\phi \, \mathrm{d}A + 2\int_{\mathcal{S}} \left[\phi \oslash \mathbf{y} - \mathbf{n}(\nabla_s \phi)^{\mathrm{T}} \nabla_s \mathbf{y}\right] \mathbf{n} \, \mathrm{d}A \\ - 4\int_{\mathcal{S}} \mathbf{n}(2H^2 - K)\phi \mathbf{n}^{\mathrm{T}}\mathbf{y} \, \mathrm{d}A = \mathbf{0}, \qquad \forall \phi \in H_0^1(\mathcal{S}).\end{aligned} \tag{10}$$

3 Subdivision Surfaces and Finite Element Space

In this section, systems (8)–(10) are discretized in a finite element space defined by the extended Loop subdivision scheme. The original Loop's subdivision scheme is usable only for control polygons without boundary. Therefore, an extension of the subdivision scheme to control polygons with boundary is required. We use Biermann et al's extension (see [2]) to achieve our goal. For saving the space, we do not describe them.

3.1 Basis Functions and Their Properties

Now let us define the basis functions of the finite element function space, denoted as $V_{\mathcal{S}(t)}$. For each control point $\mathbf{x}_i$, including the corner control point and boundary control points, of a control polygon $\mathcal{S}_d$, we associate it with a basis function ϕ_i, where ϕ_i is defined as the limit of the extended Loop's subdivision scheme with the zero control values everywhere except at $\mathbf{x}_i$ where it is one.

The control polygon $\mathcal{S}_d$, as a piecewise linear surface, is served as the definition domain of the basis function ϕ_i. The mapping from $\mathcal{S}_d$ to ϕ_i is defined by a dual subdivision process. More precisely, when the extended Loop's subdivision scheme is applied to the control function values recursively, the linear subdivision scheme (each triangle is partitioned into four equal-sized sub-triangles) is applied to the control polygon correspondingly. The limit of the former is ϕ_i and that of later is $\mathcal{S}_d$ itself.

The basis functions share some important properties:

1. *Positivity.*
   The weights of the extended subdivision rules are positive. Hence the basis function ϕ_i is nonnegative everywhere and positive around $\mathbf{x}_i$.

2. *Locality.*
   The limit value at a control point is a linear combination of the one-ring neighbor values. Hence, the limit value is zero at a control point if the control values on the one-ring neighbor control points are zeros. Therefore, the support of the basis function is within the two-ring neighborhood.
3. *Partition of Unity.*
   Since all the subdivision rules have the properties that the weights are summed to one. Therefore, if we choose all the control values as one. The control values after one subdivision step are still one. This implies that

$$\sum_{i=0}^{m} \phi_i(\mathbf{x}) = 1.$$

   This property is called partition of unity.
4. *Interpolatory Properties at the Boundary.*
   The extended subdivision rules on the boundary do not involve the interior control points. Hence the basis functions for the interior control points are zero at the boundary. This means that the given boundary curves are interpolated.
5. *Tangential Property.*
   Let $\mathbf{x}_i$ be a control point, with non of its one-ring neighbor control points is boundary control points. Then $\nabla_s \phi_i$ vanishes on the boundary. This fact can be observed by considering the eigen-decomposition of the control points. Let $\mathbf{p}^{(k)} \in \mathbb{R}^{(n+1)\times 3}$ be a vector consisting of one-ring neighbor control points of $\mathbf{x}_i^{(k)}$ at the subdivision level k, $S \in \mathbb{R}^{(n+1)\times(n+1)}$ the local subdivision matrix that convert $\mathbf{p}^{(k)}$ to $\mathbf{p}^{k+1}$, i.e.,

$$\mathbf{p}^{(k+1)} = S\mathbf{p}^{(k)} = S^k \mathbf{p}^{(1)}, \quad k = 1, 2, \cdots.$$

   Here n stands for the valence of $\mathbf{x}_i^{(k)}$. Suppose $\mathbf{p}^{(1)}$ is decomposed into

$$\mathbf{p}^{(1)} = \mathbf{e}_0 \mathbf{a}_0^T + \mathbf{e}_1 \mathbf{a}_1^T + \mathbf{e}_2 \mathbf{a}_2^T + \cdots + \mathbf{e}_n \mathbf{a}_n^T, \quad \mathbf{a}_j \in \mathbb{R}^3,$$

   where $\mathbf{e}_0$, $\mathbf{e}_1$, $\cdots$, $\mathbf{e}_n$ are the eigenvectors of S. Here we assume that these eigenvectors are arranged in the order of non-increasing eigenvalues λ_j. Then

$$\mathbf{p}^{(k+1)} = \lambda_0^k \mathbf{e}_0 \mathbf{a}_0^T + \lambda_1^k \mathbf{e}_1 \mathbf{a}_1^T + \lambda_2^k \mathbf{e}_2 \mathbf{a}_2^T + \cdots + \lambda_n^n \mathbf{e}_n \mathbf{a}_n^T,$$

   where $\lambda_0 = 1$, $\lambda_1 = \lambda_2 < 1$. The limit position at the center is $\mathbf{a}_0$. The tangent direction at this point are $\mathbf{a}_1$ and $\mathbf{a}_2$, and $\mathbf{a}_j$ is given by

$$\mathbf{a}_j^T = \tilde{\mathbf{e}}_j^T \mathbf{p}^{(1)},$$

   where $\tilde{\mathbf{e}}_j$ are the left eigenvectors of $\mathcal{S}$ with normalized condition $\tilde{\mathbf{e}}_j^T \mathbf{e}_j = 1$.
6. *Linear independency.*
   As a set of basis functions, $\{\phi_i\}_{i=0}^m$ must be linearly independent. For Loop's subdivision scheme, this fact is implied by a result from [20] on the solvability of interpolation problem:

For the given function values $\{f_i\}_0^m$, find the control function values $\{g_i\}_0^m$ such that

$$\sum_{j=0}^{m} g_j \phi_j(\mathbf{v}_i) = f_i, \quad i = 0, \cdots, m, \tag{11}$$

where $\mathbf{v}_i$ is the limit position of the subdivision surface corresponding to the control point $\mathbf{x}_i$.

3.2 Spatial Discretizations

Suppose ϕ_i is a basis function of $V_{\mathcal{S}(t)}$ corresponding to control point $\mathbf{x}_i$, $i = 0, \cdots, m$. Assume $\mathbf{x}_0, \cdots, \mathbf{x}_{m_0}$ are the interior control points, and $\mathbf{x}_{m_0+1}, \cdots, \mathbf{x}_m$ are the boundary control points. Then $\mathbf{x}(t) \in \mathcal{S}(t)$ can be represented as

$$\mathbf{x}(t) = \sum_{j=0}^{m_0} \mathbf{x}_j(t)\phi_j + \sum_{j=m_0+1}^{m} \mathbf{x}_j(t)\phi_j, \quad \mathbf{x}_j(t) \in \mathbb{R}^3, \tag{12}$$

and therefore,

$$\nabla_s \mathbf{x}(t) = \sum_{j=0}^{m_0} \nabla_s \phi_j [\mathbf{x}_j(t)]^T + \sum_{j=m_0+1}^{m} \nabla_s \phi_j [\mathbf{x}_j(t)]^T, \quad \mathbf{x}_j(t) \in \mathbb{R}^{3\times 3}. \tag{13}$$

The mean curvature vector of the surface is represented approximately as

$$\mathbf{y}(t) = \sum_{j=0}^{m} \mathbf{y}_j(t)\phi_j, \quad \mathbf{y}_j \in \mathbb{R}^3, \quad \nabla_s \mathbf{y}(t) = \sum_{j=0}^{m} \nabla_s \phi_j [\mathbf{y}_j(t)]^T \in \mathbb{R}^{3\times 3}. \tag{14}$$

Since the boundary control points are fixed and the interior control points are to be determined, the coefficients $\mathbf{x}_j$ in the first term of (12) are unknowns, while the coefficients $\mathbf{x}_j$ in the second term are the given control points on the boundary. Furthermore, since the curvature on the surface boundary involves the unknown interior control points, hence all the coefficients in (14) are treated as unknowns.

Now let us discretize equations (8)–(10) in the finite space $V_{\mathcal{S}(t)}$. Since these equations are similar in form, we treat them together. Let $\mathcal{S}$ be the limit surface of the extended Loop's subdivision scheme for the control polygon $\mathcal{S}_d$. Substituting (12)–(14) into (8)–(10), and taking the test functions ϕ as $\phi_i (i = 0, \cdots, m_0)$, ψ as $\phi_i (i = 0, \cdots, m)$, and finally noting that

$$\frac{\partial \mathbf{x}_j(t)}{\partial t} = \mathbf{0} \quad \text{if} \quad j > m_0,$$

we obtain the following matrix representations of (8)–(10):

$$\begin{cases} M_{m_0}^{(1)} \frac{\partial X_{m_0}(t)}{\partial t} + L_m^{(1)} Y_m(t) = \mathbf{0}, \\ M_m^{(2)} Y_m(t) \quad + L_m^{(2)} X_m(t) = B, \end{cases} \tag{15}$$

where

$$X_j(t) = [\mathbf{x}_0^{\mathrm{T}}(t), \cdots, \mathbf{x}_j^{\mathrm{T}}(t)]^{\mathrm{T}} \in \mathbb{R}^{3(j+1)},$$
$$Y_m(t) = [\mathbf{y}_0^{\mathrm{T}}(t), \cdots, \mathbf{y}_m^{\mathrm{T}}(t)]^{\mathrm{T}} \in \mathbb{R}^{3(m+1)},$$

are matrices consisting of the control points for the surface and the mean curvature normals, respectively, and

$$B = [\mathbf{b}_0^{\mathrm{T}}, \cdots, \mathbf{b}_m^{\mathrm{T}}]^{\mathrm{T}} \in \mathbb{R}^{3(m+1)},$$
$$M_{m_0}^{(1)} = (m_{ij}\mathrm{I}_3)_{ij=0}^{m_0,m_0}, \quad M_m^{(2)} = (m_{ij}\mathrm{I}_3)_{ij=0}^{m,m},$$
$$L_m^{(1)} = \left(l_{ij}^{(1)}\right)_{ij=0}^{m_0,m}, \qquad L_K^{(2)} = \left(l_{ij}^{(2)}\mathrm{I}_3\right)_{ij=0}^{m,K}.$$

are the coefficient matrices. The elements of these matrices are defined as follows:

$$m_{ij} = \int_{\mathcal{S}} \phi_i \phi_j \, \mathrm{d}A,$$
$$l_{ij}^{(1)} = \begin{cases} l_{ij}^{(s)} & \text{for SDF,} \\ l_{ij}^{(s)} + 4\displaystyle\int_{\mathcal{S}} \left[\mathbf{n}(H^2 - K)\phi_i\phi_j\right]\mathbf{n}^{\mathrm{T}}\mathrm{d}A & \text{for WF,} \\ l_{ij}^{(s)} - 4\displaystyle\int_{\mathcal{S}} \left[\mathbf{n}(2H^2 - K)\phi_i\phi_j\right]\mathbf{n}^{\mathrm{T}}\mathrm{d}A & \text{for QSDF,} \end{cases}$$
$$l_{ij}^{(2)} = \frac{1}{2}\int_{\mathcal{S}} \left[(\nabla_s\phi_i)^{\mathrm{T}}\nabla_s\phi_j\right] \, \mathrm{d}A,$$
$$\mathbf{b}_i = \frac{1}{2}\int_{\Gamma} \mathbf{n}_c\phi_i \, \mathrm{d}s, \tag{16}$$

with

$$l_{ij}^{(s)} = 2\int_{\mathcal{S}} \left[\phi_i \oslash \phi_j - \mathbf{n}(\nabla_s\phi_i)^{\mathrm{T}}\nabla_s\phi_j\right]\mathbf{n}^{\mathrm{T}} \, \mathrm{d}A.$$

Moving the terms relating to the known control points $\mathbf{x}_{m_0+1}, \cdots, \mathbf{x}_m$ in the second equation of (15) to the equations' right-hand side, we can rewrite (15) as

$$\begin{cases} M_{m_0}^{(1)}\frac{\partial X_{m_0}(t)}{\partial t} + L_m^{(1)}Y_m(t) = \mathbf{0}, \\ M_m^{(2)}Y_m(t) \quad + L_{m_0}^{(2)}X_{m_0}(t) = B^{(2)}. \end{cases} \tag{17}$$

Note that, matrices $M_{m_0}^{(1)}$ and $M_m^{(2)}$ are symmetric and positive definite. The integrals in defining the matrix elements are computed using Gaussian quadrature formulas over the domain triangles. The knots in the barycentric coordinate form and weights of the Gaussian quadrature formulas can be found in [1].

In the boundary integrals (16), $\mathbf{n}_c$ is the co-normal of the surface, it is infeasible to compute these co-normals $\mathbf{n}_c$ from the previous approximation, since

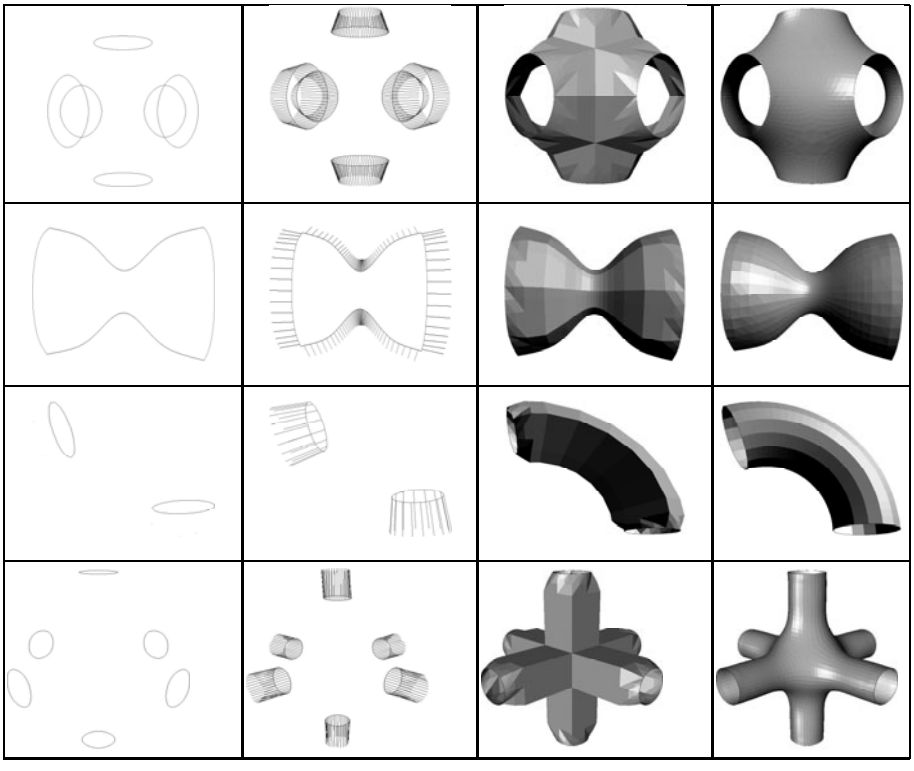


Fig. 1. First column: the boundary curves. Second column: the boundary curves with the co-normals. Third column: the initial control meshes. Last column: the control mesh of the constructed PDE subdivision surfaces.

they do not satisfy the given boundary condition. The right way is to replace $\mathbf{n}_c$ with $\mathbf{n}_c^{(\Gamma)}$. That is

$$\mathbf{b}_i = \frac{1}{2}\int_{\Gamma} \mathbf{n}_c^{(\Gamma)}\phi_i \, \mathrm{d}s.$$

3.3 Temporal Direction Discretization

Suppose we have approximate solutions

$$X_{m_0}^{(k)} = X_{m_0}(t_k) \quad \text{and} \quad Y_m^{(k)} = Y_m(t_k)$$

at $t = t_k$. We want to obtain approximate solutions $X_{m_0}^{(k+1)}$ and $Y_m^{(k+1)}$ at $t = t_{k+1} = t_k + \tau^{(k)}$ using a forward Euler scheme. Specifically, we use the following approximation

$$\frac{X_{m_0}(t_{k+1}) - X_{m_0}(t_k)}{\tau^{(k)}} \approx \frac{\partial X_{m_0}}{\partial t}.$$

The matrices $M^{(1)}$, $M^{(2)}$, $L^{(1)}$ and $L^{(2)}$ in (17) are computed using the surface data at $t = t_k$. This yields a linear system with $X_{m_0}^{(k+1)}$ and $Y_m^{(k+1)}$ as unknowns:

$$\begin{bmatrix} M_{m_0}^{(1)} & \tau^{(k)} L_m^{(1)} \\ L_{m_0}^{(2)} & M_m^{(2)} \end{bmatrix} \begin{bmatrix} X_{m_0}^{(k+1)} \\ H_m^{(k+1)} \end{bmatrix} = \begin{bmatrix} \tau^{(k)} B^{(1)} + M_{m_0}^{(1)} X_{m_0}^{(k)} \\ B^{(2)} \end{bmatrix}$$

Though the matrices $M^{(1)}$ and $M^{(2)}$ are symmetric and positive definite, the total matrix is neither symmetric nor positive definite. However the coefficient matrix of this system is highly sparse, hence a stable iterative method for its solution is desirable. We use Saad's iterative method, namely GMRES (see [15]), to solve our sparse linear system. The numerical tests show that this iterative method works very well.

4 Illustrative Examples

To illustrate our surface construction method is effective. We give several graphical examples in this section. The Given G^1 boundary conditions means specifying

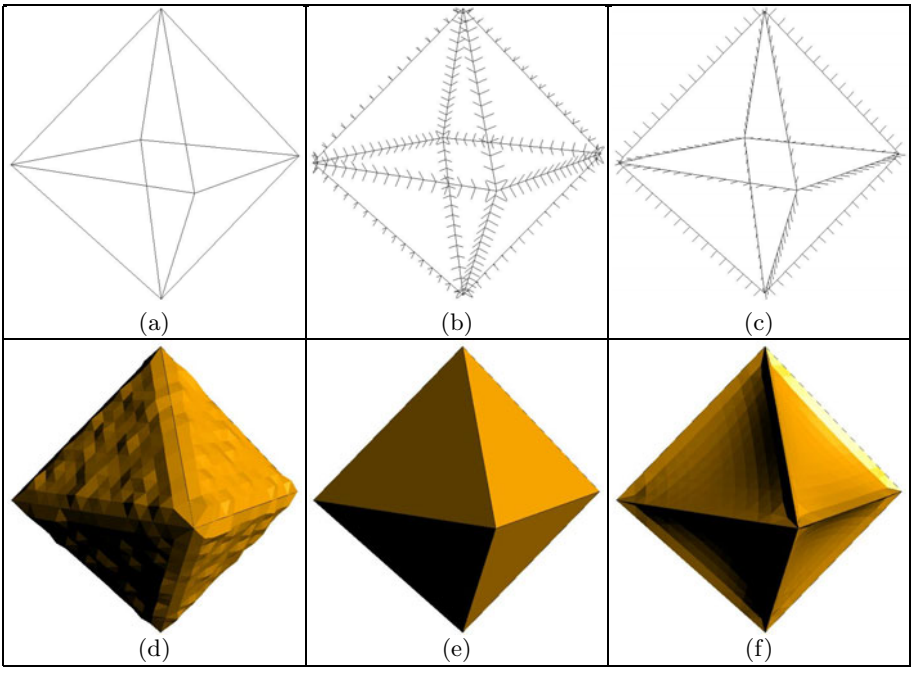


Fig. 2. (a) is the boundary curves, (b) is boundary curves with different co-normals. (c) is the boundary curves with the same co-normals. (d) is the initial control polygon. (e) and (f) are the control polygons of the PDE subdivision surfaces corresponding to the boundary curves (b) and (c), respectively.

B-spline boundary curves with co-normals on these curves where the boundary B-spline control points need to be interpolated.

For easy to illustrate, several existing triangle mesh models are used as the initial control mesh. Boundary control polygon are extracted from these models. The co-normals are computed from the initial control mesh. To show the power of our approach, the interior control vertices are sometimes perturbed.

In Fig. 1, we show one surface patch construction process. The first column in Fig. 1 shows the boundary curves. The second column shows the boundary curves as well as the co-normals on the curves. The third column shows the initial control meshes. The last column shows the control mesh of the constructed PDE subdivision surfaces. The control meshes of PDE subdivision surfaces shown in the last row, from the top down, are produced using quasi-surface diffusion flow, surface diffusion flow, Willmore flow and surface diffusion flow, respectively. The temporal step-sizes are 0.0001 for the first two, and 0.001 for the last two. The iteration numbers used are 100 for the first three, and 25 for the last one.

In Fig. 2 and 3, we join several surface patches together to form closed surfaces. For each patch, boundary curves and co-normals are provided. At the common boundaries, the co-normals may not be the same. Hence, surfaces with sharp feature can be constructed. The control meshes of the PDE subdivision surfaces as shown in Fig. 2(e) and (f) are generated using Willmore flow with 20 iterations and temporal step-size 0.01. The control mesh of the PDE subdivision surface as shown in Fig. 3(d) is generated using surface diffusion flow with 20 iterations and temporal step-size 0.0001.

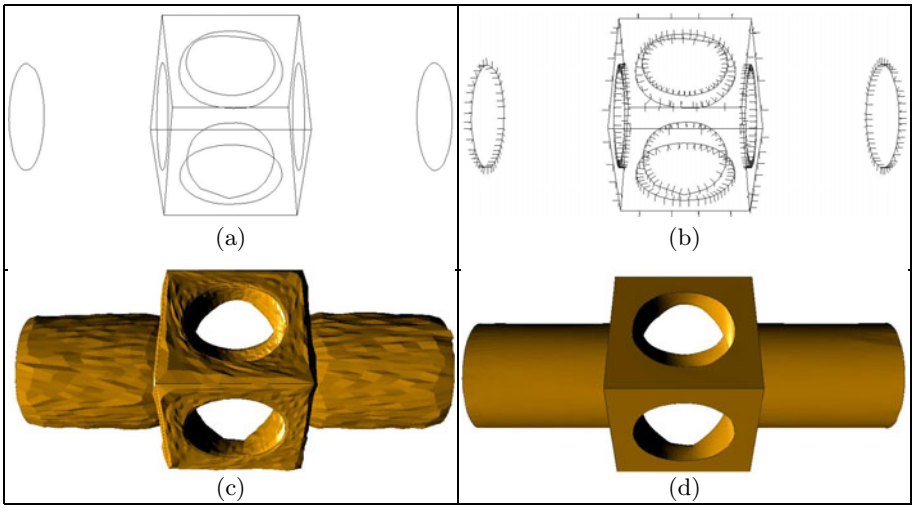


Fig. 3. (a), (b), (c) and (d) are the boundary curves, boundary curves with co-normals, initial control mesh and the PDE subdivision surface

5 Conclusions

Mesh subdivision technology can provide a simple and efficient method to construct surfaces with any topology structure, at the same time satisfy some smoothness requirement. Geometric PDEs are powerful tools for constructing high quality surfaces. In this paper, we combine these two ingredients together. We construct geometric PDE Loop's subdivision surfaces, with given G^1 boundaries condition, using three fourth-order geometric flows. A numerical solution method of the finite element based on the extended Loop's subdivision scheme is adopted, and the geometric PDE subdivision surfaces are therefore efficiently constructed.

References

1. Bajaj, C., Xu, G.: Anisotropic diffusion of subdivision surfaces and functions on surfaces. ACM Transactions on Graphics 22(1), 4–32 (2003)
2. Biermann, H., Levin, A., Zorin, D.: Piecewise-smooth Subdivision Surfaces with Normal Control. In: SIGGRAPH, pp. 113–120 (2000)
3. Bryant, R.: A duality theorem for Willmore surfaces. J. Diff. Geom. 20, 23–53 (1984)
4. Clarenz, U., Diewald, U., Dziuk, G., Rumpf, M., Rusu, R.: A finite element method for surface restoration with boundary conditions. Computer Aided Geometric Design 21(5), 427–445 (2004)
5. do Carmo, M.P.: Riemannian Geometry. Birkhäuser, Basel (1992)
6. Epstein, C.L., Gage, M.: The curve shortening flow. In: Chorin, A., Majda, A. (eds.) Wave Motion: Theory, Modeling, and Computation, pp. 15–59. Springer, New York (1987)
7. Escher, J., Simonett, G.: The volume preserving mean curvature flow near spheres. Proceedings of the American Mathematical Society 126(9), 2789–2796 (1998)
8. Jin, W., Wang, G.: Geometric Modeling Using Minimal Surfaces. Chinese Journal of Computers 22(12), 1276–1280 (1999)
9. Kuwert, E., Schätzle, R.: The Willmore flow with small initial energy. J. Diff. Geom. 57(3), 409–441 (2001)
10. Kuwert, E., Schätzle, R.: Gradient flow for the Willmore functional. Comm. Anal. Geom. 10(5), 1228–1245 (2002)
11. Man, J., Wang, G.: Approximating to Nonparameterzied Minimal Surface with B-Spline Surface. Journal of Software 14(4), 824–829 (2003)
12. Man, J., Wang, G.: Minimal Surface Modeling Using Finite Element Method. Chinese Journal of Computers 26(4), 507–510 (2003)
13. Mullins, W.W.: Two-dimensional motion of idealised grain boundaries. J. Appl. Phys. 27, 900–904 (1956)
14. Mullins, W.W.: Theory of thermal grooving. J. Appl. Phys. 28, 333–339 (1957)
15. Saad, Y.: Iterative Methods for Sparse Linear Systems, 2nd edn. SIAM, Philadelphia (2003)
16. Sapiro, G., Tannenbaum, A.: Area and length preserving geometric invariant scale–spaces. IEEE Trans. Pattern Anal. Mach. Intell. 17, 67–72 (1995)
17. Schneider, R., Kobbelt, L.: Generating fair meshes with G^1 boundary conditions. In: Geometric Modeling and Processing, Hong Kong, China, pp. 251–261 (2000)

18. Schneider, R., Kobbelt, L.: Geometric fairing of irregular meshes for free-form surface design. Computer Aided Geometric Design 18(4), 359–379 (2001)
19. Simon, L.: Existence of surfaces minimizing the Willmore functional. Commun. Analysis Geom. 1(2), 281–326 (1993)
20. Xu, G.: Interpolation by Loop's Subdivision Functions. Journal of Computational Mathematics 23(3), 247–260 (2005)
21. Xu, G.: Geometric Partial Differential Equation Methods in Computational Geometry. Science Press, Beijing (2008)
22. Xu, G., Pan, Q., Bajaj, C.: Discrete surface modelling using partial differential equations. Computer Aided Geometric Design 23(2), 125–145 (2006)
23. Xu, G., Zhang, Q.: G^2 surface modeling using minimal mean–curvature–variation flow. Computer - Aided Design 39(5), 342–351 (2007)

Efficient Computation of 3D Clipped Voronoi Diagram

Dong-Ming Yan, Wenping Wang, Bruno Lévy, and Yang Liu

The University of Hong Kong, Pokfulam Road, Hong Kong, China
Project Alice, INRIA, Nancy, France
{dmyan,wenping}@cs.hku.hk, {Bruno.Levy,Yang.Liu}@loria.fr

Abstract. The Voronoi diagram is a fundamental geometry structure widely used in various fields, especially in computer graphics and geometry computing. For a set of points in a compact 3D domain (i.e. a finite 3D volume), some Voronoi cells of their Voronoi diagram are infinite, but in practice only the parts of the cells inside the domain are needed, as when computing the centroidal Voronoi tessellation. Such a Voronoi diagram confined to a compact domain is called a clipped Voronoi diagram. We present an efficient algorithm for computing the clipped Voronoi diagram for a set of sites with respect to a compact 3D volume, assuming that the volume is represented as a tetrahedral mesh. We also describe an application of the proposed method to implementing a fast method for optimal tetrahedral mesh generation based on the centroidal Voronoi tessellation.

Keywords: Voronoi diagram, Delaunay triangulation, centroidal Voronoi tessellation, tetrahedral meshing.

1 Introduction

The Voronoi diagram (VD) is a fundamental and important geometry structure which has numerous applications in different areas, such as shape modeling [3], motion planning [18], scientific visualization [5], collision detection [19], geography [11], chemistry [16] and so on. For a finite set of sites (points in 3D), each site is associated with a Voronoi cell containing all the points closer to the site than to any other sites; all these cells constitute the Voronoi diagram of the set of sites.

Suppose that a set of sites in a compact domain in 3D are given. The Voronoi cells of those sites that on the boundary of the convex hull of all the sites are infinite. However, in many applications one often needs only the parts of the Voronoi cells inside the domain, as when computing the centroidal Voronoi tessellation. That is, the Voronoi diagram with respect to the given domain is defined as the intersection of the 3D Voronoi diagram and the domain, and is therefore called the *clipped Voronoi diagram*. The corresponding Voronoi cells are called the *clipped Voronoi cells*, see Figure 1 for 2D examples.

B. Mourrain, S. Schaefer, and G. Xu (Eds.): GMP 2010, LNCS 6130, pp. 269–282, 2010.

Computing the clipped Voronoi diagram in a convex domain is relatively easy – one just needs to compute the intersection of each Voronoi cell and the domain, both being convex. However, the operations would be more involved if the domain is non-convex and there has been no previous work on computing exact clipped Voronoi diagram for non-convex domains with arbitrary topology. A brute-force implementation would be inefficient because of the domain complexity.

Contributions: We present an efficient algorithm for computing clipped Voronoi diagrams of arbitrary closed 3D objects. The idea of our approach is to represent the input domain by a set of convex primitives. We use tetrahedron as the basic primitive in this paper – that is, the 3D domain is represented as a 3D tetrahedral mesh. Then the intersection of a 3D Voronoi cell and the input domain is reduced to computing the intersection of a 3D Voronoi cell and a set of tetrahedra, which can be done efficiently. The key to an efficient implementation is assigning each tetrahedron to its incident Voronoi cells, i.e., those Voronoi cells that intersect with the tetrahedron. Then we only need to compute the intersection between the tetrahedron and its incident cells. We identify the incident Voronoi cells for all the tetrahedra using neighborhood propagation.

This work extends our previous work [22] on computing the restricted Voronoi diagram (RVD) of a mesh surface in the following aspects. There we discuss how to compute a Voronoi diagram of sites on a triangle mesh surface by restricting the 3D Voronoi diagram of the sites to the surface, which involves the intersection of 3D Voronoi cells with individual triangles of the mesh surface. Also, in [22], we assume no connectivity information between the triangle elements and the intersection pairs of Voronoi cells and triangle elements are found with the assistance of a *kd*-tree. In this paper, we further improve the efficiency of surface RVD computation algorithm [22] by replacing the *kd*-tree query by a more efficient neighbor propagation approach, assuming the availability of the mesh connectivity information.

1.1 Previous Work

A detailed survey of the Voronoi diagram is out of the scope of this paper, the reader is referred to [4,10,15] for the properties and applications of the Voronoi diagram. Existing techniques can compute the Voronoi diagram for point sites in 2D and 3D Euclidean spaces efficiently. There are several robust implementations that are publicly available, such as CGAL [1] and Qhull [6].

To speed up the Voronoi diagram computation in specific applications, many researchers focus on computing approximated Voronoi diagram on discrete spaces with the help of the GPU (*Graphical Processing Unit*). Hoff III *et al.* [12] propose a technique for computing discrete generalized Voronoi diagram using graphics hardware. The Voronoi diagram computation is cast into a clustering problem in the discrete voxel/pixel space. Sud *et al.* [19] present an n-body proximity query algorithm based on computing the discrete 2^{nd} order Voronoi diagram on

the GPU. GPU based algorithms are fast but produce only a discrete approximation of the true Voronoi diagram.

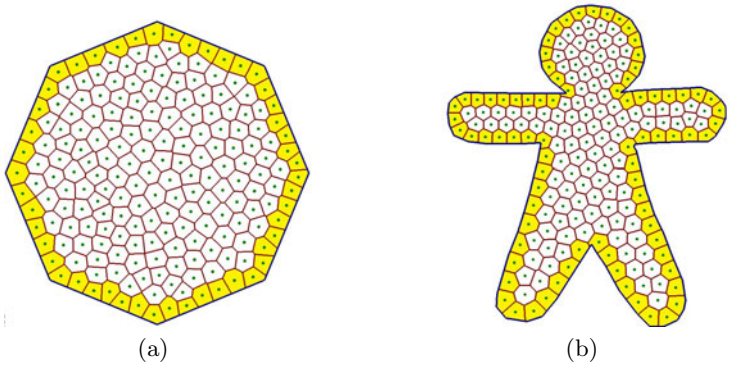
Yan *et al.* [22] present a direct algorithm for computing the restricted Voronoi diagram (RVD) [9] on mesh surfaces. In that method no connectivity information between the triangle facets is assumed, and a *kd*-tree is used to find the nearest sites of each triangle in order to identify its incident Voronoi cells. The incident 3D Voronoi cells of each triangle face are identified starting from the nearest site and the intersection between the triangle and its incident Voronoi cells is computed by Sutherland's clipping algorithm [20]. Let m be the number of triangles and n the number of sites. Then the time complexity of the method in [22] is $O(m \log n)$ assuming that the number of incident cells of each triangle is bounded, since a *kd*-tree is used for the nearest site query.

1.2 Outline

The remainder of this paper is organized as follows: We give the problem formulation in Section 2 and the overview of our algorithm in Section 3. We present our algorithm for computing clipped Voronoi diagram of 3D objects in Section 4. As an application of our algorithm, we present in Section 5 a CVT-based tetrahedral meshing method built on the top of our new method for computing the 3D clipped Voronoi diagram. Experimental results are given in Section 6 and we draw conclusions in Section 7.

2 Problem Formulation

We consider computing the exact clipped Voronoi diagram of closed 3D objects. Figure 1 illustrates the problem with two 2D examples of the clipped Voronoi diagram with a convex domain and a non-convex domain, respectively.



(a) (b)

Fig. 1. Clipped Voronoi diagram on 2D convex (a) and non-convex (b) domains. Shaded cells are boundary Voronoi cells. The number of seeds is 200 for each example.

Given a set of sites $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^n$ in 3D, the Voronoi diagram of $\mathbf{X}$ is defined by a collection of n Voronoi cells $\Omega = \{\Omega_i\}_{i=1}^n$, where

$$\Omega_i = \{\mathbf{x} \in \mathbb{R}^3, \|\mathbf{x} - \mathbf{x}_i\| \leq \|\mathbf{x} - \mathbf{x}_j\|, \forall j \neq i\}.$$

Each Voronoi cell Ω_i is the intersection of a set of 3D half-spaces, delimited by the bisecting planes of the Delaunay edges incident to the site $\mathbf{x}_i$.

Let $\mathcal{M}$ denote the input domain, which is assumed to be a connected compact set in 3D. The clipped Voronoi diagram for the sites $\mathbf{X}$ with respect to $\mathcal{M}$ is defined as the intersection of the 3D Voronoi diagram Ω and $\mathcal{M}$, denoted as $\Omega|_{\mathcal{M}} = \{\Omega_i|_{\mathcal{M}}\}_{i=1}^n$, where

$$\Omega_i|_{\mathcal{M}} = \Omega_i \bigcap \mathcal{M} = \{\mathbf{x} \in \mathcal{M}, \|\mathbf{x} - \mathbf{x}_i\| \leq \|\mathbf{x} - \mathbf{x}_j\|, \forall j \neq i\},$$

which is the intersection of the Voronoi cell Ω_i and $\mathcal{M}$. We call $\Omega_i|_{\mathcal{M}}$ the *clipped Voronoi cell* with respect to $\mathcal{M}$.

3 Algorithm Overview

In this section we describe an efficient algorithm for computing the clipped Voronoi diagram of 3D objects. We suppose that the domain $\mathcal{M}$ is represented as a set of tetrahedra; other types of convex primitives can be used for this decomposition as well.

Suppose that the input domain $\mathcal{M}$ is given by a tetrahedral mesh, that is $\mathcal{M} = \{\mathcal{V}, \mathcal{T}\}$, where $\mathcal{V} = \{\mathbf{v}_k\}_{k=1}^{n_v}$ is the set of mesh vertices and $\mathcal{T} = \{\mathbf{t}_i\}_{i=1}^m$ is the set of tetrahedral elements. Each tetrahedron (tet for short in the following) $\mathbf{t}_i$ stores the information of its four incident vertices and four adjacent tets. The four vertices are assigned indices $0, 1, 2, 3$ and so are the four adjacent tets. The index of an adjacent tet is the same as the index of the vertex which is opposite to the tet. The boundary of $\mathcal{M}$ is a triangle mesh, denoted as $\mathcal{S} = \{\mathbf{f}_j\}_{j=1}^{n_f}$, which is assumed to be 2-manifold. Each boundary triangle facet $\mathbf{f}_j$ stores the indices of three neighboring facets and the index of its containing tet.

The clipped Voronoi cells $\{\Omega_i|_{\mathcal{M}}\}$ can be classified into two types: *inner Voronoi cells* that are contained in the interior of $\mathcal{M}$ and *boundary Voronoi cells* that intersect the boundary $\mathcal{S}$ of the domain $\mathcal{M}$. Since the inner Voronoi cells of $\Omega|_{\mathcal{M}}$ are entirely inside the domain, there is no need to clip them against the boundary surface $\mathcal{S}$. So we just need to concentrate on computing the boundary Voronoi cells, see Figure 1 for 2D examples of clipped Voronoi diagrams.

Therefore the problem now is how to identify all the sites whose Voronoi cells intersect the boundary $\mathcal{S}$. To solve this problem, we first compute the surface restricted Voronoi diagram (RVD) for all the sites $\mathbf{X}$. The sites whose Voronoi cells intersect the domain boundary surface are called the *boundary sites* and their cells are the boundary Voronoi cells. So we will just compute the intersection of the boundary Voronoi cells with the domain $\mathcal{M}$. The reader is referred to [21,22] for details of surface RVD computation. In this paper, we further improve the RVD computation algorithm by replacing the *kd*-tree search in [22] by a more efficient neighbor propagation approach.

4 Clipped Voronoi Diagram Computation

There are three main steps of our algorithm for computing the clipped Voronoi diagram:

1. **Voronoi diagram construction:** This step computes the Delaunay triangulation for the input sites, from which we extract the 3D Voronoi diagram;
2. **Surface RVD computation:** We compute the surface RVD to identify all the boundary Voronoi cells;
3. **Clipped Voronoi cells construction:** The 3D clipped Voronoi cells for all the boundary Voronoi cells are computed.

In the following, we will explain each step in details.

4.1 Voronoi Diagram Construction

We first build a 3D Delaunay triangulation from input sites $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^n$, using CGAL. The corresponding 3D Voronoi diagram $\Omega = \{\Omega_i\}_{i=1}^n$ is constructed as the dual of the Delaunay triangulation, as defined in Section 2.

4.2 Surface RVD Computation

For the given set of sites $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^n$ and the boundary surface $\mathcal{S}$, the restricted Voronoi diagram (RVD) is defined as the intersection of the 3D Voronoi diagram Ω and the surface $\mathcal{S}$, denoted as $\mathbf{R} = \{\mathbf{R}_i\}_{i=1}^n$, where $\mathbf{R}_i = \Omega_i \bigcap \mathcal{S}$ [9]. Each $\mathbf{R}_i$ is called a *restricted Voronoi cell* (RVC). We compute the surface RVD using neighbor propagation, which is faster than searching using the *kd*-tree structure, as shown by our tests.

Now we are going to explain the propagation steps. Refer to Figure 2. We start from a seed triangle and one of its incident cells, which can be found by a linear search function. Here we assume that a triangle $\mathbf{f}_0$ of boundary $\mathcal{S}$ is the

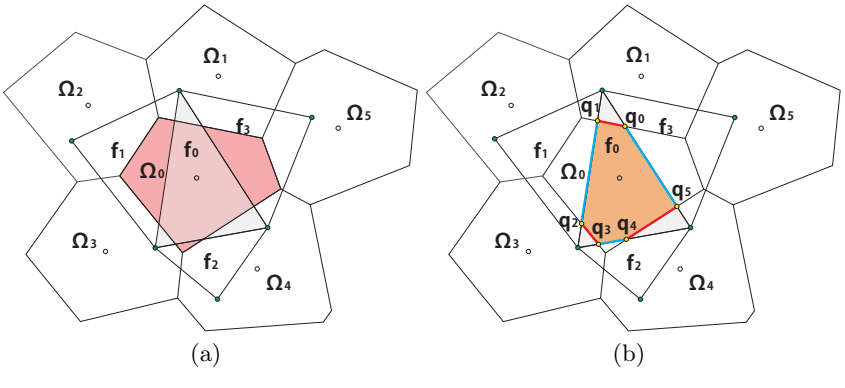


Fig. 2. Illustration of propagation process

seed triangle and the Voronoi cell Ω_0 is the corresponding cell of the nearest site of $\mathbf{f}_0$, as shown in Figure 2(a). We use an FIFO queue $\mathcal{Q}$ to store all the incident cell-triangle pairs to be processed. To start, the initial pair $\{\mathbf{f}_0, \Omega_0\}$ is pushed into the queue. The algorithm repeatedly pops out the pair in the front of $\mathcal{Q}$ and computes their intersection. During the intersection process, new valid pairs are identified and pushed back into $\mathcal{Q}$. The process terminates when $\mathcal{Q}$ is empty.

The key issue now is how to identify all the valid cell-triangle pairs during the intersection. Assume that $\{\mathbf{f}_0, \Omega_0\}$ is popped out from Q, as shown in Figure 2. We clip $\mathbf{f}_0$ against the bounding planes of Ω_0, which has five bisecting planes, i.e., $[\mathbf{x}_0, \mathbf{x}_1]$, $[\mathbf{x}_0, \mathbf{x}_2]$..., $[\mathbf{x}_0, \mathbf{x}_5]$. The resulting polygon is represented by $\mathbf{q}_0, \mathbf{q}_1, ..., \mathbf{q}_5$, as shown in Figure 2(b). Since the line segment $\overline{\mathbf{q}_0\mathbf{q}_1}$ is the intersection of $\mathbf{f}_0$ and $[\mathbf{x}_0, \mathbf{x}_1]$, we know that the opposite cell Ω_1 is also an incident cell of $\mathbf{f}_0$, thus the pair $\{\mathbf{f}_0, \Omega_1\}$ is an incident pair. Since the common edge of $[\mathbf{f}_0, \mathbf{f}_1]$ has intersection with Ω_0, the adjacent facet $\mathbf{f}_1$ also has intersection with cell Ω_0, thus the pair $\{\mathbf{f}_1, \Omega_0\}$ is also an incident pair. So is the pair $\{\mathbf{f}_2, \Omega_0\}$. The other incident pairs are found in the same manner. To keep the same pair from being processed multiple times, we store the incident facet indices for each cell. Before pushing a new pair into the queue, we add the facet index to the incident facet indices set of the cell. The pair is pushed into the queue only if the facet is not contained in the incident facets set of the cell; otherwise the pair is discarded. Each time after intersection computation, the resulting polygon is made associated with the surface RVC of the current site. The surface RVD computation terminates when the queue is empty. Those sites that have non-empty surface RVC are marked as boundary sites, denoted as $\mathbf{X}_b = \{\mathbf{x}_i | \mathbf{R}_i \neq \emptyset\}$. The pseudo code of the algorithm is given in Algorithm 4.1.

Algorithm 4.1. Surface RVD computation algorithm

```
input  : sites X, boundary mesh S
output: surface RVD R of X on S
begin
    Ω ← VoronoiDiagram(X) ;
    {f_0, Ω_0} ← FindInitialPair() ;
    queue Q ← {f_0, Ω_0} ;
    while Q ≠ ∅ do
        {f_t, Ω_t} ← Q.pop() ;
        polygon poly ← Intersect(f_t, Ω_t) ;
        R_t ← R_t ⋃ poly ;
        Q.push(NewIncidentPairs(poly)) ;
    end
end
```

4.3 Clipped Voronoi Cells Construction

Once the boundary sites $\mathbf{X}_b$ are found, we will compute the clipped Voronoi cells for these sites. The boundary Voronoi cells computation is similar to the

surface RVD computation presented in Section 4.2, with the difference that we restrict the computation on boundary cells only. For each boundary cell, we have recorded the indices of its incident boundary triangles. We know that the neighboring tet of each boundary triangle is also incident to the cell. We also store the indices of incident tet for each boundary cell. The incident tet set is initialized as the neighboring tet of the incident boundary triangle.

We use an FIFO queue to facilitate this process. The queue is initialized by a set of incident cell-tet pairs $(\Omega_i, \mathbf{t}_j)$, which can be obtained from the boundary cell and its initial incident tet set.

The pair $(\Omega_i, \mathbf{t}_j)$ in front of Q is popped out repeatedly. We compute the intersection of Ω_i and $\mathbf{t}_j$ again by Sutherland-Hodgman clipping algorithm [20] and identity new incident pairs at the same time. We clip the tet $\mathbf{t}_j$ by bounding planes of cell Ω_i one by one. If the current bounding plane has intersection with $\mathbf{t}_j$, We check the opposite Voronoi cell Ω_o that shares the current bisecting plane with Ω_i, if Ω_o is a boundary cell and $\mathbf{t}_j$ is not in the incident set of Ω_o, a new pair $(\Omega_o, \mathbf{t}_j)$ is found. We also check the neighbor tets who share the facets clipped by the current bisecting plane. Those cells that are not in the incident set of Ω_i are added to its set, and new pairs are pushed into the queue. After clipping, the resulting polyhedron is made associated with the clipped Voronoi cell $\Omega_i|_{\mathcal{M}}$ of site $\mathbf{x}_i$. This process terminates when Q is empty.

5 Tetrahedral Mesh Generation

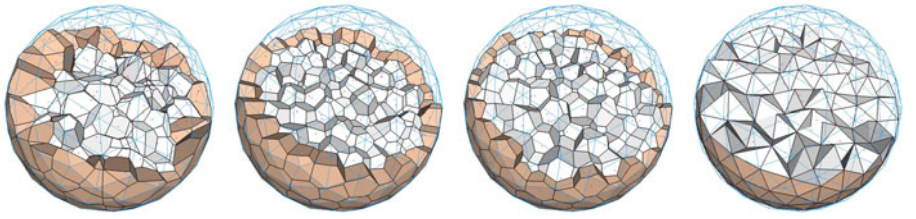
As an application, we implemented an efficient method for tetrahedral meshing based on centroidal Voronoi tessellation (CVT) [7,8], which utilizes heavily the computation of the 3D clipped Voronoi diagram. The L-BFGS method in [14] for computing CVT is used in our implementation. We will briefly describe this framework and present our experimental results in Section 6.

The centroidal Voronoi tessellation is a special kind of Voronoi tessellation such that each seed $\mathbf{x}_i$ coincides with the mass center of its Voronoi region. In the context of CVT based tetrahedral meshing, the CVT energy function is defined on the input mesh $\mathcal{M}$, *i.e.*,

$$F(\mathbf{X}) = \sum_{i=1}^{n} \int_{\Omega_i|_{\mathcal{M}}} \rho(\mathbf{x}) \|\mathbf{x} - \mathbf{x}_i\|^2 \, \mathrm{d}\sigma, \tag{1}$$

where $\rho(\mathbf{x}) > 0$ is a user-defined density function. When ρ is a constant, we get a uniform CVT. The reader is referred to [7] for preliminaries of CVT and [14] for details of convergency analysis of CVT energy functions. We omit them here since they are not the main contribution of this paper.

There are three steps of the CVT based meshing framework: initialization, iterative optimization, and mesh extraction. Our mesh generation framework is illustrated by the example in Figure 3.



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

Fig. 3. Illustration of the proposed tetrahedral meshing algorithm. The wire frame is the boundary of input mesh. (a) Clipped Voronoi diagarm of initial sites (the boundary Voronoi cells are shaded); (b) result of unconstrained CVT with $\rho = 1$; (c) result of constrained optimization. Notice that boundary seeds are constrained on the surface $\mathcal{S}$; (d) final uniform tetrahedral meshing result.

5.1 Initialization

In this step, we build a uniform grid to store the sizing field for adaptive meshing. Following the approach in [2], we first compute the *local feature size* (lfs) for all boundary vertices and then use a fast matching method to construct a sizing field on the grid. This grid is also used for efficient initial sampling (Figure 3(a)). The reader is referred to [2] for details.

5.2 Optimization

There are two phases of the global optimization part: unconstrained CVT optimization and constrained CVT optimization. In the first phase, we optimize the positions of the sites inside the input volume without any constraint, which yields a well-spaced distribution of the sites within the domain, with no sites lying on the domain boundary surface (Figure 3(b)). In the second phase, we identify all the boundary sites, i.e. those sites whose Voronoi cells intersect the domain boundary surface; we project these sites onto the boundary surface and they will be constrained to the boundary surfaces during the subsequent optimization. Then all the boundary sites and the inner sites are optimized simultaneously, again with respect to the CVT energy function (Figure 3(c)). The details of these steps are explained in the following.

CVT optimization. In the first phase, we use the L-BFGS method [14] to compute the CVT by minimizing the CVT energy function (Eqn. 1). To apply the L-BFGS method to minimize the CVT energy function we need the gradient of the CVT energy function. The partial derivative of the energy function w.r.t. each site is given by the following equation [13]:

$$\frac{\partial F}{\partial \mathbf{x}_i} = 2m_i(\mathbf{x}_i - \mathbf{x}_i^*), \tag{2}$$

here $m_i = \int_{\Omega_i|_{\mathcal{M}}} \rho(\mathbf{x})\|\mathbf{x} - \mathbf{x}_i\|^2 \, \mathrm{d}\sigma$, and $\mathbf{x}_i^*$ is the centroid given by

$$x_i^* = \frac{\int_{\Omega_i|_{\mathcal{M}}} \rho(\mathbf{x})\mathbf{x}\,\mathrm{d}\sigma}{\int_{\Omega_i|_{\mathcal{M}}} \rho(\mathbf{x})\,\mathrm{d}\sigma}. \tag{3}$$

To integrate this function, each clipped Voronoi cell $\Omega_i|_{\mathcal{M}}$ is split into a set of sub-tets $\{\tau_k\}$ by simply connecting the centroid of each clipped polyhedron of $\Omega_i|_{\mathcal{M}}$ with its triangulated facets (see Section 4.3). As discussed in [2], the exact integration of the density function may not improve the quality very much, since the density function is also discretely defined. We use a one-point approximation of the density function at the centroid of each sub-tet of the Voronoi cells, i.e.,

$$x_i^* = \frac{\sum_{\tau_k \in \Omega_i|_{\mathcal{M}}} \rho(\mathbf{c}_k)\mathbf{c}_k \cdot |\tau_k|}{\sum_{\tau_k \in \Omega_i|_{\mathcal{M}}} \rho(\mathbf{c}_k) \cdot |\tau_k|}, \tag{4}$$

where $\mathbf{c}_k$ and $|\tau_k|$ are the centroid and the volume of sub-tet τ_k, respectively.

Once the L-BFGS method is used to compute the updated sites, we need to compute the exact clipped Voronoi cells of these sites in the domain $\mathcal{M}$. Then we start the next iteration until convergence or some termination condition is met. After convergence, all the sites of the boundary Voronoi cells will be projected to the domain boundary surface.

Constrained CVT. During the second phase of optimization, all the boundary sites will be constrained on the boundary. The partial derivative of the energy function w.r.t each boundary site is computed as:

$$\left.\frac{\partial F}{\partial \mathbf{x}_i}\right|_{\mathcal{S}} = \frac{\partial F}{\partial \mathbf{x}_i} - \left[\frac{\partial F}{\partial \mathbf{x}_i} \cdot \mathbf{N}(\mathbf{x}_i)\right] \mathbf{N}(\mathbf{x}_i), \tag{5}$$

where $\mathbf{N}(\mathbf{x}_i)$ is the unit normal vector of the boundary surface at the boundary site $\mathbf{x}_i$ [14]. The partial derivative with respect to an inner site is still computed by Eqn. (2). Both boundary and inner sites will be optimized simultaneously, applying again the L-BFGS method to minimize the CVT energy function.

Sharp features are preserved in a similar way as how the boundary sites are treated. For example, we project sites on sharp edges on the boundary and allow them to vary only along these edges during the second stage of optimization. For details, please refer to [22] where these steps are described in the context of surface remeshing.

5.3 Final Mesh Extraction

After convergence, we shall extract a well-shaped surface triangle remesh for the domain boundary as well as a tetrahedral mesh for the domain interior as the dual of the final CVT. We first compute a boundary remesh $\mathcal{S}'$ from all the boundary seeds $\mathbf{X}_b$ [22]. The final tetrahedral mesh is then extracted by computing a conformal Delaunay triangulation from $\mathcal{S}'$ and all the inner seeds (Figure 3(d)).

6 Experimental Results

Our algorithm is implemented in C++ on both Windows and Linux platform. We use CGAL [1] for Delaunay triangulation and TetGen [17] for background mesh generation when the input is given only as a closed boundary mesh. All the experimental results are tested on a laptop with 2.4Ghz processor and 2Gb memory.

Efficiency. We first demonstrate the performance of the proposed clipped VD computation algorithm. We progressively sample points inside an input tetrahedral mesh, which contains $1k$ boundary triangles and $3,368$ tets. The number of sites increases from 10 to $6 \times e^5$. The results are shown in Figure 4 and the timing curves are shown in Figure 5. From the timing curve in Figure 5, we can see that the time spent on surface RVD computation is much less than Delaunay

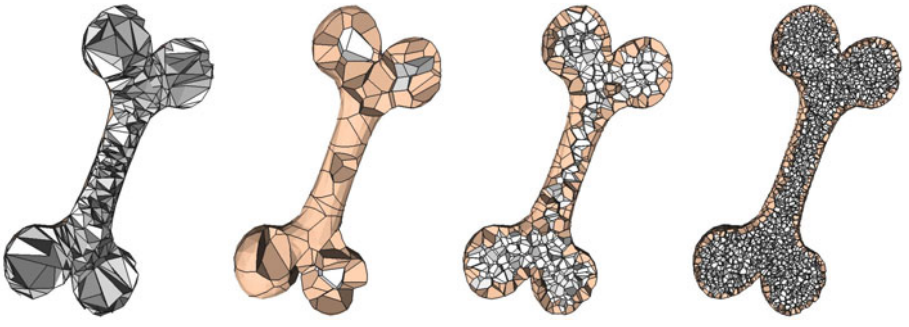


Fig. 4. (a) Input Bone model ($3,368$ tets and $1k$ boundary triangles); (b) clipped Voronoi diagram of 100 sites; (c) $1k$ sites; (d) $10k$ sites

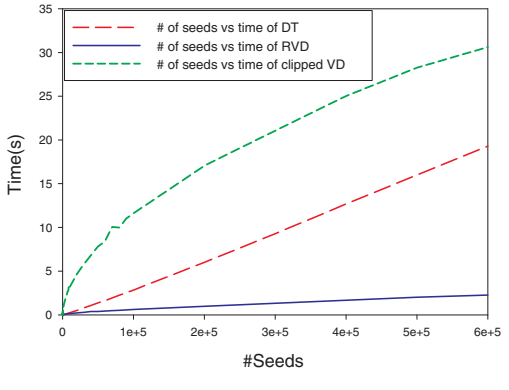


Fig. 5. Timing curve of clipped Voronoi diagram computation against the number of sites on Bone model

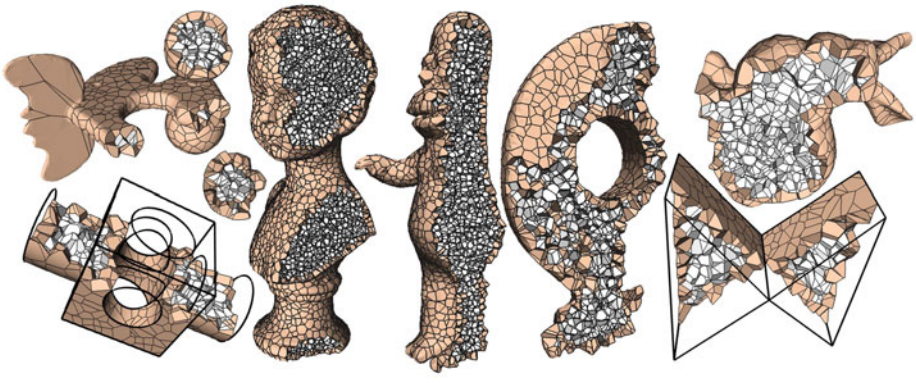


Fig. 6. Results of clipped Voronoi diagram computation

Table 1. Statistics of clipped Voronoi diagram computation on various models. $|\mathcal{T}|$ is the number of input tetrahedra. $|\mathcal{S}|$ is the number of boundary triangles. $|\mathbf{X}|$ is the number of sites. $|\mathbf{X}_b|$ is the number of boundary sites. Time (in seconds) is the total time for clipped Voronoi diagram computation, including both Delaunay triangulation and surface RVD computation.

Model	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{S}\rvert$	$\lvert\mathbf{X}\rvert$	$\lvert\mathbf{X}_b\rvert$	Time
Twoprism	68	30	1k	572	0.2
Bunny	10k	3k	2k	734	1.8
Elk	34.8k	10.4k	2k	1,173	3.1
Block	77.2k	23.4	1k	659	4.7
Homer	16.2k	4,594	10k	2,797	6.3
Rockerarm	212k	60.3k	3k	1,722	12.1
Bust	68.5k	20k	30k	5k	16.2

triangulation, since only a small portion of all the sites are boundary sites. More results of clipped Voronoi diagram computation of various 3D objects are given in Figure 6 and the timing statistics are given in Table 1.

The computational time of the clipped VD computation algorithm is proportional to the total number of incident cell-tet pairs (Section 4.3). Therefore, an input mesh with a small number of tetrahedral elements would help improve the efficiency. In our experiments, all the input tetrahedral meshes are generated by the robust meshing software TetGen [17] with conforming boundary.

Tetrahedral meshing. The complete process of the proposed tetrahedral meshing framework is illustrated in Figure 3. Figure 7 shows two adaptive tetrahedral meshing examples, with lfs as the density function. Figure 8 gives two examples with sharp features preserved. Our framework can generate high quality meshes efficiently and robustly. The running time for obtaining final results ranges from seconds to minutes, depending on the size of the input tetrahedral mesh and the desired number of sites.

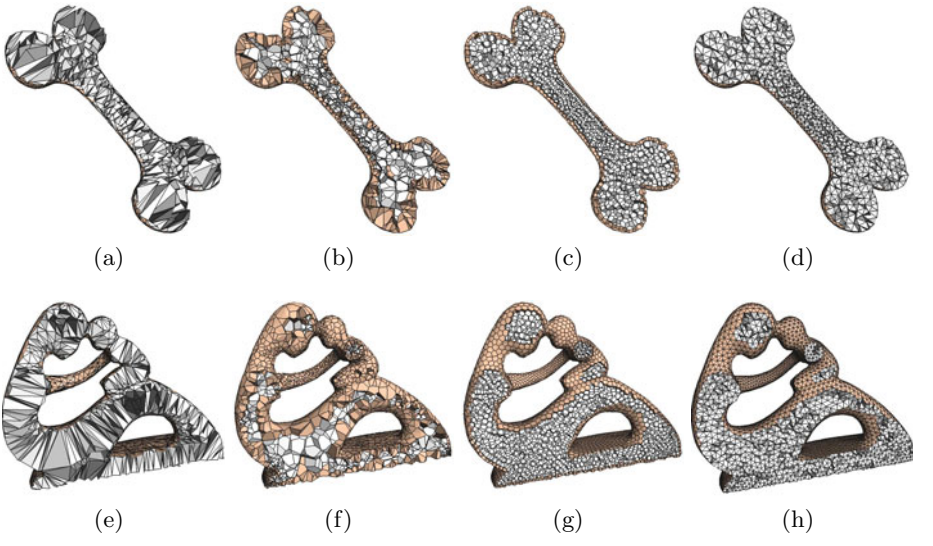


Fig. 7. Adaptive meshing result of Bone (top row) and Fertility (bottom row). (a)&(e) Cut-view of input meshes; (b)&(f) clipped Voronoi diagrams of initial samples; (c)&(g) optimization results; (d)&(h) tetrahedral meshing results.

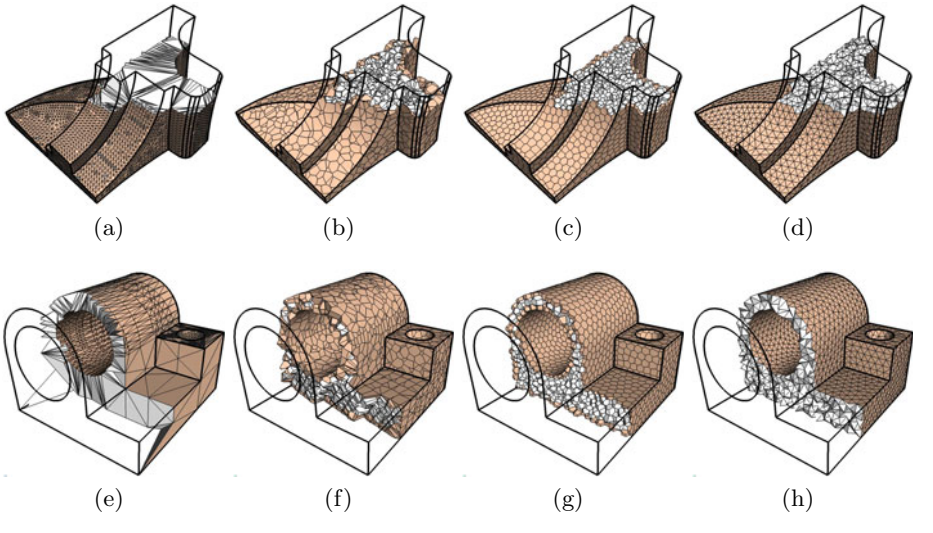


Fig. 8. Uniform meshing result of Fandisk (top row) and Joint (bottom row). (a)&(e) Cut-view of input meshes; (b)&(f) clipped Voronoi diagrams of initial samples; (c)&(g) optimization results; (d)&(h) tetrahedral meshing results with feature preserved.

7 Conclusion

We have presented an efficient algorithm for computing clipped Voronoi diagram for closed 3D objects, which has been a difficult problem without an efficient implementation. As an application, we present a new CVT based tetrahedral meshing algorithm which combines our fast clipped VD computation with fast CVT optimization [14]. In the future, we plan to investigate GPU-based methods to further improve the efficiency.

Acknowledgements

The Bunny model is the courtesy of the Stanford 3D Scanning Repository. The other 3D models used in this paper are from AIM@Shape project. We would like to thank Mr. Feng Sun for many helpful discussions during this work.

Dong-Ming Yan and Wenping Wang are partially supported by the General Research Funds (718209, 717808) of Research Grant Council of Hong Kong, NSFC-Microsoft Research Asia co-funded project (60933008), and National 863 High-Tech Program of China (2009AA01Z304). Bruno Lévy and Yang Liu are supported by the European Research Council (GOODSHAPE FP7-ERC-StG-205693).

References

1. CGAL, Computational Geometry Algorithms Library, http://www.cgal.org
2. Alliez, P., Cohen-Steiner, D., Yvinec, M., Desbrun, M.: Variational tetrahedral meshing. ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2005) 24(3), 617–625 (2005)
3. Alliez, P., Ucelli, G., Gotsman, C., Attene, M.: Recent advances in remeshing of surfaces. Shape Analysis and Structuring, 53–82 (2008)
4. Aurenhammer, F.: Voronoi diagrams: a survey of a fundamental geometric data structure. ACM Computing Surveys 23(3), 345–405 (1991)
5. Balzer, M., Deussen, O.: Voronoi treemaps. In: Proceedings of the 2005 ACM Symposium on Software Visualization, pp. 165–172 (2005)
6. Barber, C.B., Dobkin, D.P., Huhdanpaa, H.: The quickhull algorithm for convex hulls. ACM Trans. Math. Software 22, 469–483 (1996)
7. Du, Q., Faber, V., Gunzburger, M.: Centroidal Voronoi tessellations: applications and algorithms. SIAM Review 41(4), 637–676 (1999)
8. Du, Q., Gunzburger, M., Ju, L.: Advances in studies and applications of centroidal Voronoi tessellations. Numer. Math. Theor. Meth. Appl. (to appear 2010)
9. Edelsbrunner, H., Shah, N.R.: Triangulating topological spaces. Int. J. Comput. Geometry Appl. 7(4), 365–378 (1997)
10. Fortune, S.: Voronoi diagrams and Delaunay triangulations. In: Computing in Euclidean Geometry, pp. 193–233 (1992)
11. Gold, C.M.: What is GIS and what is not? Transactions in GIS 10(4), 505–519 (2006)
12. Hoff III, K.E., Keyser, J., Lin, M.C., Manocha, D.: Fast computation of generalized Voronoi diagrams using graphics hardware. In: Proceedings of ACM SIGGRAPH 1999, pp. 277–286 (1999)

13. Iri, M., Murota, K., Ohya, T.: A fast Voronoi diagram algorithm with applications to geographical optimization problems. In: Proceedings of the 11th IFIP Conference on System Modelling and Optimization, pp. 273–288 (1984)
14. Liu, Y., Wang, W., Lévy, B., Sun, F., Yan, D.-M., Lu, L., Yang, C.: On centroidal Voronoi tessellation: Energy smoothness and fast computation. ACM Transactions on Graphics 28(4), Article No. 101 (2009)
15. Okabe, A., Boots, B., Sugihara, K., Chiu, S.N.: Spatial Tessellations: Concepts and Applications of Voronoi Diagrams, 2nd edn. Wiley, Chichester (2000)
16. Poupon, A.: Voronoi and Voronoi-related tessellations in studies of protein structure and interaction. Current Opinion in Structural Biology 14(2), 233–241 (2004)
17. Si, H.: TetGen: A quality tetrahedral mesh generator and three-dimensional Delaunay triangulator, http://tetgen.berlios.de/
18. Sud, A., Andersen, E., Curtis, S., Lin, M.C., Manocha, D.: Real-time path planning in dynamic virtual environments using multiagent navigation graphs. IEEE Transactions on Visualization and Computer Graphics 14(3), 526–538 (2008)
19. Sud, A., Govindaraju, N.K., Gayle, R., Kabul, I., Manocha, D.: Fast proximity computation among deformable models using discrete Voronoi diagrams. ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2006) 25(3), 1144–1153 (2006)
20. Sutherland, I.E., Hodgman, G.W.: Reentrant polygon clipping. Communications of the ACM 17(1), 32–42 (1974)
21. Yan, D.-M.: Variational Shape Segmentation and Mesh Generation. Phd dissertation, The University of Hong Kong (2010)
22. Yan, D.-M., Lévy, B., Liu, Y., Sun, F., Wang, W.: Isotropic remeshing with fast and exact computation of restricted Voronoi diagram. Computer Graphics Forum (Proceedings of SGP 2009) 28(5), 1445–1454 (2009)

Selecting Knots Locally for Curve Interpolation with Quadratic Precision*

Caiming Zhang[1], Wenping Wang[2], Jiaye Wang[1], and Xuemei Li[1]

[1] School of Computer Science and Technology, Shandong University, Jinan 250101, China
[2] The Department of Computer Science, University of Hong Kong, Pokfulam Road, Hong Kong

Abstract. There are several prevailing methods for selecting knots for curve interpolation. A desirable criterion for knot selection is whether the knots can assist an interpolation scheme to achieve the reproduction of polynomial curves of certain degree if the data points to be interpolated are taken from such a curve. For example, if the data points are sampled from an underlying quadratic polynomial curve, one would wish to have the knots selected such that the resulting interpolation curve reproduces the underlying quadratic curve; and in this case the knot selection scheme is said to have quadratic precision. In this paper we propose a local method for determining knots with quadratic precision. This method improves on upon our previous method that entails the solution of a global equation to produce a knot sequence with quadratic precision. We show that this new knot selection scheme results in better interpolation error than other existing methods, including the chord-length method, the centripetal method and Foley's method, which do not possess quadratic precision.

Keywords: parametric curves, knots, quadratic polynomial, interpolation.

1 Introduction

The problem of computing parametric interpolating curves is of fundamental importance in computer aided geometric design, scientific computing and computer graphics. Given a sequence of data points P_i, $i = 1, 2, \ldots, n$, an interpolation scheme needs the so called *knots* t_i associated with the P_i to produce an interpolation curve $P(t)$ with $P(t_i) = P_i$. The quality of the interpolation curve, in terms of fairness and interpolation error, depends on not only the particular interpolation scheme used, but also the selection of the knots t_i. This paper addresses the problem of computing knots for a given set of data points.

* Supported by the National Key Basic Research 973 Program of China (No.2006CB303102) and the National Nature Science Foundation of China (No.60573181,60933008), Shandong Province National Nature Science Foundation(No.Z2006G05).

B. Mourrain, S. Schaefer, and G. Xu (Eds.): GMP 2010, LNCS 6130, pp. 283–295, 2010.

There are several existing methods for solving this problem. It is well known that using a uniform parametrization (that is, the knots t_i are equally spaced) to choose knots generally leads to unsatisfactory results when the distances of the data points vary greatly. The chord length parametrization is a widely accepted method for determining knots [1] [2] [3] [4] [5] [6]. This method produces satisfactory results because the accumulated chord length is a reasonable approximation to the accumulated arc length. The quality of chordal parametrisation is discussed recently in the paper [7]. Two other commonly used methods are Foley's method [9] and the centripetal method [8], which are the variations of the chord length method. Our experiments in Section 5 show that, in terms of interpolation error, none of these methods has a distinct advantage over the others. Moreover, in some cases none of these methods can produce a satisfactory result.

The problem of determining knots for constructing B-spline/NURBS curve is discussed [12] [13], where the knots are determined using an energy-optimization method. Other recent methods of determining knots can be found in [14] [15] [16].

One property shared by many of the above methods for knot selection is *linear precision*, which means that, roughly speaking, using the knots provided by such a method, the resulting interpolation curve will be a linearly parameterized straight line if the data points are sampled from a straight line. By approximation theory, in general, a smooth function with bounded derivatives is better approximated with a polynomial of higher degree. This means that a higher order precision would in general lead to an interpolation curve with a smaller interpolation error. Our contribution is a new, local knot selection method with quadratic precision.

Since the notion of quadratic precision is central to our method, it deserves some elaboration. Suppose that we have a curve interpolation scheme that takes in a set of data points $P_i = (x_i, y_i)$ and knots t_i to produce polynomial functions $x(t)$ and $y(t)$ that form a parametric curve $P(t) = (x(t), y(t))$ to interpolate the points $P_i = (x_i, y_i)$; that is, $x(t_i) = x_i$ and $y(t_i) = y_i$, $i = 1, 2, \ldots, n$. Now consider two arbitrary quadratic functions $g(t)$ and $h(t)$. Suppose that the coordinates $\{x_i\}$ and $\{y_i\}$ of the data points P_i are sampled from $g(t)$ and $h(t)$ with the same variable values t_i in the increasing order; that is, $g(t_i) = x_i$ and $h(t_i) = y_i$, $i = 1, 2, \ldots, n$. Then the interpolation scheme is said to have *quadratic functional precision* if, with the same t_i as knots, it produces an interpolation curve $P(t) = (x(t), y(t))$ of the points P_i such that $x(t) = g(t)$ and $y(t) = h(t)$. That is to say, the original curve $G(t) = (g(t), h(t))$ on which the data points lie is reproduced by the interpolation scheme.

However, in a curve interpolation problem, only the data points are specified and the knots t_i are not provided as part of the input; they need to be estimated by some knot selection method before applying a curve interpolation scheme. Hence, even when an interpolation scheme possessing the quadratic functional precision is used and the data points are sampled from a quadratic polynomial curve $G(t)$, without an appropriate set of knots t_i, the resulting interpolation curve $P(t)$ may still not reproduce the curve $G(t)$.

Now we give the definition of quadratic precision of a knot selection scheme.

Definition. *Suppose that an interpolation scheme possessing quadratic functional precision is used to compute an interpolation curve for a set of given data points* $\{P_i\}$, $i = 1, 2, \ldots, n$. *A method for computing knots* t_i *from* $\{P_i\}$ *is said to have* quadratic precision *if, for any set of data points* $\{P_i\}$ *sampled from any quadratic curve* $G(t)$, *it produces the knots* t_i *such that with these knots the interpolation scheme reproduces* $G(t)$ *as the interpolation curve.*

The first author of the present paper and his co-workers propose a method in [10], to be referred to as the ZCM method, that solves a global equation to find knots with quadratic precision. Here we preset a new, local method that computes knots with quadratic precision without having to solve a global equation. In contrast, the knots computed by the chord length, Foley's method and the centripetal methods have linear precision but not quadratic precision.

The remainder of the paper is organized as follows. The idea of the new method is described in Section 2. In Section 3, we discuss how to compute the local knot sequences for each data point from its neighboring points. In Section 4, we use a normalization scheme to merge the local knot sequences into a global, consistent knot sequence with quadratic precision.The comparison of the new method with the chord length method, Foley's method and the centripetal methods is presented in Section 5, and we conclude the paper in Section 6.

2 Basic Idea

The main idea of the method is as follows. We locally estimate the intervals between consecutive knots based on quadratic curves interpolating each set of four consecutive data points, assuming that such four points form a locally convex configuration (i.e. no inflection). Then these local knot intervals are registered together via a normalization scheme to determine a global knot sequence. When the data points are sampled from a quadratic curve, the quadratic curves interpolating each set of four consecutive data points become the same curve, since a quadratic curve (i.e. a parabola) is uniquely determined by four points on it. We shall show that the global knot sequence thus chosen possesses quadratic precision.

Let $P_i = (x_i, y_i), 1 \leq i \leq n$, be a set of distinct data points. Four consecutive data point P_{i+k}, $k = 0, 1, 2, 3$, form a *convex chain* if $P_iP_{i+1}P_{i+2}P_{i+3}P_i$ is a convex polygon. For the moment, we assume that every point $P_i = (x_i, y_i), 1 \leq i \leq n$, belongs to at least two convex chains. For example, in Figure 3, the point P_i belongs to the convex chains $\{P_{i-2}, P_{i-1}, P_i, P_{i+1}\}$ and $\{P_i, P_{i+1}, P_{i+2}, P_{i+3}\}$.

Let t_i denote the knots to be assigned for the points P_i, $1 \leq i \leq n$. Our goal is to determine the t_i in such a way that, if the P_i are taken from a parametric quadratic polynomial, i.e.,

$$P_i = A\xi_i^2 + B\xi_i + C, \qquad 1 \leq i \leq n, \tag{1}$$

then $t_i = \alpha\xi_i + \beta$, $1 \leq i \leq n$, for some constants α and β. This will ensure the quadratic precision, since a linear transform of the knots does not affect the type

of interpolation curves produced by an interpolation scheme that has quadratic functional precision.

Suppose that the data points P_i, $1 \leq i \leq n$, are taken from a parametric quadratic polynomial $P(\xi) = (x(\xi), y(\xi))$ defined by

$$\begin{aligned} x(\xi) &= X_2\xi^2 + X_1\xi + X_0, \\ y(\xi) &= Y_2\xi^2 + Y_1\xi + Y_0. \end{aligned} \tag{2}$$

Then any four consecutive data points $\{P_{i-2}, P_{i-1}, P_i, P_{i+1}\}$, $i = 3, 4, \cdots, n-1$(see Figure 1) will uniquely determine a quadratic polynomial curve $P_i(t)$ which is the same as $P(\xi)$ in Eqn. (2), but possibly with a different parameterization. Since any two proper parameterizations of a quadratic curve differ by a linear reparameterization, it follows that $t = \alpha\xi + \beta$, for some constants α and β.

Let $t_j^{(0)} = \alpha_i\xi_j + \beta_i$ denote the knots computed with respect to $P_i(t)$ for the four consecutive data points $\{P_{i-2}, P_{i-1}, P_i, P_{i+1}\}$. Suppose that the next four data points determine the quadratic curve $P_{i+1}(t)$. Let $t_j^{(1)} = \alpha_{i+1}\xi_j + \beta_{i+1}$ denote the knots computed with respect to $P_{i+1}(t)$ for the four consecutive data points $\{P_{i-1}, P_i, P_{i+1}, P_{i+2}\}$. Thus, we will have two sets of knot values $t_j^{(0)}$ and $t_j^{(1)}$ for the three data points P_j, $j = i-1, i, i+1$, derived from the two possibly different parameterizations $P_i(t)$ and $P_{i+1}(t)$ of the same quadratic curve $P(\xi)$.

Since the two sequences of knots $t_j^{(0)}$ and $t_j^{(1)}$, $j = i-1, i, i+1$, are both linearly related to ξ_i, it is possible to use a linear mapping to match up the two sequences. This is, in fact, the key idea that enables us to compute knots with quadratic precision using only local computation. At the overall level of the algorithm, suppose that we want to compute a global sequence of knots for the data points P_i, $i = 3, 4, \cdots, n-1$, that are taken from the same quadratic curve. We first consider all groups of four consecutive data points and compute locally the knots of the four points in each group with respect to the quadratic curve locally determined by these four points; thus each group of points will have its knot sequence of length 4. Since any two adjacent groups share three common data points and the two quadratic curves determined respectively by the two groups of points are the same curve, we can merge their knot sequences using a linear reparameterization to form a longer knot sequence.

To develop a complete solution based on this idea, we face two tasks: 1) computing the local knot sequence t_j from each group of four consecutive data points; 2) merging all these local knot sequences into a global knot sequence that has quadratic precision. These two steps will be explained in the following sections.

3 Computing Knots from Neighboring Data Points

In this section, we will consider how to locally compute the knots of three consecutive points $\{P_{i-1}, P_i, P_{i+1}\}$ from their neighboring points. Consider a data point P_i and its neighboring points P_{i-2}, P_{i-1}, P_{i+1}, and P_{i+2}. Let $P_i(t)$ and $P_{i+1}(t)$ be the two quadratic curves determined respectively by the first

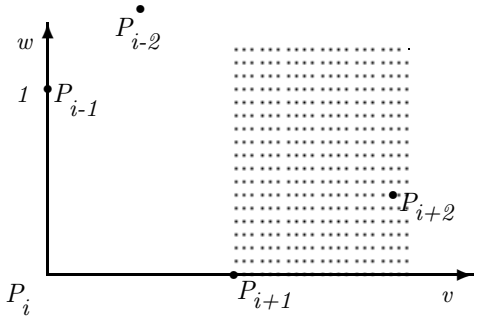


Fig. 1. Five data points

group of four points $\{P_{i-2}, P_{i-1}, P_i, P_{i+1}\}$ and the next group of four points $\{P_{i-1}, P_i, P_{i+1}, P_{i+2}\}$, assuming each group forming a convex chain. Note that the three points $\{P_{i-1}, P_i, P_{i+1}\}$ are shared by both groups. We will discuss how to compute the knots $t_j^{(0)}$ of the three points $\{P_{i-1}, P_i, P_{i+1}\}$ with respect to $P_i(t)$ and the knots $t_j^{(1)}$ of the three points with respect to $P_{i+1}(t)$. In the general case where the five points P_{i-2}, P_{i-1}, P_i, P_{i+1} are *not* taken from the same quadratic curve, since the knots $t_j^{(0)}$ and $t_j^{(1)}$, $j = i-1, i, i+1$, cannot be matched up by a linear mapping, we will need to generate the local knots for $\{P_{i-1}, P_i, P_{i+1}\}$ via an appropriate averaging of the $t_j^{(0)}$ and $t_j^{(1)}$, $j = i-1, i, i+1$.

Let $P_j = (x_j, y_j)$, $i-2 \le j \le i+2$, be five consecutive points, where $i = 3, 4, \cdots$, or $n-2$. For the moment, assume that these five points form a convex chain and no three consecutive points of them are collinear. Then, for brevity, via an affine mapping, the five points can be mapped to have the coordinates (x_{i-2}, y_{i-2}), $(0,1)$, $(0,0)$, $(1,0)$ and (x_{i+2}, y_{i+2}), respectively, as shown in Figure 1. Such an affine mapping does not affect our argument since the correspondence between a group of four points and the unique quadratic curve determined by the four points is invariant under affine mappings. Let $Q(s)$ be the quadratic curve that interpolates P_{i-1}, P_i, and P_{i+1}. Taking advantage of linear reparameterization, we may assume $Q(0) = P_{i-1}$, $Q(s_i) = P_i$, and $Q(1) = P_{i+1}$, for some constant s_i in $(0,1)$. It can be shown that $Q(s) = (x(s), y(s))$ has the expression

$$\begin{aligned} x = x(s) &= \frac{s(s-s_i)}{1-s_i}, \\ y = y(s) &= \frac{(s-s_i)(s-1)}{s_i}. \end{aligned} \tag{3}$$

The freedom of choosing $s_i \in (0,1)$ indicates that one more point is needed to uniquely determine $Q(s)$. So next we will fix s_i by requiring $Q(s)$ to further interpolate P_{i+2}. By the first equation of (3), replacing $s(s-s_i)$ of the second equation of (3) by $(1-s_i)x$, we get

$$\begin{aligned} &(1-s_i)x = s(s-s_i), \\ &s = x + (1-x-y)s_i. \end{aligned} \tag{4}$$

Substituting the second equation of (4) into the first one and arranging, the implicit equation of the quadratic curve $Q(s)$ is found to be

$$w(x,y) = a(x,y)s_i^2 + b(x,y)s_i + c(x,y) = 0, \tag{5}$$

where

$$\begin{aligned} a(x,y) &= (1-x-y)(x+y), \\ b(x,y) &= -2x(1-x-y), \\ c(x,y) &= (1-x)x. \end{aligned}$$

Since $P_{i+2} = (x_{i+2}, y_{i+2})$ is a point on this curve, substituting (x_{i+2}, y_{i+2}) for x, y in Eqn. (5) yields the two roots of s_i:

$$s_i^r = \frac{1}{x_{i+2}+y_{i+2}} \left(x_{i+2} \pm \sqrt{\frac{x_{i+2}y_{i+2}}{x_{i+2}+y_{i+2}-1}} \right). \tag{6}$$

Substituting (x_{i+2}, y_{i+2}) and (6) into (4) yields

$$s_{i+2} = \mp \sqrt{\frac{x_{i+2}y_{i+2}}{x_{i+2}+y_{i+2}-1}} + s_i^r.$$

Since $s_{i+2} > 1$ and $s_i^r < 1$, it follows

$$s_i^r = \frac{1}{x_{i+2}+y_{i+2}} \left(x_{i+2} - \sqrt{\frac{x_{i+2}y_{i+2}}{x_{i+2}+y_{i+2}-1}} \right). \tag{7}$$

Substituting $s_{i+2} > 1$ into (3) gives $x_{i+2} > 1$ and $y_{i+2} > 0$, so the point $P_{i+2} = (x_{i+2}, y_{i+2})$ should be in the dotted region in Figure 1 in order for the solution s_i^r to be real. This means that the four data points need to form a convex chain in order to uniquely determine a quadratic polynomial.

On the other hand, if we require $Q(s)$ to interpolate P_{i-2}, instead of P_{i+2}, by a similar argument we get another knot s_i^l for P_i, given by

$$s_i^l = \frac{1}{x_{i-2}+y_{i-2}} \left(x_{i-2} + \sqrt{\frac{x_{i-2}y_{i-2}}{x_{i-2}+y_{i-2}-1}} \right). \tag{8}$$

When the five points $P_j = (x_j, y_j)$, $i-2 \le j \le i+2$, are taken from the same quadratic curve, we have $s_i^l = s_i^r$. However, for data points given in general positions (but still assumed to form a convex chain), these five points may not lie on the same underlying quadratic curve, so we may have $s_i^l \neq s_i^r$. In this case we would need to reconcile the two values to determine a knot s_i for P_i. An obvious choice would be to set $s_i = (s_i^l + s_i^r)/2$. But, in the following we will propose a more elaborate scheme to get s_i from s_i^l and s_i^r, to further improve the estimate of s_i.

Substituting (x_{i-2}, y_{i-2}) and (x_{i+2}, y_{i+2}) into (5), respectively, one gets the following equations

$$\begin{aligned} a(x_{i-2}, y_{i-2})s_i^2 + b(x_{i-2}, y_{i-2})s_i + c(x_{i-2}, y_{i-2}) = 0, \\ a(x_{i+2}, y_{i+2})s_i^2 + b(x_{i+2}, y_{i+2})s_i + c(x_{i+2}, y_{i+2}) = 0. \end{aligned} \tag{9}$$

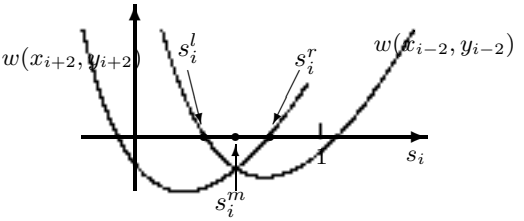


Fig. 2. Positions of s_i^l, s_i^m and s_i^r

From these equations another estimate of s_i is given by

$$s_i^m = \frac{c(x_{i+2}, y_{i+2})a(x_{i-2}, y_{i-2}) - c(x_{i-2}, y_{i-2})a(x_{i+2}, y_{i+2})}{b(x_{i-2}, y_{i-2})a(x_{i+2}, y_{i+2}) - b(x_{i+2}, y_{i+2})a(x_{i-2}, y_{i-2})}. \tag{10}$$

Thus, we have now three estimates s_i^l, s_i^m, s_i^r for s_i, as indicated in Figure 2.

Now we are going to compute s_i as a combination of s_i^l, s_i^m and s_i^r, defined by (8), (10) and (7), respectively. Substituting s_i defined by (7), (8) and (10) into $w(x, y)$ (5), the corresponding $w(x, y)$'s are denoted by $w_r(x, y)$, $w_l(x, y)$ and $w_m(x, y)$, respectively. The discussion above shows that, in general, $w_r(x, y)$ passes (x_{i+2}, y_{i+2}) and approximates (x_{i-2}, y_{i-2}), $w_l(x, y)$ passes (x_{i-2}, y_{i-2}) and approximates (x_{i+2}, y_{i+2}), while $w_m(x, y)$ approximates both (x_{i-2}, y_{i-2}) and (x_{i+2}, y_{i+2}). Let

$$\alpha_i^\tau = \sqrt{w_\tau(x_{i-2}, y_{i-2})^2 + w_\tau(x_{i+2}, y_{i+2})^2},$$
$$\tau = l, m, r.$$

If $\alpha_i^l = \alpha_i^m = \alpha_i^r = 0$, then $s_i^l = s_i^m = s_i^r$, and we simply set $s_i = s_i^l$. Otherwise, as among s_i^l, s_i^m and s_i^r, none is better for defining s_i than the rest two ones, s_i is defined by the weighted combination of s_i^l, s_i^m and s_i^r as follows

$$s_i = \frac{\alpha_i^m \alpha_i^r s_i^l + \alpha_i^l \alpha_i^r s_i^m + \alpha_i^l \alpha_i^m s_i^r}{\alpha_i^m \alpha_i^r + \alpha_i^l \alpha_i^r + \alpha_i^l \alpha_i^m}. \tag{11}$$

Note that $w_l(x_{i-2}, y_{i-2}) = 0$ and $w_r(x_{i+2}, y_{i+2}) = 0$, so, $\alpha_i^l = |w_l(x_{i+2}, y_{i+2})|$ and $\alpha_i^r = |w_r(x_{i-2}, y_{i-2})|$.

So far we have excluded the cases where the point $P_j, j = i-2, i-1, i, i+1, i+2$, do not form a convex chain or some three consecutive points of them are collinear. Now we need to address these cases.

If P_{i-1}, P_i and P_{i+1} are on a straight line, then we set

$$s_i = \frac{|P_{i-1}P_i|}{|P_{i-1}P_i| + |P_iP_{i+1}|}. \tag{12}$$

This choice makes the quadratic polynomial which passes P_{i-1}, P_i and P_{i+1} be a straight line with the magnitude of the first derivative being a constant. Such a straight line is the most naturally defined curve one can get in this case.

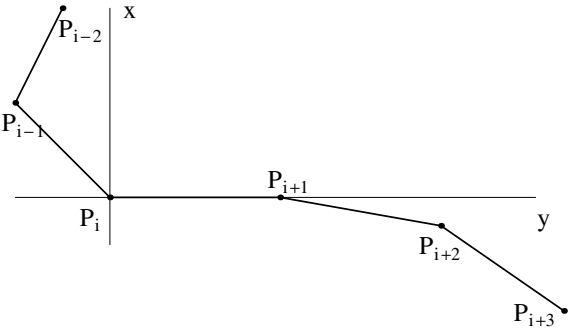


Fig. 3. Data points whose convexity changes sign

When the data points P_j, $j = i-2, i-1, i, i+1, i+2$, do not form a convex chain, as shown in Figure 3, the knot s_i for P_i is computed by (8) using the points $\{P_{i-2}, P_{i-1}, P_i, P_{i+1}\}$, and the knot s_{i+1} for P_{i+1} is computed by (7) using the points $\{P_i, P_{i+1}, P_{i+2}, P_{i+3}\}$.

Finally, for the end data points, s_2 corresponding to $Q_2(s)$ is determined using the four points $P_j, j = 1, 2, 3, 4$, and s_{n-1} corresponding to $Q_{n-1}(s)$ is determined using points the $P_j, j = n-3, n-2, n-1, n$.

4 Merging Local Knots Sequences

So far we have computed the local knots $0, s_i, 1$ for the three points P_{i-1}, P_i, P_{i+1} with respect to two locally interpolating quadratic curves $P_i(t)$ and $P_{i+1}(t)$. These knots define the knot interval $[0, s_i]$ between P_{i-1} and P_i, and the knot interval $[s_i, 1]$ between P_i and P_{i+1}; we will associate the lengths of these two intervals, i.e. s_i and $1 - s_i$, with the point P_i, and still call them knot intervals. Even when all the data points P_i are taken from the same quadratic curve, the knot intervals associated with different points may not be equal due to the different linear scales of different parameterizations. In this section, we introduce a normal form of a quadratic curve and use it to merge all the knot intervals associated with different points P_i, $i = 2, 3, \ldots, n-1$, into a consistent global knot sequence with respect to the same parameterization of a quadratic curve.

Any quadratic curve $P_i(t) = (x_i(t), y_i(t))$, where

$$\begin{aligned} x_i(t) &= X_{i,2}t^2 + X_{i,1}t + X_{i,0}, \\ y_i(t) &= Y_{i,2}t^2 + Y_{i,1}t + Y_{i,0} \end{aligned} \tag{13}$$

can be transformed by a rigid transformation and a linear reparameterization into the form

$$\begin{aligned} \bar{x}_i(s) &= s^2 + \bar{X}_1 s + \bar{X}_0, \\ \bar{y}_i(s) &= s^2 + \bar{Y}_1 s + \bar{Y}_0, \end{aligned} \tag{14}$$

with $X_{i,2} \neq 0$ or $Y_{i,2} \neq 0$. The transformations and reparameterization required are

$$\begin{aligned} \bar{x} &= x\cos\beta_i + y\sin\beta_i \\ \bar{y} &= -x\sin\beta_i + y\cos\beta_i \end{aligned} \tag{15}$$

where

$$\cos\beta_i = \frac{X_{i,2}+Y_{i,2}}{\sqrt{X_{i,2}^2+Y_{i,2}^2}}, \qquad \sin\beta_i = \frac{Y_{i,2}-X_{i,2}}{\sqrt{X_{i,2}^2+Y_{i,2}^2}},$$

and

$$\begin{aligned} \bar{X}_0 &= \cos\beta_i X_{i,0} + \sin\beta_i Y_{i,0}, & \bar{Y}_0 &= -\sin\beta_i X_{i,0} + \cos\beta_i Y_{i,0} \\ \bar{X}_1 &= \frac{\cos\beta_i X_{i,1} + \sin\beta_i Y_{i,1}}{\sqrt{\cos\beta_i X_{i,2} + \sin\beta_i Y_{i,2}}}, & \bar{Y}_1 &= \frac{-\sin\beta_i X_{i,1} + \cos\beta_i Y_{i,1}}{\sqrt{\cos\beta_i X_{i,2} + \sin\beta_i Y_{i,2}}}, \end{aligned} \tag{16}$$

and

$$s = (X_{i,2}^2 + Y_{i,2}^2)^{\frac{1}{4}}\ t. \tag{17}$$

For P_{i-1}, P_i and P_{i+1}, $i = 2, 3, \cdots, n-1$, the parametric quadratic polynomial $Q_i(s) = (x_i(s), y_i(s))$ which interpolates P_{i-1}, P_i and P_{i+1} at 0, s_i and 1, respectively, is

$$\begin{aligned} x_i(s) &= X_{i,2}s^2 + X_{i,1}s + x_{i-1}, \\ y_i(s) &= Y_{i,2}s^2 + Y_{i,1}s + y_{i-1}, \end{aligned} \tag{18}$$

where

$$\begin{aligned} X_{i,2} &= \frac{x_{i-1}-x_i}{s_i} + \frac{x_{i+1}-x_i}{1-s_i}, \\ X_{i,1} &= -\frac{(x_{i-1}-x_i)(s_i+1)}{s_i} - \frac{(x_{i+1}-x_i)s_i}{1-s_i}, \\ Y_{i,2} &= \frac{y_{i-1}-y_i}{s_i} + \frac{y_{i+1}-y_i}{1-s_i}, \\ Y_{i,1} &= -\frac{(y_{i-1}-y_i)(s_i+1)}{s_i} - \frac{(y_{i+1}-y_i)s_i}{1-s_i}. \end{aligned} \tag{19}$$

When we convert the quadratic curve $Q_i(s)$ in Eqn. (18) to the normal form in Eqn. (14), by the reparameterization (17), the knot intervals s_i and $1-s_i$ associated with P_i become

$$\begin{aligned} \Delta_{i-1}^i &= (X_{i,2}^2 + Y_{i,2}^2)^{\frac{1}{4}}\ s_i, \\ \Delta_i^i &= (X_{i,2}^2 + Y_{i,2}^2)^{\frac{1}{4}}\ (1-s_i), \end{aligned} \tag{20}$$

where $X_{i,2}$ and $Y_{i,2}$ are defined in (19).

If P_{i-1}, P_i and P_{i+1} are on a straight line, then with (16) and (12), it is easy to prove that

$$\begin{aligned} \Delta_{i-1}^i &= |P_{i-1}P_i|, \\ \Delta_i^i &= |P_iP_{i+1}|. \end{aligned} \tag{21}$$

Hence, by mapping each $Q_i(s)$ into the normal form, for each pair of consecutive points P_i and P_{i+1} there are two knot intervals Δ_i^i and Δ_i^{i+1}, $2 \leq i \leq n-1$. We have $\Delta_i^i = \Delta_i^{i+1}$ if all the data points are taken from the same quadratic curve. But, in general, $\Delta_i^i \neq \Delta_i^{i+1}$. Furthermore, for end data points, there is only one

knot interval, Δ_1^1, for the pair P_1 and P_2; and there is one knot interval, Δ_{n-1}^{n-2}, for the pair P_{n-1} and P_n.

We first average the two sequences of knot intervals, $\{\Delta_i^i\}$ and $\{\Delta_i^{i+1}\}$, into a single sequence of knot intervals, $\{\Delta_i\}$, $i = 1, 2, \ldots, n-1$, as follows.

$$\begin{aligned} &\Delta_1 = \Delta_1^1, \\ &\Delta_i = \frac{2\Delta_i^i \Delta_i^{i+1}}{\Delta_i^i + \Delta_i^{i+1}}, \quad i = 2, 3, \cdots, n-2, \\ &\Delta_{n-1} = \Delta_{n-1}^{n-2}. \end{aligned}$$

From the knot intervals $\{\Delta_i\}$, we compute the global knot sequence $\{t_i\}$, $i = 1, 2, \ldots, n$, as follows.

$$\begin{aligned} &t_1 = 0, \\ &t_{i+1} = t_i + \Delta_i, \qquad i = 1, 2, \cdots, n-1. \end{aligned} \tag{22}$$

5 Experiments

We will present two test examples to compare our new local knot selection method with the chord length method, Foley's method, and the centripetal method. The comparison is performed by using the knots computed with these methods in the construction of a parametric cubic spline interpolant. In the first example, the data points used in the comparison are taken from the ellipse, $F(s) = (x(s), y(s)$ defined by

$$\begin{aligned} x(s) &= 3\cos(2\pi s), \\ y(s) &= 2\sin(2\pi s). \end{aligned}$$

The four methods are compared on non-uniform data points produced by dividing the interval $s \in [0, 1]$ into n subintervals, $n = 20, 40, 80$, i.e., s_i is defined as follows:

$$s_i = [i + \lambda sin(i * (n - i))]/n, \qquad i = 0, 1, 2, \cdots, n, \tag{23}$$

where $\lambda = 0.1$, which makes the distance between two adjacent points being unequal.

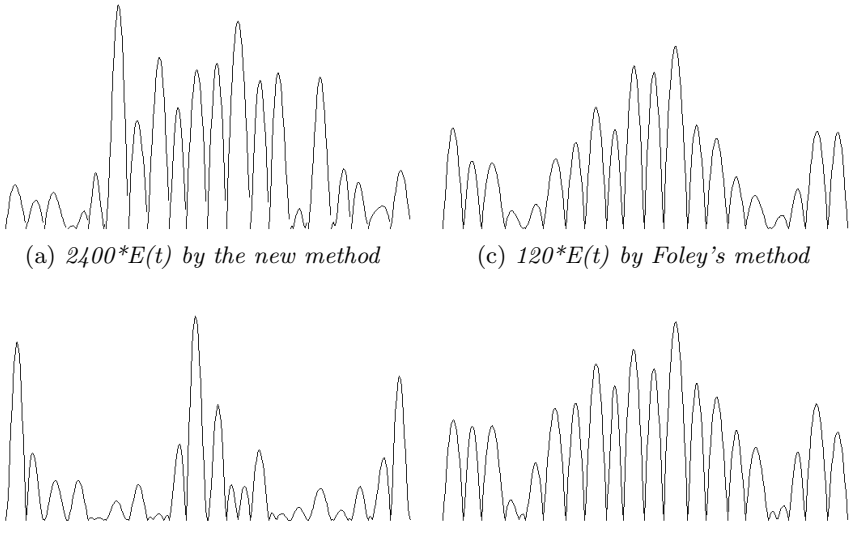
The four methods are evaluated in terms of the absolute error $E(t)$ defined by

$$\begin{aligned} E(t) &= \min_s \{|P(t) - F(s)|\} \\ &= \min_s \{|\tilde{P}_i(t) - F(s)|, s_{i-1} \le s \le s_i\}, \qquad i = 0, 1, 2, \cdots, n-1, \end{aligned}$$

where $P(t)$ denotes one of the splines constructed by the four methods, $\tilde{P}_i(t)$ is the corresponding part of $P(t)$ on the subinterval $[t_{i-1}, t_i]$, and $F(s)$ is the above ellipse. For point $\tilde{P}_i(t)$, $min\{|\tilde{P}_i(t) - F(s)|, s_{i-1} \le s \le s_i\}$ means the shortest distance from the point $\tilde{P}_i(t)$ to the curve segment $F(s)$.

Table 1. Maximum absolute errors

Error	New	chord	Foley	centripetal
n=20	4.6e-4	8.41e-4	7.48e-3	1.02e-2
n=40	2.02e-5	7.11e-5	5.54e-4	1.17e-3
n=80	3.01e-6	4.25e-6	1.06e-4	4.60e-4



(a) *2400*E(t) by the new method* (c) *120*E(t) by Foley's method*

(b) *1200*E(t) by the chord length method* (d) *96*E(t) by the centripetal method*

Fig. 4. Error curves by the four methods

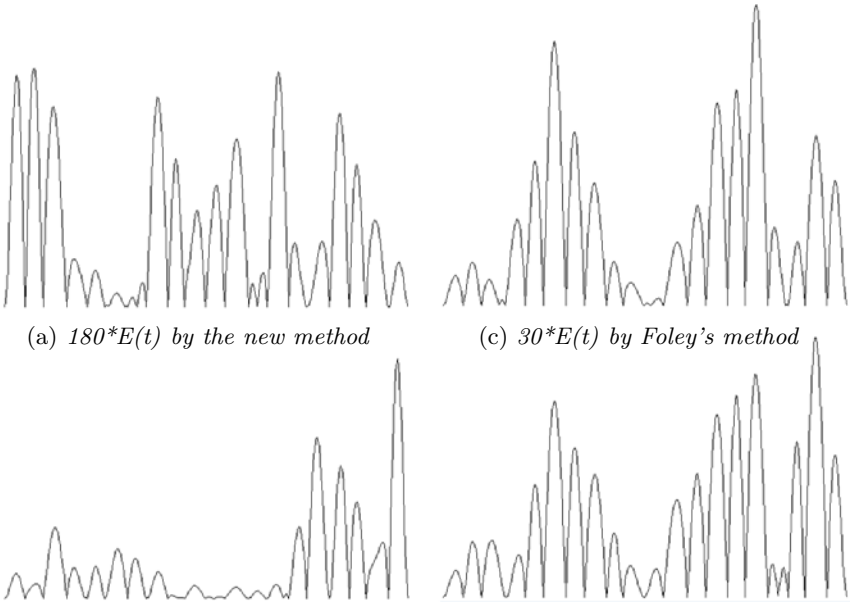
The four methods have been compared on data points defined by (23). The maximum values of the error curve $E(t)$ generated by the four methods are shown in Table 1. For $n = 20$, the error curves by the four methods are shown in Figure 4.

In the second example, the four methods are compared using data points taken from a Bézier curve of degree eight, $F(s) = (x(s), y(s))$. The control points of the curve are $\{(0,0), (19,24), (39,43), (58,37), (78,0), (98,-50), (141,-81), (164,-64), (188,0)\}$. To prevent large errors from occurring near the end points (x_0, y_0) and (x_n, y_n), $n = 20, 40, 80$, the tangent vectors of $F(s)$ at $s = 0$ and $s = 1$ are used as the end conditions to construct the cubic splines. The maximum values of the error curve $E(t)$ generated by the four methods are shown in Table 2. For $n = 20$, the error curves by the four methods are shown in Figure 5.

The two test examples show that our method leads to smaller interpolation error than all the other three methods. It is about half of the error of the chord length method, and about one order of magnitude smaller than Foley's method and the centripetal method.

Table 2. Maximum absolute errors

Error	New	chord	Foley	centripetal
n=20	4.92e-3	2.47e-2	4.33e-2	9.60e-2
n=40	1.60e-3	2.16e-3	7.64e-3	2.26e-2
n=80	3.348e-4	3.94e-4	1.75e-3	7.71e-3



(a) *180*E(t) by the new method* (c) *30*E(t) by Foley's method*

(b) *45*E(t) by the chord length method* (d) *12*E(t) by the centripetal method*

Fig. 5. Error curves by the four methods

6 Conclusions and Future Work

A new method for choosing knots in parametric curve interpolation has been presented. The new method can be used in polynomial curve interpolation as well as in spline curve interpolation. The chosen knots have a quadratic precision, meaning that, from the approximation point of view, the new method is better than the chord length method, the centripetal method, and Foley's method. Our experiment results also indicate that if the polygon formed by the data points does not has any inflection, that is, globally convex, then the new method in general gives better approximation than the other three methods.

While the approximation error produced with the new method is relatively small, it is however more involved than the other existing methods. Therefore a further research problem is to devise a simpler method for computing knots with quadratic precision.

It is known that, when constructing a cubic spline interpolant, with the suitable two end conditions and the knots, the constructed parametric cubic spline reproduces parametric cubic polynomials. Our next work is to investigate whether there is a method of choosing knots with cubic precision.

References

1. Ahlberg, J.H., Nilson, E.N., Walsh, J.L.: The theory of splines and their applications. Academic Press, New York (1967)
2. de Boor, C.: A practical guide to splines. Springer, New York (1978)
3. Brodlie, K.W.: A review of methods for curve and function drawing. In: Brodile, K.W. (ed.) Mathematical methods in computer graphics and design, pp. 1–37. Academic Press, London (1980)
4. Su, B., Liu, D.: Computational Geometry, pp. 47–48. Shanghai Science and Technology Press, Shanghai (1982)
5. Faux, I.D., Pratt, M.J.: Computational Geometry for Design and Manufacture. Ellis Horwood (1979)
6. Späth, H.: Spline algorithms for curves and surfaces Utilitas Mathematica. Winnipeg, Canada (1974)
7. Floater, M.S.: Chordal cubic spline interpolation is fourth order accurate. IMA J. Numer. Anal. 26, 25–33 (2006)
8. Lee, E.T.Y.: Choosing nodes in parametric curve interpolation. CAD 21(6), 363–370 (1989)
9. Farin, G.: Curves and surfaces for computer aided geometric design: A practical guide. Academic Press, London (1988)
10. Zhang, C., Cheng, F., Miura, K.: A method for determing knots in parametric curve interpolation. CAGD 15, 399–416 (1998)
11. Zhang, C., Cheng, F.: Constructing Parametric Quadratic Curves. Journal of Computational and Applied Mathematics 102, 21–36 (1999)
12. Xie, H., Qin, H.: A Novel Optimization Approach to The Effective Computation of NURBS Knots. International Journal of Shape Modeling 7(2), 199–227 (2001)
13. Xie, H., Qin, H.: Automatic Knot Determination of NURBS for Interactive Geometric Design. In: Proceedings of International Conference on Shape Modeling and Applications, SMI 2001, pp. 267–277 (2001)
14. Marin, S.P.: An approach to data parametrization in parametric cubic spline interpolation problems. J. Approx. Theory 41, 64–86 (1984)
15. Zhang, C., Han, H., Cheng, F.: Determining Knots by Minimizing Energy. Journal of Computer Science and Technology 216, 261–264 (2006)
16. Hartley, P.J., Judd, C.J.: Parametrization and shape of B-spline curves for CAD. CAD 12(5), 235–238 (1980)

Eigenmodes of Surface Energies for Shape Analysis

Klaus Hildebrandt, Christian Schulz,
Christoph von Tycowicz, and Konrad Polthier

Freie Universität Berlin

Abstract. In this work, we study the spectra and eigenmodes of the Hessian of various discrete surface energies and discuss applications to shape analysis. In particular, we consider a physical model that describes the vibration modes and frequencies of a surface through the eigenfunctions and eigenvalues of the Hessian of a deformation energy, and we derive a closed form representation for the Hessian (at the rest state of the energy) for a general class of deformation energies. Furthermore, we design a quadratic energy, such that the eigenmodes of the Hessian of this energy are sensitive to the extrinsic curvature of the surface.

Based on these spectra and eigenmodes, we derive two shape signatures. One that measures the similarity of points on a surface, and another that can be used to identify features of the surface. In addition, we discuss a spectral quadrangulation scheme for surfaces.

1 Introduction

The spectrum and the eigenfunctions of the Laplace-Beltrami operator of a surface have stimulated much recent work in shape analysis and geometry processing, ranging from parametrization, segmentation, and symmetry detection to shape signatures and mesh filtering. Such methods profit from the properties of the eigenfunctions of the Laplace-Beltrami operator. For example, on a curved surface they form an orthogonal basis of the space of L^2-functions on the surface. Furthermore, the Laplacian depends only on the metric of the surface, hence the eigenvalues and eigenfunctions are invariant under isometric deformations of the surface. However, there are disadvantages as well. For example, a consequence of the invariance to isometric deformations is an insensitivity to extrinsic feature of the surface, like sharp bends, that are of essential importance for some applications.

Contributions. In this work we derive operators, whose eigenmodes and spectra can serve as alternatives to the spectrum and modes of the Laplacian for applications in geometry processing and shape analysis. On the one hand, the eigenfunctions of these operators share properties with the eigenfunctions of the Laplacian, *e.g.*, they form an orthogonal basis of an adequate space of variations of the surface. On the other hand, there are fundamental differences, *e.g.*, these eigenfunctions depend (not only on intrinsic quantities but also) on the extrinsic

B. Mourrain, S. Schaefer, and G. Xu (Eds.): GMP 2010, LNCS 6130, pp. 296–314, 2010.

curvature of the surface. We consider two different settings: vibration modes and frequencies of surfaces derived from deformation energies, and eigenvalues and eigenmodes of the Hessian of quadratic energies that are defined on a space of functions on a surface.

On a planar domain, the eigenfunctions of the Laplacian serve as a model for the vibration modes of a flat plate (Chladni plates). For curved surfaces more elaborate models are required to describe the vibration modes of a surface. We consider a physical model that describes vibration modes of a surface mesh through the eigenfunctions of the Hessian of a deformation energy. In general, computing the Hessian of a deformation energy is a delicate and laborious task. But, to compute the vibration modes we do not need to compute the Hessian at all points in the space of possible surfaces, but only at the point that represents the reference surface. We derive a simple formula, that can be used to compute the Hessian at the reference surface for a general class of deformation energies. We hope that this framework will stimulate further exploration of the eigenmodes and eigenfrequencies of deformation energies.

The Dirichlet energy of a surface is a quadratic functional on an appropriate space of functions on a surface. The Hessian of this energy is the Laplace- Beltrami operator of the surface. We propose a quadratic functional that can be derived from the Dirichlet energy, but is not intrinsic. The eigenfunctions of this energy are sensitive to the extrinsic curvature of the surface.

We discuss three applications that use the proposed eigenmodes and spectra. We define two (multi-scale) signatures, the *vibration signature*, based on the vibration modes, and the *feature signature*, based on the eigenmodes of the modified Dirichlet energy. To each of the two signatures we associate a (multi-scale) pseudo-metric on the surface. The resulting *vibration distance* can be used as a similarity measure on the surface and the *feature distance* can identify features of a mesh. Furthermore, we test the *spectral surface quadrangulation* method of Dong et al. [6] with specific vibration modes, instead of eigenfunctions of the Laplacian. The resulting quadrangulation, in our opinion, aligns better with the extrinsic curvature of the surface.

Related work. Recently, we have seen a boom of papers that use the eigenvalues and eigenfunctions of the Laplace-Beltrami operator as an ingredient to algorithms in geometry processing and shape analysis. An overview of this development can be found in the recent survey by Zhang et al. [29] and in the course notes of a *Siggraph Asia 2009* course held by Lévy and Zhang [16]. Here, we can only briefly outline the work that has been most relevant for this paper.

The spectrum of the Laplace-Beltrami operator of a Riemannian manifold contains a significant amount of information about the manifold and the metric. Though it does not fully determine the Riemannian manifold, it can be used as a powerful shape descriptor of a class of isometric Riemannian manifolds. Reuter et al. [21,22] use the spectrum of the Laplace-Beltrami operator to construct a finger print of surfaces, which they call the *Shape-DNA*. By construction this finger print is invariant under isometric deformations of a surface. Among other applications the Shape-DNA can be used for shape matching, copyright

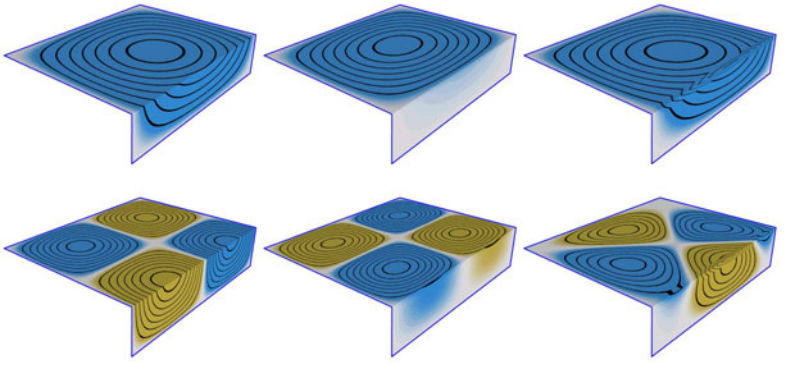


Fig. 1. Visualization of modes of different energies. First column shows the Laplacian eigenmodes, second column the eigenmodes of the modified Dirichlet energy E_D^N, and third column the vibrations modes derived from the thin shell energy restricted to normal variations.

protection, and database retrieval. Rustamov [23] developed the *Global Point Signature (GPS)*, a signature that can be used to classify shapes up to isometry. Based on GPS, Ovsjanikov et al. [17] developed a method for the detection of global symmetries in shapes. Dong et al. [6] present an elegant technique that uses the Morse-Smale complex (and the quasi-dual complex) of a carefully chosen Laplace eigenfunction to generate a coarse quadrangulation of a surface mesh. This approach was extended by Huang et al. [14], who design a least-squares optimization routine that modifies the selected Laplace eigenfunction (and hence its Morse-Smale complex) and provides a user with control of the shape, size, orientation, and feature alignment of the faces of the resulting quadrangulation. The computation of the spectrum and eigenfunctions of the Laplacian is a delicate and computationally expensive task, even for medium sized meshes. Vallet and Lévy [27] propose an efficient shift-and-invert Lanczos method and present an implementation that is designed to handle even large meshes. Using the eigenfunctions of the Laplacian, one can compute the heat kernel of the surface. Sun et al. [26] propose a surface signature based on the heat kernel and use the signature to derive a measure for the geometric similarity of different regions of the surface. Due to its construction, this measure is invariant under isometric deformations of the surface. Independent of this work, Gebal et al. [10] propose a similar signature, named the *Auto Diffusion Function*, and use it for mesh skeletonization and segmentation.

Modal analysis is a well established technique in structural mechanics and mechanical engineering, that aims at computing the modes and frequencies of an object during vibration. In graphics, it is mainly used to speed up physical simulations, see [18,12,2,4]. Recently, Huang et al. [15] use vibration modes of a surface to decompose it into physically meaningful parts. They compute the modes of the surface from the Hessian of the *as-rigid-as-possible* deformation energy, which was proposed by Sorkine and Alexa [25].

In physical simulation, thin shell models describe the dynamics of a thin flexible structure that has a curved undeformed configuration. For example, in cloth simulation thin shells are used to describe folds and wrinkles [3]. Common discrete models [1,3,11,9] describe the middle surface of a thin shell by a mesh and measure the bending of the surface at the edges of the mesh. Of particular interest for this work is the model of Grinspun et al. [11] that uses a discrete energy to simulate thin shells.

2 Deformation Energies

In this section, we consider discrete deformation energies that are defined for surface meshes in $\mathbb{R}^3$. Such energies measure the deformation of a mesh from a reference mesh. A surface mesh is given by the positions of the vertices and the combinatorial information which vertices form triangles. Here, we vary only the positions of the vertices and leave the combinatorial information unchanged. The positions of the vertices can be written in one $3n$-vector x, where n is the number of vertices. Hence, we can identify the space of meshes (with fixed combinatorics) with $\mathbb{R}^{3n}$.

A general deformation energy. We consider deformation energies of the following form:

$$E(x) = \frac{1}{2} \sum\nolimits_i \omega_i(\bar{x}) \left(f_i(x) - f_i(\bar{x})\right)^2, \tag{1}$$

where x is a surface mesh and $\bar{x}$ a fixed reference mesh. In this equation, the sum can run over the edges, the vertices, or the triangles of x, and the f_i's and ω_i's are elementary functions, which *e.g.* measure angles, length of edges, or area of triangles. The weights ω_i must be positive and we require E to be twice continuously differentiable around $\bar{x}$. Then, E has global minimum at $\bar{x}$, which implies that the gradient of E at $\bar{x}$ vanishes and that the Hessian of E at $\bar{x}$ is positive semi-definite.

As an example of such an energy we consider a discrete energy that is designed for thin shell simulation.

Discrete shells. If we regard the surface mesh as a thin shell, then a physical model of thin shells provides us with a deformation energy. Here, we consider the discrete shell model of Grinspun et al. [11]. The energy that governs this model of thin shells is a weighted sum of two components: a flexural energy and a membrane energy. The weight reflects properties of the material to be simulated, *e.g.*, in cloth simulation the membrane energy usually gets a high weight due to the stretching resistance of cloth.

The discrete flexural energy is given as a summation over the edges of the mesh:

$$E_F = \frac{3}{2} \sum\nolimits_i \frac{\|\bar{e}_i\|^2}{\bar{A}_{e_i}} \left(\theta_{e_i} - \bar{\theta}_{e_i}\right)^2, \tag{2}$$

where θ_{e_i} is the dihedral angle at the edge e_i, A_{e_i} is the combined area of the two triangles incident to e_i and $\|e_i\|$ is the length of the edge. The quantities

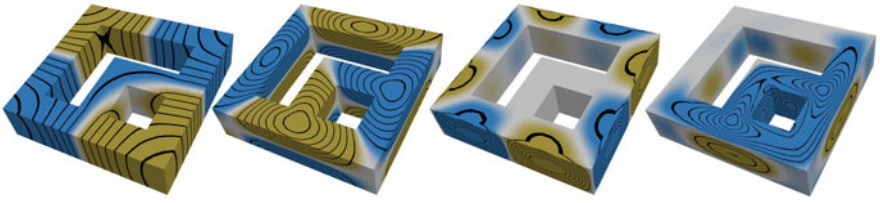


Fig. 2. Two eigenmodes of the lower spectrum on the double torus with sharp features, left: Laplacian, and right: modified Dirichlet energy

$\|\bar{e}_i\|$, $\bar{A}_{e_i}$, and $\bar{\theta}_{e_i}$ are measured on the reference mesh. To write this flexural energy in the general form (1) we set

$$f_i = \theta_{e_i} \quad \text{and} \quad \omega_i = \frac{3\,\|e_i\|^2}{A_{e_i}}.$$

The membrane energy consists of two terms: one measuring the stretching of the edges,

$$E_L = \frac{1}{2}\sum\nolimits_i \frac{1}{\|\bar{e}_i\|}(\|e_i\| - \|\bar{e}_i\|)^2, \tag{3}$$

and one measuring the change of the triangle areas A_i

$$E_A = \frac{1}{2}\sum\nolimits_i \frac{1}{\bar{A}_i}(A_i - \bar{A}_i)^2. \tag{4}$$

Here the second sum runs over the triangles of the mesh. We can describe E_L in the general form (1) by setting

$$f_i = \|e_i\| \quad \text{and} \quad \omega_i = \frac{1}{\|e_i\|},$$

and to describe E_A we set

$$f_i = A_i \quad \text{and} \quad \omega_i = \frac{1}{A_i}.$$

3 Modes of Deformation Energies

Modal analysis provides ways to compute the modes of a surface with respect to a deformation energy. To inspect the modes of a mesh, given by a $3n$-vector $\bar{x}$, we consider a deformation energy $E(x)$ that has $\bar{x}$ as a reference surface. Then, we are interested in the eigenvalues and eigenmodes of the Hessian of the deformation energy E at the mesh $\bar{x} \in X$.

The Hessian of a deformation energy (or more general of a function) does not depend solely on the differentiable structure of X, but also on the metric

on X, hence belongs to Riemannian geometry. Therefore, before considering the Hessian of E we equip X with a metric. Since X equals $\mathbb{R}^{3n}$, the tangent space T_xX at a mesh x can be identified with $\mathbb{R}^{3n}$. We can interpret an element of T_xX as a vector field on x, that assigns a vector in $\mathbb{R}^3$ to every vertex of x. Then, a natural choice of a scalar product on T_xX is a discrete L^2-product, *e.g.*, the mass matrix used in FEM [28] or the discrete L^2-product used in DEC [5,27]. We denote the matrix, that describes the scalar product on T_xX by M_x. For completeness, we would like to mention that if x is a mesh that has degenerate triangles, the discrete L^2-product on T_xX may be only positive semi-definite. However, away from the closed set of meshes that have at least one degenerate triangle, X equipped with the discrete L^2-product is a Riemannian manifold.

We denote by ∂E_x the $3n$-vector containing the first partial derivatives of E at x and by $\partial^2 E_x$ the matrix containing the second partial derivatives at x. We would like to emphasize that ∂E_x and $\partial^2 E_x$ do not depend on the metric on X, whereas the gradient and the Hessian of E do. The gradient of E at x is given by

$$\mathrm{grad}_x E = M_x^{-1} \partial E_x. \tag{5}$$

The Hessian of E at a mesh x is the self-adjoint operator that maps any tangent vector $v \in T_xX$ to the tangent vector $\mathrm{hess}_x E(v) \in T_xX$ given by

$$\mathrm{hess}_x E(v) = \nabla_v \mathrm{grad}_x E, \tag{6}$$

where ∇ is the covariant derivative of X.

Hessian computation. In general, it is a delicate task to derive an explicit representation of the Hessian of a deformation energy and often only approximations of the Hessian are available. Here, we derive a simple explicit formula for the Hessian of a deformation in the general form (1) at the point $\bar{x}$, which involves only first derivatives of the f_i's.

Since the gradient of E vanishes at $\bar{x}$, one can show that at $\bar{x}$ the Hessian of E takes the following form

$$\mathrm{hess}_{\bar{x}} E = M_{\bar{x}}^{-1} \partial^2 E_{\bar{x}}.$$

Hence, at $\bar{x}$ we do not need derivatives of the metric to compute $\mathrm{hess}_{\bar{x}} E$. Furthermore, to compute the second partial derivatives of E at $\bar{x}$ we do not need to calculate second derivatives, but we only need the first derivatives of the f_i's. We present an explicit formula for $\partial^2 E_{\bar{x}}$ in the following Lemma.

Lemma 1 (Explicit Hessian). *Let E be a deformation energy of the form (1). Then, the matrix $\partial^2 E_{\bar{x}}$ containing the second derivatives of E at $\bar{x}$ has the form*

$$\partial^2 E_{\bar{x}} = \sum\nolimits_i \omega_i(\bar{x})\, \partial f_{i\,\bar{x}}\, \partial f_{i\,\bar{x}}{}^T, \tag{7}$$

where $\partial f_{i\,\bar{x}}{}^T$ denotes the transpose of the vector $\partial f_{i\,\bar{x}}$.

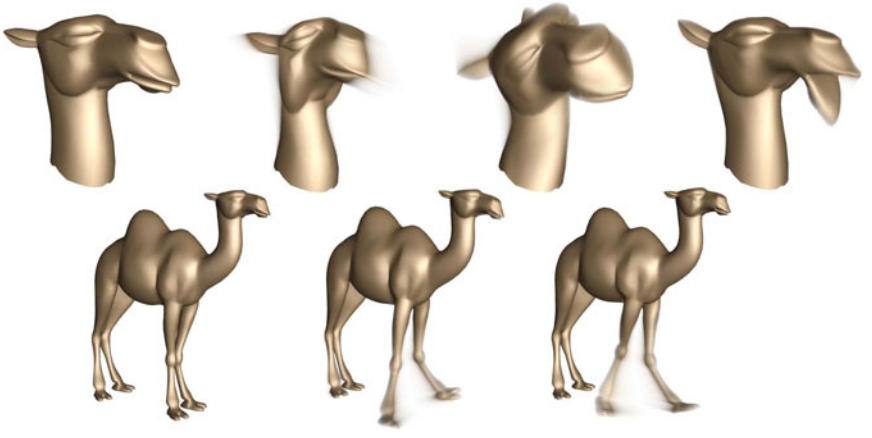


Fig. 3. Visualization of vibration modes derived from the discrete thin shell energy. In each row the left most image shows the rest state followed by some deformations captured by a vibration mode.

The computation of the first derivatives of the f_i's is usually straight forward, and, in addition, the first derivatives of many elementary quantities are explicitly stated in the literature. For example, a formula for the first derivative of the dihedral angle θ can be found in [28] and a formula for the first derivative of the area of a triangle is contained in [20].

Eigenvalue problem. To get the eigenmodes of $\mathrm{hess}_{\bar{x}} E$, we need to solve the generalized eigenvalue problem

$$\partial^2 E_{\bar{x}}\, \Phi = \lambda\, M_{\bar{x}}\, \Phi, \tag{8}$$

where $\Phi \in T_{\bar{x}} X$ and $\lambda \in \mathbb{R}$. The matrix $\partial^2 E_{\bar{x}}$ is symmetric and at least positive semi-definite ($\bar{x}$ is a minimum of E) and $M_{\bar{x}}$ is symmetric and positive definite. Hence, the structure of this problem is similar to the generalized eigenvalue problem arising in manifold harmonics [23,27]. Fast solvers for this problem are discussed in [24,27]. Since $\mathrm{hess}_{\bar{x}} E$ is self-adjoint with respect to the discrete L^2-product (given by $M_{\bar{x}}$), all eigenvalues of $\mathrm{hess}_{\bar{x}} E$, *i.e.* the solutions of (8), are real and the eigenmodes $\mathrm{hess}_{\bar{x}} E$ form an orthogonal basis of $T_{\bar{x}} X$. If we L^2-normalize the eigenmodes, they form an orthonormal basis of $T_{\bar{x}} X$ in which both Matrices $\partial^2 E_{\bar{x}}$ and $M_{\bar{x}}$ are diagonal matrices.

Vibration modes. To illustrate the concept of eigenmodes of the Hessian of a deformation energy, we look at the vibrations of a mesh in a force field induced by the energy. For simplicity, we consider the case of free vibrations. In general, the dynamics of a time-dependent mesh $x(t)$ in the space X is governed by a system of non-linear second-order ODEs of the form

$$M_{x(t)} \ddot{x}(t) = f(t, x(t), \dot{x}(t)),$$

see [1]. Here, the mass matrix M_x represents the physical mass of x and f represents the acting forces. We consider the force field that has E as its potential, *i.e.*,

$$f(t, x(t), \dot{x}(t)) = -\partial E_{x(t)}.$$

In the case of free vibrations, this is the only force. In a more general setting, we could include damping and exterior forces, see [18]. The equations that govern the motion of a time-dependent mesh $x(t)$ during free vibration are

$$\operatorname{grad}_{x(t)} E + \ddot{x}(t) = 0, \tag{9}$$

where we use the definition of the gradient, eq. (5), to simplify the formula. Since we are interested in meshes x that are (arbitrarily) close to $\bar{x}$, we expand the force $\operatorname{grad}_x E$ into a Taylor series around $\bar{x}$. Using $\partial E_{\bar{x}} = 0$ ($\bar{x}$ is a minimum of E) we get

$$\operatorname{grad}_x E = \operatorname{hess}_{\bar{x}} E(x - \bar{x}) + \mathcal{O}(\|x - \bar{x}\|^2). \tag{10}$$

Then, if we omit the second order term in (10) and plug (9) and (10) together, we get

$$\operatorname{hess}_{\bar{x}} E\, u(t) + \ddot{u}(t) = 0, \tag{11}$$

where $u(t) = x(t) - \bar{x}$. This is a system of second-order linear ODEs that are coupled by $\operatorname{hess}_{\bar{x}} E$. To solve the system we consider a normalized eigenbasis B of $\operatorname{hess}_{\bar{x}} E$. Written in such a basis, both matrices $\partial^2 E_{\bar{x}}$ and $M_{\bar{x}}$ are diagonal matrices and equation (11) takes the form

$$\Lambda\, w(t) + \ddot{w}(t) = 0, \tag{12}$$

where w is the representation of u in the basis B and Λ is a diagonal matrix that contains the eigenvalues. The system (12) is decoupled and can be solved row by row. Each row describes an oscillation around $\bar{x}$ with frequency $\sqrt{\lambda}$ in the direction of the eigenmode Φ corresponding to the eigenvalue λ. This means, that the eigenmodes of problem (8) describe the vibration modes of the mesh $\bar{x}$ (with respect to the deformation energy E).

The vibrations of a physical system are usually not free, but are affected by damping forces. Common models for such forces are Rayleigh damping, see [12], and, even simpler, mass damping, see [18]. We would like to mention that if Rayleigh (or mass) damping forces are added to the system, it still has the same vibration modes, see [12].

Normal variations. In addition to arbitrary variations of the vertices of $\bar{x}$, we consider variations that restrict every vertex to vary only in direction of the surface normal at the vertex. Let us fix a normal direction at each vertex of the mesh. Then, a normal variation is determined by a function on the mesh. This reduces the eigenvalue problem (11) to an n-dimensional problem. We denote the restriction of $\partial^2 E_{\bar{x}}$ to the subspace of $T_{\bar{x}} X$ spanned by the vertex normals of $\bar{x}$ by $\partial^2 E^N_{\bar{x}}$. Then, equation (11) reduces to

$$\operatorname{hess}_{\bar{x}} E^N\, \varphi + \ddot{\varphi} = 0,$$

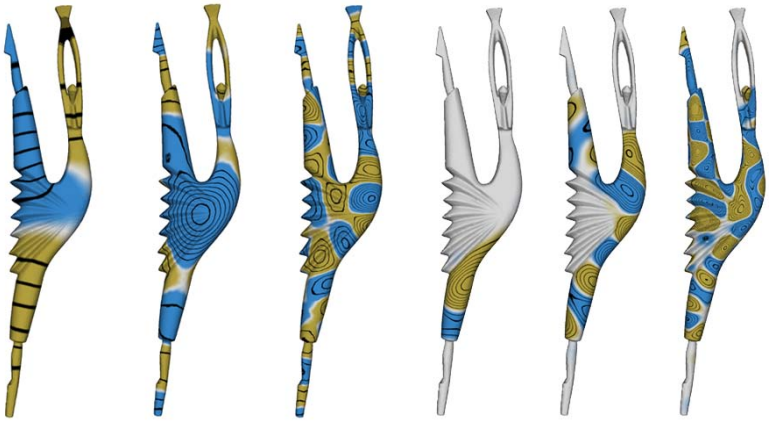


Fig. 4. Visualization of modes of the Laplacian (three images on the left) and modes of the thin shell energy restricted to normal variations (three images on the right)

where φ is a function on the mesh. The corresponding eigenvalue problem is

$$\partial^2 E^N_{\bar{x}}\, \varphi = \lambda\, M_{\bar{x}}\, \varphi,$$

where $M_{\bar{x}}$ is the mass matrix that represents the discrete L^2-product of functions on the mesh $\bar{x}$.

4 Quadratic Energies

In addition to energies defined on the space of meshes X, we consider energies that are defined on an appropriate space of functions on a surface. In particular, we consider the Dirichlet energy, that on a compact smooth surface Σ is defined for weakly differentiable functions $\varphi : \Sigma \to \mathbb{R}$ (that vanish at the boundary of Σ) by

$$E_\Delta(\varphi) = \frac{1}{2} \int_\Sigma \|\operatorname{grad} \varphi\|^2 \, \mathrm{d}A. \tag{13}$$

This is an intrinsic energy, therefore, the energy and its eigenmodes do not change under isometric deformations of the surface Σ. For some applications this is a desired feature, for other applications it is not. By modifying the Dirichlet energy, we construct a new energy that is extrinsic.

Assume that Σ is an orientable surface in $\mathbb{R}^3$ and let ν denote the normal of Σ. Then, all three coordinates ν^k of ν are smooth functions and for a weakly differentiable function φ the product $\varphi\,\nu^k$ is weakly differentiable. We define

$$E^N_\Delta(\varphi) = \sum_{k=1}^{3} E_\Delta(\varphi\,\nu^k). \tag{14}$$

This energy satisfies the equation

$$E_\Delta^N(\varphi) = E_\Delta(\varphi) + \frac{1}{2}\int_\Sigma \varphi^2(\kappa_1^2 + \kappa_2^2)\mathrm{d}A, \tag{15}$$

where κ_1 and κ_2 are the principal curvatures of Σ. This means that $E_D^N(\varphi)$ is the sum of the Dirichlet energy of φ and the φ^2-weighted total curvature of Σ.

Discrete energies. In the discrete setting, we consider a mesh $x \in X$ and the space F_x of continuous functions $u : x \to \mathbb{R}$ that are linear in every triangle. Such a function is differentiable in the interior of every triangle, and hence one can directly evaluate the integral (13). For a rigorous treatment of this discrete Dirichlet energy and a convergence analysis see [7,13]. A function in F_x is already determined by its function values at the vertices of x, hence it can be represented by an n-vector. Then, the discrete Dirichlet energy E_D is a quadratic functional on F_x, and for a function $u \in F_x$ (represented by an n-vector) $E_D(u)$ is explicitly given by

$$E_D(u) = \frac{1}{2}u^T S\, u, \tag{16}$$

where S is the usual *cotan*-matrix, see [19,28].

To discretize the energy E_Δ^N we fix a normal direction at every vertex of the mesh, and we denote the oriented unit normal vector at a vertex v_i by $N(v_i)$. Then, we say a continuous and piecewise linear vector field V on x is a *normal vector field* if for every vertex v_i of x the vector $V(v_i)$ is parallel to $N(v_i)$. The space of normal vector fields on x is an n-dimensional vector space and the map that maps a function $u \in F_x$ to the normal variation V_u, given by $V_u(v_i) = u(v_i)\, N(v_i)$ for all $v_i \in x$, is a linear isomorphism. The three coordinate functions V_u^k of V_u are functions in F_x and we define the discrete energy E_D^N analog to eq. (14) by

$$E_D^N(u) = \sum_{k=1}^{3} E_D(V_u^k).$$

A simple calculation shows that the energy E_D^N satisfies

$$E_D^N(u) = \frac{1}{2}u^T A\, u, \tag{17}$$

where the formula

$$A_{ij} = \langle N(v_i), N(v_j)\rangle\, S_{ij}$$

relates the entries A_{ij} of the matrix A to the entries S_{ij} of the cotan-matrix S.

The computation of the Hessian of the Dirichlet energy E_D and the energy E_D^N is simple. Both energies are quadratic, therefore the Hessian is constant and the matrices $\partial^2 E_D$ and $\partial^2 E_D^N$ are the matrices S and A given in equations (16) and (17). The metric we consider on F_x is the discrete L^2-product given by the mass matrix M. The eigenvalues and eigenfunctions of the Hessian of E_D (resp. E_D^N) satisfy the generalized eigenvalue problem $S\,\phi = \lambda\, M\,\phi$ (resp.

Fig. 5. Vibration distance to the marked vertex v of the Armadillo model in three colorings: continuous coloring from white being similar to red being dissimilar to v and binary colorings with two different thresholds where blue vertices are similar to v

$A\,\phi = \lambda\ M\,\phi$). To abbreviate the terminology, we call these eigenvalues and eigenfunctions the eigenvalues and eigenmodes of the energy E_D (resp. E_D^N). Similar to the vibration modes of a deformation energy, the eigenmodes of E_D and E_D^N are orthogonal with respect to M. We would like to remark that the Hessian of the E_D is the discrete Laplace-Beltrami operator and therefore the eigenmodes and eigenvalues of E_D agree with the eigenfunctions and eigenvalues of the discrete Laplace-Beltrami operator.

Modes of E_D^N. As illustrated in Figures 1 and 2, the eigenmodes of E_D and E_D^N differ significantly. Whereas the eigenmodes of the Laplacian are insensitive to the extrinsic curvature, the modes of E_D^N corresponding to lower eigenvalues hardly move in regions of high curvature, see Fig 1. A possible explanation for this behavior is the following. The energy has its minimum at the origin of the space of normal vector fields. Therefore, at the origin the gradient of the energy vanishes and the modes of the Hessian corresponding to small eigenvalues point into directions of least expenditure of energy. Now, equation (15) shows that E_D and E_D^N differ by a zero's order term, that measures a weighted L^2-norm of the function, where the weight is the sum of the squared principal curvatures. Therefore, eigenmodes of E_D^N that correspond to small eigenvalues have small function values in areas of high curvature, because then a variation in this direction causes less increase of energy.

5 Modal Signatures

In this section we introduce two multi-scale surface signatures: the *vibration signature*, based on the vibration modes of the surface, and the *feature signature*, which uses the eigenfunctions and eigenvalues of the modified discrete Dirichlet energy E_D^N. The construction of the signatures follows the construction of the heat kernel signature defined in [26].

The signatures we consider are multi-scale signatures, which take a positive scale parameter t as input. For every t such a signature is a function on the mesh, *i.e.*, it associates a real value to every vertex of the mesh. Let v be a vertex of a mesh x and let t be a positive value. Then, we define the *vibration signature* of x at vertex v and scale t by

$$S_t^{Vib}(v) = \sum_j e^{-\lambda_j t} \left\| \Phi_j(v) \right\|^2, \tag{18}$$

where λ_j and Φ_j denote the eigenvalues and the L^2-normalized vector-valued vibration modes of a mesh x. The value $\|\Phi_j(v)\|$ describes the displacement of the vertex v under the L^2-normalized vibration mode Φ_j. For a fixed t the vibration signature of v measures a weighted average displacement of the vertex over all vibration modes, where the weight of the j^{th} eigenmode is $e^{-\lambda_j t}$. The weights depend on the eigenvalues and on the scale parameter. For increasing λ, the function $e^{-\lambda t}$ rapidly decreases, *e.g.*, the modes with smaller eigenvalue receive higher weights than the modes with large eigenvalues. Furthermore, for increasing t all weights decrease, and, more importantly, the weights of the vibration modes with smaller eigenvalues increases relative to the weights of the modes with larger eigenvalues.

The *feature signature* is constructed in a similar manner, but it uses the eigenmodes and eigenvalues of the modified Dirichlet energy E_D^N. We define

$$S_t^{Feat}(v) = \sum_j e^{-\lambda_j t} \, \phi_j(v)^2 \tag{19}$$

where the λ_j are the eigenvalues and the $\phi_j(v)$ are the L^2-normalized eigenmodes of the Hessian of the modified discrete Dirichlet energy E_D^N.

Multi-scale distances. From each of the two signatures we can construct the following (multi-scale) pseudo-metric on the mesh: let v,$\tilde{v}$ be vertices of the mesh x, then we define

$$\delta_{[t_1,t_2]}(v,\tilde{v}) = \left(\int_{t_1}^{t_2} \frac{\left(S_t(v) - S_t(\tilde{v})\right)^2}{\sum_k e^{-\lambda_k t}} \mathrm{d}\log t \right)^{\frac{1}{2}}. \tag{20}$$

By construction, for any pair of scale values $t_1 < t_2$, $\delta_{[t_1,t_2]}$ is positive semi-definite and symmetric, and one can show that it satisfies the triangle inequality. We call the pseudo-metrics constructed from S_t^{Vib} and S_t^{Feat} the *vibration distance* and the *feature distance*.

The idea behind the construction of the pseudo-metric is to use the integral $\int_{t_1}^{t_2} \left(S_t(v) - S_t(\tilde{v})\right)^2 \mathrm{d}t$. However, the actual definition additionally includes two heuristics. First, since for increasing t the values $S_t(v)$ decreases for all v, we normalize the value $\left(S_t(v) - S_t(\tilde{v})\right)^2$ by dividing it by the discrete L^1-norm of S_t,

$$\|S_t\|_{L^1} = \textstyle\sum_k e^{-\lambda_k t}.$$

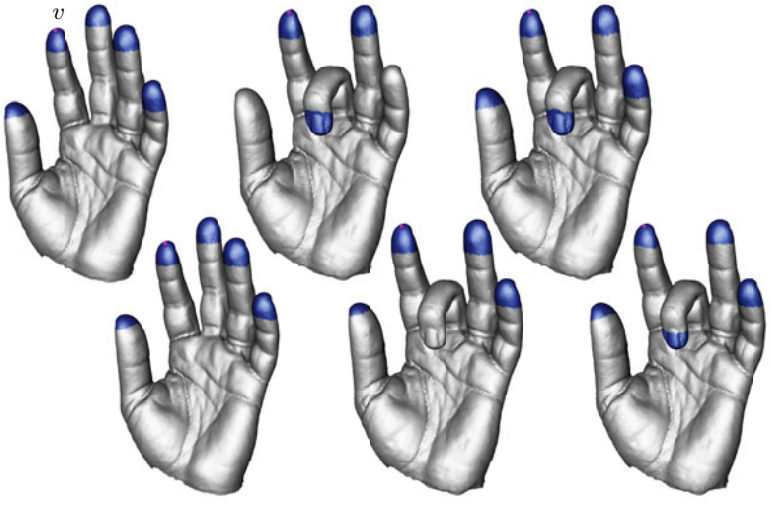


Fig. 6. Vertices (blue) similar to vertex v based on heat kernel signature [26] (top row) and our vibration signature (lower row). Left and right column depict similarity based on a small range of t's and middle column on a large range of t's.

Second, for a fixed vertex v the signature $S_t(v)$, in general, varies stronger for small values of t than for large t's. To increase the discriminative power of the pseudo-metric, we associate a higher weight to the small t's and a lower weight to the larger t's. We achieved this by using a weighted integral with weight function $\mathrm{dlog}\, t = \frac{1}{t}\mathrm{d}t$. To discretize this weighted integral we use a uniform decomposition of the logarithmically scaled interval $[t_1, t_2]$.

6 Results and Discussion

We experiment with the vibration modes of the discrete thin shell energy, the eigenmodes of E_D^N, and, for comparison, the eigenfunctions of the *cotan*- Laplace operator. In addition, we restrict the space of variations to normal variations of the mesh and inspect the modes of the thin shell energy in this setting. As a discrete L^2-scalar product we use the diagonal (or lumped) mass matrix M, which comes from FEM. The diagonal entry in the i^{th} row of the matrix is a third of the combined area of the triangles adjacent to the i^{th} vertex of the mesh. To compute the eigenmodes of a mesh, we solve the generalized eigenvalue problem (8). Since M is a diagonal matrix, this problem can be transformed into a standard eigenvalue problem as described in [27]. Then, we solve the resulting standard eigenvalue problem with the shift-and-invert Lanczos scheme described in [27]. For most examples and applications we do not need to compute the full spectrum, but only the lower part of the spectrum.

Spectral zoo. We compare the eigenmodes of the Laplacian to the ones of the modified Dirichlet energy E_D^N and to the vibration modes of the thin shell energy

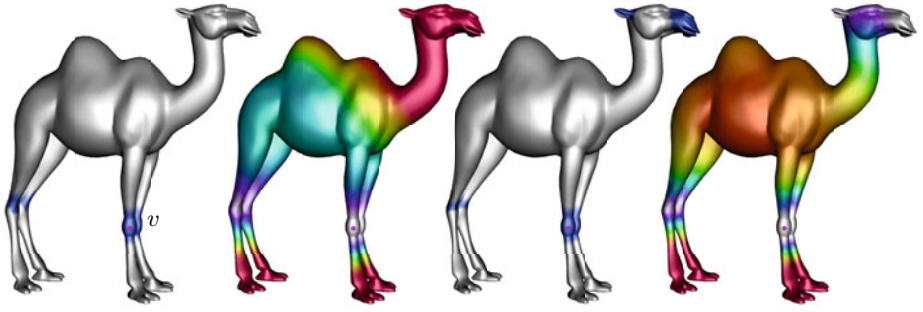


Fig. 7. Comparison of two similarity measures. Distance to vertex v in binary as well as continuous coloring based on our vibration signature (left most) and the heat kernel signature (right most).

restricted to normal variations. To convey an impression of the characteristics of the modes of the different energies, we show some examples in Figures 1, 2 and 4. To visualize the modes we use blue color for positive values, white for zero crossings, and orange for negative values. Additionally, we draw isolines as black lines.

As a first example, we study how the eigenmodes change when we isometrically deform a flat plate, see Fig. 1. On the undeformed flat plate, the eigenmodes of E_D^N equal the eigenmodes of the Laplacian. As shown in Fig. 1, there are certain differences between the three types of considered modes when computed on the deformed plate. Due to its intrinsic nature the Laplacian eigenmodes ignore the newly introduced feature, Fig. 1 left. In contrast, the eigenmodes of E_D^N and the vibration modes are sensitive to the feature, Fig. 1 middle and right. The eigenmodes of E_D^N corresponding to lower eigenvalues almost vanish at the feature and the vibration modes place additional extrema on the fold.

Investigating the differences between the eigenmodes of the Laplacian and E_D^N further, we compute them on the double torus with sharp features shown in Fig. 2. It can be seen that each of the shown Laplacian eigenmodes contains a more or less equally distributed set of extrema as well as a certain reflection symmetry, Fig. 2 left. The corresponding isolines suggest a low influence of the sharp features to the considered Laplacian eigenmodes. Similar to the Laplacian modes the two eigenmodes for E_D^N also posses a reflection symmetry, Fig. 2 right. But here we find that the eigenmodes of the lower part of the spectrum correspond to oscillations of flat areas surrounded by sharp edges, Fig. 2 right. This matches our considerations in Section 4.

For a third comparison, we choose a model without sharp edges, the dancer (25k vertices). We compare the eigenmodes of the Laplacian to the modes of the thin shell energy restricted to normal variations, see Fig. 4. As in the case of the torus we notice that the Laplacian eigenmodes oscillate equally over the whole surface, see Fig. 4 left. In contrast, the vibration modes respect the extrinsic geometry features, *e.g.*, they align to the creases on the dancer model. In addition,

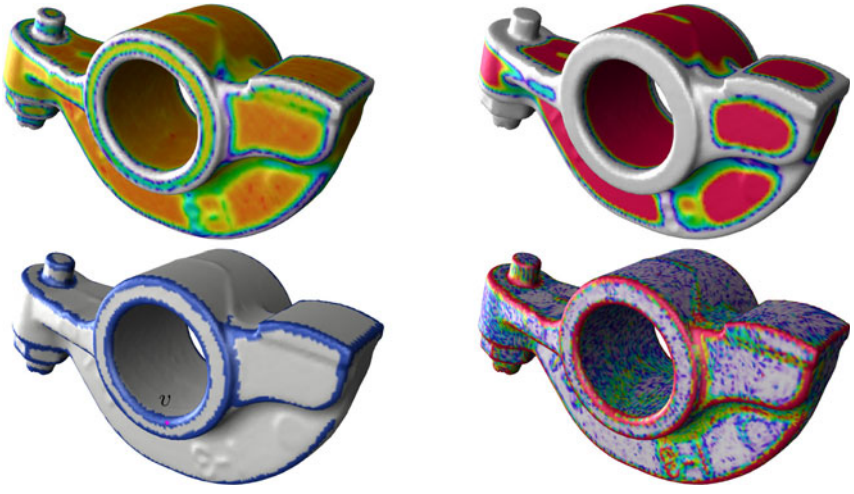


Fig. 8. Results of the feature signature on the rocker arm model. The top row shows the feature signature for increasing scale values. The bottom row shows on the left the feature distance to the marked vertex v binary colored by a threshold, and, on the right, the surface colored by curvature ($\sqrt{\kappa_1^2 + \kappa_2^2}$).

on the dancer model vibration modes corresponding to the lower eigenvalues of the thin shell energy spectrum tend to concentrate on some parts of the surface, *e.g.*, a leg of the dancer. Though the whole surface vibrates, the amplitude of the vibration varies strongly across the surface.

Fig. 3 shows eigenvibrations with respect to the discrete thin shell energy. The images on the left (top and bottom row) show the reference mesh and each of the other images visualizes a vibration mode. The discrete thin shell energy is a weighted sum of a flexural and a membrane energy. If we decrease the weight of the membrane energy, the resulting vibration modes include stretching and squashing of the surface, Fig. 3 top row 2^{nd} and 3^{rd} image. In contrast, if we put a large weight on the membrane energy, the resulting eigenmodes try to preserve the metric. Examples of such modes are given in Fig. 3 top row 4^{th}, bottom row 2^{nd} and 3^{rd} image.

Vibration Signature. In the following we examine the properties of the vibration signature S_t^{Vib} defined in eq. (18) and compare it to the heat kernel signature (HKS) introduced in [26]. As noted in Section 5, $S_t^{Vib}(\mathrm{v})$ encodes the vibration behavior of a vertex v on multiple scales, *i.e.*, vertices that oscillate with similar intensity throughout the eigenmodes, will be close in terms of the vibration distance $\delta_{[t_1,t_2]}(\cdot,\cdot)$. We illustrate this property in Fig. 5 for the Armadillo model (16k vertices). On the left we color plot the vibration distance $\delta_{[t_1,t_2]}(v,\cdot)$ to the marked vertex v. Two further binary colorings are given, colorizing vertices that are closer to v than a threshold in blue and the other vertices

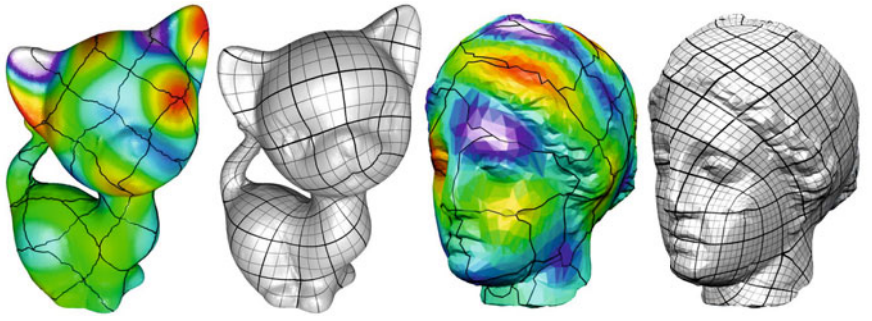


Fig. 9. Quadrangulation of the Kitten and the Venus model based on eigenmodes of the discrete thin shell energy restricted to normal variations

in white. For a small threshold the vertices on both feet are close to v; increasing the threshold causes parts of the hands to be colored in blue as well.

Fig. 6 compares S_t^{Vib} to the HKS. Every image of the hand model (40k vertices) depicts the vertices that are closer to the marked vertex v. In the first column similar results are achieved for HKS and S_t^{Vib}. Since the HKS is constructed using the spectrum and eigenfunctions of the Laplacian, the signature depends only on intrinsic properties of the surface. Thus the signature is incapable to discern isometrically deformed parts of a surface. The vibration signature however is sensitive to extrinsic information and hence represents an alternative to the HKS. This characteristic of S_t^{Vib} becomes especially apparent in the second column of Fig. 6. Here the middle finger of the hand is almost isometrically deformed. The HKS cannot distinguish this situation from the undeformed one; hence it recognizes the tips of the three longest fingers of the hand as similar to vertex v. As the deformation alters the vibration behavior of the bent finger, S_t^{Vib} detects only the tips of the unbent ones. Alike the HKS, the vibration distance can be evaluated at different scales (different choices of $[t_1, t_2]$). Choosing smaller t's increases the weights (*cf.* eq. 18) for eigenmodes with higher frequency. Therefore, more local vibrations described by these eigenmodes contribute more to the vibration distance. An example is shown on the right side of the lower row of Fig. 6. For smaller t's, $\delta_{[t_1,t_2]}(v, \cdot)$ captures vibrations of the fingertips as well and thus classifies the vertices on all tips as similar to v.

In Fig. 7 we provide a last comparison of the vibration signature and the HKS for the camel model (10k vertices). The vibration distance shown on the left, finds both pairs of knees (at the forelegs and at the hind legs) to be the closest to vertex v. For the HKS, shown on the right, the results are not as intuitive: the vertices at the mouth resp. ears of the camel are closer to the vertex v than the vertices at the hind legs, even closer than the vertices at the knees of the hind legs. This behavior of the HKS was the same at different scales and it is difficult to interpret the results. An indication for this behavior can be found by inspecting the Fiedler vector, which is the eigenfunction of the discrete Laplacian associated to the lowest (non-zero) eigenvalue. Of all eigenfunctions,

this one gets the highest weight in the signature. On the camel model, the Fieldler vector has one type of extrema (*e.g.* its minima) at tips of the toes of the hind legs at the tip of the tail and the other type of extrema (*e.g.* its maxima) at the tips of the toes of the forelegs, at the tips of the ears, and the tip of the snout. The function values at the tips of the ears and the tip of the snout are about the same as the function values at the knees of the forelegs. Hence, the contribution of this eigenfunction to the vibration distance is almost zero. This behavior repeats at some of the higher modes.

Feature Signature. The feature signature and the feature distance can be used to identify features of the surface like sharp bends or sharp corners. It is our impression that the signature could serve as an indicator function to surface segmentation algorithms. Fig. 8 shows the feature signature on the rockerarm model for different scale values. Vertices of the mesh that have a signature value close to zero are colored white in these images. The white areas seem to include the important features of the rocker arm model. The lower left image shows in blue all the vertices that are close (with respect to the feature distance) to a vertex on a sharp bend. For comparison we show a curvature plot ($\sqrt{\kappa_1^2 + \kappa_2^2}$) on the rocker arm.

Concerning the applicability as a feature indicator, an advantage of the feature signature over curvature is that the feature signature naturally comes with a scale parameter. Whereas the curvature is noisy and would require some smoothing operations, the feature distance even for low scale values seems to be much smoother. Another interesting difference is the following. Some areas of the rockerarm model have high curvature but do not indicate features, *e.g.*, the curved area inside the hole has a much higher curvature than for example the flat parts on the sides of the model. Still, the feature distance associates similar function values to both of these parts.

Quadrangulation. We investigate the applicability of the spectral quadrangulation approach by Dong et al. [6] to the eigenmodes of the thin shell energy restricted to normal variations. The Morse-Smale complex of an eigenfunction decomposes the surface into four-sided regions, see Fig. 9. Critical points emerging from high frequency noise in the functions are removed using *cancellations*, see Edelsbrunner et al. [8]. Based on the positions of the irregular (non valance 4) vertices and the connectivity of the Morse-Smale complex, a quadrangulation of the surface is constructed. The results of the quadrangulation algorithm for the Kitten (25k vertices) and the Venus (4k vertices) model are shown in Fig. 9. Two images are shown for every model: first the Morse-Smale complex together with the generating eigenmode and second the final quadrangulation. For the Kitten model the lines of the quadrangulation run diagonally while for the Venus the lines are aligned with the quadrilaterals of the Morse-Smale complex. Since the quadrangulation method depends on various factors, the initial function being only one of these, it is difficult to compare the results produced by eigenmodes of the restricted thin shell energy to those of Laplacian eigenfunctions. Still, our impression is that the quadrangulations produced by eigenmodes of the restricted thin shell energy align more with the features of the surface.

7 Future Work

Using the eigenmodes of the thin shell energy restricted to normal variations for the quadrangulation of a surface yields promising results. Many critical points of the eigenmodes are placed on the characteristic features of the surface. In its current form the quadrangulation method varies the position of the critical points in order to produce quadrilaterals of similar size, and, therefore, moves the vertices of the quadrangulation away from the features. As future work we would like to modify this scheme such that the vertices of the quadrangulation are not allowed to move away from the feature, but still may vary along the feature. The goal would be to produce quadrangulations that include features (like sharp bends) of the surface. Furthermore, it would be interesting to include the feature signature into the quadrangulation process.

Acknowledgements. This work was supported by the DFG Research Center MATHEON "Mathematics for Key Technologies" in Berlin. We would like to thank the anonymous reviewers for their comments and suggestions.

References

1. Baraff, D., Witkin, A.: Large steps in cloth simulation. In: Proceedings of ACM SIGGRAPH, pp. 43–54 (1998)
2. Barbič, J., James, D.L.: Real-time subspace integration for St. Venant-Kirchhoff deformable models. ACM Transactions on Graphics 24(3), 982–990 (2005)
3. Bridson, R., Marino, S., Fedkiw, R.: Simulation of clothing with folds and wrinkles. In: Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 28–36 (2003)
4. Choi, M.G., Ko, H.S.: Modal warping: Real-time simulation of large rotational deformation and manipulation. IEEE Transactions on Visualization and Computer Graphics 11(1), 91–101 (2005)
5. Desbrun, M., Hirani, A.N., Leok, M., Marsden, J.E.: Discrete exterior calculus (2005), arXiv:math/0508341, arXiv preprint
6. Dong, S., Bremer, P.T., Garland, M., Pascucci, V., Hart, J.C.: Spectral surface quadrangulation. ACM Transactions on Graphics 25(3), 1057–1066 (2006)
7. Dziuk, G.: Finite elements for the Beltrami operator on arbitrary surfaces. In: Partial Differential Equations and Calculus of Variations. Lecture Notes in Mathematics, vol. 1357, pp. 142–155. Springer, Heidelberg (1988)
8. Edeslbrunner, H., Harer, J., Zomorodian, A.: Hierarchical Morse-Smale complexes for piecewise linear 2-manifolds. Discrete Computational Geometry, 87–107 (2003)
9. Garg, A., Grinspun, E., Wardetzky, M., Zorin, D.: Cubic Shells. In: Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 91–98 (August 2007)
10. Gebal, K., Bærentzen, J.A., Aanæs, H., Larsen, R.: Shape analysis using the auto diffusion function. Computer Graphics Forum 28(5), 1405–1413 (2009)
11. Grinspun, E., Hirani, A.N., Desbrun, M., Schröder, P.: Discrete shells. In: Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 62–67 (2003)

12. Hauser, K.K., Shen, C., O'Brien, J.F.: Interactive deformation using modal analysis with constraints. In: Graphics Interface, pp. 247–256 (2003)
13. Hildebrandt, K., Polthier, K., Wardetzky, M.: On the convergence of metric and geometric properties of polyhedral surfaces. Geometricae Dedicata 123, 89–112 (2006)
14. Huang, J., Zhang, M., Ma, J., Liu, X., Kobbelt, L., Bao, H.: Spectral quadrangulation with orientation and alignment control. ACM Transactions on Graphics 27(5), 1–9 (2008)
15. Huang, Q., Wicke, M., Adams, B., Guibas, L.: Shape decomposition using modal analysis. Computer Graphics Forum 28(2), 407–416 (2009)
16. Lévy, B., Zhang, H.: Spectral mesh processing. In: ACM SIGGRAPH ASIA Courses, pp. 1–47 (2009)
17. Ovsjanikov, M., Sun, J., Guibas, L.: Global intrinsic symmetries of shapes. Computer Graphics Forum 27(5), 1341–1348 (2008)
18. Pentland, A., Williams, J.: Good vibrations: modal dynamics for graphics and animation. In: Proceedings of ACM SIGGRAPH, pp. 215–222 (1989)
19. Pinkall, U., Polthier, K.: Computing discrete minimal surfaces and their conjugates. Experimental Mathematics 2(1), 15–36 (1993)
20. Polthier, K.: Computational aspects of discrete minimal surfaces. In: Hass, J., Hoffman, D., Jaffe, A., Rosenberg, H., Schoen, R., Wolf, M. (eds.) Global Theory of Minimal Surfaces, Clay Foundation (2005)
21. Reuter, M., Wolter, F.E., Peinecke, N.: Laplace-spectra as fingerprints for shape matching. In: Proceedings of the ACM Symposium on Solid and Physical Modeling, pp. 101–106 (2005)
22. Reuter, M., Wolter, F.E., Peinecke, N.: Laplace-Beltrami spectra as "Shape-DNA" of surfaces and solids. Computer-Aided Design 38(4), 342–366 (2006)
23. Rustamov, R.M.: Laplace-Beltrami eigenfunctions for deformation invariant shape representation. In: Proceedings of Eurographics/ACM SIGGRAPH Symposium on Geometry Processing, pp. 225–233 (2007)
24. Saad, Y.: Numerical Methods for Large Eigenvalue Problems. Manchester University Press (1992)
25. Sorkine, O., Alexa, M.: As-rigid-as-possible surface modeling. In: Proceedings of Eurographics/ACM SIGGRAPH Symposium on Geometry Processing, pp. 109–116 (2007)
26. Sun, J., Ovsjanikov, M., Guibas, L.J.: A concise and provably informative multi-scale signature based on heat diffusion. Computer Graphics Forum 28(5), 1383–1392 (2009)
27. Vallet, B., Lévy, B.: Spectral geometry processing with manifold harmonics. Computer Graphics Forum (2008)
28. Wardetzky, M., Bergou, M., Harmon, D., Zorin, D., Grinspun, E.: Discrete quadratic curvature energies. Computer Aided Geometric Design 24(8-9), 499–518 (2007)
29. Zhang, H., van Kaick, O., Dyer, R.: Spectral mesh processing. Computer Graphics Forum (to appear 2010)

Author Index