Surpassing Local Optimality in Geometry Processing

by

Paul Zhang

S.B., California Institute of Technology (2015)
S.M., Massachusetts Institute of Technology (2020)

Submitted to the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

JUNE 2023

Authored by: Paul Zhang
Department of Electrical Engineering and Computer Science
May 19, 2023

Certified by: Justin Solomon
Associate Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by: Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Surpassing Local Optimality in

Geometry Processing

by

Paul Zhang

Submitted to the Department of Electrical Engineering and Computer Science
on May 19, 2023, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Geometry processing is a diverse field studying questions such as *how to represent a shape*, *how to modify a shape*, and *how shapes respond to perturbation*. These questions lie at the heart of many applications in computational design, simulation, fabrication, and animation. We approach these problems through the lens of geometric optimization where solutions are acquired by defining application specific objective functions paired with geometry dependent constraints. These problems can generically be solved with Newton's method to initialization dependent local optima. In this thesis, we examine whether problem specific knowledge can be leveraged to employ more advanced optimization techniques and obtain better results. We specifically explore the use of convex relaxation, variable augmentation and sum-of-squares programming to target cross-field based quad meshing, hexahedral mesh quality enhancement, and algebraic collision detection. With these tools, we manage to avoid shallower local minima and sometimes to reach or even surpass globally optimal solutions. We conclude with some principles by which these methods can be generally applied to other parts of geometry processing or optimization.

Thesis Supervisor: Justin Solomon
Title: Associate Professor of Electrical Engineering and Computer Science

Acknowledgements

The work in this dissertation could not have been completed without the help of many remarkable people. Below I highlight just a few of those who helped me along the way.

I got my first taste of success in research during an exchange program with Prof. Ryutaroh Matsumoto at the Tokyo Institute of Technology. He helped me understand various components of mentoring and scientific writing, and also encouraged me to explore a diverse set of ideas. After that, Prof. Peter Schröder and Prof. Alan Barr's classes in computer graphics showed me the elegant connections between math and computation and attracted me to pursue research in geometric computing.

During my PhD I have been lucky to collaborate with many people at MIT including Ed Chien, David Palmer, Dmitriy Smirnov, Lingxiao Li, S. Mazdak Abulnaga, Daryl DeFord, Junyi Zhu and Klara Mundilova. In addition to various research conversations I enjoyed our eclectic discussion about adjacent topics like gastronomy, fermentation, rock climbing, bees, crocheting, origami, and random other interests through the years. It's been my privilege to work with some stellar undergraduates - Zoë Marschner, Katherine Y. Cheng, Wanlin Li - whose quick progress always encouraged me to work harder and keep up. Most importantly, I would like to thank my advisor, Justin Solomon, whose optimistic outlook, infectious enthusiasm, and supportive lab group has made the last six years very enjoyable.

Many of my projects have come to fruition thanks to various sources outside MIT including David Bommes, Heng Liu, Josh Vekhter, Etienne Vouga, Ryan Viertel, Judy (Hsin-Hui) Chiang, Xinyi (Cynthia) Fan, and Rasmus Tamstorf. Their support and patience during endless video calls across many time zones have been invaluable to

shaping my research.

I would like to thank my partner Jade Yang and her family for their constant support and for keeping me grounded over the last eleven years.

My PhD research has been supported by the Department of Energy Computational Science Graduate Fellowship, and the Mathworks Engineering Fellowship.

Contents

1

Introduction

1.1 Motivation

Geometry processing is a broad field studying questions such as *how to represent a shape*, *how to modify a shape*, and *how shapes respond to perturbation*. This sequence of questions forms the digitization pipeline by which real world geometries can be understood digitally through computation and lies at the heart of many applications in computational design, simulation, fabrication, and animation. While many tools have been built to tackle parts of the pipeline, the vast diversity of geometries and applications provides a seemingly endless supply of open problems.

A key contributor to the diversity of problems in geometry processing comes from the different choices of representation. Common 2D geometry representations are triangle/quadrilateral meshes, Catmull-Clark/Loop subdivision surfaces (subsurfs) [Loo87; CC78], NURBS patches or signed distance fields (SDF). Similarly, 3D geometries can be encoded with Lagrangian point clouds, or tetrahedral/hexahedral meshes, some of which extend to subdivision volumes. The number of different representations increases as new geometry types emerge such as polyhedral meshes or neural radiance fields (NERFs).

Assuming we have picked a geometry type, we need to understand how to manipulate it. Typically modifications can be classified as *geometric* or *topological*. Geometric modifications deal with quantities like control point displacements, while

topological modifications deal with changing the connectivity between control points. These modifications are often performed to bring out desirable properties of that representation. For example, among triangle meshes, one may prefer triangle elements to contain their own circumcenters, leading to geometric deformations of triangle mesh to become Delaunay. On the topological side, Loop subdivision surfaces often prioritize the property that most of its vertices should connect to 6 neighboring triangles. Any vertex failing that criterion is *singular* and introduces visual artifacts. Since this criteria deals with connectivity, it requires topological modifications to change.

Once a shape is satisfactorily digitized, one may inquire how it reacts to perturbations. Of particular interest to us is how multiple geometries react to each other. Answering this requires solving the key problem of collision detection that asks whether multiple geometries intersect, or will intersect on a given trajectory. Since collision detection algorithms between specific geometry types typically don't extend to other geometry types, the number of collision detection algorithms required expands quadratically with the number of geometry types.

1.2 Geometric Optimization

Despite the diversity of challenges in geometry processing, solutions can frequently be obtained via geometric optimization. In essence, geometric optimization problems are posed by defining application specific objective functions paired with geometry dependent constraints. In this general form, black box solvers for optimization problems can be applied to reach initialization dependent local optima.

Two main difficulties affect this approach. First, it is often non-obvious how to pose a mathematically well-defined objective function that accurately captures the qualitative goal. Second, generic optimization problems are often solved with Newton or quasi-Newton methods which are susceptible to getting stuck in local optima or failing to converge at all. Even obtaining a feasible initialization can be difficult so generic solvers are non-ideal especially when applications can necessitate globally optimal solutions.

In this thesis, we leverage problem specific knowledge to enable more sophisticated optimization techniques, yielding improvements across the pipeline. We specifically target cross-field based quadrilateral (quad) meshing, hexahedral (hex) mesh quality enhancement, and algebraic collision detection as representative problems in geometry processing. Each of these topics have specific exploitable properties that are fortuitously compatible with advanced optimization techniques such as convex relaxation, variable augmentation and sum-of-squares programming. This allows us to avoid shallower local minima and sometimes to reach or even surpass globally optimal solutions. We conclude with some principles by which these methods can be more generally applied to other parts of geometry processing.

1.3 Related work

Out of all the different geometry types, we primarily focus our attention on quadrilateral and hexahedral meshes. Quad meshes are often preferred by artists due to their compatibility with Catmull-Clark subsurfs. Hexahedral meshes are preferred in solving numerical PDE because the results they produce with tri-linear basis functions are often superior to those produced with linear basis functions on tetrahedral meshes [Wei94; CK92]. They have also been shown to perform better with quadratic basis functions in the context of nonlinear elasto-plastic simulation [Ben+95]. Due to the persistent demand for quad and hex meshes, a variety of techniques exist to generate them.

1.3.1 Cross Fields and Quad Meshing

The generation of tangential n-RoSy fields over surfaces has many applications ranging from surface BRDF modification [Bra+18] to fabrication [Plu+21; Zhu+20; Cig+14] to texture synthesis [Knö+15] and sketch-based modeling [IBB15; BS19]. Surveys of n-RoSy field design methods are provided in [Vax+16] and [GDT15].

The $n = 4$ case (cross fields) have been especially well-studied for their applications in quadrilateral (quad) meshing where their $\frac{\pi}{2}$-symmetry allows them to guide the

layout of quad elements without rotational seams. Methods to compute intrinsically-smooth cross fields with alignment and singularity constraints are studied in [Ray+08; CDS10; Knö+13]. To contrast, [Jak+15] formulate an *extrinsic* smoothness functional on cross fields i.e. crosses are compared without a shared tangent space or connection. The resulting energy is non-convex but is minimized to local optimality, often resulting in many regions of irregular topology. Huang and Ju [HJ16] analyze this extrinsic energy, and find that it is decomposable into an intrinsic smoothness energy combined with an extrinsic curvature alignment term.

Once a cross field is obtained, standard procedures for quad meshing compute a parameterization i.e. a seamless function whose differential aligns to the cross field. Finally isolines of the parameterization yield edges of a discrete quad mesh [BZK09; KNP07]. Alternative approaches can skip parameterization by instead solving for a position field encoding quad mesh vertices directly [Jak+15].

1.3.2 Octahedral Fields and Hexahedral Meshing

The three-dimensional counterpart to a cross field is an *octahedral field*. A single octahedral frame consists of three mutually-orthogonal vectors and their negations. Huang et al. [Hua+11] introduced a particularly convenient representation of an octahedral frame as a rotation of the spherical function $g(x, y, z) : (x, y, z) \in S^2 \mapsto x^4 + y^4 + z^4$, encoded by coefficients in the spherical harmonic (SH) basis. Ray, Sokolov, and Lévy [RSL16] and Solomon, Vaxman, and Bommes [SVB17] use this representation to generate volumetric normal-aligned octahedral fields.

Analogously to cross fields in quad meshing, octahedral fields are often used to generate hexahedral meshes. Once an octahedral field is obtained, [NRP11a] computes a volumetric parameterization so that its differential aligns to the octahedral field. Isoplanes of the parameterization then partition a volume into hexes in the *hex extraction* step [LBK16]. Unsurprisingly, these steps are significantly more challenging than in the 2D case. Octahedral fields contain *singular curves* and *singular nodes* that can be topological barriers to hex meshability. Methods like [Li+12; Jia+14] try to repair singularities in existing octahedral fields [Li+12; Jia+14] to make them

meshable. Other works have derived conditions and algorithms to compute frame fields with constraints imposed on the singularities [Liu+18; CC19]. Despite these improvements, frame field based hexahedral meshing remains an open problem. Recent hex mesh generation methods have taken very different approaches by leveraging mesh-cutting operations and poly-cubing [Liv+20; Li+21].

1.3.3 Collision Detection and Intersection

Once a shape is satisfactorily digitized, we can answer how that shape responds to perturbation by simulating it. Trusty, Chen, and Levin [TCL21] develop a method for simulation of elasticity on volumes enclosed by NURBS patches without volumetric remeshing. Catmull-Clark surfaces have been used in [WHP11] for thin shell simulation. Schneider et al. [Sch+19; Sch+18] use higher-order basis functions to solve various PDEs, including linear elasticity. While these methods enable dynamic simulation in algebraically modeled surfaces, there is a notable lack of exact collision detection and intersection prevention.

Classic approaches exist for adjacent problems such as surface-surface intersection (SSI) where one computes intersection curves between static curved surfaces. SSI is frequently used in computer aided design (CAD), where curves are obtained by first linearizing the surfaces to find an initial intersection point, followed by stepping along the common tangent direction of the two surfaces. This method requires tuning of various tolerance parameters, as well as a sufficiently dense initial linearization. We refer the reader to [Bar+87] for a summary of SSI in CAD. Unfortunately, failure to find an intersection does not guarantee non-intersection.

1.4 Overview

In this thesis, we target problems across geometry processing through the lens of geometric optimization. In Chapter 2 we target the cross field generation problem, a key step in quad mesh generation. Through an uncommon formulation of cross field generation we enable convex relaxation to achieve better representations of surfaces.

In Chapter 3 we examine the problem of improving hex meshes, the volumetric counterpart of quad meshes. By identifying topological barriers to quality, we create a mesh augmentation strategy to surpass locally optimal hex meshes. In Chapter 4, we explore the use of sum-of-squares programming to tackle collision detection and adjacent problems between algebraically formulated geometries. Finally in Chapter 5 we reflect on generalizations of these methods to the rest of geometry processing.

2

Feature-Aligned Cross Field Generation via Convex Relaxation

This chapter is based on work in [Zha+20] and focuses on a method for designing feature-aligned cross fields on surfaces. The intro and related works have been trimmed to remove redundancy with Chapter 1. A greater emphasis is also put on the convex relaxation aspect of the final optimization problem.

Inspired by work in the study of octahedral fields, we borrow their encoding and apply it to crosses. We pair this representation with a class of convex energy functionals generalizing the commonly used Dirichlet energy. Our theoretical analysis of these energies in the smooth setting shows they penalize deviations from surface creases while otherwise promoting intrinsic smoothness. Furthermore, this representation lends itself well to convex relaxation resulting in high quality cross fields that automatically align to sharp features of an underlying geometry. We demonstrate the applicability of our method to quad-meshing and include an extensive benchmark comparing our fields to other automatic approaches for generating feature-aligned cross fields on triangle meshes.

Figure 2-1: A variety of feature-aligned cross fields computed using our cross field formulation.

2.1 Introduction

N-rotationally symmetric (RoSy) tangential vector fields over surfaces are ubiquitous in computer graphics. Depending on the application, n-RoSy field design algorithms must trade off between several desirable properties of the field. In almost all cases, n-RoSy fields are expected to be as smooth as possible. For surfaces with boundary, constraints on how the field aligns to the boundary are common, and for artistic applications, users may wish to prescribe a sparse set of streamlines that the field must follow. For meshing applications, alignment of n-RoSy fields to salient geometric features is also critical as a means to identify or preserve mesh detail. Our target is to improve this latter aspect for the important case of 4-RoSy fields.

There are two broad strategies for achieving feature alignment. The first is to optimize only for smoothness, under the assumption that a well-chosen functional for measuring cross-field smoothness will automatically penalize fields that fail to align to geometric features. The most commonly-used smoothness functionals (including the Dirichlet energy and its variants) are *intrinsic*, and recover solutions that are unique only up to rotation [Knö+13]. These are ambivalent to isometric deformations of the surface and ignore *extrinsic* features such as creased folds.

An alternative strategy is to include energy terms that explicitly enforce alignment to an input guiding field of principal curvature directions during cross-field design [Knö+13; Bra+18]. Drawbacks include the difficulty of robustly computing

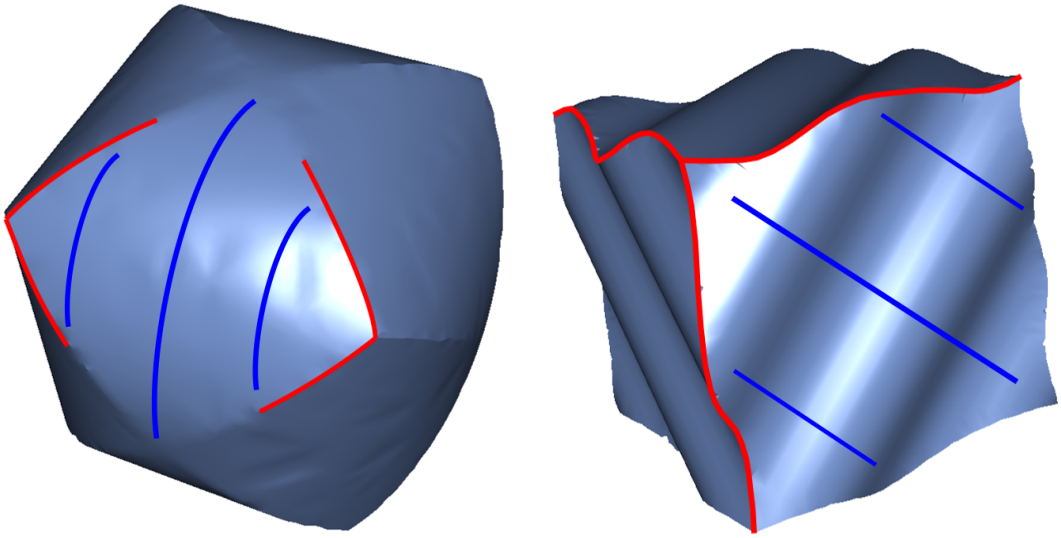


Figure 2-2: Two surfaces (the **three-cylinder-intersection** and **wavy-box**) whose maximal curvature directions (blue lines) contradict its feature curves (red lines).

principal curvature directions on noisy meshes, the fact that forcing alignment to a guiding field based on local geometry may exclude cross-field designs that are globally more optimal, and more critically the fact that principal curvature directions are often different from features e.g. Figure 2-2.

Our main observation is that neither of the above strategies adequately identify those features most important to generating high-quality quad meshes. Often the surface being modelled is constructed from smooth patches that are joined along *sharp extrinsic feature curves* where the normal direction is discontinuous or changes rapidly. On the one hand, such features are invisible to intrinsic smoothness functionals; on the other, the orientation of the feature curves often contradict that of nearby curvature lines.

Consider the surfaces shown in Figure 2-2: Neither existing strategy will promote alignment to the features curves shown in red. Both of these shapes are developable away from a sparse set of cone singularities at the corners; the gaussian curvature is nearly zero at creased edges and curved facets, and so purely intrinsic approaches have no hope of aligning to the creases. Augmenting with a guiding field based on extrinsic curvature is counter-productive, as the curvature lines (blue) are not compatible with the surface's more important crease features curves (red).

We approach feature alignment in a new way, which detects and aligns cross fields to sharp features in a stable fashion. Our method is based on an extrinsic representation of cross fields using spherical harmonic (SH) basis functions. SH functions have been used successfully in volumetric octahedral field problems for hexahedral meshing [SVB17; Hua+11; RSL16], and we argue that this representation is well-suited not only for computing octahedral fields in volumes but also for field computation on surfaces. In particular, we apply an SH representation of octahedral frames, or frames of three orthogonal directions in $\mathbb{R}^3$, proposed by Huang et al. [Hua+11]; when one of its directions is constrained to the surface normal, it exhibits the same symmetry as a two-dimensional cross. We use this fact to devise a class of cross field energies that promote intrinsic smoothness in smooth regions of the surface. Over sharp creases, however, our energy aligns the field to the crease direction, achieving automatic feature alignment without the need for explicit computation of extrinsic curvature directions or feature curves.

Contributions In this work, we

- introduce spherical harmonic functions for the computation of surface cross fields;
- propose a family of convex field smoothness energies whose optima are feature-aligned cross fields;
- provide a theoretical analysis of the behavior of a few important members of this family; and
- introduce cross fields with soft normal alignment for increased versatility/robustness.

Our approach is able to extract feature-aligned fields with comparable levels of efficiency to those of purely intrinsic algorithms. We test our algorithm extensively on over 200 different meshes with results in both § 4.6 and in supplementary materials of [Zha+20]. We leverage our algorithm to produce feature-aligned cross fields and demonstrate their usefulness for quad meshing.

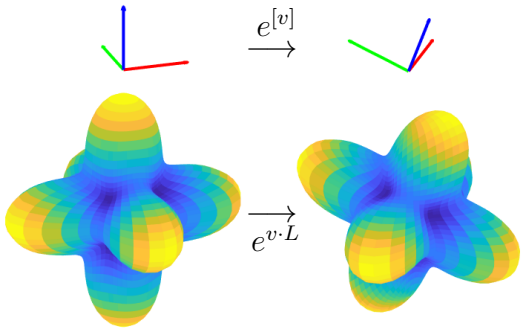


Figure 2-3: rotation of octahedral frame

2.2 Related Work

When using cross fields for quad mesh parameterization or processing, methods [BZK09; Cam+16; Knö+13; Bra+18] often promote feature alignment by including a loss term penalizing disagreement with curvature directions. However, as we argue in the introduction and illustrate in Figure 2-2, alignment to curvature directions is often less important than alignment to sharp creases. Other parameterization methods such as [BZK09; Bom+13; CBK15] allow feature alignment but just assume that such feature curves are provided as input.

2.3 Preliminaries

Since our formulation relies heavily on both the spherical harmonic representation of octahedral frames and vectorial total variation, we present a preliminary introduction to these topics.

2.3.1 Spherical Harmonic (SH) Octahedral Frames

As introduced by Huang et al. [Hua+11], the canonical axis-aligned octahedral frame can be represented by spherical harmonics as a function $g_0 : \mathbb{S}^2 \to \mathbb{R}$ written as $g_0 = \sqrt{\frac{5}{12}}Y_{44} + \sqrt{\frac{7}{12}}Y_{40}$, where Y_{lm} denotes the basis for real spherical harmonics. The function g_0 can be understood as the scaled projection of $x^4 + y^4 + z^4$ onto the fourth band ($l = 4$) of spherical harmonics. Written differently, we can encode g_0 as a vector

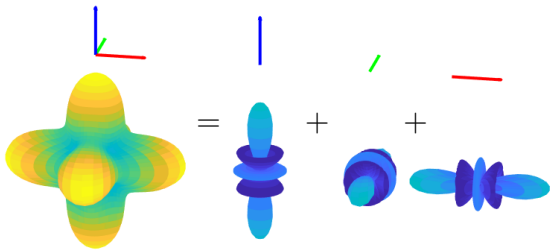


Figure 2-4: SH frame as sum of three lobes

of coefficients in the full basis of fourth-band spherical harmonics $Y_{4(-4)}, \ldots, Y_{44}$:

$$f_0 = \left[0, 0, 0, 0, \sqrt{\frac{7}{12}}, 0, 0, 0, \sqrt{\frac{5}{12}}\right]^T.$$

The space of octahedral frames can be described as all rotations of the canonical octahedral frame, that is, the orbit of f_0 under the group of 3D rotations SO(3) [PBS19, Definition 3.5]. We write this via exponentiation of the Lie algebra elements: the set of octahedral frames is

$$\mathcal{V} = \left\{ f \;\middle|\; \text{there exists } v \in \mathbb{R}^3 \text{ with } f = e^{v \cdot L} f_0 \right\},$$

where $v \cdot L = v_x L_x + v_y L_y + v_z L_z$ and L_x, L_y, L_z are the angular momentum operators expressed in the basis of band-four spherical harmonics. In this basis, L_x, L_y, L_z are each 9×9 matrices. The angular momentum operators are explicitly written in supplementary materials, §4. In this description, v can be interpreted as an axis-angle representation of rotation, with corresponding rotation matrix $e^{[v]}$. $[v]$ denotes the skew-symmetric matrix that acts as $[v]u = v \times u$. Accordingly, $e^{v \cdot L} g_0$ encodes the octahedral frame whose directions are $\hat{x}, \hat{y}, \hat{z}$ rotated by $e^{[v]}$, where $\hat{}$ denotes normalization (see Figure 2-3).

Using such SH rotations, we can present an alternative interpretation of the octahedral frame f_0 as the sum of three orthogonal SH lobe-shaped functions. The z-aligned lobe is $l = [0, 0, 0, 0, \sqrt{\frac{7}{12}}, 0, 0, 0, 0]$ and is depicted in the inset. Lobes can be rotated in the same way that frames can i.e. by applying $e^{v \cdot L}$. The canonical

octahedral frame can therefore be equivalently expressed as $f_0 = l + e^{\frac{\pi}{2}L_x}l + e^{\frac{\pi}{2}L_y}l$ (see Figure 2-4).

The space of octahedral frames that are aligned to a unit vector $\hat{n}$ can be described by the set

$$\left\{ e^{v_n \cdot L} e^{\theta L_z} f_0 \;\middle|\; \theta \in S^1 \right\},$$

where v_n is any axis-angle rotation taking $\hat{z}$ to $\hat{n}$, (e.g., the vector parallel to $\hat{z} \times \hat{n}$ and has magnitude equal to the angle from $\hat{z}$ to $\hat{n}$) and θ encodes an additional twist of the frame about $\hat{n}$. The first rotation about $\hat{z}$ can be written in explicit form [Hua+11] as

$$e^{\theta L_z} f_0 = \left[\sqrt{\frac{5}{12}} \cos 4\theta, 0, 0, 0, \sqrt{\frac{7}{12}}, 0, 0, 0, \sqrt{\frac{5}{12}} \sin 4\theta \right]^T.$$

The above allows us to formulate the set of all octahedral frames g aligned to a given direction $\hat{n}$ in terms of two constraints:

$$\|f\|_2 = 1, \quad W_n f = u_0 = \left[0, 0, 0, \sqrt{\frac{7}{12}}, 0, 0, 0 \right]^T, \tag{2.1}$$

where W_n is the second through eighth rows of $e^{-v_n \cdot L}$. The linear constraint rotates the frame from normal alignment to $\hat{z}$ alignment, and the norm constraint ensures that the first and last components are of the appropriate form.

Lastly, we will make use of the projection operator $\pi_{\mathcal{V}} : \mathbb{R}^9 \to \mathcal{V}$ onto the space of octahedral frames $\mathcal{V}$, as defined in [PBS19, § 2.5.5].

2.3.2 Vectorial Total Variation

We will later make use of a total variation energy (amongst others) to analyze the behavior of our cross fields on creased surfaces. Here, we introduce total variation and vectorial total variation definitions in $\mathbb{R}^n$ and provide intuition about their use. The extension to functions on a Riemannian manifold is straightforward, using the standard intrinsic gradient and divergence operators.

The total variation of a differentiable scalar function $h : \Omega \to \mathbb{R}$ is $TV[h] =$

$\int_\Omega \|\nabla h\|_2 \, dA$ where $\Omega \subset \mathbb{R}^n$ [AFP00]. For non-differentiable h, the relevant definition is:

$$TV[h] = \sup_{\phi \in C_c^1, \forall_x \|\phi(x)\|_2 \leq 1} \left(\int_\Omega h \nabla \cdot \phi \, dA \right),$$

where C_c^1 denotes differentiable, compactly-supported vector fields. For smooth h, equivalence to $\int_\Omega \|\nabla h\|$ follows from integration by parts and Stokes's theorem. In this case, the maximizing ϕ is $-\nabla h / \|\nabla h\|$. If h is the indicator function of a suitably regular (e.g., non-fractal) subset $A \subset \Omega$, then $TV[h]$ is the perimeter of A.

When $h : \Omega \to \mathbb{R}^m$ is vector-valued rather than scalar-valued, there are many different definitions for the vectorial total variation $VTV[h]$ [Sap96]. We use one proposed by Di Zenzo [Di 86], which for differentiable h is given by $VTV[h] = \int_\Omega \|\nabla h\|_F \, dA$, where $\| \cdot \|_F$ is the Frobenius norm. More generally, we can take

$$VTV[h] = \sup_{\phi \in C_c^1, \forall_x \|\phi(x)\|_F \leq 1} \left(\sum_{i=1}^{m} \int_\Omega h_i \nabla \cdot \phi_i \right), \tag{2.2}$$

where $h = (h_1, h_2, \ldots, h_m)$, and $\phi = (\phi_1, \phi_2, \ldots \phi_m)$ is a differentiable, compactly-supported m-tuple of vector fields. This definition is *not* equivalent to a sum of m independent scalar total variations: The constraint on ϕ introduces nontrivial coupling between the dimensions. This definition of total vectorial variation is considered in the case where Ω is a surface in $\mathbb{R}^3$ by Bresson and Chan [BC08], but without specific analysis for discontinuous h.

2.4 Spherical Harmonic Octahedral Frames as Cross Fields

We use normal-aligned octahedral fields to encode tangent cross fields on surfaces, with the goal of computing a smooth cross field on a surface aligned to sharp features. The SH representation will enable us to capture features even when they are purely extrinsic. To this end, our next task is to define a means of measuring smoothness by examining the gradient of a SH field along the surface.

2.4.1 Derivatives of SH Octahedral Frames

To calculate $\|\nabla f\|^2$, we first express it in an appropriate local coordinate system that simplifies the formulas in coordinates and better reveals the structure. Following the notation in § 2.3.1, an octahedral field $f(r) : \Omega \to \mathcal{V} \subset \mathbb{R}^9$ can be parameterized relative to a point r^* by $v(r) : \Omega \to \mathbb{R}^3$, where $v(r)$ is the axis-angle rotation from $f(r^*)$ to $f(r)$. This implies that $v(r^*) = [0, 0, 0]$. Without loss of generality, we rotate the surface so that the normal of Ω at r^* is $\hat{z}$. We can then compute the gradient ∇f at the point r^* from the formula $f(r) = e^{v(r)\cdot L} f(r^*)$:

$$\nabla f(r)|_{r^*} = \begin{bmatrix} | & | & | \\ L_x f(r^*) & L_y f(r^*) & L_z f(r^*) \\ | & | & | \end{bmatrix} \begin{bmatrix} | & | & | \\ \nabla_x v & \nabla_y v & \nabla_z v \\ | & | & | \end{bmatrix}_{r^*} . \tag{2.3}$$

As the field $f(r)$ encodes an extrinsically embedded frame at each point, we take the gradient ∇ to be the component-wise derivative of the field's nine scalar functions rather than a covariant or Lie derivative along the surface in order to capture the extrinsic geometry of the surface. We use [Ros02, §1.2.5] and the fact that $v(r^*) = [0, 0, 0]$ to derive Equation 2.3.

By combining facts about the SH representation and standard results in differential geometry we show that the squared norm $\|\nabla f(r)\|^2$ at r^* can then be expressed in the following more intuitive way:

Proposition 1. Let $f(r) : \Omega \to \mathcal{V} \subset \mathbb{R}^9$ be a normal-aligned octahedral field over a smooth surface Ω. Then at every point $r^* \in \Omega$, $\|\nabla f\|_F^2 = k_1^2 + k_2^2 + w$, where k_1 and k_2 are the principal curvatures and w measures the intrinsic tangential twist of the octahedral field. Using mean and Gauss curvatures H and K, we can write $\|\nabla f\|_F^2 = 2H^2 - K + w$.

Full proof of this formula is provided in § 2.8.1. Proposition 1 gives a more intuitive form for Equation 2.3 and relates the spherical harmonic representation of an octahedral frame to properties of the frame it represents. Most notably, the Dirichlet energy of the SH representation can be effectively decoupled into extrinsic

dependence of $\|\nabla f\|_F^2$ on the surface Ω and the intrinsic tangential twisting of the normal-aligned octahedral field $f(r)$. The values of H and K simply contribute a fixed quantity depending on Ω rather than the field. Therefore, the influence of f on $\|\nabla f\|_F^2$ is just in w, the intrinsic twist of the cross field it represents. We stress that this behavior is quite different from the behavior of the component-wise derivative evaluated on vectors, as studied in [HJ16], where their smoothness energy promotes alignment to extrinsic curvature directions.

2.4.2 L^p Smoothness Energy of SH Cross Fields

Suppose we wish to measure smoothness of a normal-aligned octahedral field in the SH representation. We define the following class of convex smoothness energies using the L^p-norm of $\|\nabla f\|_F$ over the surface Ω for $p \geq 1$:

$$E_p(\Omega, f) = \left(\int_\Omega \|\nabla f\|_F^p \, dA \right)^{\frac{1}{p}}. \tag{2.4}$$

We now analyze the behavior of the E_p energy for cross fields in several select cases.

Case $p = 2$: Dirichlet Energy

We begin with a common choice in geometry processing when smoothness is desirable: the Dirichlet energy E_2. Given Proposition 1, we can write the Dirichlet energy as $\int_\Omega 2H^2 - K + w$. Since H and K are independent of the octahedral field f, they have no influence over the f that minimizes E_2. Therefore on smooth Ω, we recover intrinsically smooth cross fields.

Since the Dirichlet energy may diverge at singularities [Knö+13], this choice of energy has the theoretical drawback of diverging for all f in the neighborhood of creases which break octahedral symmetry. In the discretized setting, however, the behavior of E_2 is dependent on mesh resolution and empirically leads to strong feature alignment as demonstrated in §4.6. It also leads to an easily-solved optimization problem described in §2.5.2.

Case $p = 1$: Vectorial Total Variation

As noted in the previous section, the conventional means of measuring field smoothness fails to be well-defined for our field representation on creased surfaces. We show here that the E_1 energy is not only finite across sharp edges and around singular points but also provides an intuitive measure of field quality that captures both smoothness and feature alignment. It is also known as the *vectorial total variation*.

Consider a function $f : \Omega \to \mathbb{R}^9$ that is piecewise smooth on n closed patches Ω_j intersecting in a finite-length curve network $\Gamma = \bigcup_{k=1}^{s} \gamma_k$, where each γ_k is a C^1 curve. Equivalently $\Gamma = \bigcup_{j=1}^{n} \partial\Omega_j$, and the vectorial total variation can be decomposed into integrals over each patch and Γ.

Proposition 2. For compact Ω and f as above, $VTV[f]$ is finite and given by the following equation:

$$VTV[f] = \sum_{j=1}^{n} \int_{\mathring{\Omega}_j} \|\nabla f\|_F \, dA + \sum_{k=1}^{s} \int_{\gamma_k} \|f^+ - f^-\|_2 \, dL, \tag{2.5}$$

where f^+ and f^- refer to the limiting values of f on either side of γ_k, and $\mathring{\Omega}_j$ denotes the interior of Ω_j.

The basic argument starts from Equation 2.2, splits it into integrals over the patches, applies integration by parts, and utilizes partitions of unity to construct a maximizing sequence of ϕ's. The full argument is contained in § 2.8.2. An analogous result, which applies to arbitrary functions on $\mathbb{R}^n$ with bounded variation, is contained in [AFP00], with the addition of a third term representing the *Cantor* part of f. Since our f is piecewise smooth, however, we can safely ignore the Cantor part. The second term is often referred to as the *jump* part in the total variation literature.

Equation 2.5 provides an intuitive description of the total variation of an octahedral field in the SH basis as a measure of intrinsic smoothness with extra jump terms. Letting f represent a normal-aligned octahedral field we obtain:

$$VTV[f] = \sum_{j=1}^{n} \int_{\mathring{\Omega}_j} \sqrt{2H^2 - K + w} \, dA + \sum_{k=1}^{s} \int_{\gamma_k} \left\|f^+ - f^-\right\|_2 \, dL \tag{2.6}$$

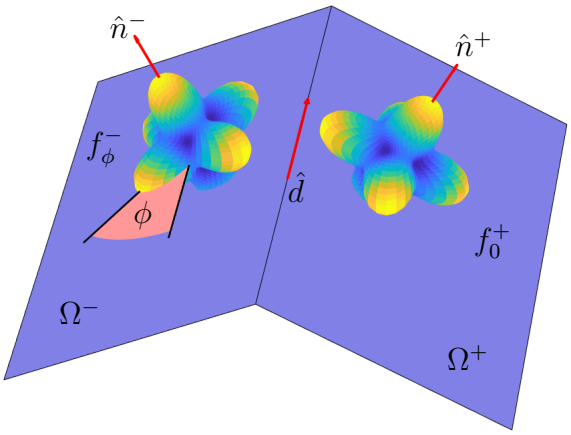


Figure 2-5: Octahedral frames near crease

Generalizing to Creased Surfaces While the above result is derived for smooth surfaces Ω and discontinuous f, we can further generalize the result to a surface Ω constructed from smooth open patches Ω_j joined along a network of sharp creases $\Gamma = \bigcup_{k=1}^{s} \gamma_k$. As there is neither a consistent metric nor a consistent tangent space on Ω across Γ, there is no well-defined choice of gradient. We therefore use Equation 2.6 as the *definition* for E_1 on such a creased surface. Since f is a normal-aligned octahedral field, it is necessarily discontinuous across creases, resulting in contributions to the jump term.

The jump $\|f^+ - f^-\|_2$, where f^+ and f^- represent octahedral frames aligned to different normal directions, is minimized if f^+ and f^- are both aligned to the axis of rotation from one normal to the other. We formalize this property by Proposition 3

Proposition 3. Let Ω^+ and Ω^- be smooth patches of a surface with normal directions $\hat{n}^+$ and $\hat{n}^-$ that meet at a crease. Let $\hat{d}$ denote the intersection of their tangent spaces at the crease. Let f_θ^+ and f_ϕ^- be the octahedral frames on either side of the crease aligned to $\hat{n}^+$ and $\hat{n}^-$ respectively. θ and ϕ denote their deviation from alignment to $\hat{d}$. The cost $\|f_\theta^+ - f_\phi^-\|_2$ is minimized by $\theta = \phi = 0$.

Proof of this proposition is left to § 2.8.3. The setup is depicted in Figure 2-5, showing discontinuous normal directions $\hat{n}^+$ and $\hat{n}^-$ as the left and right red arrows respectively. The crease direction $\hat{d}$ is shown by the middle red arrow. We emphasize that this proposition implies (locally) crease alignment always minimizes the VTV.

We extensively test and show in § 4.6 that this crease alignment tends to globally hold on surfaces with complicated geometry and topology as well.

General $p \geq 2$

By Equation 2.4, and Proposition 1, E_p incentivizes intrinsic smoothness for all p on smooth domains. On creased domains, we have demonstrated (local) crease-alignment for the $p = 1$ case. For $p \geq 2$, the value of E_p diverges for a creased surface. However, we find empirically that minimizing E_p (by recovering solutions to Equation 2.7) on a discretized surface leads to stronger feature alignment as p increases. This behavior may be explained by Proposition 3, which affects all edges regardless of p. The p simply exponentiates the energy across each edge before accumulating it into the total E_p. Local to a single edge, the energy-minimizing configuration is unaffected by p. Based on our experiments, we further conjecture that the sequence of fields obtained by minimizing E_2 on an increasingly dense discrete approximation of Ω converges to a feature-aligned cross field. This intrinsically smooth feature-alignment is empirically shown in Figure 2-10. We leave proof of this conjecture to future work.

Relation to Polycube Surfaces

We achieve an additional property for all values of p through our use of SH octahedral frames. Consider the case of Ω being a cube: minimizers of E_p will have zero energy, despite the cube's sharp corners, since the field's octahedral symmetry allows it to simultaneously align to all three creases at each corner. Effectively a surface with many angle-$\frac{\pi}{2}$ turns and cube-corners can have just as low of an energy as one with no creases at all. More generally, if Ω is a polycube surface, $E_p(\Omega, f) = 0$ by choosing f to be a facet aligned uniform frame field.

2.5 Optimizing for an Octahedral Frame Field

Our discussion above provides a new class of energies based on the SH representation of cross fields, which naturally promote both intrinsic smoothness and extrinsic crease

alignment without the need for feature curve detection or reliance on potentially noisy local curvature estimates. For this reason, we propose solving the following variational problem to find a cross field f^* over a surface:

$$\begin{aligned} f^* = \arg\min_f \quad & E_p(\Omega, f) \\ \text{subject to} \quad & W_{n(x)} f(x) = u_0. \end{aligned} \tag{2.7}$$

Recall that the constraint encodes normal alignment of the frame field (see equation (2.1)). Some past algorithms have an extra $||f(x)||_2 = 1$ constraint which results in uniform-scale isotropic octahedral fields over Ω. This constraint makes the problem nonconvex, and causes the functional to diverge in the neighborhood of field singularities, which are unavoidable on generic surfaces by the Poincaré-Hopf Theorem. Accordingly, a relaxation is naturally required; we drop the constraint $\|f(x)\|_2 = 1$, yielding a convex problem with globally-optimal solution. Dropping the $\|f(x)\|_2 = 1$ constraint allows the frame's two tangential components to scale independently from its normal component, resulting in anisotropic octahedral fields. We obtain octahedral fields with uniform-magnitude normal-lobes, and varying scale in the magnitude of the tangential cross field. This relaxation is similar in spirit to the one which appears in [Knö+13] , and has similar benefits, including automatic placement of singularities, and bounded-energy minimizers in the smooth limit (which is necessary in order for the field to be insensitive to the underlying mesh).

2.5.1 Soft Normal Alignment

It is sometimes beneficial to relax the normal alignment constraint, e.g., in cases where the mesh contains sliver triangles with unstable normal directions. In these cases, a smoother cross field can be obtained by deviating slightly from exact normal alignment. This relaxation changes the optimization problem from Equation 2.7 into

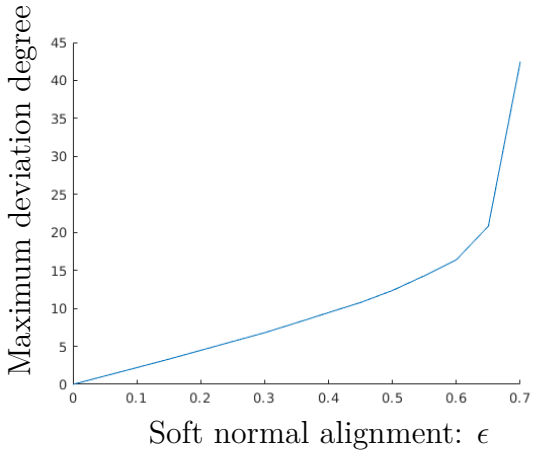


Figure 2-6: Maximum angle deviation w.r.t. soft normal alignment

the following:

$$\begin{aligned} f^* = \underset{f}{\text{argmin}} \quad & E_p(\Omega, f) \\ \text{subject to} \quad & \|W_{n(x)} f(x) - u_0\|_2 \leq \epsilon. \end{aligned} \tag{2.8}$$

This problem imposes a point-wise normal alignment constraint with tolerance ϵ. When $\epsilon = 0$, we recover the hard normal alignment formulation Equation 2.7. On the opposite side of the spectrum, as $\epsilon \to \|u_0\|_2 = \sqrt{\frac{7}{12}} \approx 0.76$, the solution to Equation 2.8 approaches a constant octahedral field. This is the case where normal alignment has relaxed so far that the octahedral frames are effectively unconstrained.

For values in between we perform the following experiment to obtain a rough correspondence between soft normal alignment parameter ϵ and maximum angle deviation from normal alignment: For each value of ϵ between 0 and .7 (at intervals of .05), we sample 100000 ϵ-perturbations of a $\hat{z}-$aligned frame, extract the frame they represent, and compute its maximum angle deviation from the $\hat{z}$-axis. Results are shown in Figure 2-6.

We highlight that this parameter encodes a point-wise constraint uniformly applied over the mesh. As such its interpretation does not change with different meshes.

The benefit of soft normal alignment is demonstrated in Figure 2-7. Due to the influence of a sliver triangle in the **buste** mesh with unstable normal direction, the hard-normal-aligned cross field is forced to create a localized artifact. By using soft

normal alignment, the sliver triangle's unstable normal direction has less influence over the resulting cross field, therefore increasing the quality of the result. A similar benefit is demonstrated on the **duck** and **armchair** meshes shown in the supplementary materials.

Additionally we test soft normal alignment on a **cube-mesh** with artificial noise added in Figure 2-9. With hard normal alignment the cross fields exhibit undesirable alignment to noise which increases with p. With soft-normal alignment, the cross fields show significantly decreased sensitivity to noise.

2.5.2 Discretization

Now we describe how to construct smooth cross-fields by numerically optimizing a discretization of E_p. We assume the surface Ω has been triangulated into a manifold mesh $\mathcal{M} = (V, E, F)$. Let n_t be the normal direction of triangle $t \in F$. We represent a cross on M as a normally-aligned octahedral frame $f_t \in \mathcal{V} \subset \mathbb{R}^9$ per triangle. We use the shorthand f to denote the concatenation of all f_t into a single $9|F| \times 1$ vector. V is a $n_v \times 3$ matrix of vertex positions, where n_v is the number of vertices. E denotes the $n_e \times 2$ matrix of edges, where n_e is the number of edges. The energy E_p can be discretized as

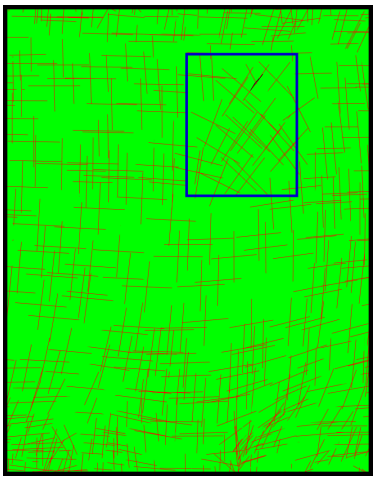
$$E_p = \left(\sum_{e \in E} w_e \| f_{t_1} - f_{t_2} \|_2^p \right)^{\frac{1}{p}} \tag{2.9}$$

where t_1 and t_2 are triangles adjacent to edge e, and w_e are weights corresponding to the dual Laplacian. We use $w_e = \frac{\|e\|}{\|e^*\|}$, where $\|e\|$ is the length of edge e and $\|e^*\|$ is the distance between barycenters of t_1 and t_2.

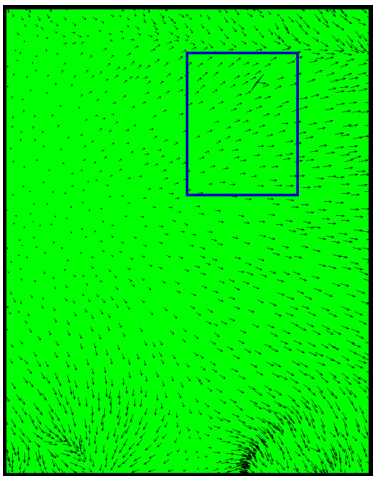
For $\epsilon = 0$, the normal alignment constraint is discretized by the linear constraint $Wf = u$, where W is a sparse block-diagonal matrix with a block W_{n_t} for each triangle. It has dimensions $7|F| \times 9|F|$. The vector u is a repetition of u_0 for each triangle, resulting in a $7|F| \times 1$ vector. For $\epsilon > 0$, the normal alignment constraint is discretized by a second-order cone constraint: $\|W_{n_t} f_t - u_0\|_2 \leq \epsilon$ per triangle.

For the case $p = 1$ and a completely flat surface Ω, our discretization agrees with the standard discretization of total variation in image processing [ROF92; Cha+10].

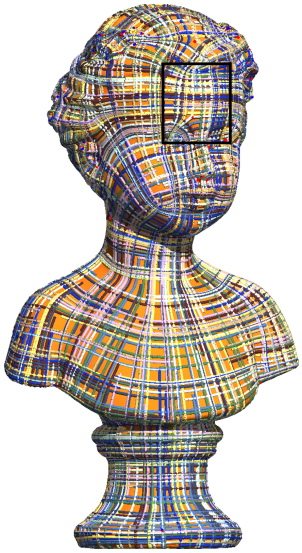
(a) hard normal-aligned streamlines



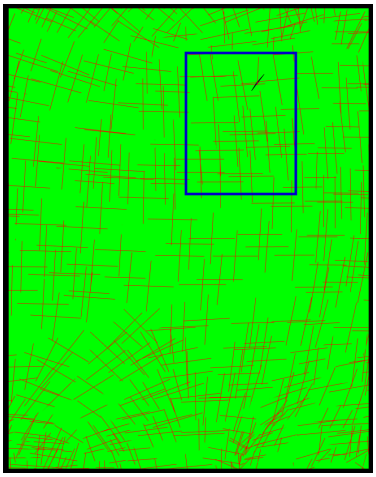
(b) hard normal-aligned crosses



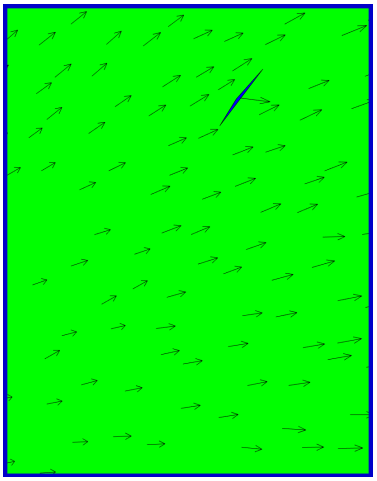
(c) mesh normal directions near sliver triangle (zoom × 1)



(d) soft normal-aligned streamlines



(e) soft normal-aligned crosses



(f) mesh normal directions near sliver triangle (zoom × 2)

Figure 2-7: Soft normal alignment increases quality of the cross field and decreases the influence of mesh artifacts. The **buste** mesh is shown with $p = \infty$ and varying normal alignment: $\epsilon = 0$ for the top figure and $\epsilon = 0.5$ on the bottom. (a) Hard normal aligned streamlines; (b) magnified crosses shows a small patch of diagonal crosses in an otherwise regular region; (c) magnified triangle normals visualized with sliver triangle 4611 shaded in blue; (d) soft normal aligned streamlines; (e) magnified crosses no longer shows diagonal artifacts; (f) extra-magnified triangle normals visualized with sliver triangle 4611 shaded in blue. While the normal direction of the region points diagonally up and right, the sliver triangle's normal direction points almost completely to the right.

2.5.3 Meshes with Boundary

When the mesh contains boundaries, we do not enforce any boundary conditions; in PDE parlance, this choice corresponds to "natural boundary conditions." While it is tempting to expect the natural boundary conditions for $p = 2$ to imply zero Neumann boundary conditions [Ste+18], the SH representation vector is complicated by being constrained to a spatially varying linear subspace. We simply allow the cross on the boundary to be that which minimizes total energy. If desired, one can enforce a constraint that the cross field on the boundary be aligned to the boundary through the method described in § 2.5.4.

2.5.4 Manual Guidance

To support manual guidance of the octahedral frame field, we can prescribe alignment of the frame field to streamlines. Streamline constraints combined with normal alignment result in a fully-determined frame. Therefore, prescribing streamlines is equivalent to prescribing the value of f_t on a subset of triangles T_p. Denote the prescribed octahedral frame on triangle t as F_t. We then add a new linear constraint that

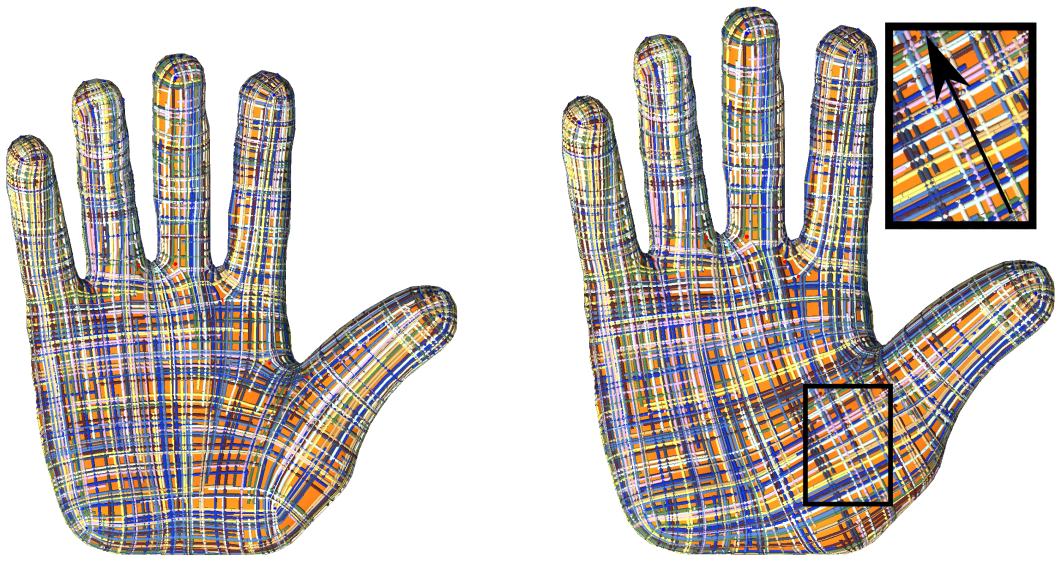
$$\forall t \in T_p, \quad f_t = F_t. \tag{2.10}$$

This technique is demonstrated in Figure 2-8.

2.5.5 Non-Triviality Constraint

As a result of dropping the unit-norm constraint from Equation 2.7, we have no explicit guarantee that the tangential components of octahedral frames do not degenerate to zero. On a surface with a crease, however, the normal alignment constraint on one side of the crease imposes that the magnitude of the tangential component on the other side of the crease is close to one. As a result, we observe empirically that the vast majority of our octahedral frames do not degenerate.

In the case that octahedral frames do degenerate significantly, their norms can



(a) before (b) after

Figure 2-8: Octahedral fields obtained by minimizing E_∞ on the **hand** mesh before and after adding manual direction. The manually-added streamline is shown by the inset black arrow. This constraint removes a singularity from the original octahedral field.

be too small to project robustly. We locate these by using the octahedral projection from [PBS19] to measure the distance from f_t to the octahedral variety $\mathcal{V}$: $d(f_t) = \|\pi_{\mathcal{V}}(f_t) - f_t\|_2$, and thresholding by $d(f_t) > .665$. If such frames are found, we run the optimization again while holding non-degenerate frames to their projected values. In our experiments, just one round of re-solving results in 99.8% non-degenerate frames.

2.5.6 Solving For an Octahedral Field

In its most general form, our problem formulation consists of minimizing a mixed-norm objective, with both linear and second-order cone constraints. This results in a convex problem that we solve with Mosek 9 to global optimality [ApS17]. The normal alignment constraint becomes $\left[\epsilon, (W_{n_t} f_t - u_0)^T\right] \in \mathcal{L}^8$, where $\mathcal{L}^8$ is the 8-dimensional Lorentz cone. Likewise, the energy is formulated using a single p-norm cone. Our code is written in Matlab with a mex interface to Mosek; it builds cross-platform. Since our problem is convex, any dependence on initialization would entirely be due to non-unique solutions, which we do not observe in practice. Furthermore, we use

the interior point method, which does not require or accept manual initialization. In the specific case of $\epsilon = 0, p = 2$, solving this optimization is equivalent to solving a linear system.

2.6 Results

We begin with a comparison of the behavior of our energy for different values of p. This experiment is depicted in Figure 2-11. We observe that our cross fields naturally align to features with increasing strength for higher p. In the case $p = 1$, our cross field is discontinuous over all creases, but while it is provably incentivized to align, it sometimes deviates due to the influence of neighboring creases, e.g. on the top surface of the fandisk. For $p = 2$, our cross fields achieve close alignment to the upper half of the shallow crease, as well as alignment on the top face where the $p = 1$ case failed. Finally for $p = \infty$ our fields align down the entirety of the shallow crease. While in theory, E_p for $p \geq 2$ diverges on creases, we observe that its discretization yields empirically strong crease alignment. This may be due to application of Proposition 3 over all edges of the mesh. We show our fields for different discretizations of the same geometry in Figure 2-12 and observe that in all cases, we achieve crease alignment.

Supplementary Materials In our supplementary document we perform an empirical study to evaluate the performance of our method. We evaluate our method on a number of models drawn from the Thingi10k [ZJ16] dataset, as well as a number of other commonly used benchmark models to demonstrate effective crease alignment on real-world models. We also compare our approach to several baseline methods ([Jak+15; Bra+18; Knö+13]), by generating fields on the models in the "Robust Field-Aligned Global Parametrization" dataset [MPZ14], taking care to sample the relevant parameter space for each formulation. While it is difficult to precisely quantify the quality of a vector field, we highlight a number of cases where our method recovers fields which more faithfully conform to mesh features than baseline methods on real-world models.

Number of triangles	Bases setup	Biharmonic solve	Instant meshes	Globally optimal	Ours
3K	5	.005	.026	.85	2.8
12K	24	.005	.053	20.46	15.058
20K	44	.005	.080	21.913	25.895
69K	170	.006	.141	62.733	135.09
80K	181	.006	.222	71.15	112.3

Table 2.1: Runtimes in seconds for computing cross-fields using different methods on meshes with a varying number of triangles. Methods listed are those of Brandt et al. [Bra+18], Jakob et al. [Jak+15], Knöppel et al. [Knö+13], and our own. Runtimes for fields from [Bra+18] are split into time needed for the setup of 500 bases eigenfields and the field computation separately because of drastically differing timescales.

Our runtimes are shown for a set of meshes with 240 to 76K vertices and 480 to 152K faces in Figure 2-15. Runtimes naturally increase with mesh size and appear to grow linearly with number of triangles in our mesh test set. Memory costs are incurred to store a W_{n_t} per triangle, a single u_0, w_e per edge, and f_t per triangle. Hence, storage is linear in size of the mesh. More detailed information regarding parameter choices and runtimes is provided in the supplementary materials. Table 2.1 shows a summary of our runtimes in comparison to that of other methods. Our runtimes are on the same scale as [Knö+13] and to the bases setup step in [Bra+18].

Comparison to Explicit Feature Curves. Next, we compare our feature-aligned cross fields to those produced with the help of explicitly-computed feature curves. We obtain feature curves on the 1904-triangle **Moai** mesh from [GLK16, Fig. 9]. We compute a cross field with additional hard constraints as described in § 2.5.4 to enforce alignment to the precomputed feature curves. We compare the resulting field with and without explicit feature-curve alignment in Figure 2-13. While the feature curves help guide the cross field, just a few artifacts in the computed features drastically influence the resulting cross field to have more singularities and be less smooth without clear benefit. The **Moai** is shown from an angle where these differences are most pronounced.

Effect of Mesh Resolution on Crease-Alignment vs Extrinsic Curvature. In this experiment we test on a geometry where a sharp crease is mis-aligned to extrinsic curvature directions. We generate meshes of this geometry at varying resolution to see how crease-alignment interacts with extrinsic curvature. Results of this experiment are depicted in Figure 2-10. As mesh resolution increases our cross fields become crease-aligned and intrinsically smooth, agreeing with the theory. For very low mesh resolution, the cross fields are more sensitive to the underlying meshing pattern.

Comparison to 3D Octahedral Fields. Due to similarity of frame representation, we compare our method to surface cross fields obtained by optimizing a volumetric octahedral field. Algorithms like those of [Hua+11; RSL16] can generate surface cross fields by approximating the surface with the limiting behavior of a thin layer of tetrahedra or prism elements. However, prism elements are non-standard and both element types will be poorly conditioned without introducing further restrictions such as zero normal gradient to mimic a triangle mesh. We instead opt to compare with the Boundary Element Method (BEM) [SVB17] which acts directly on surface triangle meshes. We use the 2500 triangle **fandisk** mesh for this comparison. As observed earlier, our method has increasing feature alignment with increased values of p. In comparison, Figure 2-14 shows that the BEM field fully ignores the shallow crease of the **fandisk**, running through it at a 45° offset. Moreover, despite the fact that the BEM only needs boundary data as input, its runtime is close to 50 times slower than ours.

Challenging Test Cases We compare feature alignment of our cross fields with that of existing methods on several meshes illustrative examples in Figure 2-16. As pointed out in the introduction, a key advantage of our technique is that it recovers crease aligned fields on models whose maximal curvature directions *disagree* with their creases. This occurs naturally when models are specified by the intersections of developable patches, a very common primitive in CAD tools. We introduce two benchmark models for testing crease alignment when creases disagree with intrinsic

notions of curvature. The **three-cylinder-intersection** mesh is composed of 12 quadrilateral patches where each patch is a subset of a cylinder and has maximal curvature directions making $\frac{\pi}{4}$ angles with its boundary creases. The **wavey-box** example has the same creases as a standard cube, with the modification that each of its faces has a sine wave ripple running diagonally through it. These two cases are shown in Figure 2-2. The **fandisk** mesh is another example of a challenging case for feature-alignment due to its shallow crease with strong non-aligning neighboring creases which is representative of one way that such features arise in real-world models.

Our cross fields on these test cases are shown in Figure 2-16. We observe proper feature alignment in our fields and while other methods can sometimes be tuned per model to achieve the same feature alignment, there is no choice of parameters that worked on all test cases. In particular:

- fields from [Jak+15] are distracted by extrinsic curvature on the **three-cylinder-intersection** and entirely pave over the shallow crease of the **fandisk**. Their results on **wavey-box**, and **wedge** are successfully aligned to the creases.;

- fields from [Bra+18] are challenging to tune with λ representing alignment to a guiding extrinsic curvature field. We show their method for the biharmonic energy ($m = 2$) as a point of contrast to Dirichlet energy. We choose two values of λ, $\lambda = -.0001$ for slight extrinsic curvature alignment, and $\lambda = -.1$ for stronger extrinsic curvature alignment. Their fields are unable to align to features of the **three-cylinder-intersection** in both cases, and specifically for $\lambda = -.1$ the field strongly aligns to noise on the flat upper face of the **fandisk** mesh. Their fields are successfully crease-aligned for the **wedge** mesh;

- we compare against both the anti-holomorphic and Dirichlet energies of [Knö+13] with the curvature alignment parameter λ set to -0.1. This results in good alignment on the **three-cylinder-intersection**, but noisy or unaligned fields for the remaining test cases.

In contrast, our method for $p = 2$ achieves feature alignment on all test cases without unnecessary discontinuities in the field over flat faces.

Quad Meshing Feature alignment is especially important when using cross fields to guide high-fidelity quad meshing. We generate quad meshes using [CBK15] to parameterize our cross fields. We compare against a standard quad meshing pipeline using cross fields from [BZK09] and [CBK15] for parameterization. We also test against parameterization by [Cam+16], which introduces extra guidance to encourage extrinsic curvature alignment.

For the **fandisk** mesh prior methods generate quad meshes that are influenced by the shallow crease, but do not manage to capture it sharply (see Figure 2-17). We observe that by placing singularities near the shallow crease of the **fandisk**, our quad meshes manage to align much more sharply. The quad mesh generated by minimizing E_∞ aligns even better than for E_2.

We also compare quad meshes generated from our cross fields against the prior art on the **anchor**, **spot**, **moomoo**, and **three-cylinder-intersection** meshes. These results are shown in Figure 2-18. We observe generally better alignment in the quad meshes generated from our method. By placing singularities on the cylindrical region of the **anchor**, our quad meshing manages to align better to its creases. On the **spot** mesh we see a straighter connection between the ear and the head. For the **three-cylinder-intersection**, the quad mesh generated from our fields clearly align better. Since the **moomoo** is a relatively smooth mesh, we do not see particularly defining differences in quality.

2.7 Discussion

Feature alignment is a desirable property in many geometry processing applications. In the context of cross fields and remeshing, we consider features to be creases where the surface changes non-smoothly. Quality of feature detection and alignment can significantly impact quality of the remeshing and the usefulness of the resulting cross

fields. While significant effort has been put into extrinsic alignment of cross fields to curvature directions, they are not always appropriate substitutes for crease alignment. By specifically targeting discontinuities of the surface we have created a new class of octahedral frame field energies parameterized by $p \geq 1$ for computing crease-aligned cross fields on surfaces. The resulting fields are intrinsically smooth over smooth surfaces, and can be used for crease-aligned quad meshing. Moreover, alignment is fully automatic and does not rely on explicit extraction of feature curves, itself an open problem and active area of research.

We find the behavior of E_p for $p \geq 2$ over creases of a discretization to be an interesting point for further exploration since all practical computations on surfaces are necessarily discrete and we observe strong feature alignment, despite the problem being ill-posed in the smooth setting. Theoretical analysis of anisotropic normally-aligned octahedral frame fields combined with Proposition 3 may be able to explain this behavior. Since all edges of a mesh are creases of a piecewise linear domain, the behavior of geometry processing algorithms on creased domains merits further study.

There are also further extensions of soft-normal-aligned octahedral frame fields. While in this paper we fix ϵ as a single parameter per mesh, it could also be defined as a scalar field representing "trust" in the quality of a mesh. It would be interesting to explore a spatially-varying ϵ dependent on triangle quality or other metrics in the future. If we treat the mesh itself as variables, soft normal alignment enables a surface flow towards meshes with lower cross-field energy. Our analysis can be further extended to SH representations of n-RoSy fields or even platonic solid symmetries [She+16].

Even without these extensions, our method provides a practical solution to a challenging problem. By pairing a new representation of cross fields with convex relaxation we achieve crease-aligned cross fields on surfaces.

2.8 Appendix: Proofs

2.8.1 Proof of Proposition 1

We carry over assumptions §4.1. While Equation (2.3) parameterizes the octahedral field by a single axis-angle rotation $v(r)$ per point $r \in \Omega$, for the case of a surface, it is more intuitive to decompose $v(r)$ into two rotations, one that accounts for intrinsic twisting, and one that accounts for the change in normal direction of the surface.

First, we set a local parametrization of the embedding, via $\exp|_{r^*} : T_{r^*}\Omega \to \Omega$. WLOG, we set coordinates (μ, ν) on $T_{r^*}\Omega$ such that axes μ and ν align with principal curvature directions (with $r^* = (0, 0)$ of course).

We split $v(\mu, \nu)$ into one rotation about the normal direction $\hat{z}$ by an angle amount $t(\mu, \nu)$, and a second rotation about a vector $k(\mu, \nu) = \hat{z} \times \hat{n}(\mu, \nu)$ by angle $\arccos(\hat{z} \cdot \hat{n}(\mu, \nu))$. Note that this means $k(\mu, \nu)_z = 0$. We can re-express $g(\mu, \nu)$ as the following

$$g(\mu, \nu) = e^{k(\mu,\nu)\cdot L} e^{[0,0,t(\mu,\nu)]\cdot L} g(r^*).$$

This can be interpreted as first applying a twisting rotation about the normal, followed by a normal adjustment rotation. This implies that similar to $v(\mu, \nu)$, $k(r^*) = [0, 0, 0]$ and $t(r^*) = 0$ as well. For shorthand, let $w(\mu, \nu) = k(\mu, \nu) + [0, 0, t(\mu, \nu)]$.

We can derive a very similar result to Equation (2.3) with the following changes.

$$\nabla g(\mu, \nu)|_{r^*} = \begin{bmatrix} | & | & | \\ L_x g(r^*) & L_y g(r^*) & L_z g(r^*) \\ | & | & | \end{bmatrix} \begin{bmatrix} | & | \\ \nabla_\mu w & \nabla_\nu w \\ | & | \end{bmatrix}_{r^*} \tag{2.11}$$

This implies that our quantity of interest the squared norm $\|\nabla g(\mu, \nu)\|^2$ at r^* is

$$\begin{aligned} \|\nabla g(\mu, \nu)\|^2 &= \operatorname{Tr}\left[\begin{bmatrix} | & | \\ \nabla_\mu w & \nabla_\nu w \\ | & | \end{bmatrix}^T \frac{20}{3} I_3 \begin{bmatrix} | & | \\ \nabla_\mu w & \nabla_\nu w \\ | & | \end{bmatrix} \right] \\ &= \frac{20}{3} \operatorname{Tr}\left[(\nabla w)^T \nabla w \right] \end{aligned} \tag{2.12}$$

The $\frac{20}{3}I_3$ comes from the fact that $gL_i^T L_j g = \frac{20}{3}\delta_{ij}$ for $i, j \in x, y, z$ and any $g \in \mathcal{V}$[PBS19, Equation 3] . We now expand the expression $\|\nabla w(\mu,\nu)\|_F^2$ as $|\nabla k(\mu,\nu)_x|_2^2 + |\nabla k(\mu,\nu)_y|_2^2 + |\nabla t(\mu,\nu)|_2^2$.

Since $k(\mu,\nu)$ is an axis-angle rotation describing the change in normal from $\hat{n}(r^*)$ to $\hat{n}(\mu,\nu)$, we recognize its relationship to the principal curvatures. As μ and ν were chosen to align with principal curvature directions, $\frac{\partial k_x}{\partial \mu} = 0$, $\frac{\partial k_y}{\partial \nu} = 0$, $\frac{\partial k_x}{\partial \nu} = k_1$, $\frac{\partial k_y}{\partial \mu} = k_2$, where k_1 and k_2 are the principal curvatures of Ω at r^*. Finally denote t_μ and t_ν to be $\frac{\partial t}{\partial \mu}$ and $\frac{\partial t}{\partial \nu}$ respectively. We are left with $\|\nabla g(\mu,\nu)\|_F^2 = k_1^2 + k_2^2 + t_\mu^2 + t_\nu^2$.

Substituting in w for $t_\mu^2 + t_\nu^2$, and re-expressing with Gauss and mean curvatures, K and H, we obtain

$$\|\nabla g(\mu,\nu)\|_F^2 = k_1^2 + k_2^2 + w = 2H^2 - K + w.$$

Recall that $t(r^*) = 0$. This motivates our view of w as tangential intrinsic twisting.

2.8.2 Proof of Proposition 2

For more compact notation, let $\Phi := \{\phi \mid \forall_x |\phi(x)|_F \leq 1, \phi \in C_c^1\}$. Starting from Equation (2), we may split the term into integrals over the patches Ω_j:

$$\begin{aligned}
VTV[f] &= \sup_{\Phi} \sum_{i=1}^{m} \int_{\Omega} f_i \nabla \cdot \phi_i \\
&= \sup_{\Phi} \sum_{i=1}^{m} \sum_{j=1}^{n} \int_{\Omega_j} \tilde{f}_i^j \nabla \cdot \phi_i \\
&= \sup_{\Phi} \sum_{i=1}^{m} \sum_{j=1}^{n} \left(\int_{\partial\Omega_j} \tilde{f}_i^j \phi_i \cdot \hat{n} - \int_{\mathring{\Omega}_j} \nabla f_i \cdot \phi_i \right) \\
&= \sup_{\Phi} \sum_{j=1}^{n} \left(\left(\sum_{i=1}^{m} \int_{\partial\Omega_j} \tilde{f}_i^j \phi_i \cdot \hat{n} \right) - \left(\sum_{i=1}^{m} \int_{\mathring{\Omega}_j} \nabla f_i \cdot \phi_i \right) \right).
\end{aligned}$$

Here, we've applied integration-by-parts and Stokes' theorem, and switched sum orders for simpler argument below.

Let us consider the pointwise maxima over Ω for the integrands. For the in-

tegrals over $\mathring{\Omega}_j$, they are maximized for ϕ set to the negated, normalized $\nabla f = (\nabla f_1, \ldots, \nabla f_m)^T$. This results in an integrand value of $|\nabla f|_F$ pointwise.

The integrals over $\partial\Omega_j$ may be considered as integrals over Γ, the curve network separating the patches $\mathring{\Omega}_j$. For all but a finite set, points of Γ lie on the interior of a curve γ_k and separate the neighborhood into patches. For each γ_k, choose one patch and let f^+ denote the limiting value from this side, and let f^- denote the other limiting value. The maximizing ϕ will be such that $\phi_i = ((f^+ - f^-)_i / |f^+ - f^-|_2)\, \hat{n}^+$ where $\hat{n}^+$ is the unit normal to the $+$ patch. This results in an net integrand value of $|f^+ - f^-|_2$ once over the total curve network Γ.

The final step is to construct a sequence of differentiable ϕ's that converge pointwise to these optimal values, achieving the maximum in the limit and proving our theorem. This can be done with partitions of unity, subordinate to a constructed cover. For simplicity, we assume that $\nabla f \neq 0$ on $\mathring{\Omega}_j$ and $f^+ - f^- \neq 0$ on γ_k. In this case, consider an $\epsilon > 0$ and the open cover consisting of n sets $U_j^\epsilon := \{x | x \in \Omega_j, d(x, \partial\Omega_j) > \epsilon\}$ and s sets $V_k^\epsilon := \{x | d(x, \gamma_k) < 2\epsilon\}$.

On the sets U_j^ϵ, we multiply the associated partition function by the negated normalized ∇f. On the sets V_k^ϵ, we multiply the associated partition function by any differentiable extension of $\phi|_{\gamma_k} = (\ldots, \phi_i, \ldots)$ where $\phi_i = ((f^+ - f^-)_i / |f^+ - f^-|_2)\, \hat{n}^+$. The sum of these is a differentiable ϕ that approaches the pointwise maximizers as $\epsilon \to 0$. In the case when ∇f or $f^+ - f^-$ vanishes, note that the value of ϕ is irrelevant at those points, and a slight modification of this argument with additional open covers (about the vanishing regions) will provide the result.

2.8.3 Proof for Proposition 3

First we derive an expression for difference between two normally aligned octahedral frames f_0 and f_1 across a crease. Let f_0 be the canonical frame, and $f_1 = e^{b[\cos a, \sin a, 0]\cdot L} e^{tL_z} f_0$. The angle t represents an initial twist in the $\hat{z}$ direction, and the angle b represents a bend across a crease in the xy-plane. The crease direction is described by angle a relative to $\hat{x}$. The difference between these two octahedral frames is then $E(a, b, t) = |f_0 - e^{b[\cos a, \sin a, 0]\cdot L} e^{tL_z} f_0|_2^2$.

Consider the case where $a = 0$, we can interpret f_0 as an octahedral frame aligned to the normal $\hat{z}$ and the crease direction $\hat{x}$. f_1 is then the octahedral frame on the other side of a crease of angle $\pi - b$. t describes how misaligned f_1 is to the crease direction $\hat{x}$. The cost of deviating from crease-alignment is then $E(0, b, t) - E(0, b, 0)$.

Now consider cases where $a \neq 0$. Since a describes the direction of the crease, its deviation from 0 corresponds to deviation of f_0 from crease alignment. To prove that cost is minimized by crease alignment, it suffices to show that $E(a, b, t) - E(0, b, 0)$ is non-negative for all values of a,b, and t.

We will first derive an expression for $E(a, b, t)$ without matrix exponentials and then use Mathematica code in § 2.8.4 to prove

$$E(0, b, t) - E(0, b, 0) = \frac{5}{24}(7 + \cos 4b)\ \sin^2 2t \geq 0,$$

and that $E(a, b, t) - E(0, b, 0)$ is non-negative for all values of a, b, and t. The reason we need to first derive a form without matrix exponentials is so that Mathematica has an easier time manipulating our equations of interest. With matrix exponentials, Mathematica often hangs and is unable to output a result.

The derivation for an expression for $E(a, b, t)$ without matrix exponentials is as follows. First we split an octahedral frame f_0 into its three lobes. $f_0 = l_x + l_y + l_z$, where $l_z = [0, 0, 0, 0, \sqrt{\frac{7}{12}}, 0, 0, 0, 0]$, $l_x = e^{\frac{\pi}{2}L_x} l_z$ and $l_y = e^{\frac{\pi}{2}L_y} l_z$. We can therefore compute $E(a, b, t)$ as follows.

$$\begin{aligned} E(a, b, t) = & \quad |f_0 - e^{b[\cos a, \sin a, 0]\cdot L} e^{tL_z} f_0|_2^2 \\ = & \quad |(l_x + l_y + l_z) - e^{b[\cos a, \sin a, 0]\cdot L} e^{tL_z} (l_x + l_y + l_z)|_2^2 \end{aligned}$$

Denote the rotation $e^{b[\cos a, \sin a, 0]\cdot L} e^{tL_z}$ by $R(a, b, t)$. This results in

$$\begin{aligned} E(a, b, t) = & \quad |(l_x + l_y + l_z) - R(a, b, t) l_x - R(a, b, t) l_y - R(a, b, t) l_z)|_2^2 \\ = & \quad |l_x + l_y + l_z - R(a, b, t) l_x - R(a, b, t) l_y - R(a, b, t) l_z)|_2^2 \end{aligned}$$

The above expression is be composed of many terms of the following two forms $l_{d_1}^T R(a, b, t) l_{d_2}$ for $d \in \{x, y, z\}$ and $|l_{d_1} - R(a, b, t) l_{d_2}|$. Note that both these expressions

are variables depending on the angle between two lobes. We therefore compute $|l_{v_1} - l_{v_2}|_2^2$ and $l_{v_1}^T l_{v_2}$ in Mathematica, where l_v is a lobe oriented in the v direction as a function of the angle θ between v_1, v_2. We obtain

$$|l_{v_1} - l_{v_2}|_2^2 = \frac{5}{42}(9 + 7\cos 2\theta)\sin^2\theta$$

and

$$l_{v_1}^T l_{v_2} = \frac{1}{336}(9 + 20\cos 2\theta + 35\cos 4\theta).$$

In the notebook these are denoted by "d2t[a]" and "kt[a]" respectively, where "a" in the notebook represents θ. Finally we substitute these terms into $E(a, b, t)$. The final expression is derived by the Mathematica code in § 2.8.4.

2.8.4 Feature Alignment Proof Notebook

```
In[60]:= (*First we set up the E(a,b,t) function using helpers d2t \
and kt. d2t is |l1-l2|^2 and kt is l1'l2*)

In[61]:= n = {0, 0, 1}; t1 = {1, 0, 0}; t2 = {0, 1, 0};
R12 = {{0, -1, 0}, {1, 0, 0}, {0, 0, 1}}; Rn1 = {{0, 0, 1}, {0, 1,
   0}, {-1, 0, 0}}; Rn2 = {{1, 0, 0}, {0, 0, 1}, {0, -1, 0}};

d2t[theta_] := 35/96*(9 + 7*Cos[2*theta])*Sin[theta]^2*(16/49);

kt[theta_] := 7/768*(9 + 20*Cos[2*theta] + 35*Cos[4*theta])*(16/49);

v2r[v_] :=
  MatrixExp[{{0, -v[[3]], v[[2]]}, {v[[3]], 0, -v[[1]]}, {-v[[2]],
     v[[1]], 0}}];

R[eangle_, bend_, twist_] :=
```

```
  FullSimplify[
   ComplexExpand[
    v2r[(Cos[eangle]*t1 + Sin[eangle]*t2)*bend].v2r[n*twist]]];

R2A[R_] := ArcCos[(Simplify[TrigExpand[Tr[R]]] - 1)/2];

Energy[R_] :=
  d2t[ArcCos[t1.R.t1]] + d2t[ArcCos[t2.R.t2]] + d2t[ArcCos[n.R.n]] +
   12*kt[Pi/2] - 2*kt[ArcCos[t1.R.R12.t1]] -
   2*kt[ArcCos[t2.R.Transpose[R12].t2]] - 2*kt[ArcCos[n.R.Rn1.n]] -
   2*kt[ArcCos[n.R.Rn2.n]] - 2*kt[ArcCos[t1.R.Transpose[Rn1].t1]] -
   2*kt[ArcCos[t2.R.Transpose[Rn2].t2]];

Energy[eangle_, bend_, twist_] :=
  Simplify[TrigExpand[Energy[R[eangle, bend, twist]]]];

In[70]:= (*Final expression of the energy*)

In[71]:= Energy[a, b, t]

Out[71]= -(1/1536)
 5 (-564 + 84 Cos[4 a] - 56 Cos[4 a - 2 b] + 14 Cos[4 (a - b)] +
    112 Cos[2 b] + 196 Cos[4 b] + 14 Cos[4 (a + b)] -
    56 Cos[2 (2 a + b)] + 70 Cos[8 a - 4 t] -
    8 Cos[8 a - 3 b - 4 t] - 56 Cos[4 a - 2 b - 4 t] +
    28 Cos[8 a - 2 b - 4 t] - 56 Cos[8 a - b - 4 t] +
    56 Cos[b - 4 t] - 56 Cos[8 a + b - 4 t] + 8 Cos[3 b - 4 t] -
    8 Cos[8 a + 3 b - 4 t] + 28 Cos[2 (b - 2 t)] -
    56 Cos[2 (2 a + b - 2 t)] + 28 Cos[2 (4 a + b - 2 t)] +
    84 Cos[4 (a - t)] + 14 Cos[4 (a - b - t)] + Cos[4 (b - t)] +
```

```
    14 Cos[4 (a + b - t)] + Cos[4 (2 a + b - t)] + 70 Cos[4 t] +
    Cos[4 (b + t)] + 28 Cos[2 (b + 2 t)] + 56 Cos[b + 4 t] +
    8 Cos[3 b + 4 t] + Cos[8 a - 4 (b + t)])

In[72]:= (*This is the one-sided cost of deviating from \
crease-alignment*)

In[73]:= ECt = Energy[0, b, t] - Energy[0, b, 0] // Simplify

Out[73]= 5/24 (7 + Cos[4 b]) Sin[2 t]^2

In[74]:= (* We look for and find no solutions where E(a,b,t)-E(0,b,0) \
is negative *)

In[75]:= diff = Energy[a, b, t] - Energy[0, b, 0];
diffSimp = diff // FullSimplify;
Solve[diffSimp < 0, {a, b, t}]

{}
```

2.8.5 Angular Momentum Operators

$$L_x = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\sqrt{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\sqrt{\frac{7}{2}} & 0 & -\sqrt{2} \\ 0 & 0 & 0 & 0 & 0 & -\frac{3}{\sqrt{2}} & 0 & -\sqrt{\frac{7}{2}} & 0 \\ 0 & 0 & 0 & 0 & -\sqrt{10} & 0 & -\frac{3}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 0 & \sqrt{10} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{3}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sqrt{\frac{7}{2}} & 0 & \frac{3}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 \\ \sqrt{2} & 0 & \sqrt{\frac{7}{2}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sqrt{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$L_y = \begin{pmatrix} 0 & \sqrt{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\sqrt{2} & 0 & \sqrt{\frac{7}{2}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\sqrt{\frac{7}{2}} & 0 & \frac{3}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{3}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\sqrt{10} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sqrt{10} & 0 & -\frac{3}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{3}{\sqrt{2}} & 0 & -\sqrt{\frac{7}{2}} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \sqrt{\frac{7}{2}} & 0 & -\sqrt{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \sqrt{2} & 0 \end{pmatrix}$$

$$L_z = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

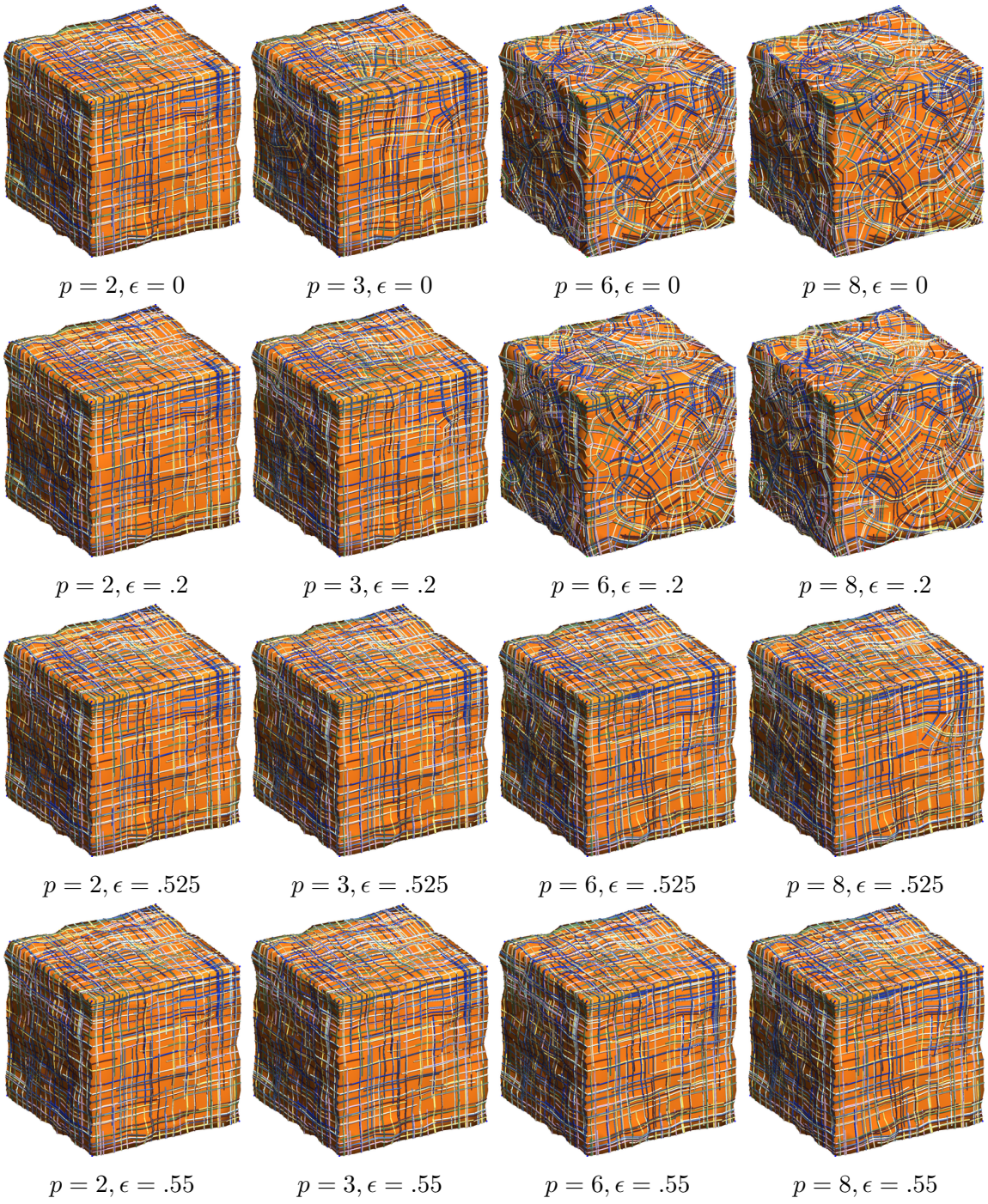


Figure 2-9: As ϵ increases or as p decreases, the cross fields become less sensitive to noise added to the **cube-mesh**.

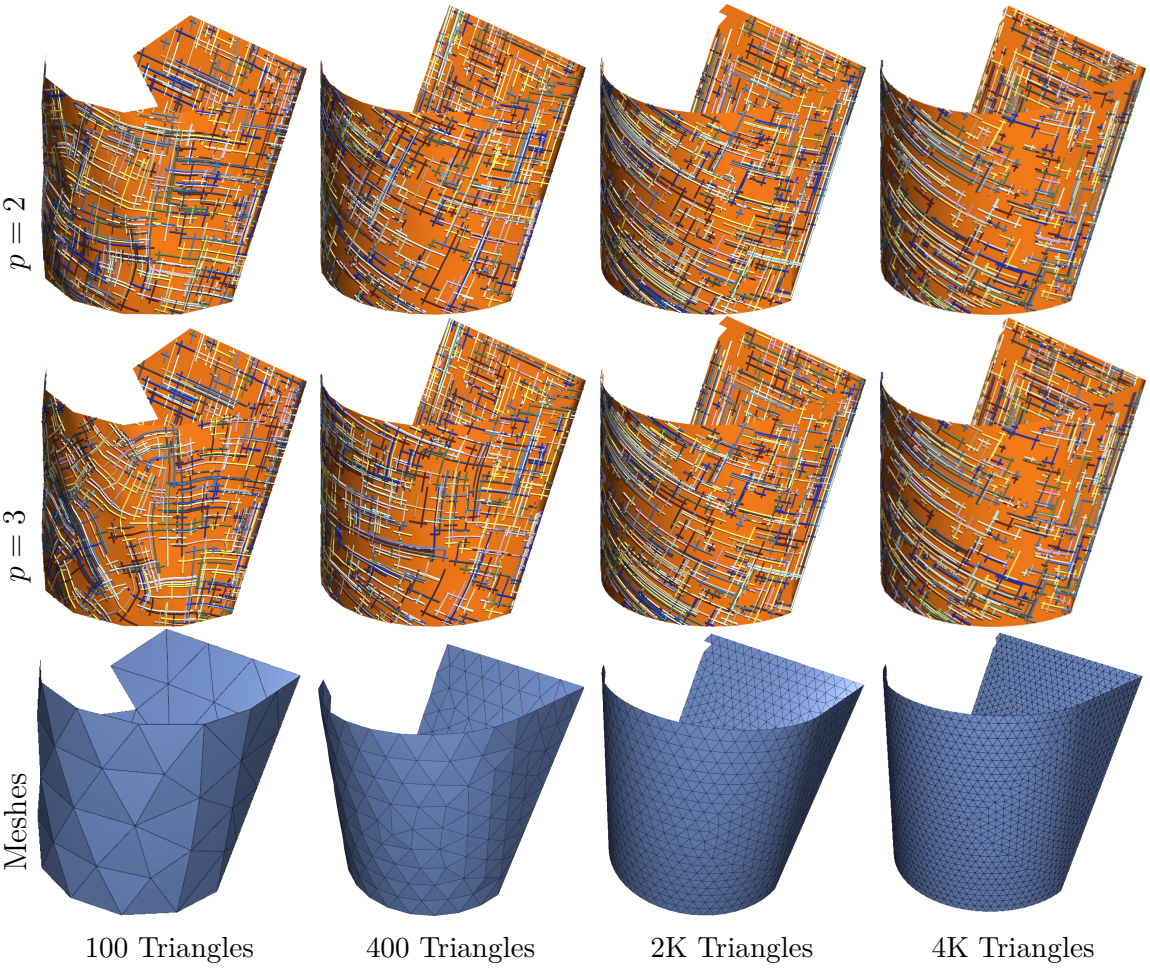
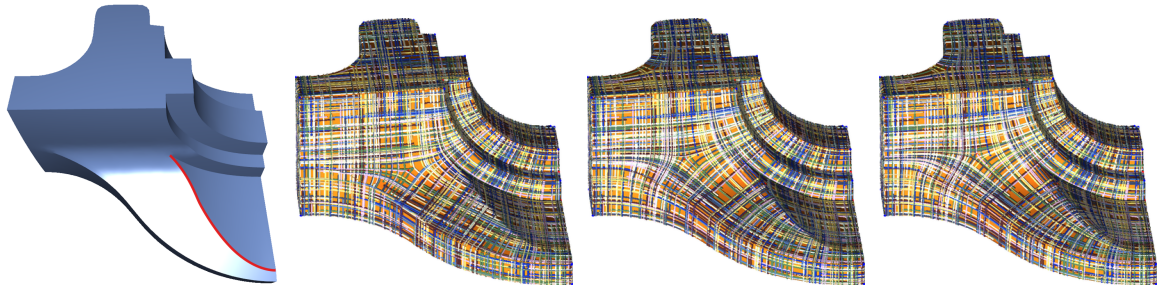


Figure 2-10: On this developable surface, our cross fields are intrinsically smooth in the limit of refinement, but exhibit some mesh sensitivity on coarse meshes, particularly for higher p values. They are crease-aligned for all resolutions. Note that the extrinsic curvature of the cylindrical bend has no effect on the cross fields at higher resolutions.



(a) Fandisk mesh (b) $p = 1$ (c) $p = 2$ (d) $p = \infty$

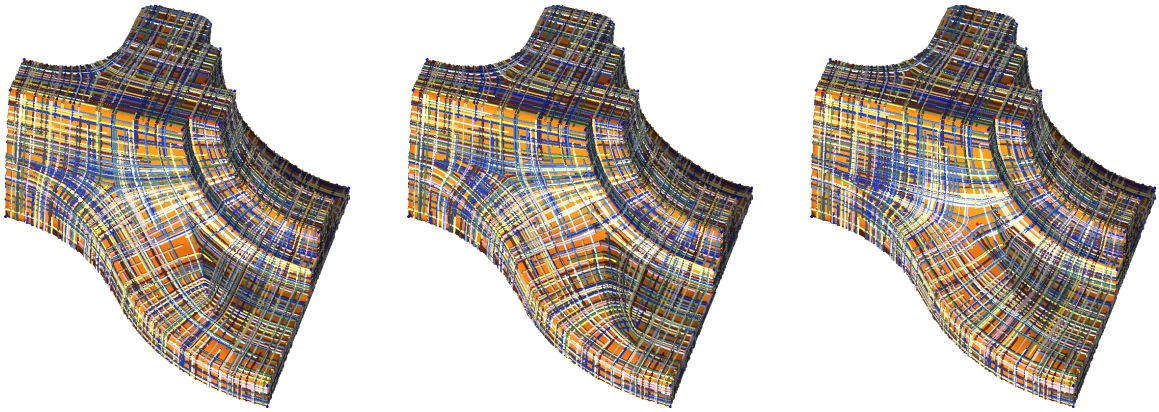
Figure 2-11: Cross fields generated by minimizing E_p for $p = 1, 2, \infty$ on the **fandisk** mesh. The shallow crease of the **fandisk** mesh is marked in red. Our cross fields naturally align to the shallow crease with increasing strength for higher p.

(a) 1.3k (b) 5.4k (c) 21.8k (d) 11.6k (e) Multires

Figure 2-12: Cross fields generated by minimizing E_2 on different meshings of the **three-cylinder-intersection** with number of faces indicated. Cross field (d) is computed on the multi-resolution mesh (e). Notice that we obtain the same feature-aligned cross field each time.

(a) **Moai** and explicit feature curves(red edges) (b) Explicit Feature Curve Alignment: $p = 2, \epsilon = 0$ (c) Our method: $p = 2$, $\epsilon = 0$

Figure 2-13: Comparison of our feature-aligned cross fields to those generated when adding additional explicit feature curve alignment constraints. Explicit feature curves were obtained from [GLK16, Fig. 9] Despite the extra cost of precomputing explicit feature curves, slight artifacts in the feature curves (most pronounced on the side) force the explicitly guided cross field to have lower quality.



(a) Ours $p = 2$, $\epsilon = 0$: 3.9s (b) Ours $p = \infty$, $\epsilon = 0$: 3.5s (c) BEM: 161s

Figure 2-14: Cross field and runtime comparison of our method to a method optimizing volumetric octahedral frames [SVB17]. The **fandisk** used contains 2.5k triangles.

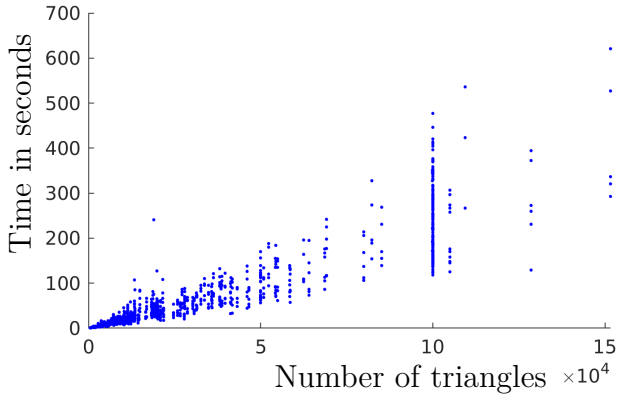


Figure 2-15: Runtimes to compute cross fields over various mesh sizes.

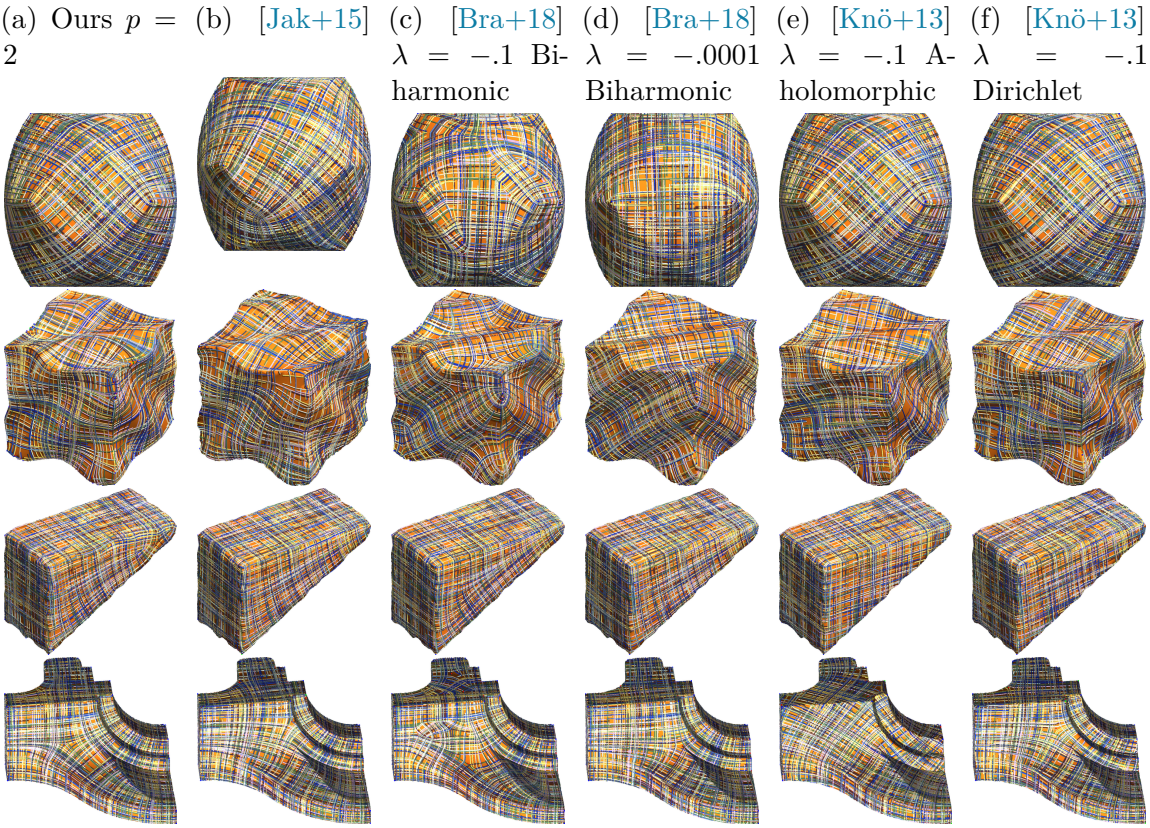
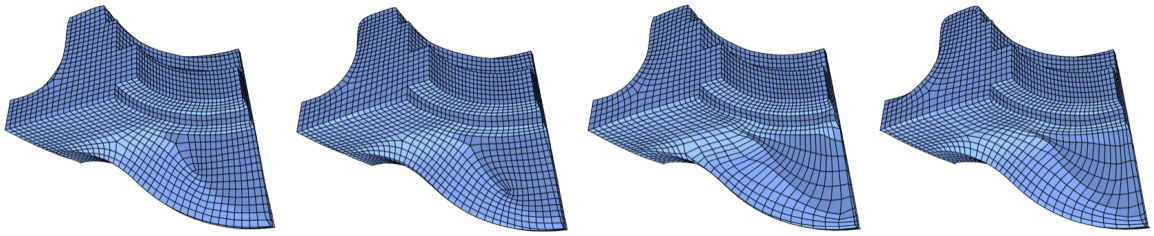
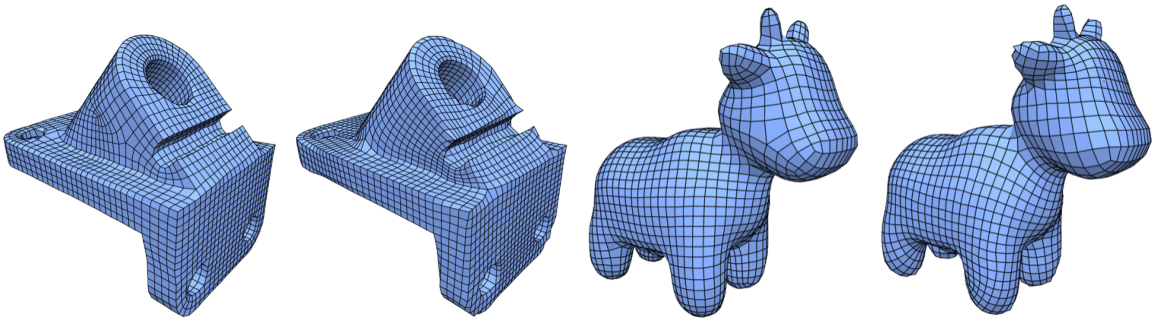


Figure 2-16: Various cross field methods compared on several meshes with complex features and geometry. We test on the three-cylinder-intersection, wavey-box, wedge, and fandisk meshes and compare against the following works: [Jak+15; Bra+18; Knö+13] with various parameters. We use normal aligned octahedral fields generated by minimizing E_2. We achieve crease-alignment on all test cases where other methods succeed sporadically.

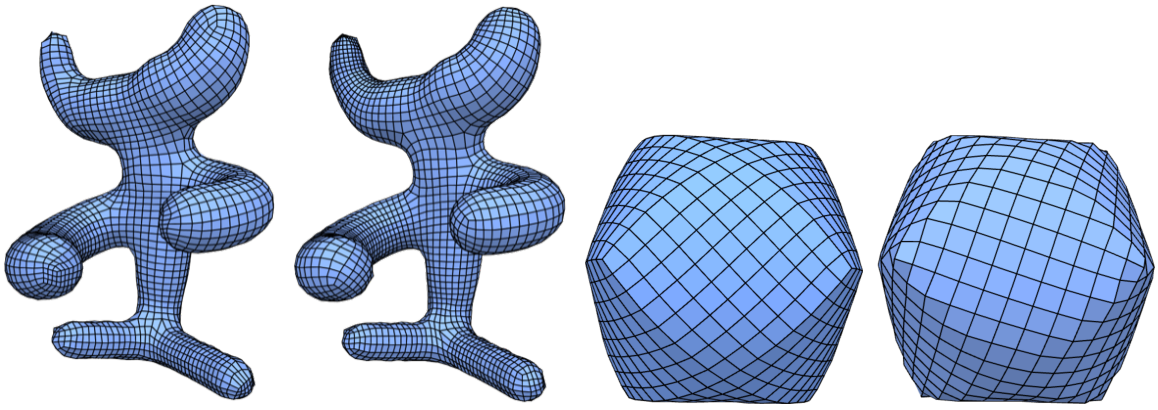


(a) E_2 (b) E_∞ (c) QGP (d) Curvature filter

Figure 2-17: Quad meshes of the **fandisk** mesh generated using cross fields from E_2, E_∞, MIQ + QGP [CBK15], and MIQ + Curvature filter [Cam+16] respectively. Our methods achieve sharp alignment to the shallow crease with increased depth for higher p. Alternative methods are influenced by the crease only to a shallower extent.



(a) Anchor Mesh with E_2 (b) Anchor Mesh with [BZK09; CBK15] (c) Spot Mesh with E_2 (d) Spot Mesh with [BZK09; CBK15]



(e) Moomoo Mesh with E_2 (f) Moomoo Mesh with [BZK09; CBK15] (g) 3 Cylinder Intersection with E_2 (h) 3 Cylinder Intersection with [BZK09; CBK15]

Figure 2-18: Quad meshes of the **anchor**, **spot**, **moomoo**, and **three-cylinder-intersection** meshes. We compare quad meshes generated using cross fields from our E_2 energy with quad meshes generated through [CBK15] and [BZK09]. Our methods achieve sharper feature alignment on the **anchor**, **spot** (on the ear), and **three-cylinder-intersection** meshes.

3

Hexahedral Quality Enhancement with Variable Augmentation

This chapter is based on work in [Zha+23] and focuses on topological modifications of hexahedral meshes. The introduction has been re-arranged to emphasize mesh quality and how it is affected by singularities. A greater emphasis is also put on the interpretation of sheet inflation as variable augmentation in the scaled Jacobian optimization. Lastly, we extrapolate from our new understanding of the formation of singular nodes to infer that Dirichlet energy is a poor objective function for octahedral field computation.

3.1 Introduction

Hexahedral (hex) meshing is a long studied topic in geometry processing with many challenging associated problems. In essence, the goal is to tessellate a volume with minimally distorted cubes (hexahedra). This distortion is scored by a geometric measure, the scaled Jacobian [Qua21]. On the topological side, hex meshes can vary from structured to unstructured. Fully structured meshes require that all interior mesh edges be adjacent to four hexes each. Edges failing this criteria are singular and indicate an unstructured hex mesh. Singular edges join together into singular curves that can collide to form singular nodes, a complex junction of multiple singular curves.

Unsurprisingly, the topology and geometry of hex meshes are intertwined; Hex meshes with more complex singular nodes are forced to have more distorted elements with smaller scaled Jacobian scores.

In this chapter, we study topological modifications of hex meshes to increase the scaled Jacobian. We find that the prescence of singular nodes significantly hinders the quality of a hex mesh. Fortuitously, we discover that all eight of the most common singular nodes are also decomposable into just singular curves through topological modifications of the mesh. We further show that all singular nodes, regardless of edge valence, are locally decomposable. Finally we demonstrate these decompositions on hex meshes, thereby decreasing their distortion and converting all singular nodes into singular curves. With this decomposition, the enigmatic complexity of 3D singular nodes becomes effectively 2D.

Our contributions are as follows:

- We show by construction that all eight of the most practically relevant singular nodes are decomposable into just singular curves.
- We show that all singular nodes, regardless of valence, are locally decomposable.
- We apply our decompositions to hex meshes demonstrating that entire singular graphs can be separated into independent singular curves.
- We leverage these topological decompositions to increase the geometric quality of hexahedral meshes

3.2 Preliminaries

Our exploration of singular nodes is motivated by the following question. What if a singular node is formed when singular curves just barely graze each other as they pass by? If that were the case, then we could separate the curves with a sheet and increase its thickness to force the curves away from each other, thereby unmaking the singular node. To formalize this idea, we begin with the following definitions.

3.2.1 Singular Vertices, Curves, and Nodes

We denote a hex mesh as $\{\mathcal{V}, \mathcal{H}\}$ where $\mathcal{V}$ is a list of vertices embedding the mesh and $\mathcal{H}$ is a list of hexes of the mesh. Let $\mathcal{F}$ denote the quadrilateral faces of the mesh, $\mathcal{E}$ denote the edges of the mesh, and $\deg(e \in \mathcal{E})$ denote the number of hexes adjacent to edge e, i.e., its degree or valence. A *regular edge* is an interior edge e satisfying $\deg(e) = 4$. All other interior edges are *singular edges*. We will not address cases where $\deg(e) \leq 2$ since these are not typically accepted as valid hex meshes. We will also not consider singular boundary edges in this paper but refer the interested reader to [Liu+18] for the corresponding definition. For our purposes, one can ignore boundary singularities or push them all to the interior by adding one layer of padding to the hex mesh boundary. A *singular vertex* is a vertex of the mesh that is adjacent to any singular edge. A *singular node* is a vertex of the mesh that is adjacent to more than two singular edges. A *singular curve* is an alternating sequence of singular edges and singular vertices that either forms closed cycles or, ends at a boundary vertex or a singular node. Note that we have deviated from the language of [Liu+18] by distinguishing singular nodes from singular vertices. Singular nodes are reserved for the junctions of multiple singular curves and will be the primary focus of this work. Figure 3-1 depicts a summary of this terminology.

For a singular node $v \in \mathcal{V}$, let $\mathcal{T}(v)$ be the triangle mesh in bijection with that node according to [NRP11b]. This bijection is formed by intersecting the singular node of the hex mesh with an infinitesimally small sphere. Since the intersection of a corner of a hex with a sphere forms a triangle, the hexes adjacent to the singular node partition the sphere into triangular regions thus forming a sphere triangulation. Triangulations formed this way are naturally regular i.e. they contain no loop edges or multi-edges. This is depicted on the right of Figure 3-1. The sphere triangulation encodes the *singular node type*. Two isomorphic sphere triangulations encode singular nodes of the same type. The *signature* of a singular node is a list of numbers indicating how many of its adjacent singular edges are of each degree. Since singular edges have degree 3 or higher, the signature of a node starts with the number of edges with

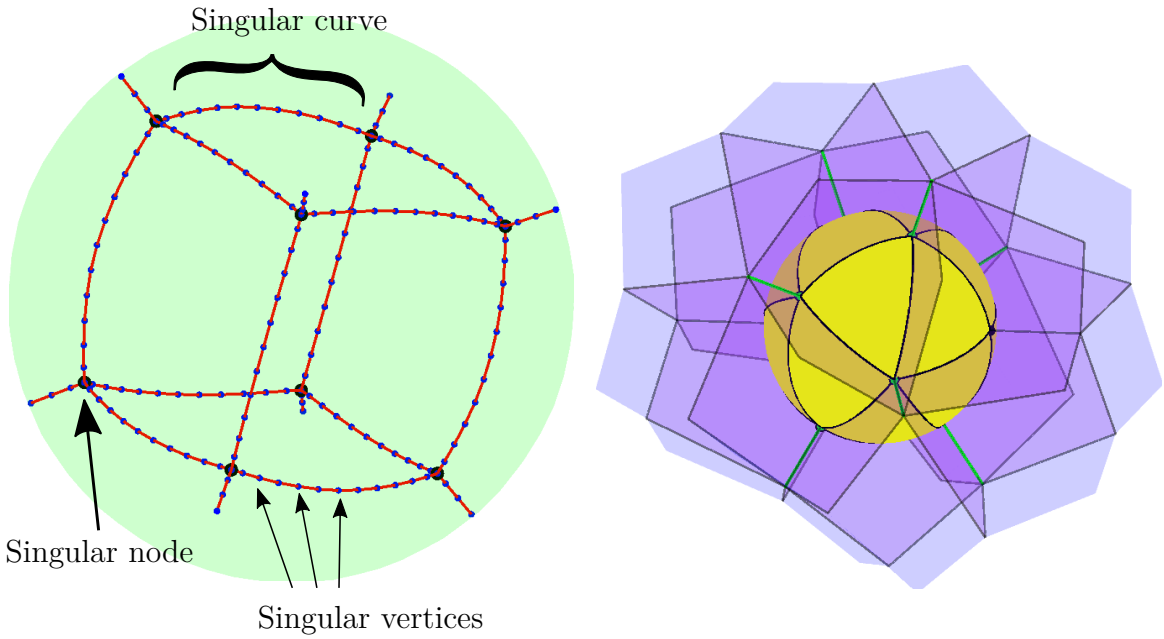


Figure 3-1: (Left) We show the singular graph of a hex mesh of a sphere. Singular edges are red, singular nodes are large black circles and singular vertices are small blue vertices. (Right) A close-up view of one singular node. Faces adjacent to the singular node are displayed in purple. A yellow sphere is overlayed on top of the singular node. Its intersection with the hex mesh partitions the sphere into triangular regions thus forming a sphere triangulation.

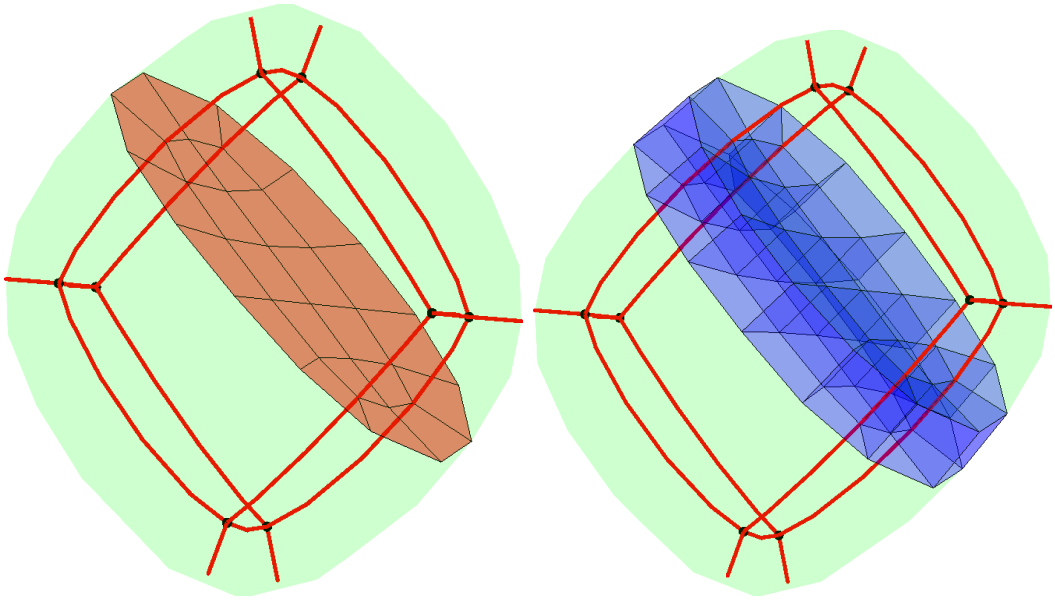


Figure 3-2: (Left) Red quads indicate a sheet inside of a hex mesh of an ellipsoid. Red curves depict the singular graph of that ellipsoid. The sheet is manifold with boundary on the boundary of the hex mesh. (Right) Blue quads indicate all faces of the inflated sheet.

degree 3. For singular nodes whose adjacent singular edges have only valence 3, 4, or 5, the signature uniquely identifies the singular node type [Liu+18]. For this reason we identify singular nodes by their signature e.g. $(4, 0, 0)$ is the signature identifying the singular node type generated by subdividing a single tetrahedron into four hexes as illustrated in the first image of Figure 3-4.

3.2.2 Sheet Inflation

Let a *sheet* $\mathbb{Q} \subset \mathcal{F}$ be a manifold quad mesh whose boundary is a subset of the boundary of $\mathcal{H}$. A *sheet inflation* of $\mathbb{Q}$ is a mesh modifying operation by which each $q \in \mathbb{Q}$ is inflated from a 2D quad face to a hex element [LS10]. One such sheet inflation is depicted in Figure 3-2. If the inflated sheet passes through a singular node, the singular type of that node may change. One can interpret this as the sheet slipping through an infinitesimally small gap between singular curves and forcing them apart, or alternatively as cutting a singular node into separate pieces.

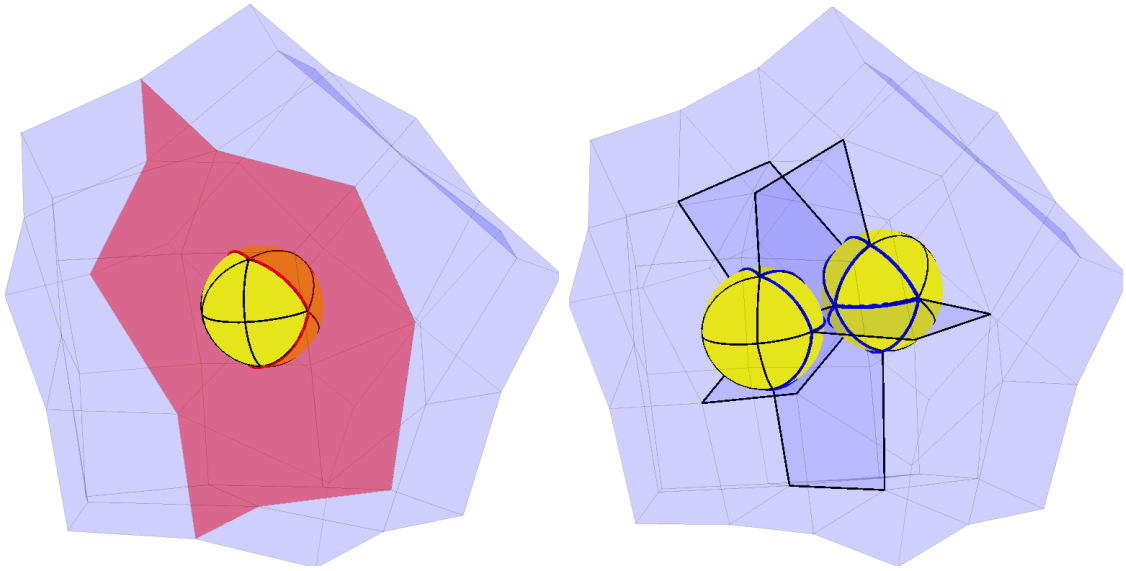


Figure 3-3: (Left) The yellow sphere triangulation indicates the structure of a singular node. Red quads indicate a sheet intersecting the singular node. The sheet intersects the sphere triangulation on a cycle of red curves that divide the sphere triangulation into two disks. (Right) Two singular nodes are visualized post sheet inflation with their sphere triangulations. Blue quad faces are newly created faces from the inflation. Blue edges indicate newly created edges in each sphere triangulation.

3.2.3 Effect of Sheet Inflation on $\mathcal{T}(v)$

As singular nodes are often represented and visualized as sphere triangulations we describe here how sheet inflation through a singular node corresponds to a *splitting* of the sphere triangulation. This splitting is illustrated by Figure 3-3. Given a sheet $\mathbb{Q} \subset \mathcal{F}$ passing through singular node v, the faces of $\mathbb{Q}$ adjacent to v correspond to edges of $\mathcal{T}(v)$. These edges trace out a cycle in the graph of $\mathcal{T}(v)$ partitioning the sphere into two disk triangulations $\mathcal{D}_1$ and $\mathcal{D}_2$. When the sheet is inflated into a layer of hexes, $\mathcal{D}_1$ and $\mathcal{D}_2$ are separated. Each disk is restored back into sphere triangulations by attaching triangles from the boundary of each disk to a respective new vertex. The end result is two sphere triangulations one built from $\mathcal{D}_1$ and one built from $\mathcal{D}_2$. These sphere triangulations track the modifications to v on either side of the inflated sheet and are both regular since they are still in bijection with hex mesh nodes.

3.3 Singular Node Decomposition (Valences 3, 4, and 5)

We are now equipped with the mechanism by which all practically relevant singular nodes are decomposed into singular curves. These nodes are enumerated in [Liu+18, Figure 6] and will also be shown at the beginning of each respective decomposition. Various singular decompositions will be depicted throughout the remainder of this paper. Valence 3 singular curves will be red, valence 5 singular curves will be green and higher valence singular curves will be blue. We omit drawing interior regular edges to minimize visual clutter.

In the first row of Figure 3-4 we start with the (4,0,0) singular node, which consists of four valence 3 singular curves joined at a junction. With a single sheet inflation, this singular node is revealed to actually be two valence 3 singular curves that pass each other transversally. A similar theme follows for (2,2,2) in the second row which is revealed to be a valence 3 and a valence 5 singular curve passing each other transversally. Finally in the third row, the (0,4,4) node decomposes into two valence 5 singular curves passing each other transversally.

From these three examples, it is tempting to think all singular nodes may consist of singular curves glued together transversally. One might conclude as well that the number of singular curves of a particular valence meeting at a singular node from this construction must be even. The $(1, 3, 3)$ singular node, shown in Figure 3-5, presents a curious counterexample. Since it consists of one valence 3 singular curve and three valence 5 singular curves, it is impossible to decompose this node into just two singular curves passing each other transversally in a valid hex mesh.

This conundrum is resolved by realizing that one of the regular edges adjacent to this singular node is actually a pair of valence 3 and valence 5 singular curves, glued together in parallel. Another way to understand this node is that one valence 5 singular curve has split an otherwise parallel pair of valence 3 and 5 curves. The decomposition of this node into one valence 3 and two valence 5 singular curves is

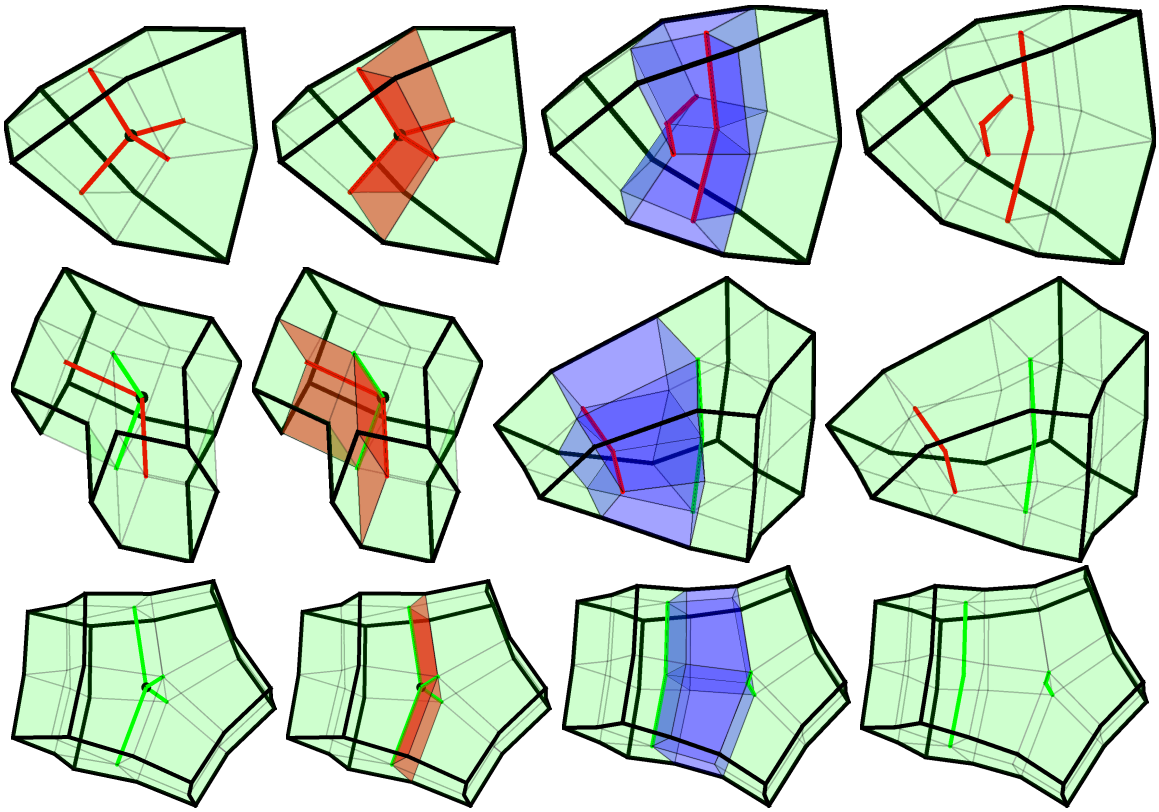


Figure 3-4: From top to bottom the (4,0,0), (2,2,2), and (0,4,4) singular nodes are depicted. Left to right indicates steps to decompose each singular node. Red quads indicate the sheet to be inflated. Blue quads indicate faces of the newly inflated hexes. One sheet inflation is sufficient to decompose each of these nodes.

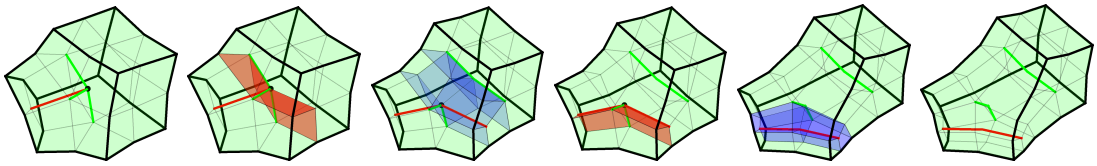


Figure 3-5: The (1,3,3) singular node is decomposed into two valence 5 and one valence 3 curve via two sheet inflations.

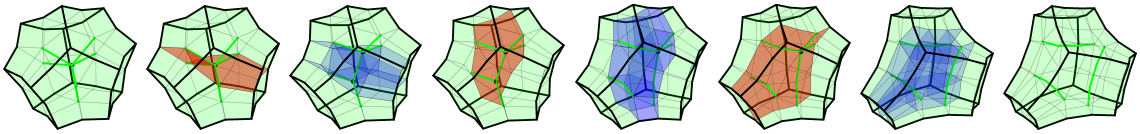


Figure 3-6: The (0,3,6) singular node is decomposed into three valence 5 curves via three sheet inflations.

illustrated in Figure 3-5.

In Figure 3-6 we decompose the (0,3,6) singular node into three valence 5 singular curves. It is also valuable to think of the (0,3,6) node as a combination of two (0,4,4) singular nodes. In this way, one only needs to decompose a singular node into constituent nodes that are already known to be decomposable. The fourth image of Figure 3-6 shows exactly this decomposition which we will notate as:

$$(0,3,6) = (0,4,4) +_5 (0,4,4)$$

The subscript 5 on the plus symbol denotes that two singular nodes are joined along a valence 5 edge, followed by an inverse sheet inflation (sheet collapse). This notation serves only as a shorthand and does not uniquely encode how to glue two singular nodes together. The non-uniqueness is clear from the fact that $(0,5,2) +_4 (2,3,0)$ can be either $(0,6,0)$ or $(2,2,2)$. These additions serve more as schematics than as an equation with any algebraic properties. For completion, we decompose the constituent $(0,4,4)$ in the latter half of Figure 3-6.

The (0,2,8) singular node is decomposed in Figure 3-7 into four valence 5 curves. In the fourth image, we see that

$$(0,2,8) = (0,3,6) +_5 (0,4,4)$$

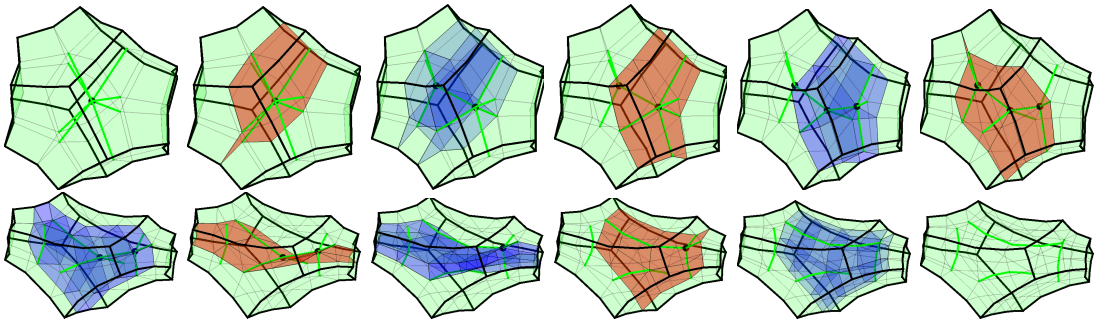


Figure 3-7: The (0,2,8) singular node is decomposed into four valence 5 curves via five sheet inflations.

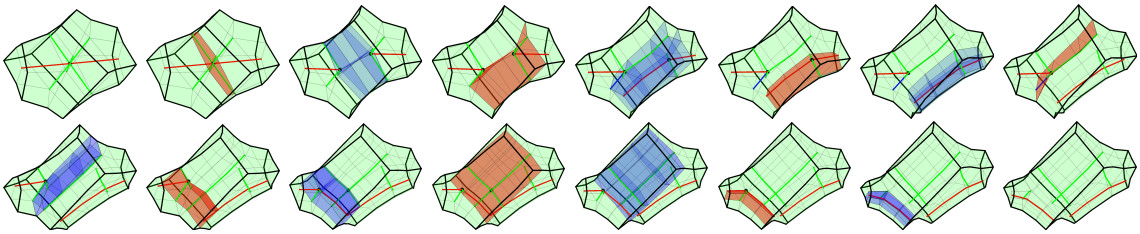


Figure 3-8: The (2,0,6) singular node is decomposed into four valence 5 and two valence 3 curves via seven sheet inflations.

and both constituent singular nodes have already been shown to be decomposable. For completion, the rest of the decomposition steps are also shown.

The (2,0,6) singular node is decomposed in Figure 3-8 into four valence 5 and two valence 3 curves. By the fourth image we see

$$(2, 0, 6) = (1, 3, 3) +_4 (1, 3, 3).$$

While similar to the already shown decompositions, this decomposition introduces a valence 6 singularity in the fifth image and a singular node with signature (1,3,3,1). This valence 6 singular curve is removed in image 9 by splitting a valence 5 curve off of its node. This node exemplifies that the singularities of a certain degree does not have to strictly decrease from sheet inflations or a decomposition.

Finally the (0,0,12) singular node decomposition is shown in Figure 3-9 to become six valence 5 curves. This singular node is counter-intuitive because we were unable

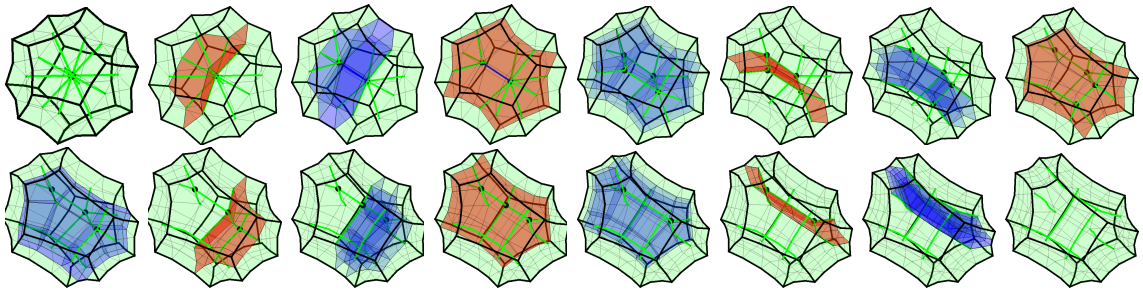


Figure 3-9: The (0,0,12) singular node is decomposed into six valence 5 curves via seven sheet inflations.

to show a decomposition of the form

$$(0, 0, 12) = (0, 4, 4) +_n (0, 2, 8).$$

Even though the number of singular curves present is sufficient, we were not able to split the six valence 5 curves into a group of four and two. This demonstrates that the order in which singular curves are combined matters. As this figure is especially complex to comprehend, we offer the following roadmap of how the decomposition is performed.

$$(0, 0, 12) = \underbrace{((0, 3, 6) +_5 (0, 3, 6))}_{(0,2,8,1)} +_6 \underbrace{((0, 4, 4) +_5 (0, 4, 4))}_{(0,4,4,1)}$$

Again, a valence 6 curve is created at an intermediate step of the decomposition.

While this paper focuses primarily on singular node decomposition via sheet inflation, it is also possible to simplify singular graphs via sheet collapse e.g.

$$(1, 3, 3) +_3 (4, 0, 0) = (0, 6, 0).$$

In this case, two singular nodes joined by a valence 3 curve can be entirely annihilated into a regular node: $(0, 6, 0)$. Furthermore, the symmetries of the $(4, 0, 0)$ node make the resulting combination independent of the ambiguities of $+_3$. Having such a configuration within a hex mesh is fortuitous but unreliable.

3.4 Decomposing General Singular Nodes

Given that the eight singular nodes of valence 3, 4, or 5 are decomposable into singular curves, it's natural to ask whether decomposition extends to higher valence singular nodes. In fact, the decompositions of the (0,0,12) and (2,0,6) nodes both already required decomposing the following singular nodes with valence 6: (0,4,4,1), (0,2,8,1) and (1,3,3,1). We will refer to previously known decomposable singular nodes and their associated sphere triangulations as *base cases*. To generalize decomposability of singular nodes we offer the following result.

Proposition 4. Given a regular sphere triangulation $\mathcal{T}$ with some vertex u of degree larger than 5, there exists a splitting such that either the number of vertices in both resulting triangulations decreases or the resulting triangulations are base cases.

Proof. The local neighborhood of u is an umbrella $\mathcal{U}$ of at least 6 triangles. The boundary of this umbrella is a cycle of at least 6 vertices denoted by $\mathbb{C}$. To construct a splitting of $\mathcal{T}$ into triangulations of fewer vertices, we need a pair of vertices a and b adjacent to u that are at least 3 edges apart from each other in $\mathbb{C}$ such that there is path p from a to b through the interior of $\mathcal{T} - \mathcal{U}$. This construction is illustrated in Figure 3-10. The sequence of edges $[(ua), p, (bu)]$ partitions $\mathcal{T}$ into $\mathcal{D}_1$ and $\mathcal{D}_2$ where each disk triangulation has at least 2 interior vertices. Since splitting a sphere triangulation replaces all vertices on the interior of either side with just one new vertex each, both resulting triangulations will have fewer vertices than $\mathcal{T}$. For readability, we leave more detailed construction of the splitting to § 3.8 □

Applying the splitting in Proposition 4 could result directly in base cases, where the rest of the recipe for decomposition into singular curves is already known. If the splitting does not result in base cases, then it produces triangulations with fewer vertices. This can be repeated until there are not enough vertices to have a degree 6 vertex. Since sheet inflation at a singular node corresponds to splitting of a sphere triangulation, Proposition 4 allows us to recursively find a sequence of sheets whose inflation results in singular nodes that have lower than valence 6 singular edges. We

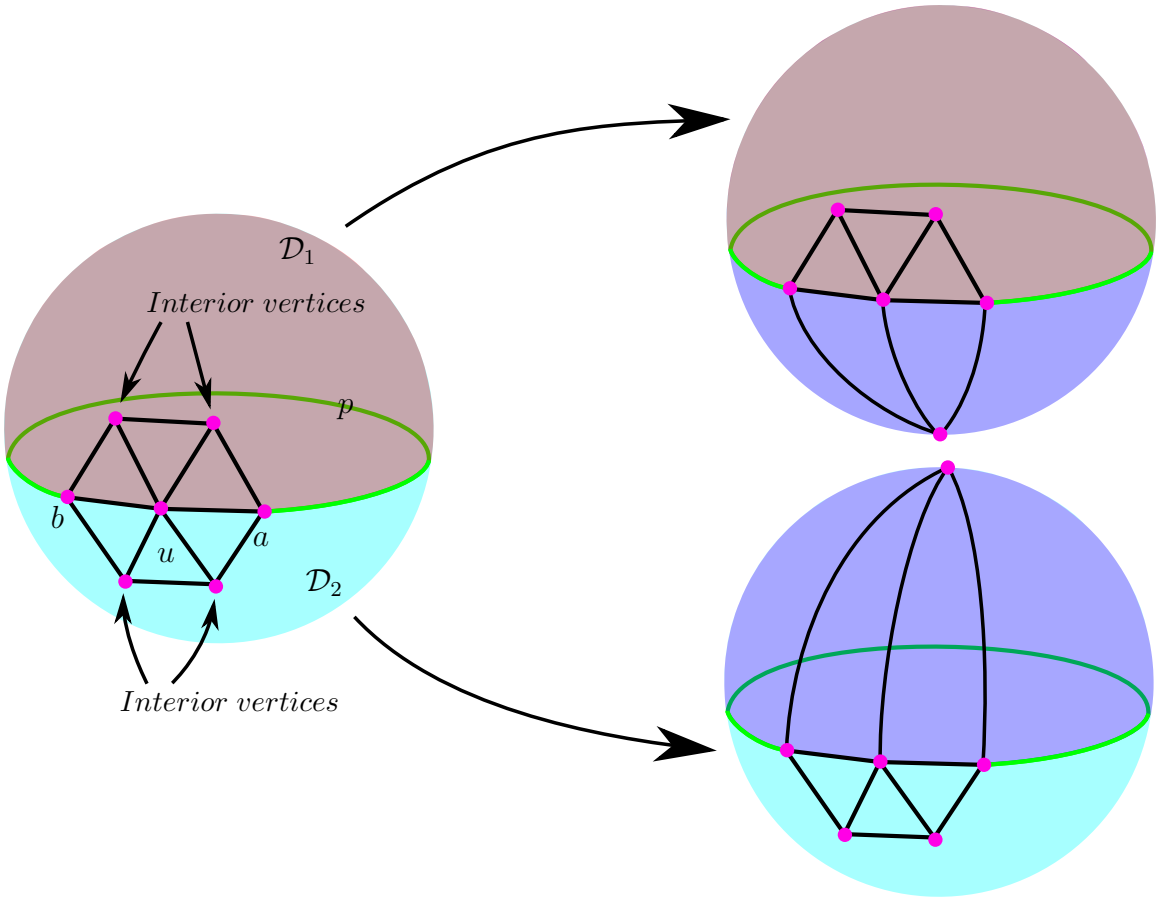


Figure 3-10: Illustration of how to find a cycle such that splitting along that cycle results in two sphere triangulations, each with fewer vertices. The only requirement is that there is a vertex u of degree ≥ 6. The required cycle is then $[(ua), p, (bu)]$.

Algorithm 1 Decomposes all singular nodes of a hex mesh into singular curves.

```
1: procedure DECOMPOSE-SINGULAR-GRAPH(H)
2:     do
3:         N ← GETRANDOMSINGULARNODE(H)
4:         if ONLYHASVALENCE345(H,N) then
5:             C ← GETHARDCODEDCUT(H, N)
6:         else
7:             C ← GETGENERALCUT(H, N)
8:         end if
9:         S ← PROPAGATECUT(H, C)
10:        H ← SHEETINFLATION(H, S)
11:    while N ≠ ∅
12:    return H
13: end procedure
```

have already enumerated singular decompositions for all singular nodes with valence lower than 6 and can therefore decompose any singular node into singular curves.

The primary limitation of Proposition 4 is that it restricts attention to individual singular nodes while ignoring the full singular graph of the mesh. Doing so implicitly assumes that the prescribed local splitting of a singular node can be extended to a sheet inflation on the hex mesh. Removing this assumption separates the proven *local decomposability* of singular nodes from the desired *global decomposability* of entire hexahedral meshes. We present our empirical bridge between local and global decomposition in § 3.5.

3.5 Singular Graph Decomposition

We develop a procedure to perform singular graph decompositions on full hex meshes rather than on individual singular nodes. Pseudocode for this procedure is given in Algorithm 1 and Algorithm 2 First, we randomly select a singular node. For any singular node with valence restricted to 3, 4, or 5, we hard-code a subset of faces adjacent to the node to be inflated. If the node has valence 6 or higher, we use Proposition 4 (denoted GetGeneralCut in Algorithm 1) to select these faces. These faces form a partial sheet that locally decomposes the initially selected node, but need to be extended through the rest of the mesh in order to be inflatable.

Next we propagate the partial sheet throughout the hex mesh following Algorithm 2. Let a face be *parallel* to the partial sheet if they share a regular edge but share no adjacent hexes. We greedily add parallel faces to the partial sheet until no more parallel faces can be found. Next we look for any interior singular vertices on the boundary of the partial sheet. If such a vertex is found, then we compute the smallest number of new faces that need to be added to the partial sheet so that its boundary excludes this singular vertex. This is denoted by Put-v-In-S in Algorithm 2 and is equivalent to a graph shortest path computation on the triangulation representing this singular vertex.

These two steps are repeated until no more parallel faces can be found, and the boundary of the partial sheet is entirely on the boundary of the hex mesh. If at any stage of the algorithm, the partial sheet became non-manifold then the sheet propagation algorithm has failed. This is because sheet inflation in a hex mesh is not well defined for general non-manifold sheets. Note that self-intersecting sheets are also counted as non-manifold and while specific sheet inflation algorithms could be used to handle them, we make the simplistic demand that all sheets are manifold. If the sheet is manifold then we inflate it resulting in the decomposition of at least one singular node. All results shown were generated by Algorithm 1.

3.6 Results

Applying our decomposition strategy to a hex mesh of a sphere reveals that it has the same singular graph structure as that of a padded tetrahedron. Figure 3-11 shows this correspondence where inflating one sheet that passes through seven singular nodes, simultaneously decomposes three of them. The end result is a singular graph composed of four (4,0,0) singular nodes. One of these nodes has singular curves that all connect directly to the boundary. The other four of these nodes connect to each other and the boundary via valence 3 singular curves in a tetrahedral arrangement. This singular graph is exactly what one obtains by padding a hex mesh of a regular tetrahedron i.e. padding a (4,0,0) node.

Algorithm 2 Propagates a partial sheet into a full sheet recursively.

procedure PROPAGATECUT($\mathcal{H}$, S)
  $\mathbb{Q} \leftarrow$GETFACES($\mathcal{H}$)
  while $\exists f \in \mathbb{Q}$:PARALLEL($\mathcal{H}$, S, f) **do**
    $S \leftarrow S \cup f$
  end while
  $\mathcal{V} \leftarrow$GETVERTICES($\mathcal{H}$)
  $\mathcal{V}_S \leftarrow$GETINTERIORSINGULARVERTICES($\mathcal{H}$)
  if $\exists v \in (\partial S \cap \mathcal{V}_S)$ **then**
    $S \leftarrow$ PUT-V-IN-S($\mathcal{H}$, S, v)
  else
    if NONMANIFOLD($\mathcal{H}$, S) **then**
      return ERROR
    else
      return S
    end if
  end if
  return PROPAGATECUT($\mathcal{H}$, S)
end procedure

Changing how the sheet cuts through the singular graph produces different intermediate and final singular graphs. In Figure 3-12, we decompose a padded cube in two different sequences and show the their intermediate singular graphs. To improve clarity, we provide schematics of a subset of the singular graphs. The ending singular graphs from both sequences are also topologically distinct i.e. no purely geometric deformation maps one singular graph into the other. They do however appear to invariably contain a single singular cycle.

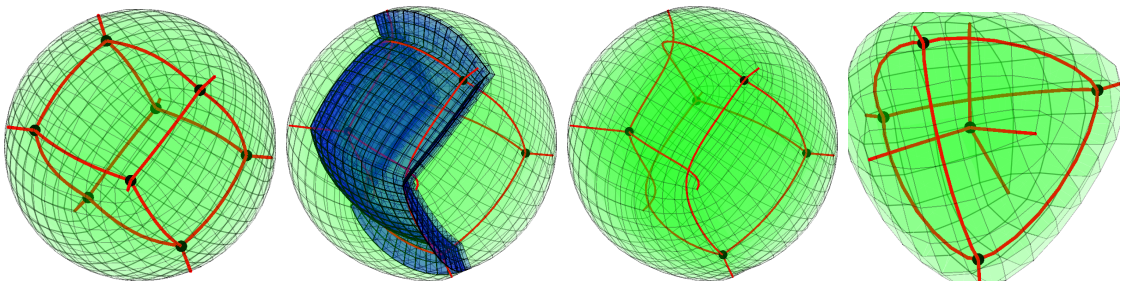


Figure 3-11: (Left) Hex mesh of sphere with singular graph. (Mid-left) Blue hexes are newly inflated hexes. (Mid-right) Hex mesh post-inflation. (Right) Singular graph of a padded hex mesh of a tetrahedron. The last two images have topologically equivalent singular graphs.

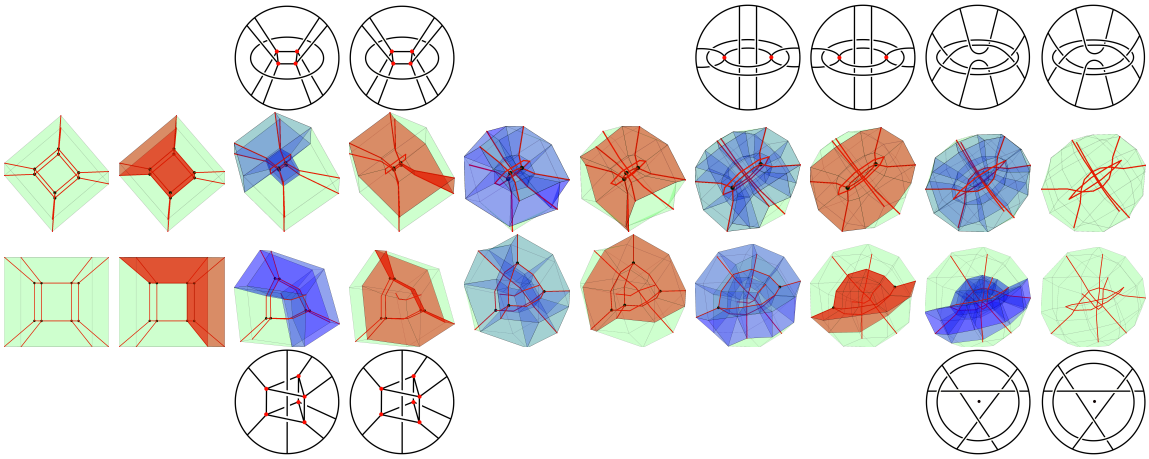


Figure 3-12: We show two sequences of singular graph decomposition starting from the same hex mesh on the left to a fully decomposed singular graph on the right. The first row indicates select singular graph schematics for the first sequence. The last row indicates select singular graph schematics for the second sequence. Even though both singular graphs start out identical, the ending singular graphs are different due to different sheet inflations.

The first sheet inflation of the second sequence results in the same singular graph as a padded hex mesh of a triangular prism: a padded (2,3,0). Since the hex mesh of a sphere has the same singular graph as the padded cube, these results indicate that singular graphs for a padded cube, padded tet, and padded triangular prism are identical up to a series of sheet inflations and collapses.

In Figure 3-13, we apply our decomposition to more complex singular graphs. The first two rows depict the decomposition of the **G1** hex mesh. The starting singular graph consists of 12 nodes connected by 36 singular curves. This graph is successfully decomposed into seven singular curves, one of which is a closed cycle. The last two rows depict the decomposition of the **G2** hex mesh. The starting singular graph consists of 16 nodes connected by 40 singular curves. This graph is successfully decomposed into 12 singular curves, two of which are closed cycles. While the starting singular graphs are different, both meshes are fully decomposed with the same number of sheet inflations.

Finally, we apply our decomposition in Figure 3-14 to the cactus mesh from [Bra+19]. While the mesh starts with only singularities of valence 3, 4 and 5, the decomposition results in intermediate singular graphs with nodes of signature (2,3,0,2).

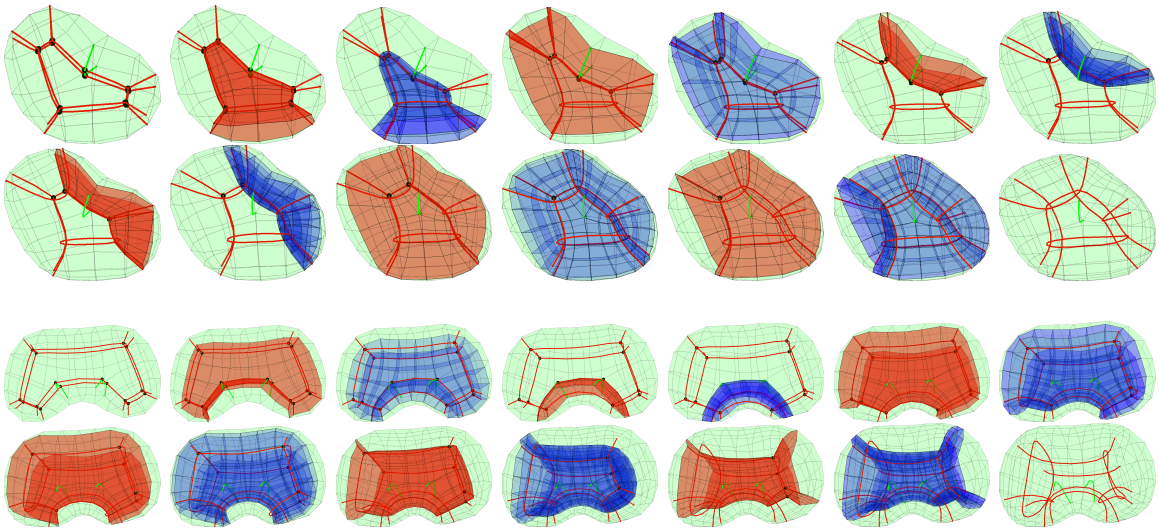


Figure 3-13: We apply singular decomposition to the **G1** and **G2** hex meshes. The first two rows correspond to the sequence of singular graphs from decomposing **G1**. The last two rows correspond to the sequence of singular graphs from decomposing **G2**. The number of singular nodes decreases each sheet inflation ultimately resulting in a singular graph with no nodes at all.

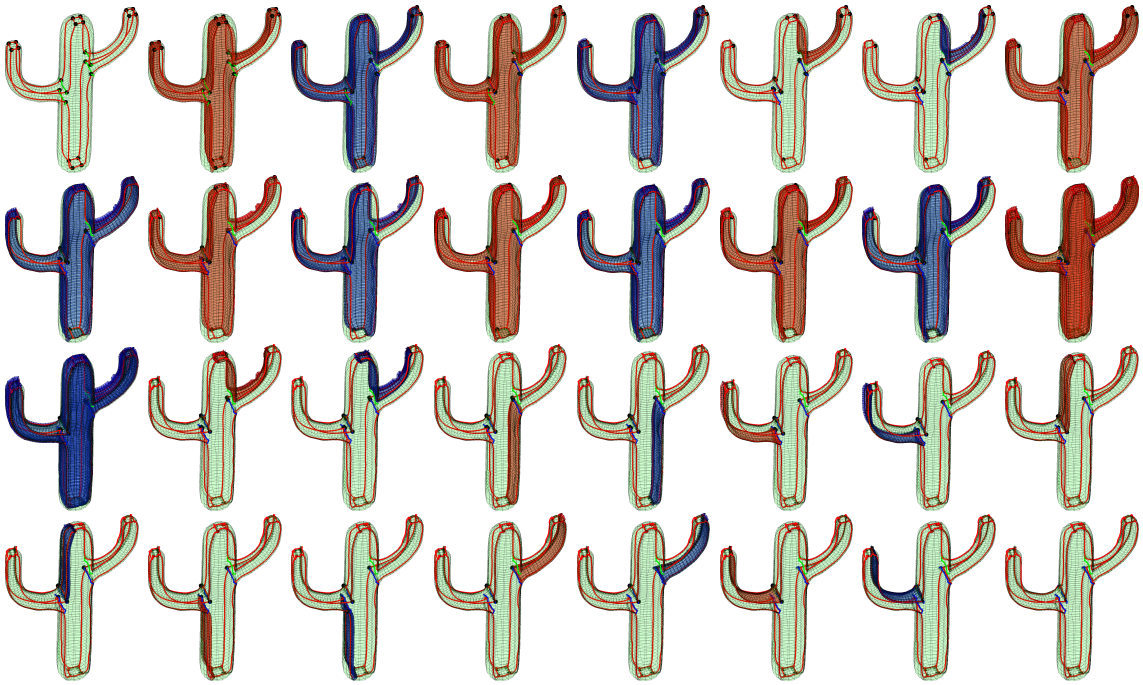


Figure 3-14: We apply singular decomposition to the cactus mesh from [Bra+19]. The number of singular nodes decreases with each sheet inflation ultimately resulting in a singular graph with no nodes at all. While the original singular graph consisted of only valence 3, 4 and 5 nodes, intermediate singular graphs from this sequence contain singular nodes with signature (2,3,0,2). The fully decomposed singular graph has valence 6 curves.

The final configuration is seen to contain singular curves of valence 6. Since our goal is only to remove singular nodes, we terminate with valence 6 curves.

3.6.1 Scaled Jacobians

The minimum scaled Jacobian of a hex mesh is a common metric by which to evaluate distortion of the mesh [Qua21]. We maximize the minimum scaled Jacobian before and after singular decomposition of each singular node with the following geometric optimization. First, we assemble the $8\mathcal{H} \times 1$ vector J of scaled Jacobians, one per corner of each hex. We then minimize the optimization energy $E_p = \|1 - J\|_p^p$ via gradient descent on mesh vertices with free boundaries. We perform 100 iterations of gradient descent on E_2, E_4, and E_8 consecutively while using line search to ensure energy decrease. As $p \to \infty$, the p-norm emulates a max operation, but in practice we see no significant difference past $p = 8$. The resulting minimum scaled Jacobians from this optimization are summarized in Table 3.1. Unsurprisingly, singular nodes have lower scaled Jacobians than singular curves.

By symmetry, the minimum scaled Jacobian of any hex mesh, regardless of resolution, containing a (4,0,0) node is upper bounded by $\frac{4}{3\sqrt{3}} = .7698$. The same bound for a hex mesh containing a (0,0,12) node is $\frac{\sqrt{2(5+\sqrt{5})}}{5} = .761$. These bounds are exactly attained in Table 3.1 for the (4,0,0) and (0,0,12) nodes. The same upper bound computed for meshes containing valence 3 singular curves is $\sin(\frac{2\pi}{3}) = .866$ and for meshes containing valence 5 singular curves is $\sin(\frac{2\pi}{3}) = .951$.

By performing a sheet inflation to split singular nodes into singular curves, the minimum scaled Jacobian of the (4,0,0) node is increased to .86, almost the theoretic upper bound. For (0,4,4) as well, decomposing the singular node into two valence 5 curves brings the minimum scaled Jacobian to almost the theoretic upper bound. Decomposing (0,0,12) node into six valence 5 curves brings significant improvement to the minimum scaled Jacobian, though it is not as close to the theoretic upper bound due to interactions between singular curves.

Moving towards full singular graphs, we perform the same scaled Jacobian opti-

Mesh	Original	Decomposed	UpperBound
(4,0,0)	0.769	0.86	0.866
(2,2,2)	0.807	0.862	0.866
(0,4,4)	0.896	0.943	0.951
(1,3,3)	0.822	0.865	0.866
(0,3,6)	0.863	0.939	0.951
(0,2,8)	0.82	0.937	0.951
(2,0,6)	0.745	0.856	0.866
(0,0,12)	0.761	0.926	0.951
Sphere	0.768	0.849	0.866
Padded Tet	0.715	0.812	0.866
G1	0.769	0.811	0.866
G2	0.757	0.820	0.866
Ellipsoid	0.767	0.825	0.866

Table 3.1: For various hex meshes, we indicate the maximized minimum scaled Jacobian before and after singular decomposition. The first column indicates the mesh, the second column indicates before singular decomposition, and the third column indicates after. The fourth column indicates a theoretic upper bound on the minimum scaled Jacobian for the decomposed mesh. It essentially indicates the presence of a valence 3 or 5 singular curve. The maximized minimum scaled Jacobians are frequently at global optimality before decomposition, and unable to increase due to a singular node. The scores are invariably higher post singular decomposition.

mization for a sphere mesh. Maximization of its minimum scaled Jacobian results in a value of .768, close to the upper bound for any mesh containing a (4,0,0) node. We apply our decomposition to this mesh and re-optimize its scaled Jacobian resulting in a significant improvement to .849. We run the same optimizations on the padded tetrahedron, **G1**, and **G2** resulting in similar increases in the minimum scaled Jacobian. These results are summarized in Table 3.1. The topological process of singular decomposition on **cactus** unfortunately created geometric inversions we could not untangle. This reflects the difficulty of untangling meshes, but does not affect singular node removal nor our conclusion that the minimum scaled Jacobians can be increased through singular decomposition.

3.6.2 Implications for Octahedral Field Based Hex Meshing

Our results have indicated so far that singular nodes in a hex mesh tend to increase scaled Jacobian. In this subsection we explore how singular nodes form to begin with in the context of octahedral field based hex meshing. This approach starts with the construction of an octahedral field (also frequently called a *frame field*): a map from the domain of interest into the space of rotated cubes SO(3)/O [Hua+11; SVB17]. The frame field map is invariably computed by minimizing Dirichlet energy, thus promoting smooth solutions. Parameterization and hex extraction ultimately convert the frame field into a hex mesh where singularities of the field match singularities of the mesh.

It may be tempting to think of singular nodes as some kind of point defect of the frame field, however, $\pi_2(\mathrm{SO}(3)/\mathrm{O}) = \emptyset$, so frame fields necessarily do not contain point defects. Instead, [Mer79, Figure. 42-45] most crucially show that when two singular curves pass through each other they necessarily create a new singular curve with type equal to their commutator. Frame field singular curve types are elements of $\pi_1(\mathrm{SO}(3)/\mathrm{O}) = \mathrm{BO}$, the binary octahedral group, which is non commutative [Mer79]. Therefore, having singular curves pass through each other is highly dis-incentivized in frame field computations since singularities increase Dirichlet energy. The effect of minimizing Dirichlet energy is essentially that singular curves try not to pass through each other.

Since Dirichlet energy promotes smoothness, it also shortens the length of singular curves. When this shortening effect pulls singular curves into each other, they may either pass through incurring the cost of creating a new singular curve, or the curves can remain tautly in contact thus creating a singular node. The optimization of Dirichlet energy is therefore responsible for the formation of singular nodes in frame fields that are then replicated in the output hex meshes. As we have seen, singular nodes hurt hex mesh quality so Dirichlet energy is not a good optimization objective in frame field computations. Unfortunately, it is by far the most popular objective function in frame field computation.

3.7 Discussion

This chapter presents singular nodes as the result of gluing singular curves together at a point and shows that the reverse can be done via sheet inflation to unmake singular nodes into simple singular curves. This removes the 3D complexity of singular nodes leaving meshes with lower distortion. We demonstrate this procedure on a variety of meshes showing in all cases that no singular nodes are left behind and that mesh distortion is reduced.

The main limitation of our work is that the local sheets we prescribe for decomposing a singular node are not guaranteed to propagate globally while avoiding self-intersection. This can result in the inability to decompose a singular graph entirely. We expect that a valid sheet inflation can always be found and leave its efficient computation to future work.

While our method decreases the number of singular nodes in a mesh, its *base complex* [GDC15] may increase in size. This tradeoff should be considered by the user as they may have to choose between a larger scaled Jacobian or maintaining a small number of base complex cells.

Our results can be extended to design new ways of modifying the singular graph of a mesh. Instead of only decomposing nodes into curves, one can *rewire* singular curves by merging them at a node with sheet collapse, and decomposing them in a different way from how they were combined. For example, the sphere triangulation of the (4,0,0) node contains three distinct cycles of length four. Therefore, it is possible to bring two valence 3 singular curves together to form a (4,0,0) node and split them apart again in three distinct ways.

Many works aim to build minimal degree smooth parameterizations of quad meshes with singularities [KP16; KP19]. These methods do not clearly generalize to the volumetric case where singular nodes may suffer decreased continuity from methods designed for 2D singularities. A promising approach following our work is then to decompose any given singular graph so that no singular nodes exist. We expect that it is easier to adapt quad mesh singular parameterization methods to

singular curves that are just 2D singularities extruded into 3D than it is to adapt parameterization methods for singular nodes. Even if one derived a singular node parameterization method for a specific node type, it may not extend to other node types. This problem is made easier by only needing to consider singular curves after decomposition.

3.8 Appendix: Proof of Proposition 4

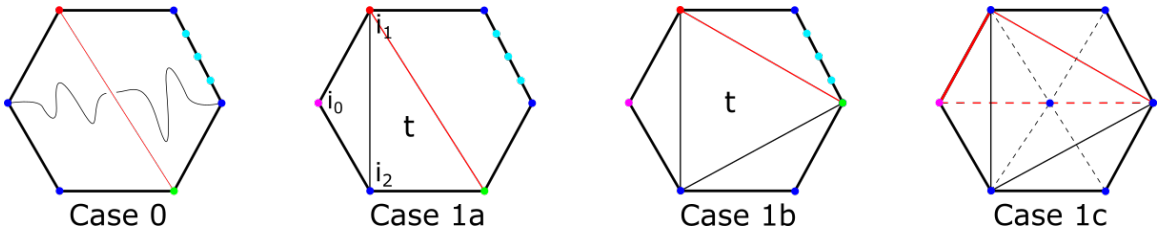


Figure 3-15: Illustrations of $\mathcal{T} - \mathcal{U}$ in various situations. This figure considers the cases where there are no isolated vertices, or when there is at least one isolated vertex and t has all its vertices on $\mathbb{C}$.

We show here how to construct a splitting satisfying Proposition 4. For the majority of cases we will construct a path p such that the splitting is $[(ua), p, (bu)]$ as described in the main document. In a few cases, we will explicitly construct the splitting cycle without using u or p. Recall from the main document that in order for the splitting to result in sphere triangulations with fewer vertices, it must partition $\mathcal{T}$ into regions with at least two interior vertices each.

We define an *isolated vertex* to be a boundary vertex of $\mathcal{T} - \mathcal{U}$ with degree 2. Isolated vertices are impossible to build path p from since they have no path to the interior of $\mathcal{T} - \mathcal{U}$. For the rest of the proof we will repeatedly reference the labeled cases in Figure 3-15, Figure 3-16 and Figure 3-17. These figures depict $\mathcal{T} - \mathcal{U}$ with the following color coding. The purple vertex is an arbitrary isolated vertex. The red vertex is labeled a, and the green vertex is labeled b. Cyan vertices are optionally present to indicate that $\mathbb{C}$ has more than or equal to 6 vertices.

Case 0 First, we consider the case where no vertices on $\mathbb{C}$ are isolated. We can choose a and b arbitrarily as long as they are 3 or more edges apart in $\mathbb{C}$ and compute p as the shortest path between them in the interior of $\mathcal{T} - \mathcal{U}$. If such a path is found, then we have our splitting. If the path does not exist, there must be an interior edge e between boundary vertices on $\mathbb{C}$ separating a from b. Since no isolated vertices exist, we can simply choose endpoints of e as the new a and b with $p = e$. This is illustrated by the first image of Figure 3-15.

The rest of the proof considers the case where at least one isolated vertex i_0 exists. Let its neighboring vertices be i_1, i_2. (i_0, i_1, i_2) must be a triangle in $\mathcal{T} - \mathcal{U}$ with an interior edge (i_1, i_2). Since every interior edge is adjacent to two triangles, flipping across (i_1, i_2) brings us to a triangle we will denote t. t can be arranged in a few different ways.

Case 1a The triangle t is completed with a boundary vertex adjacent to i_2. Without loss of generality, i_1 is exchangeable with i_2. This is illustrated in the second image of Figure 3-15. The single red edge in t is p.

Case 1b The triangle t can be completed with a boundary vertex not adjacent to either i_1 or i_2. This is illustrated in the third image of Figure 3-15. As long as one or more cyan vertices are present, the red edge separates $\mathbb{C}$ such that both sides have at least 2 interior vertices.

Case 1c This case is identical to Case 1b except there are no cyan vertices i.e. $\mathbb{C}$ has exactly 6 vertices. This configuration is special in that we are unable to split $\mathcal{T}$ into two triangulations each with fewer than 7 vertices. Thus the full sphere triangulation is depicted including u and its neighborhood $\mathcal{U}$ with dashed edges in the fourth image of Figure 3-15. For this configuration, we specify the full splitting by the red edges. This splitting turns the sphere triangulation into

$$(3, 0, 3, 1) = (2, 2, 2) +_4 (1, 3, 3).$$

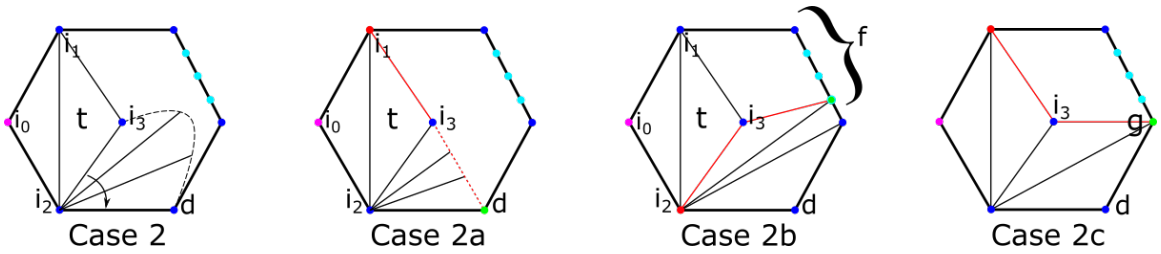


Figure 3-16: Illustrations of $\mathcal{T} - \mathcal{U}$ in various situations. This figure considers the cases where t has one vertex on the interior of $\mathcal{T} - \mathcal{U}$ and $\mathbb{C}$ has more than 6 vertices.

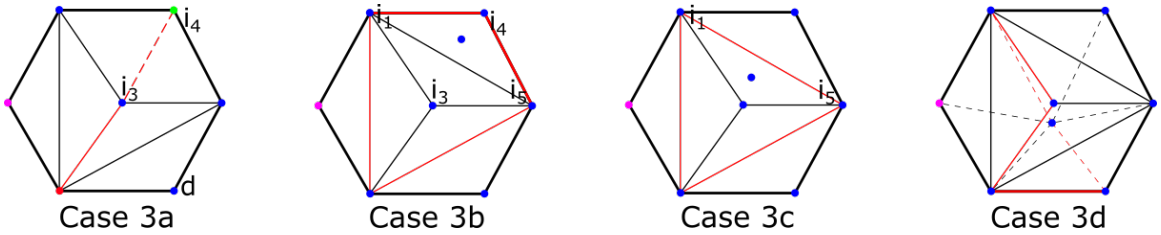


Figure 3-17: Illustrations of $\mathcal{T} - \mathcal{U}$ in various situations. This figure considers the cases where t has one vertex on the interior of $\mathcal{T} - \mathcal{U}$ and $\mathbb{C}$ has exactly 6 vertices.

Even though the number of vertices in one of the resulting triangulations is still 7, all vertex degrees are less than or equal to 5, thus they are base cases.

Case 2 The last possibility is that triangle t is completed with one interior vertex i_3 of $\mathcal{T} - \mathcal{U}$. We illustrate this in the first image of Figure 3-16 and refer to its labels. Since i_2 is adjacent to both i_3 and d there must be a path from i_3 to d that is entirely adjacent to i_2. This is indicated by the dashed black edge. The rest of Case 2 enumerates different possibilities for this path.

Case 2a If the path is fully on the interior of $\mathcal{T} - \mathcal{U}$, then p is indicated by the red edges in the second image of Figure 3-16.

Case 2b If the path connects i_3 to any vertex labeled f in the third image of Figure 3-16, then p is indicated by the red edges.

Case 2c The only remaining possibility is that the path connects i_3 to g depicted in the fourth image of Figure 3-16. Assuming at least one cyan vertex is present, the red edges form p.

Case 3 If there are no cyan vertices, i.e. $\mathbb{C}$ contains only 6 vertices, then the red edges from Case 2c do not partition $\mathbb{C}$ into two regions of at least two interior vertices. In this setting we consider the last few cases.

Case 3a If there is an interior path from i_3 to i_4 as labeled in the first image of Figure 3-17, then the red edges form p.

Case 3b, 3c The only way for the interior path from case 3a to not exist is if an edge prevents an interior path between i_3 and i_4. This edge is labeled (i_1, i_5) in the second and third images of Figure 3-17. These two images show explicit splittings for $\mathcal{T}$ in the case that there is an interior vertex of $\mathcal{T} - \mathcal{U}$ that is not i_3.

Case 3d Finally, if there are no interior vertices of $\mathcal{T} - \mathcal{U}$ except for i_3, then we know the full sphere triangulation $\mathcal{T}$. This triangulation has signature (4,0,0,4) and is fully depicted in the last image of Figure 3-17. The red edges indicate the following splitting:

$$(4, 0, 0, 4) = (3, 0, 3, 1) +_5 (1, 3, 3, 1).$$

Both of the resulting sphere triangulations are base cases.

In summary, we have explicitly constructed splittings of $\mathcal{T}$ that result in two sphere triangulations each with fewer vertices than $\mathcal{T}$. The two exceptions are cases 1c and 3d where we construct splittings that result in base cases.

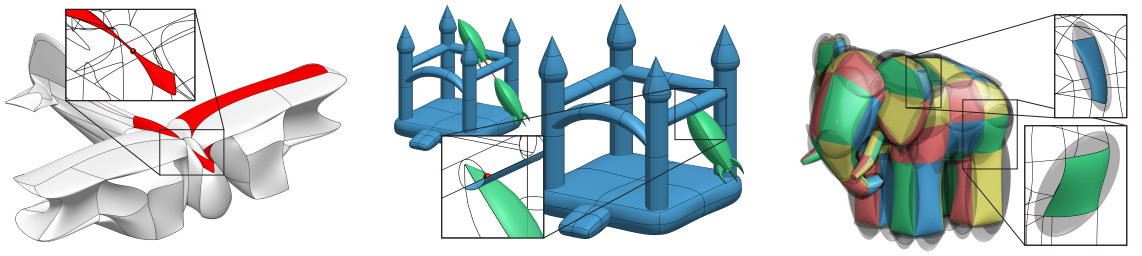
4

Algebraic Collision Detection with Sum-of-Squares Programming

This chapter is based on work in [Mar+21], which spun out as a sequel to [Mar+20]. [Mar+20] originated as an Undergraduate Research Opportunities Program (UROP) project where David Palmer and I co-mentored Zoë Marschner. We initially aimed for the modest task of repairing tangled hexahedral meshes using sum-of-squares (SOS) programming. After that initial success, we realized the sum-of-squares machinery was extremely general and started brainstorming other problems to tackle, eventually resulting in [Mar+21]. Zoë implemented at least the first working iteration of each problem formulation, and generated all the figures. I ran the batch experiments generating the bulk of the empirical results. David and I wrote and revised most of the text, and David helped elucidate various aspects of the SOS theory. We all contributed to the problem formulations presented. Compared to [Mar+21], the discussion section has been modified to reflect more recent understanding of open problems in SOS programming.

4.1 Introduction

Of the many geometric representations available today, *polynomial patches* are exceptionally powerful. They are a mainstay of computer-aided design and digital sculpting,



Self intersections in Coons patches | Continous collision detection in NURBS patches | Bounding ellipsoids of tensor product patches

Figure 4-1: We describe a framework for solving many different geometry processing problems on polynomial patches using sum-of-squares relaxation. Here, we show three of the problems we discuss in this paper: Self-Intersection, Continuous Collision Detection, and Minimal Bounding Ellipsoid. Our formulation is very general, working for any piecewise polynomial or rational patch type. We show each problem here on a model with a different patch type: cubic Coons patches from [SBS20], bicubic NURBS patches from [TCL21], and bicubic Bézier tensor patches.

where they provide piecewise-smooth parametrization. They can represent complex shapes with just a few control points. And in finite-element modeling, high degree polynomial bases can be used to construct high-fidelity solutions to partial differential equations.

Polynomial patches have recently made inroads into geometry processing problems involving PDEs and simulation [Sch+19; Sch+18; MC20; TCL21; Jia+21]. However, basic geometric kernels that are straightforward to implement for piecewise-linear meshes remain challenging for higher-order patches. These kernels include detecting self-intersections and collisions, computing bounding volumes, and measuring distances. Such kernels become especially important in *dynamics* problems, which in principle require continuous-time maintenance of physical feasibility.

We propose to bring the methods of *sum-of-squares programming* (SOS) to bear on this domain. The core idea of SOS programming is the replacement of polynomial positivity constraints with more computationally tractable SOS constraints of bounded degree d, which can be represented by semidefinite programs. The miracle of SOS programming comes from the Positivstellensatz [BPT12], which states that for a large enough d, the globally-optimal solution to the modified problem is certifiably equivalent to the globally-optimal solution of the original problem. This certificate of

correctness comes from the theory of SOS programming and is known as *exact recovery*. Thus, the SOS machinery allows us in many cases to solve seemingly non-convex optimization problems to global optimality.

A caveat to SOS relaxation is that the cost of solving the relaxed problem increases factorially with d [BPT12]. While the minimal required d for exact recovery is problem-dependent and can be large in general, it is often close to the degrees of the polynomials in the original problem formulation, thereby maintaining tractability. Furthermore, we find that in practice the d required to obtain the correct solution to the original problem is often lower than the d required to obtain an exact recovery certificate. Since one often only cares about correctness of the solution, choosing the lower d allows the cost of the problem to be driven down in this case. If an algebraic proof of correctness is desired, the higher d can be chosen, at the expense of an increase in the cost of the problem.

Returning to geometry processing on polynomial patches, many geometric kernels can be formulated as polynomial optimizations. For example, surface-surface intersection (SSI) of quadratic triangles can be written as the minimization of a quartic objective function with linear inequality constraints. Applying SOS relaxation then yields a convex problem directly, and it only remains to verify that the relaxation successfully produces a solution to the original problem, i.e., that d was large enough. As illustrated by this example, the simplicity of SOS relaxation makes it readily adaptable to a wide variety of problems.

In this paper, we apply SOS methodology to several core problems in geometry processing on polynomial surfaces of varying degree. We show that a minor modification to our problem formulation allow us to support rational surfaces as well—namely, those consisting of NURBS patches. We verify the success of our SOS formulations on an exhaustive suite of test data. An overview of these results is provided in Table 4.1, where we show the experimentally determined minimum degree d_{correct} required to solve each problem with 100% accuracy. Finally, we apply these geometric kernels on various higher-order meshes, demonstrating the extensibility of geometry processing methods on linear meshes to higher-order surfaces. With these low-level operations

out of the way, we pave the way for development of higher-level geometry processing techniques on higher-order surfaces.

4.2 Related Work

SOS programming Sum-of-squares (SOS) programming is a type of convex relaxation in which polynomial positivity constraints are replaced by SOS constraints. These can be transformed further into semidefinite optimization problems (SDP), which are solvable in polynomial time via interior-point methods [Ali95; NN94; BBV04]. Modeling frameworks such as YALMIP [Löf04; PPP02] convert SOS formulations into SDPs, which can be solved by black-box solvers such as MOSEK and SDPT3 [ApS17; TTT01]. We provide the mathematical background relevant to this paper in § 4.3. For a comprehensive review of this field, see [BPT12].

4.3 Preliminaries

For a complete review of sum-of-squares theory we refer the reader to [BPT12; Par19]; Marschner et al. [Mar+20] also present similar background in the context of a geometry processing problem. For completeness, we recall the most relevant concepts here.

4.3.1 Positive Polynomials and SOS Polynomials

Let $\mathbb{R}[\mathbf{u}] = \mathbb{R}[u_1, ..., u_k]$ be the ring of real multivariate polynomials in $\mathbf{u}$. $\mathbb{R}[\mathbf{u}]_d$ denotes the subset of polynomials of degree at most d. We will use $[\mathbf{u}]_d$ to denote the basis of monomials up to degree d. Any member $f(\mathbf{u}) \in \mathbb{R}[\mathbf{u}]_d$ can be written in this basis: $f(\mathbf{u}) = [\mathbf{u}]_d^\top \mathbf{f}$, where $\mathbf{f}$ denotes the vector of coefficients of monomials in f.

A special subset of $\mathbb{R}[\mathbf{u}]$ (resp. $\mathbb{R}[\mathbf{u}]_d$) is the cone of *positive polynomials* P (resp. P_d). As the name suggests, positive polynomials $f(\mathbf{u}) \in P$ satisfy $f(\mathbf{u}) \geq 0$ for all $\mathbf{u}$. Many polynomial optimization problems can naturally be written with positive polynomial constraints, and thus it is highly desirable to be able to optimize over P. Unfortunately, even membership testing in P is NP-hard in general [BPT12, §3.4.3].

This leads us to the more restrictive subset Σ of *sum-of-squares (SOS) polynomials*; we use Σ_d to denote SOS polynomials of bounded degree d. Members of this set $f(\mathbf{u}) \in \Sigma$ can be decomposed into sums of squares of polynomials: $f(\mathbf{u}) = \sum_i s_i(\mathbf{u})^2$ for $s_i \in \mathbb{R}[\mathbf{u}]$. Naturally, they form a subset of P, giving us the following inclusions:

$$\Sigma_d \subset P_d \subset \mathbb{R}[\mathbf{u}]_d \tag{4.1}$$

Unlike P_d, Σ_d is computationally tractible—feasibility and optimization problems over Σ_d translate naturally in to SDPs. This will be made explicit in § 4.3.2.

4.3.2 SOS Optimization

Membership in Σ_d can be expressed via semidefinite programming. Using $\langle \cdot, \cdot \rangle$ to denote the Frobenius inner product, an SOS polynomial is equivalently written as

$$\begin{aligned} f(\mathbf{u}) &= \begin{pmatrix} s_1(\mathbf{u}) \\ \vdots \\ s_k(\mathbf{u}) \end{pmatrix}^\top \begin{pmatrix} s_1(\mathbf{u}) \\ \vdots \\ s_k(\mathbf{u}) \end{pmatrix} \\ &= [\mathbf{u}]_{\lfloor d/2 \rfloor}^\top \underbrace{\begin{pmatrix} \mathbf{s}_1 & \cdots & \mathbf{s}_k \end{pmatrix} \begin{pmatrix} \mathbf{s}_1^\top \\ \vdots \\ \mathbf{s}_k^\top \end{pmatrix}}_{S} [\mathbf{u}]_{\lfloor d/2 \rfloor} \\ &= \left\langle S, [\mathbf{u}]_{\lfloor d/2 \rfloor} [\mathbf{u}]_{\lfloor d/2 \rfloor}^\top \right\rangle, \end{aligned} \tag{4.2}$$

where the coefficients $\mathbf{s}_i$ are now encoded in the matrix S. Equation 4.2 provides a linear relationship between $\mathbf{f}$ and S, which is positive semidefinite by construction. Indeed, we have shown that the existence of such an $S \succeq 0$ is equivalent to $f \in \Sigma_d$. See Table 4.1 for rough dimensions of the SDP problem.

Using the above transformation to render SOS constraints into semidefinite constraints, one can perform optimization over $f(\mathbf{u}) \in \Sigma_d$ with relative ease. For this reason, it is often profitable to transform a problem involving a positivity constraint

$f \in P$ to one with a set of constraints of the form $s_i \in \Sigma_d$—resulting in an *SOS program*. A key theorem in SOS programming, the *Positivstellensatz*, explains how to effect this transformation so that the global optimum of the original problem is recovered in the limit of increasing SOS degree d [BPT12]. We will use a specialized version of the Positivstellensatz below.

4.3.3 SOS Optimization on a Compact Domain

In the context of geometry processing, one frequently seeks to optimize a functional over a compact domain, rather than over all of $\mathbb{R}^k$. Thus, one often encounters constraints of the form $f(\mathbf{u}) \geq 0$ for $\mathbf{u} \in \mathcal{D}$ where $\mathcal{D}$ is compact. SOS programming can be extended to handle such constraints. Here the key theorem is Putinar's variant of the Positivstellensatz:

Theorem 1 (Putinar's Positivstellensatz [Put93]; see also [BPT12], Theorem 3.138)**.** Let $\mathcal{D} = \{\mathbf{u} \in \mathbb{R}^k : g_i(\mathbf{u}) \geq 0\}$ be a domain with an algebraic certificate of compactness. Any polynomial $f(\mathbf{u})$ that is strictly positive on $\mathcal{D}$ admits a decomposition

$$f(\mathbf{u}) = s_0(\mathbf{u}) + \sum_{i=1}^{m} s_i(\mathbf{u}) g_i(\mathbf{u}), \tag{4.3}$$

with SOS polynomials $s_i \in \Sigma_d$ for high enough degree d.

Observe that the decomposition Equation 4.3 provides a certificate of nonnegativity by construction, since all s_i are nonnegative and all g_i are nonnegative on $\mathcal{D}$. For brevity we omit details of the required algebraic certificate of compactness and refer the interested reader to [BPT12]. For all domains encountered in this paper, compactness certificates are readily computable. The theory of SOS programming has an elegant dual formulation that allows us to determine a sufficiently large d, but we will defer discussion of this dual to § 4.5 to prioritize presenting concrete examples of what SOS programming can achieve for geometry processing.

Equality Constraints It is often convenient to consider *semialgebraic* domains $\mathcal{D}$ defined by polynomial inequalities *and equations*. This generalization does not pose

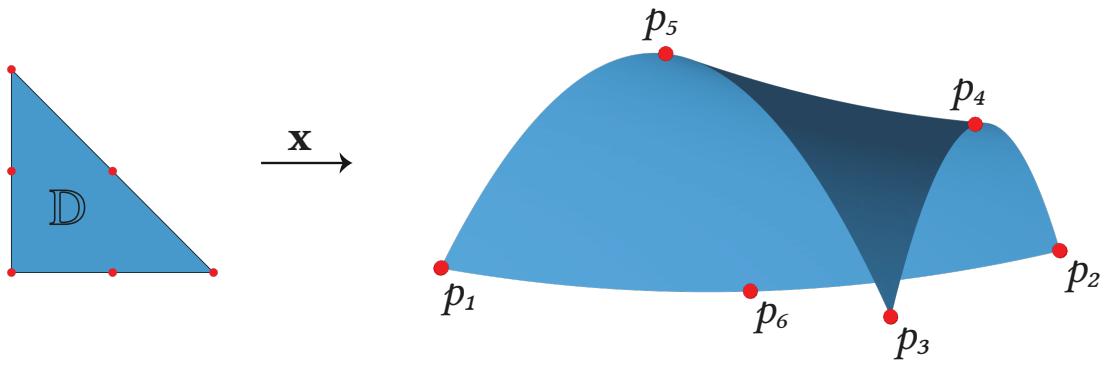


Figure 4-2: Quadratic triangle patch

a significant problem for SOS methods, for the reasons we will outline below.

Given a domain defined by equations and inequalities $\mathcal{D} = \{\mathbf{u} : g_i(\mathbf{u}) \geq 0, h_i(\mathbf{u}) = 0\}$, a decomposition of the form

$$f(\mathbf{u}) = s_0(\mathbf{u}) + \sum_{i=1}^{m} s_i(\mathbf{u}) g_i(\mathbf{u}) + \sum_{i=1}^{n} p_i(\mathbf{u}) h_i(\mathbf{u}), \tag{4.4}$$

where the s_i are SOS and the p_i are arbitrary polynomials, certifies the nonnegativity of f on $\mathcal{D}$.

One can formally rewrite each equality constraint $h_i(\mathbf{u}) = 0$ as a pair of inequality constraints $h_i(\mathbf{u}) \geq 0$ and $-h_i(\mathbf{u}) \geq 0$, reducing $\mathcal{D}$ to the form required in Theorem 1. Then the theorem shows existence of a certificate of the form

$$f(\mathbf{u}) = s_0(\mathbf{u}) + \sum_{i=1}^{m} s_i(\mathbf{u}) g_i(\mathbf{u}) + \sum_{i=1}^{n} (t_i^+(\mathbf{u}) - t_i^-(\mathbf{u})) h_i(\mathbf{u}), \tag{4.5}$$

where the s_i, t_i^+, and t_i^- are SOS. But such a certificate is *a fortiori* of the form Equation 4.4. In this way, Theorem 1 extends to the case of mixed constraints.

4.3.4 Polynomial Patches

Definition 1. A *polynomial patch* is a map $\mathbf{x} : \mathbb{D} \subset \mathbb{R}^k \to \mathbb{R}^n$ from a compact semialgebraic base domain $\mathbb{D} = \{\mathbf{u} \in \mathbb{R}^k : g_i(\mathbf{u}) \geq 0\}$, and such that each component of $\mathbf{x}$ is a multivariate polynomial of bounded degree $d_{\mathbf{x}}$.

In practice, the base domain is typically a canonical triangle, square, cube, or similar, and so the polynomials g_i are affine. The dimension of the patch for most geometry processing tasks is $k \in \{1, 2, 3\}$, and n controls the embedding dimension of the patch.

It is often useful to parameterize these polynomials $\mathbf{x}$ by linear combinations of a few basis functions, with coefficients that are obtained from control points $\mathbf{p}_i \in \mathbb{R}^n$. Patches of this form include Bézier and Coons patches, and they generalize the simplest polynomial patch, the *linear triangle*, which we will describe here for illustrative purposes. For the linear triangle patch, $\mathbb{D}$ is the triangle with vertices $(1, 0), (0, 1), (0, 0)$, $n = 3$, and the basis functions (in barycentric coordinates) are $\phi_1(\mathbf{u}) = u_1$, $\phi_2(\mathbf{u}) = u_2$, and $\phi_3(\mathbf{u}) = 1 - u_1 - u_2$. If the vertices of the linear triangle in $\mathbb{R}^3$ are $\mathbf{p}_1$, $\mathbf{p}_2$ and $\mathbf{p}_3$, then $\mathbf{x}(\mathbf{u}) = \phi_1\mathbf{p}_1 + \phi_2\mathbf{p}_2 + \phi_3\mathbf{p}_3$.

By varying the degree and number of basis functions, one can realize a variety of patch types. For example, a quadratic triangle is parameterized by six basis functions with control points p located at the vertices and edge midpoints. Such a quadratic triangle, along with its control points, is depicted in Figure 4-2. We will use n_B to denote the number of basis functions needed for a particular patch. For the purposes of this paper, we will focus our attention on quadratic and cubic triangles, quadratic Bézier curves, bicubic Bézier tensor patches, cubic Coons patches, and B-spline surfaces.

4.4 Geometric Kernel Problems

In this paper, we show that a variety of geometric problems on polynomial patches can be solved with SOS programming. Our key observation is that most of these can be formulated in a very similar way. In particular, we offer the following template problem:

$$\begin{aligned} f^* &= \min_{\mathbf{u} \in \mathcal{D}} \quad f(\mathbf{u}) \\ \mathbf{u}^* &= \arg\min_{\mathbf{u} \in \mathcal{D}} f(\mathbf{u}). \end{aligned} \tag{4.6}$$

where

$$\mathbb{D} = \{\mathbf{u} \in \mathbb{R}^k : g_i(\mathbf{u}) \geq 0, h_i(\mathbf{u}) = 0\}, \tag{4.7}$$

is a compact semialgebraic domain. In what follows, we will outline how to compute the globally optimal value f^* using the machinery of SOS programming, and we will apply this to a variety of patch problems. In § 4.5, we will discuss how to compute $\mathbf{u}^*$ and determine the correct degree for the relaxation.

To apply SOS methods, we rewrite the problem in terms of an additional variable λ, which acts as a lower bound on $f(\mathbf{u})$ for $\mathbf{u} \in \mathcal{D}$. This is equivalent to asking that the polynomial $f(\mathbf{u}) - \lambda$ be positive for $\mathbf{u} \in \mathcal{D}$. Applying Theorem 1, such a requirement is equivalent to the SOS constraints

$$f - \lambda - \sum_i h_i p_i - \sum_i g_i s_i \in \Sigma, \quad s_i \in \Sigma, \quad p_i \in \mathbb{R}[\mathbf{u}]. \tag{4.8}$$

We have thus relaxed the problem Equation 4.6 into the SOS form

$$\lambda^* = \left\{ \begin{array}{ll} \max\limits_{\lambda \in \mathbb{R}} & \lambda \\ \text{s.t.} & f - \lambda - \sum\limits_i h_i p_i - \sum\limits_i g_i s_i \in \Sigma_d, \\ & s_i \in \Sigma_d \\ & p_i \in \mathbb{R}[\mathbf{u}]_d \end{array} \right\}. \tag{4.9}$$

As d increases, Theorem 1 guarantees that λ^* converges to the globally optimal value f^*. For high-enough d, the relaxation will be accurate to within numerical precision, and as we observe in § 4.6, the convergence is dramatic at a fairly low d.

4.4.1 Optimization over a Polynomial Patch

The template problem Equation 4.6 allows us to optimize arbitrary polynomial objective functions over a polynomial patch. One only needs to choose the g_i that encode a base domain $\mathbb{D}$ along with an appropriate objective function f. We now give several examples of objective functions of interest to geometry processing.

Closest Point (CP) The closest point problem aims to find the minimum distance between a target point $\mathbf{t}$ and a polynomial patch. Let $\mathbf{x}(\mathbf{u})$ be the shape function of the patch. Then one simply chooses the objective function $f(\mathbf{u}) = \|\mathbf{x}(\mathbf{u}) - \mathbf{t}\|_2^2$ in the template problem.

Minimal Axis Aligned Bounding Box (MBB) The axis aligned bounding box problem finds the smallest-volume AABB containing a polynomial patch $\mathbf{x}(\mathbf{u})$. The bounds can be obtained by choosing the objective function $f(\mathbf{u}) = \pm x_i(\mathbf{u})$.

Hexahedron Quality Evaluation The hexahedral quality evaluation problem presented by Marschner et al. [Mar+20] also fits into the template. Here $\mathcal{D}$ is a unit cube, $\mathbf{x}(\mathbf{u})$ encodes trilinear interpolation, and $f(\mathbf{u})$ is the Jacobian determinant of $\mathbf{x}(\mathbf{u})$, i.e., $f(\mathbf{u}) = \det(\nabla_{\mathbf{u}}\mathbf{x})$.

4.4.2 Multiple Patches in One Optimization

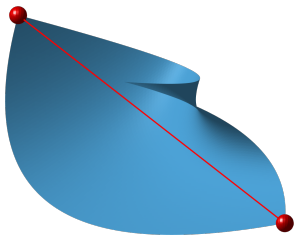
Often it is desirable to optimize an objective over multiple patches, or multiple copies of the same patch, simultaneously. Such problems can often be recast as optimization problems over a higher-dimensional base domain formed from the Cartesian product of the original base domains.

To wit, let polynomial patches $\mathbf{x}^1$, and $\mathbf{x}^2$ map respectively from base domains $\mathbb{D}_1 = \{\mathbf{u}^1 : g_i^1(\mathbf{u}^1) \geq 0\}$, and $\mathbb{D}_2 = \{\mathbf{u}^2 : g_i^2(\mathbf{u}^2) \geq 0\}$ into $\mathbb{R}^n$. Let

$$\begin{aligned} \mathbb{D} &= \{(\mathbf{u}^1, \mathbf{u}^2) : g_i^1(\mathbf{u}^1) \geq 0, g_i^2(\mathbf{u}^2) \geq 0, q_j(\mathbf{u}^1, \mathbf{u}^2) \geq 0\} \\ &\subseteq \mathbb{D}_1 \times \mathbb{D}_2, \end{aligned} \tag{4.10}$$

where the q_j are additional problem-dependent and/or symmetry-breaking constraints. One can then define a new patch $\mathbf{x}(\mathbf{u}^1, \mathbf{u}^2) = (\mathbf{x}^1(\mathbf{u}^1), \mathbf{x}^2(\mathbf{u}^2))$ and rewrite f relative to $\mathbf{x}$ and $\mathcal{D}$. For notational convenience, we will continue to use $\mathbf{u}^1$, $\mathbf{u}^2$, $\mathbf{x}^1$, and $\mathbf{x}^2$ directly.

Patch Diameter (PD) The *diameter* of a patch is the distance between the two most distant points on that patch. This can be found using the template problem with two identical base domains $\mathbb{D}_1 = \mathbb{D}_2$ and maps $\mathbf{x}^1 = \mathbf{x}^2$. Then one simply chooses the objective $f(\mathbf{u}^1, \mathbf{u}^2) = -\|\mathbf{x}^1(\mathbf{u}^1) - \mathbf{x}^2(\mathbf{u}^2)\|_2^2$.



Surface-Surface Intersection (SSI) This problem detects when two polynomial patches intersect each other. As described previously, we use an augmented constraint set consisting of constraints from both patches. We further add the equality constraints $h(\mathbf{u}^1, \mathbf{u}^2) = \mathbf{x}^1(\mathbf{u}^1) - \mathbf{x}^2(\mathbf{u}^2) = 0$. This restricts optimization from the domain $\mathcal{D}_1 \times \mathcal{D}_2$ to its subset on which intersections occur. Feasibility of this problem determines if the two surfaces intersect. We can arbitrarily choose as an objective function $f(\mathbf{u}^1, \mathbf{u}^2) = u_1^1$.

Self-Intersection (SI) The self-intersection problem finds places where a single polynomial patch intersects itself. To fit this into the template problem, we start with the constraints of the SSI problem for two identical domains $\mathcal{D}_1 = \mathcal{D}_2$, $\mathbf{x}^1 = \mathbf{x}^2$. Then we choose the objective to be $f(\mathbf{u}^1, \mathbf{u}^2) = -\|\mathbf{u}^1 - \mathbf{u}^2\|_2^2$. This gives us points that are as far apart as possible in the base domain but map to the same embedded point in $\mathbb{R}^n$.

Continuous Collision Detection (CCD) The Continuous Collision Detection problem aims to find out if and when two patches on a time varying trajectory will intersect in a given time interval. Figure 4-3 shows an example of the CCD problem. Let two patches $\mathbf{x}^1$, $\mathbf{x}^2$ be defined by control points $\mathbf{p}_i^1$, $\mathbf{p}_i^2$ via $\mathbf{x}(\mathbf{u}) = \sum_i^{n_B} \mathbf{p}_i \phi_i$. Let control points have velocities v_i^1 and v_i^2. We then define time-dependent patches by linear interpolation $\mathbf{x}(\mathbf{u}, t) = \sum_i^{n_B} (\mathbf{p}_i + \mathbf{v}_i t)\phi_i$, mapping from the augmented base domain $\mathcal{D}_1 \times \mathcal{D}_2 \times [0, t_{max}]$ to $\mathbb{R}^n$. We wish to determine if the provided velocities will result in a collision within t_{max} units of time.

As in the SSI case, we augment the constraint set with an equality constraint $h(\mathbf{u}^1, \mathbf{u}^2, t) = \mathbf{x}^1(\mathbf{u}^1, t) - \mathbf{x}^2(\mathbf{u}^2, t) = 0$. This restricts the optimization to the subset of the joint base domain on which space-time collisions occur. It just remains to

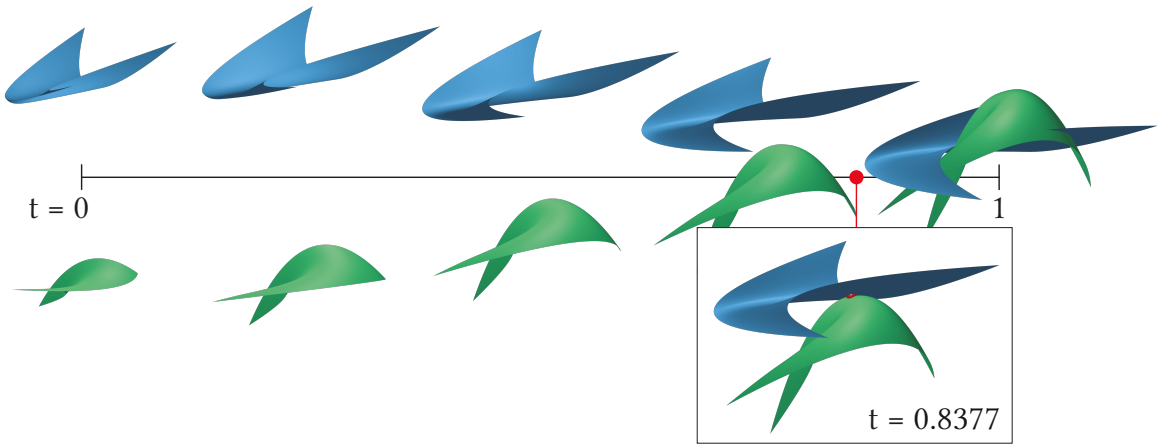


Figure 4-3: The CCD problem is demonstrated on two time-varying quadratic triangle patches over the time interval $[0, 1]$. The patches are shown at the time of first collision, at $t = 0.8377$, in the inset, with the collision point marked in red.

choose $f(\mathbf{u}^1, \mathbf{u}^2, t) = t$ to find the earliest collision.

4.4.3 Outside the Template

Rational Surfaces A natural generalization of polynomial surfaces are *rational surfaces*—surfaces whose shape functions $\mathbf{x}$ are rational. Using control points $\mathbf{p}_i$ and basis functions ϕ_i, rational surfaces are decomposable as follows

$$x_j(\mathbf{u}) = \frac{\sum_i p_{ij}\phi_i(\mathbf{u})}{b(\mathbf{u})} = \frac{a_j(\mathbf{u})}{b(\mathbf{u})}, \tag{4.11}$$

for polynomials $a_j, b, \phi_i \in \mathbb{R}[\mathbf{u}]$. Straightforward application of problem formulations from § 4.4.1 and § 4.4.2 to rational surfaces may seem not to fit the template Equation 4.6. The key observation for applying SOS programming to rational surfaces is that we simply need to clear the denominators. We demonstrate here the SSI problem on rational surfaces, with all other problems following *mutatis mutandis*. The SSI formulation applied to rational surfaces starts with

$$\begin{aligned} \min_{\mathbf{u}^1 \in \mathcal{D}_1,\, \mathbf{u}^2 \in \mathcal{D}_2} \quad & u_1^1 \\ \text{s.t.} \quad & \frac{\mathbf{a}^1(\mathbf{u}^1)}{b^1(\mathbf{u}^1)} = \frac{\mathbf{a}^2(\mathbf{u}^2)}{b^2(\mathbf{u}^2)} \end{aligned} \tag{4.12}$$

By cross-multiplying denominators, the rational constraints are transformed back into polynomial constraints.

$$\begin{aligned} \min_{\mathbf{u}^1 \in \mathcal{D}_1,\, \mathbf{u}^2 \in \mathcal{D}_2} \quad & u_1^1 \\ \text{s.t.} \quad & \mathbf{a}^1(\mathbf{u}^1)\, b^2(\mathbf{u}^2) = \mathbf{a}^2(\mathbf{u}^2)\, b^1(\mathbf{u}^1) \end{aligned} \tag{4.13}$$

Thus problems that can be solved on polynomial patches via SOS programming can be extended to rational surfaces.

Minimal Enclosing Ellipsoid (MEE) The Minimal Enclosing Ellipsoid problem finds the ellipsoid of smallest volume that fully contains a polynomial patch. Smallest ellipsoids enclosing point sets have a long history in convex optimization, where they are known as *Löwner-John ellipsoids* [Tod16]. The MEE of a finite collection of points or ellipsoids can be computed exactly via SDP. This involves a reduction from the problem of computing the smallest ellipsoid with arbitrary center to the case of an ellipsoid centered at the origin. We will combine this reduction with SOS methods to compute the MEE of a polynomial patch.

We can parameterize an ellipsoid in $\mathbb{R}^3$ by a positive definite matrix $A \in \mathcal{S}^3_{++}$ and center point $\mathbf{c} \in \mathbb{R}^3$:

$$\begin{aligned} \mathcal{E}(A, \mathbf{c}) &= \{\mathbf{p} \in \mathbb{R}^n : (\mathbf{p} - \mathbf{c})^\top A (\mathbf{p} - \mathbf{c}) \leq 1\}, \\ \mathrm{Vol}(\mathcal{E}(A, \mathbf{c})) &\propto \det(A^{-1}) = (\det A)^{-1}. \end{aligned} \tag{4.14}$$

Given $\mathcal{D}$ and $\mathbf{x}$ describing a 3D polynomial patch, the MEE can be computed as the solution to

$$(A^*, \mathbf{c}^*) = \left\{ \begin{array}{ll} \operatorname{argmin}_{A \in \mathcal{S}^3_{++},\, \mathbf{c} \in \mathbb{R}^3} & \mathrm{Vol}(\mathcal{E}(A, \mathbf{c})) \\ \text{s.t.} & 1 \geq \max_{\mathbf{u} \in \mathcal{D}} (\mathbf{x}(\mathbf{u}) - \mathbf{c})^\top A (\mathbf{x}(\mathbf{u}) - \mathbf{c}) \end{array} \right\} \tag{4.15}$$

The problem in this form does not clearly fit into the template due to the inner maximization, which can also be viewed as a universal quantification. But as we will

now show, we can still apply SOS optimization to obtain the MEE. The first step is to reduce the general MEE problem Equation 4.15 to an equivalent *centered* MEE problem in one higher dimension:

$$B^* = \left\{ \begin{array}{ll} \underset{B \in \mathcal{S}^4_{++}}{\text{argmin}} & \text{Vol}(\mathcal{E}(B, 0)) \\ \text{s.t.} & 1 \geq \max\limits_{\mathbf{u} \in \mathcal{D}} \mathbf{y}^\top(\mathbf{u}) B \mathbf{y}(\mathbf{u}) \\ & \mathbf{y}(\mathbf{u}) = \begin{bmatrix} \mathbf{x}(\mathbf{u}) \\ 1 \end{bmatrix} \end{array} \right\} \tag{4.16}$$

From B^*, one can recover A^* by completing the square [Tod16]. The constraints of Equation 4.16 require that the polynomial $1 - \mathbf{y}^\top B \mathbf{y} \in \mathbb{R}[\mathbf{u}]$ be nonnegative for all $\mathbf{u} \in \mathcal{D}$. Theorem 1 allows us to encode this requirement with SOS constraints, resulting in the following SOS program.

$$B^* = \left\{ \begin{array}{ll} \arg\min & \log(\text{Vol}(\mathcal{E}(B, 0))) \\ \text{s.t.} & s_i \in \Sigma, \quad s_i \in \mathbb{R}[\mathbf{u}]_d \\ & 1 - \mathbf{y}^\top B \mathbf{y} - \sum\limits_i g_i s_i \in \Sigma \\ & B \in \mathcal{S}^4_{++} \end{array} \right\}, \tag{4.17}$$

where we have also convexified the objective by taking its log. Both the objective and constraints are now convex, and we can solve Equation 4.17 to global optimality, and by extension the MEE problem.

Our ability to solve the MEE problem hinges on the fact that MEE has an equivalent centered formulation wherein $\mathbf{c} = 0$. Consider what would happen if we tried to relax Equation 4.15 directly. We would require $1 - (\mathbf{x} - \mathbf{c})^\top A (\mathbf{x} - \mathbf{c}) \geq 0$ for $\mathbf{u} \in \mathcal{D}$. Our constraints would contain bilinear terms like $\mathbf{x}^\top A \mathbf{c}$ and linear-quadratic terms like $\mathbf{c}^\top A \mathbf{c}$, thus breaking convexity. While our solution to the MEE problem does not generalize directly to other problems outside the template, it illustrates what can and cannot be handled by SOS programming.

Minimal Surrounding Sphere (MSS) The Minimal Surrounding Sphere problem finds the sphere of smallest volume that fully contains a polynomial patch. This can be solved with a minor modification to Equation 4.17. Let B_3 be the top left 3×3 block of B. We simply add the constraint that $B_3 = aI_3$, where I_3 is the 3×3 identity matrix and a is a scalar variable. This additional constraint reduces the MEE program to an MSS program without affecting convexity.

4.5 Moment Relaxation in Theory and Practice

While the previous section lists a variety of problems that can be solved with SOS programming, two points remain. First, while the SOS approach of the previous section gives us a way to compute the optimal value of Equation 4.6 by way of Equation 4.9, we do not have a way to compute the arg min of Equation 4.6. Using CP as an example, we can find the distance from a target point to its projection on a patch, but do not have the projected point. Second, we have not mentioned how to certify correctness of a solution and determine if d is large enough for Theorem 1 to hold. This section will address both points.

4.5.1 Moment Relaxation in Theory

Consider an alternative formulation of Equation 4.6 as a *measure relaxation*:

$$f^* \coloneqq \min_{\mu \in \mathcal{P}(\mathcal{D})} \mathbf{E}_\mu[f], \tag{4.18}$$

where $\mathcal{P}(\mathcal{D})$ is the space of probability measures on $\mathcal{D}$, and $\mathbf{E}_\mu[f] = \int_{\mathcal{D}} f \, d\mu$ denotes integration against μ. If $\mathbf{u}^* = \arg\min_{\mathbf{u} \in \mathcal{D}} f(\mathbf{u})$, then the optimum of Equation 4.18 is achieved by placing all the mass of μ at $\mathbf{u}^*$—that is, the delta measure $\delta_{\mathbf{u}^*}$ minimizes Equation 4.18 [Las01]. While this relaxation convexifies Equation 4.6, optimization over the infinite-dimensional space of measures $\mathcal{P}(\mathcal{D})$ is intractable. To realize the measure relaxation computationally, μ must be represented by its moments. Using the multi-index $\boldsymbol{\alpha}$ to index monomial exponents of f, the $\boldsymbol{\alpha}$-moment of μ is defined as

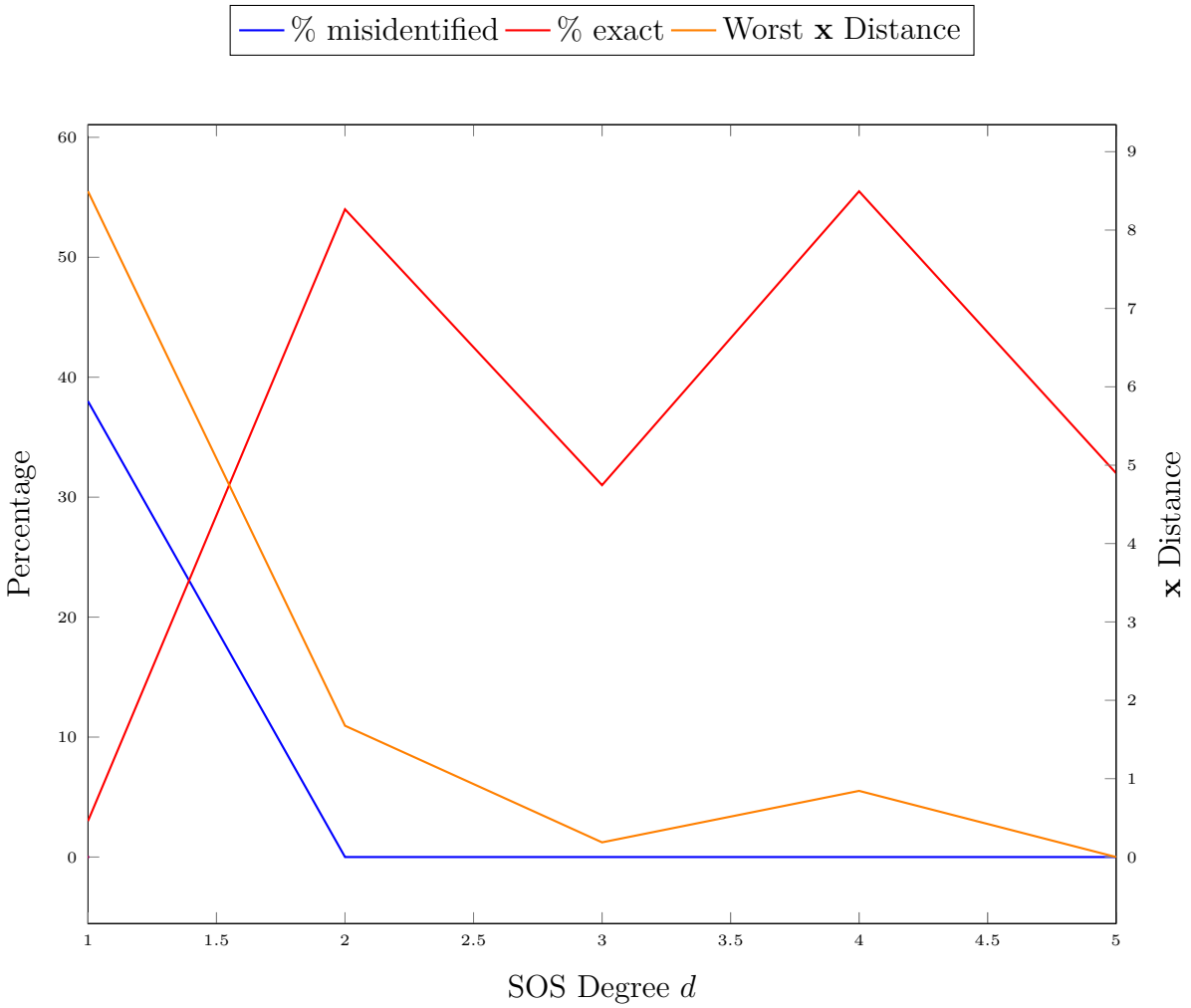


Figure 4-4: For the SSI problem on quadratic triangles, all intersections are correctly identified for degree $d \geq 2$, and the distance between extracted points of intersection drops to 6.7e-05 at degree $d = 5$, even though exact recovery does not yet occur at these degrees. Triangle control points were sampled from i.i.d. standard normal distributions.

$\mu_{\boldsymbol{\alpha}} \coloneqq \mathbf{E}_\mu[\mathbf{u}^{\boldsymbol{\alpha}}]$. Let $f_{\boldsymbol{\alpha}}$ be the coefficient of $\mathbf{u}^{\boldsymbol{\alpha}}$. Then the expectation in Equation 4.18 can be expressed as

$$\mathbf{E}_\mu[f] = \sum_{\boldsymbol{\alpha}} f_{\boldsymbol{\alpha}} \mu_{\boldsymbol{\alpha}}, \tag{4.19}$$

a simple linear function of *finitely many* moments. It remains to constrain $\mu_{\boldsymbol{\alpha}}$ to be valid moments of a distribution over $\mathcal{D}$, a task fulfilled by the following theorem:

Theorem 2 ([Las01], Theorem 4.2)**.** Suppose $\mathcal{D}$ is a compact domain with algebraic certificate of compactness. Let d be large enough for $f(\mathbf{u}) - f^*$ to admit the decomposition in Equation 4.3. Then the following **moment relaxation** SDP computes f^*:

$$f^* = \left\{ \begin{array}{lll} \min\limits_{\boldsymbol{\mu} \in \mathcal{M}_d} & \mathbf{E}_{\boldsymbol{\mu}}[f(\mathbf{u})] & \\ \text{s.t.} & \mathbf{E}_{\boldsymbol{\mu}}[q(\mathbf{u})^2] \geq 0, & \forall q \in \mathbb{R}[\mathbf{u}]_{\lceil d/2 \rceil} \\ & \mathbf{E}_{\boldsymbol{\mu}}[q(\mathbf{u})^2 g_i(\mathbf{u})] \geq 0, & \forall q \in \mathbb{R}[\mathbf{u}]_{\lceil d/2 \rceil - w_i} \\ & \mathbf{E}_{\boldsymbol{\mu}}[1] = 1, & \end{array} \right\} \tag{4.20}$$

where $w_i = \lceil \deg g_i / 2 \rceil$, and $\mathcal{M}_d$ is the space of moment vectors up to degree d. Furthermore, the moment vector corresponding to $\delta_{\mathbf{u}^*}$ minimizes Equation 4.20.

We now have two SDP-based approaches to solving the same polynomial optimization problem—one derived by SOS programming, and one via moment relaxation. As the reader may have guessed, the two SDPs are dual to each other [Las01].

If the moment vector $\boldsymbol{\mu}$ solving Equation 4.20 corresponds to a valid probability measure, then we know the relaxation was tight and we have achieved the global optimum of the original measure relaxation Equation 4.18, and in turn of the polynomial problem Equation 4.6. Verifying this when the optimal measure is a delta measure is particularly simple—this will be the case if and only if the semidefinite matrix corresponding to the first constraint in Equation 4.20 has rank one. This phenomenon is known as *exact recovery*, and it provides an *optimality certificate* verifying that d was chosen sufficiently high for the relaxed problem to solve the original problem Equation 4.6. Finally, when we have exact recovery, $\mathbf{u}^*$ can be recovered as the mean

Table 4.1: Overview of SOS results on quadratic and cubic triangle patches. For each problem, we randomly generate 1000 test cases. Specifically for CCD on cubic triangles we test on a reduced set of 300 cases. We show the minimum SOS polynomial degree d_{correct} for which all problem instances are solved correctly. This degree is generally less than the degree d_{exact} required for exact recovery. For problems where it is applicable (SSI,SI,CCD), we provide the percentage of the data that intersects or collides. We also provide the number of optimization variables for each problem, which indicate the dimensions of the ultimately solved SDP problem. Additionally, we provide the median runtimes over 50 instances of each problem with standard deviations. We separate out the time spent in YALMIP converting to MOSEK's input form, and the time spent in MOSEK actually solving the SDP problem. CCD on cubic triangles takes the longest, which is unsurprising given the large number of optimziation variables. Lower-degree problems like MBB and MSS are already fast enough to be used on large meshes out of the box. *For SI, 100% exact recovery is taken out of cases where the patch did self-intersect. Exact recovery is not expected if the patch does not self-intersect.

	d_{correct}		% Correct		% Exact		% Intersects		# Vars		YALMIP Time (s)		MOSEK Time (s)	
Triangle $d_{\mathbf{x}}$	2	3	2	3	2	3	2	3	2	3	2	3	2	3
CP	3	5	100	100	100	100	-	-	31	64	.56±.04	.52±.04	.14±.02	.10±.08
MBB	2	4	100	100	100	100	-	-	19	46	.48±.02	.49±.02	.002±.0003	.004±.0007
PD	4	6	100	100	100	100	-	-	491	1471	.49±.02	.5±.02	.05±.009	1.24±.19
SSI	5	6	100	100	27	0	60	85	757	1261	.6±.02	.66±.08	.78±.12	6.19±1.4
SI	4	4	100	100	100*	0	51	84	491	491	.38±.02	.5±.03	.29±.06	.39±.09
CCD	5	6	100	100	1	0	82	90	2017	3697	.69±.04	1.31±.09	25.9±4.4	699±49
MSS	2	4	100	100	-	-	-	-	36	63	.41±.03	.38±.008	.006±.002	.012±.002
MEE	4	6	100	100	-	-	-	-	68	107	.42±.04	.37±.03	.13±.02	.13±.01

(i.e., vector of first moments) of $\delta_{\mathbf{u}^*}$:

$$\mathbf{u}^* = (\mu_{\mathbf{u}_1}, ..., \mu_{\mathbf{u}_k}) \tag{4.21}$$

Revisiting PD and SI With Equation 4.21 we are equipped to extract $\mathbf{u}^*$ for problems with a unique global minimum. Problems like PD, however, have multiple global minima due to the exchange symmetry $\mathbf{u}_1 \leftrightarrow \mathbf{u}_2$, which leaves the objective function unchanged. Such a symmetry means that the computed optimum $\boldsymbol{\mu}$, even if it corresponds to a valid measure, will not in general be a delta measure, making exact recovery elusive. Moreover, the mean extraction Equation 4.21 will yield the Euclidean mean of optimal values, which may not itself be optimal.

To address these issues, we break the exchange symmetry by adding the generic constraint

$$g(\mathbf{u}_1, \mathbf{u}_2) = (\mathbf{u}_1 - \mathbf{u}_2) \cdot \vec{v} \geq 0, \tag{4.22}$$

where $\vec{v}$ is a randomly sampled unit vector. With this modification, generic uniqueness of the arg min is restored, allowing for exact recovery.

SI has the same symmetry and requires the same symmetry-breaking constraint. Furthermore, exact recovery is only possible for SI if the patch has a self-intersection. Otherwise, any pair of points $\mathbf{u}_1 = \mathbf{u}_2 \in \mathcal{D}$ are globally optimal with an optimal value of 0. Despite the possible lack of exact recovery, we address in § 4.5.2 how to solve problems like SI reliably in practice.

4.5.2 Moment Relaxation in Practice

A general algorithm to achieve exact recovery might be to incrementally increase the SOS degree d from 1 until exact recovery is achieved. While this strategy may succeed eventually, Marschner et al. [Mar+20] observe that moment relaxations of a specific problem class tend to achieve exact recovery at the same degree, independent of the specific numbers involved. In particular, they observe that for a trilinear hexahedron, the hexahedron validity problem consistently achieves exact recovery

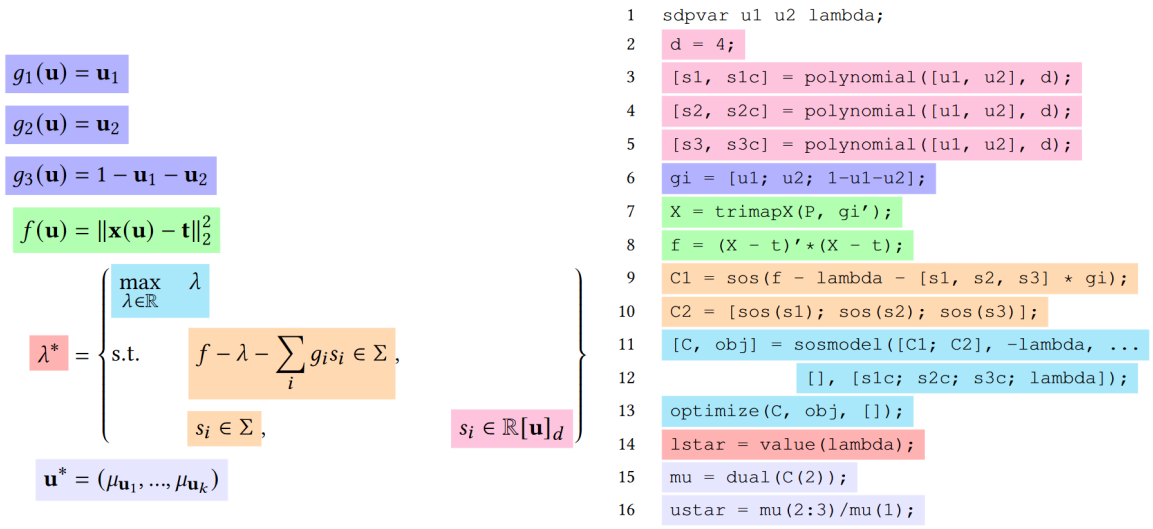


Figure 4-5: Conversion from Equation 4.9 to functioning YALMIP code for CP. The YALMIP code matches the mathematical formulation almost line to line. After specifying the the primal SOS problem, the dual moment vector and $\mathbf{u}^*$ are easy to extract. To solve other problems on different patches, only `gi`, `trimapX` and `f` need to be changed.

with $d = 4$. Given a problem type, one might determine d empirically by generating many randomized instances of the same problem and increasing d until these generic instances can be solved to exact recovery. Then the empirically-determined degree can be applied to untested instances of the same problem.

Curiously, we observe that it is frequently possible to obtain f^* and $\mathbf{u}^* \in \mathcal{D}$ even without exact recovery. When d is smaller than required for exact recovery, the moment vector μ does not encode a delta measure. Nevertheless, we can take its mean following Equation 4.21 and treat that as a proxy for $\mathbf{u}^*$. While this leaves the realm of formal SOS theory, we find in practice that this strategy is extraordinarily robust. Consider the SSI problem on quadratic triangles in Table 4.1. Choosing degree $d = 5$ provides us with exact recovery for only 27% of the instances of this problem. However, we can correctly detect intersection 100% of the time, and our extracted $\mathbf{u}^*$ gives us the correct intersection points 100% of the time. The same phenomenon is observed for SI and CCD—exact recovery is not required to obtain the correct solution.

Thus we distinguish the degree d_{exact} at which exact recovery is achieved from

the degree d_{correct} at which a correct solution is achieved. In problems like SSI where numerical verification of intersection is easy given the candidate points of intersection, having an accurate $\mathbf{u}^*$ is just as effective a certificate of intersection as exact recovery. Since the cost of solving an SOS program is primarily dictated by d, it is extremely fortunate that for practical purposes one does not require exact recovery.

We demonstrate in Figure 4-4 the percentage of correctly solved SSI problems out of 200 for each of $d \in 1, 2, 3, 4, 5$ using randomly sampled quadratic triangles. Our sampling strategy is described in § 4.6.2. 128 of these problems have intersections. Out of problems with intersections, we plot the percentage of them that exhibit exact recovery for each degree. This percentage never reaches 100% and our key observation is that it does not need to! We do not require exact recovery to obtain the correct solution. We plot the percentage of problems where the SOS formulation misidentifies whether an intersection exists. At degree only $d = 2$, the percentage misidentified is already zero. Thus we are already able to determine whether a pair of quadratic triangles intersect with 100% accuracy. Finally, for problems that have intersections, using Equation 4.21 we can extract the points on each surface that intersect. When the SSI problem is correctly solved, their distance must be 0. We plot the maximum distance between the extracted points over all intersecting examples at each degree. At $d = 5$, the points of intersection are identified with high precision. While these problem instances do not exhaustively cover SSI, they provide very strong empirical evidence that exact recovery is not required.

4.6 Results and Applications

4.6.1 Implementation

Modern problem modeling languages such as YALMIP [Löf04] allow one to specify an SOS program in the concise form Equation 4.9, automatically converting it to a primal-dual SDP formulation from which one can obtain the moment vector μ of Equation 4.20. Figure 4-5 demonstrates how easy it is to specify and solve the primal

and dual problems. The details of converting SOS or moment constraints into an SDP can be entirely left to the modeling language.

All batch experiments were run on an Intel i7-8700K CPU @ 3.70GHz with 16 GB of RAM. Problems were all written in MATLAB 2021a using YALMIP version 20200116 to formulate the problems and MOSEK 9.2 to solve. Default modeling and solver parameters were maintained. Table 4.1 lists the median runtimes and standard deviations for 50 random instances of each problem for quadratic and cubic triangles. Time spent in YALMIP converting the problem into a form that MOSEK can handle does not vary significantly across problems or patch types. Time spent actually solving the problem in MOSEK varies much more. Clear trends are that solver time increases for each problem with SOS degree, which increases with patch degree. While CCD seems to be a significantly challenging problem, MBB is fairly easy. For most problems, the table shows that the majority of the time is spent in YALMIP. This time can be cut out entirely by formulating the problem directly as an SDP problem, or by taking advantage of YALMIP's ability to precompile problems. We expect the runtime can be further improved by engineering tailor made optimization strategies for these problems.

4.6.2 SOS Problems Overview

A key parameter of our SOS formulations is the degree d. Since it controls the size of the SDP in Equation 4.2, we naturally want to choose a small d. However, if d is too small, the SDP may not successfully solve the unrelaxed problem. In Table 4.1, we show the minimal degree d_{correct} required to solve each problem for 100% of our test cases on various triangular patches. As mentioned in § 4.5.2, this is not necessarily the same degree d_{exact} needed to achieve exact recovery.

For each problem, 1000 test instances were generated, except for CCD on cubic triangles, for which we used 300 instances. For CP, MBB, PD, SI, SSI, MSS, and MEE, we randomly sampled quadratic and cubic triangles by picking all dimensions of all control points independently according to a normal distribution with variance 1. For CCD, pairs of triangles are sampled the same as before, but then shifted by

$(0, 0, \frac{\sqrt{2}}{2})$ and $(0, 0, -\frac{\sqrt{2}}{2})$ respectively. Velocity vectors are sampled the same way, but shifted by $(0, 0, -\frac{\sqrt{2}}{2})$ and $(0, 0, \frac{\sqrt{2}}{2})$ respectively. This is done to increase the number of problem instances that successfully collide without starting out in an intersecting configuration. With this sampling strategy, 82% (90%) of the sampled quadratic (cubic) triangles collide in spacetime while 22.5% (28%) of configurations start out intersecting. We verify correctness of the SOS solutions by comparing to the same problem solved on a linearization of the polynomial patch. Patches are uniformly subdivided so that each edge is split into 10 segments. We then check that solutions for the SOS solution and the linearization match up to a threshold of 10^{-2} in both the parametric and embedded spaces. In a few cases where they do not match, we increase density of the linearization until they do.

Table 4.1 demonstrates how challenging it can be to guess d_{correct} in advance. The SOS degrees we end up with are not clearly correlated with the degrees of the objective or constraints. The clearest trend is that d_{correct} increases with the degree of the shape function $d_{\mathbf{x}}$. There does not seem to be a monotonic relationship between d_{correct} and the degree of the objective function. For example, CP has an objective degree of $2d_{\mathbf{x}}$ while needing fairly low d_{correct}. On the other hand, CCD has an objective degree of just 1 but needs a much higher d. Even if we consider the constraint degrees as well, the degree of the CCD constraint is only $d_{\mathbf{x}} + 1$ which is still less than CP's objective degree $2d_{\mathbf{x}}$.

4.6.3 Closest Point

As described in 4.4.1, our method of formulation is applicable to the problem of finding the closest point on a polynomial patch to a target point.

Applying this method for all patches in a model lets us find the closest point on the model to a target. In Figure 4-6, we show this procedure applied to a collection of points surrounding the bicubic Bézier tensor patch teapot model, where $d_{\mathbf{x}} = 6$ and we chose $d = 5$. We can solve an analogous problem in one lower ambient dimension, finding the closest point to a Bézier curve in $\mathbb{R}^2$. This formulation allows for the creation of Voronoi diagrams of 2D objects formed from Bézier curves, as shown in

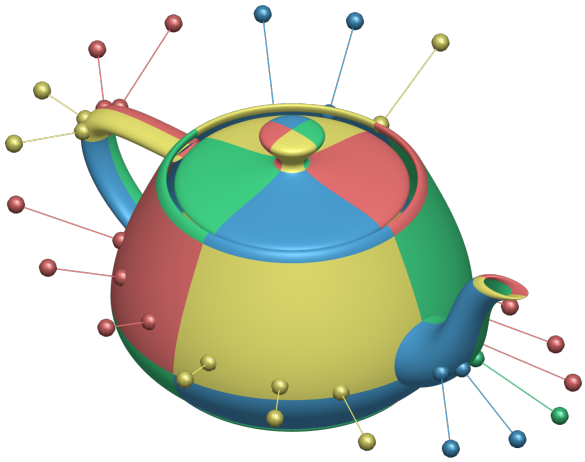


Figure 4-6: Closest point projections onto a bicubic Bézier tensor patch teapot.

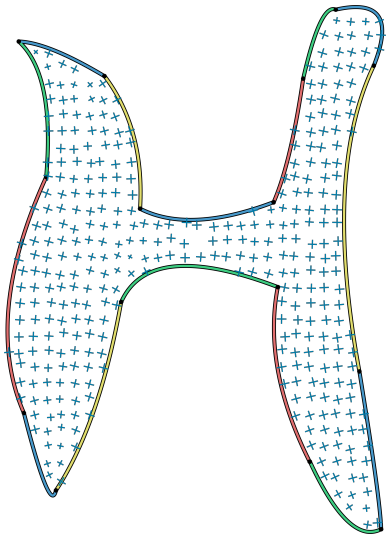


Figure 4-7: Cross field generated via walk-on-spheres

Figure 4-8. In this figure, the 2D objects are letters consisting of quadratic Bézier curves, where $d_{\mathbf{x}} = 2$ and $d = 3$. We color points in the 2D domain by which curve segment they are closest to, forming curve-segment Voronoi cells.

Closest point queries allow us to build more complex operations on domains with curved boundaries. For example, the Monte Carlo walk-on-spheres procedure introduced to geometry processing in [SC20] builds a stochastic solver for linear PDE out of closest point queries. Using the SOS formulation of CP as a building block, such a procedure can solve a PDE on a domain defined by polynomial bounding curves with exact boundary conformation, without linear remeshing. In Figure 4-7,

we demonstrate walk-on-spheres applied to computing a boundary aligned cross-field. Each frame is computed by averaging complex fourth powers sampled from 100 random walks, each stopping when it arrives within 10^{-3} of the boundary. No mesh of the domain interior is required, and in principle, samples can be adapted to resolve singularities precisely.

4.6.4 Intersections

It is worth revisiting the intersection problems (SSI, SI, CCD) in Table 4.1. First, for SI on quadratic triangles, $d_{\text{correct}} = d_{\text{exact}}$. The same property does not extend to cubic triangles or SSI on quadratic triangles since the percentage that achieve exact recovery is different from the percentage that have intersection.

An interesting feature of SI is that the degree d_{correct} is the same for both quadratic and cubic triangles. This is surprising given that for other problems the cubic triangles require higher d_{correct}. We note here that CCD was almost the same in that 93% of cubic triangle CCD instances succeeded using $d = 5$ with runtime comparable to CCD for quadratic triangles. Only for the remaining percentage was $d = 6$ necessary. Due to the large time cost of CCD on cubic triangles with $d = 6$, one can imagine an optimization where $d = 5$ is used first to prune cases where a collision happens and is certifiably found. Only in the absence of this collision would one resort to $d = 6$.

In Figure 4-10 we demonstrate application of SI and SSI tests to airplane meshes from Smirnov, Bessmeltsev, and Solomon [SBS20]. These models use cubic Coons patches for which $d_{\mathbf{x}} = 4$. The SI and SSI problems are solved with $d = 4$. We are able to find intersections for both SI and SSI and provide their exact locations. This enables users in a CAD pipeline to discover and manually fix problems in a design.

In Figure 4-9 we demonstrate application of the SOS relaxed CCD problem to detect collision of a rigid bicubic Bézier tensor patch teapot and elephant. Velocities were chosen to guarantee collision within 1 unit of time. This problem was solved with $d = 4$. Using CCD we are able to find the earliest instance of collision, the patches that collide, and the location of their collision. With the location of the collision we can get exact surface normals at the collision point, which can then be passed onto

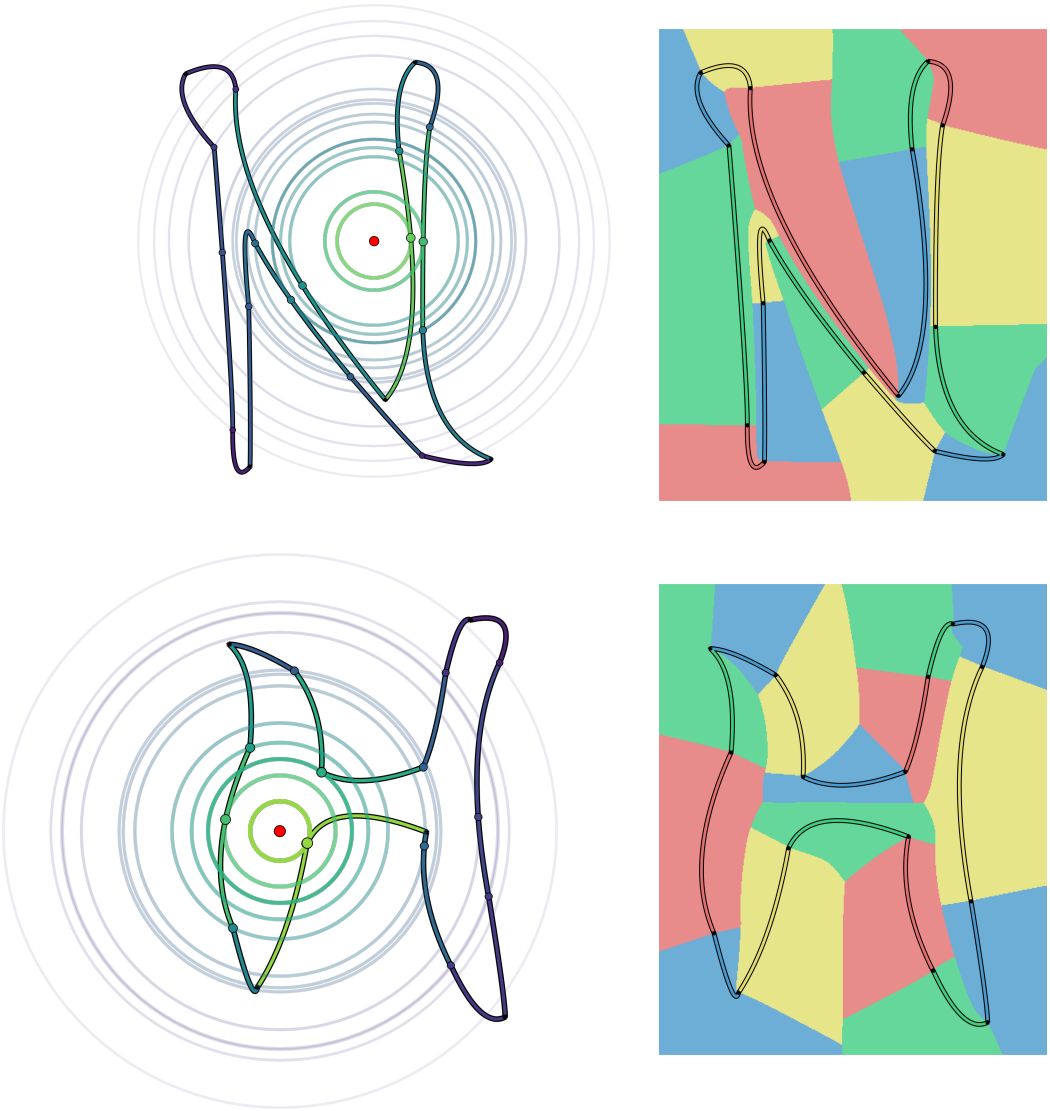
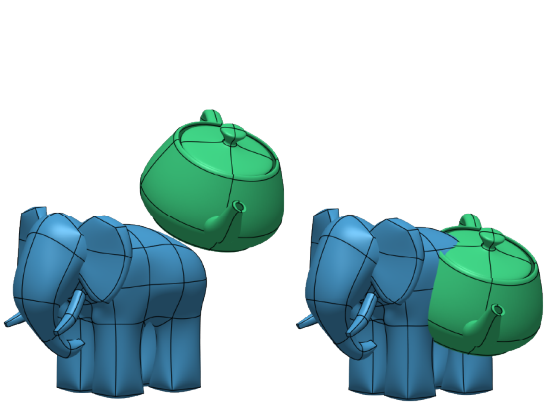
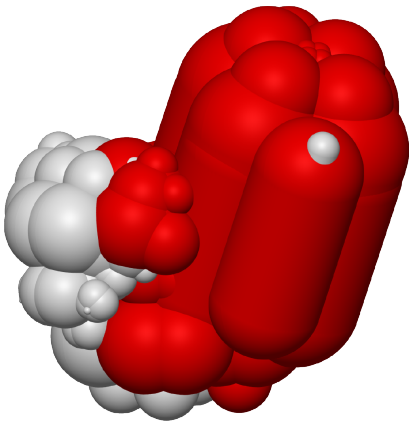


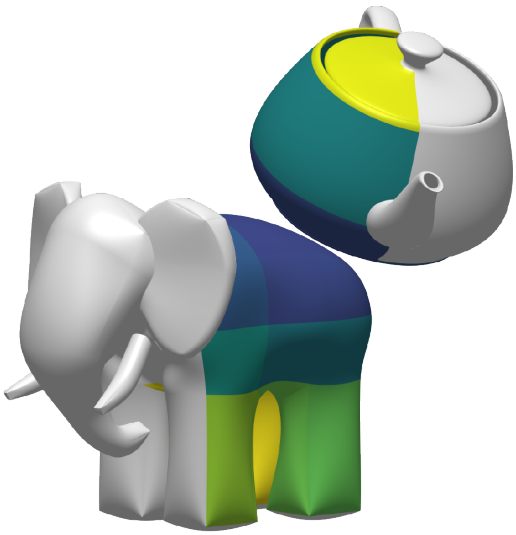
Figure 4-8: Curve segment Voronoi diagrams for the glyphs H and N from [Smi+20], each composed of 15 quadratic Bézier curves. On the left, we show the calculation of the closest point on each of the curve segments to the red point, with the distance to each curve represented by a circle. On the right, we show the Voronoi diagram, computed by finding the curve with the minimum distance to each point in a 500 by 500 grid around each letter.



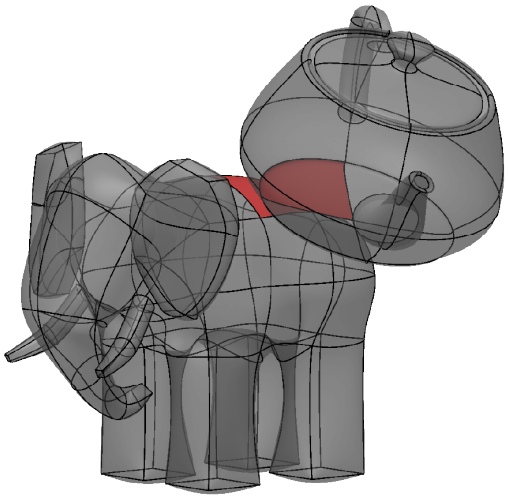
(a) Problem Setup



(b) Bounding Sphere CCD

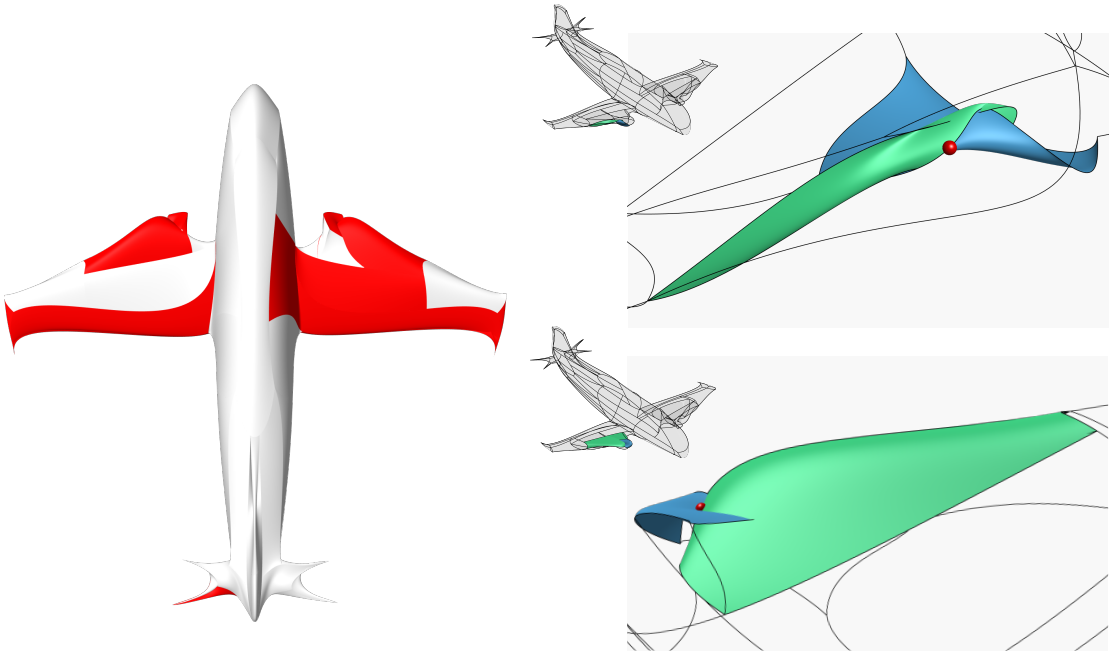


(c) Exact CCD

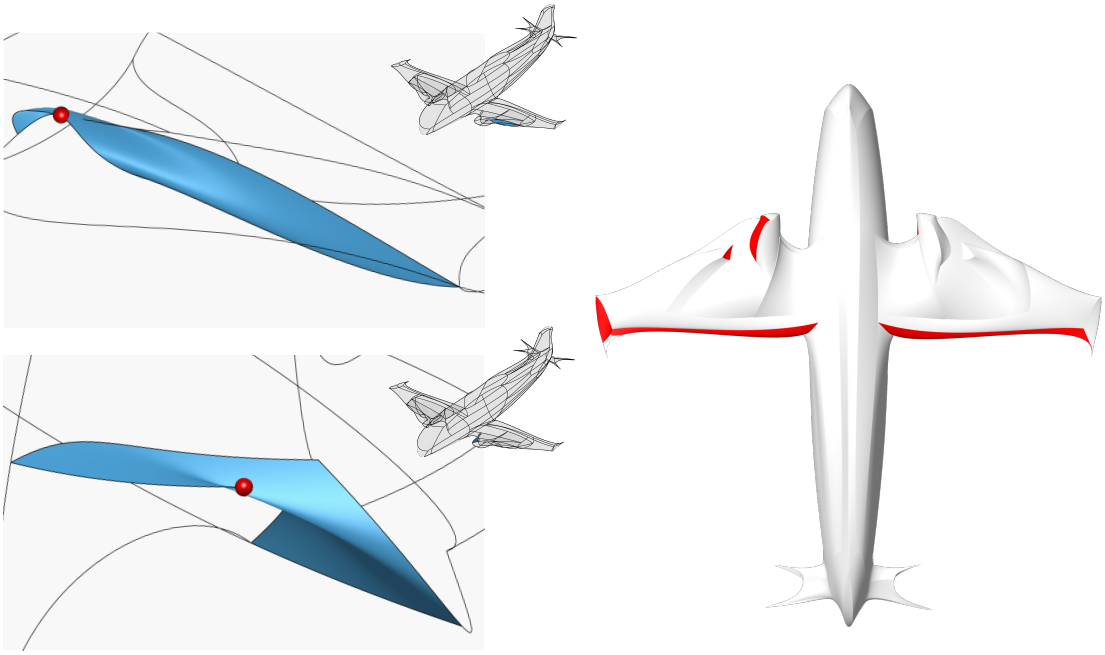


(d) First Intersection

Figure 4-9: We show the application of our CCD solution to the problem shown in (a), where the teapot moves rigidly into the elephant. The teapot and elephant models are comprised of 32 and 128 bicubic Bézier tensor patches, respectively. To accelerate the calculation of the collisions, we first use our MSS method to pre-compute a set of bounding spheres for each model, and use these to determine which pairs of patches may intersect. In (b) we color in red the bounding spheres of the elephant and bounding capsules of the teapot for which at least one intersection was found. We then solve the SOS patch CCD problem on the remaining candidate intersections. The results of this are shown in (c), where each patch is colored based on the time at which it first collides (with darker colors representing earlier times). We can also determine the exact moment in time when the two models first intersect, shown in (d).



Surface-Surface Intersections



Self-Intersections

Figure 4-10: Our method for finding self-intersections and surface-surface intersections applied to an invalid airplane mesh, obtained from the pipeline of [SBS20]. In the top row, we show on the left all the patches that intersect with another patch in red. On the right, two of those surface-surface intersections are shown, with a red point marking the intersection point calculated by our method. In the bottom right, all the patches that include self-intersections are shown in red. Two of these self-intersections are shown on the left, with the intersection points plotted in red.

later steps in a simulation pipeline such as collision response.

4.6.5 Bounding Volumes

Bounding volumes are useful to speed up various intersection-type problems. They can be used in ray tracing to quickly detect if a ray will not intersect an object. They can also be used to quickly detect if rigid objects will not collide. The tighter a bounding volume is, the more non-collisions/intersections can be quickly pruned. On the other hand, employing a more complex bounding volume shape can result in increased computation and updating costs in cases when bounding volumes cannot be precomputed. As such, there is a tradeoff between the computation saved by having a tighter bounding volume and the cost of maintaining the bounding volume itself. SOS optimization is flexible enough to target multiple points along this tradeoff e.g. MBB, MSS, and MEE. In Figure 4-11 we compute the MBB, MSS, and MEE evaluated on bicubic Bézier tensor patches of the teapot mesh. As expected the total volume of MEEs is less than that of MSSs or that of MBBs. The MBB problem fits into the framework of § 4.4.1, and so we can generically expect exact recovery for high enough d. The MSS and MEE problems are different in that the MSS (MEE) generically intersects the polynomial patch it bounds at multiple points. Thus we do not ever expect exact recovery. Nevertheless, we can obtain the minimal volume and the parameters of the MSS (MEE) achieving that volume from Equation 4.17.

In Figure 4-9, we demonstrate the MSS on rigid bicubic Bézier tensor patches as a way to shorten computation time of CCD problems between meshes. Instead of running CCD between all pairs of patches, we can drastically cut down computation time by precomputing the MSS for each patch. It is straightforward to check if two moving spheres will collide or not, allowing us to reduce the number of CCD computations required.

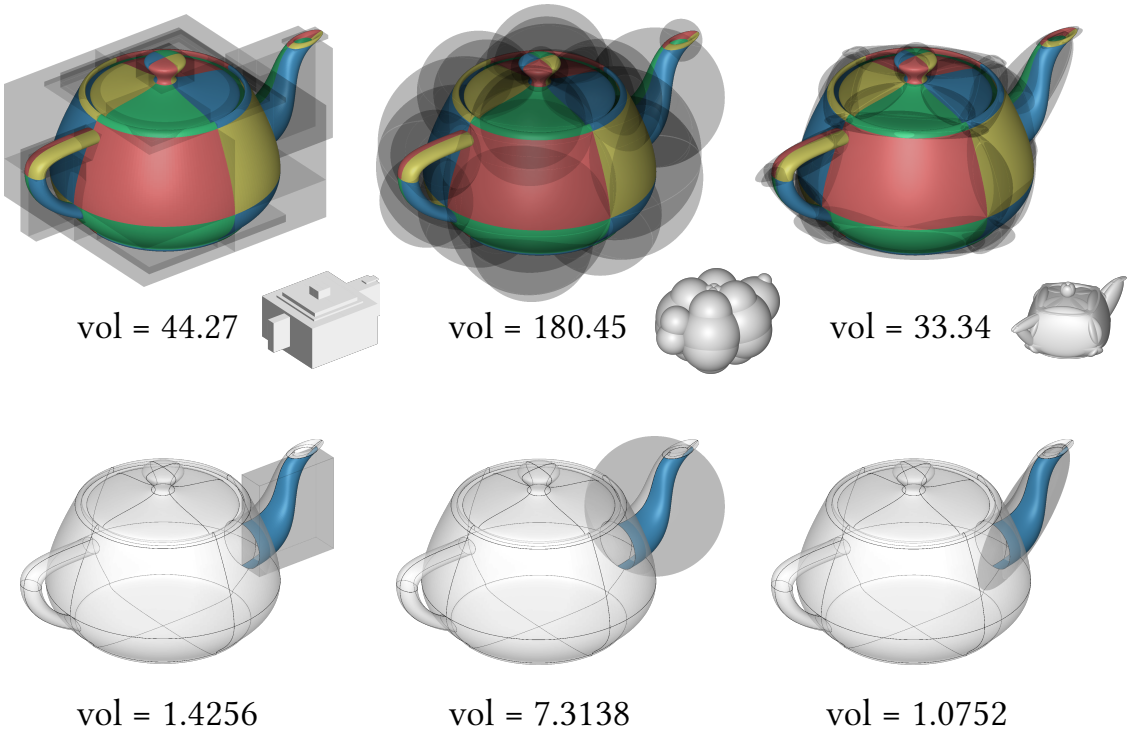


Figure 4-11: In the top row, we show a comparison of the MBB, MSS, and MEE for the bicubic Bézier tensor patch teapot. The total volume of bounding elements is computed for each type; ellipsoids provide the tightest bounding volume. The bottom row shows a comparison of the three different bounding types for an individual patch.

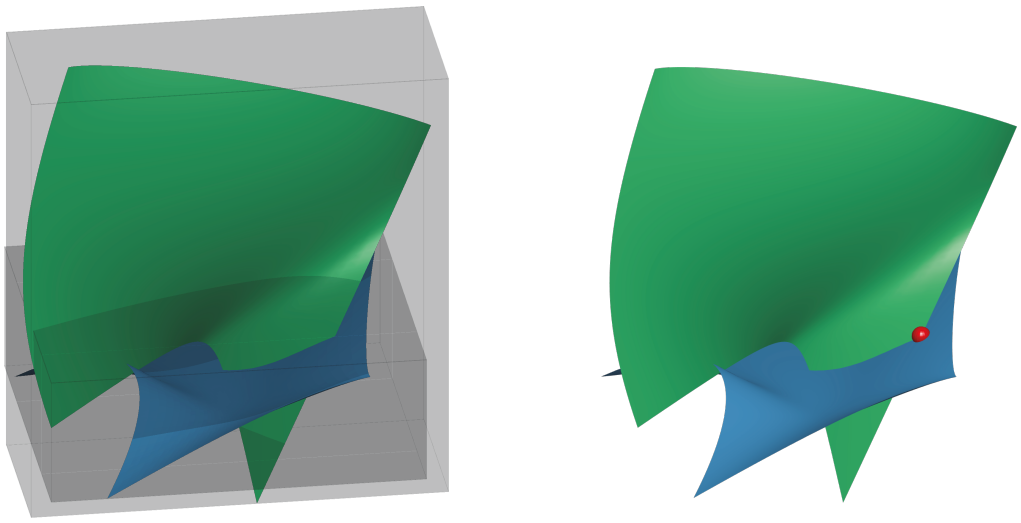


Figure 4-12: The MBB and SSI problems solved on generic NURBS patches. On the left, two bounding boxes are drawn around the blue and green patches. On the right, the red point indicates where these patches intersect.

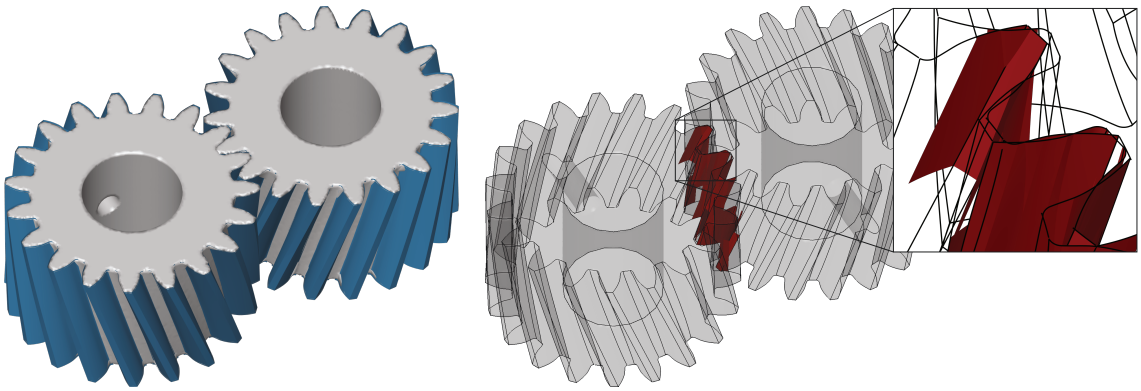


Figure 4-13: This helical gear, from the ABC dataset [Koc+19], is modeled with NURBS surfaces defining its teeth, shown in blue in the left figure. To determine whether the two gears intersect, we can apply our SSI formulation to the NURBS surfaces in the two gears. To speed up the runtime, we first compute the bounding boxes of each patch with our MBB method and use these to identify patches which may intersect. The patches in the two gears that intersect are shown in red on the right.

4.6.6 Rational Surfaces

We demonstrate application of SSI to NURBS in Figure 4-13 on gears obtained from [Koc+19]. Intersecting teeth are easily detected and highlighted. We also demonstrate CCD in Figure 4-1 on the castle from [TCL21] to find which patches of the rocket will first collide with patches of the castle. These collisions could then be fed back into their elasticity simulation to render colliding elastic objects without need for linearization or volumetric remeshing.

While handling of rational surfaces allows us to process generic NURBS surfaces, we find that many of the models in [Koc+19] and [TCL21] have mainly patches with constant denominators $b = 1$, making them simply B-Splines. The primary exceptions in these datasets are patches that comprise spherical or cylindrical features. In Figure 4-12, we show MBB and SSI on generic rational surfaces without constant denominator computed using SOS programming.

4.7 Discussion

Many problems in geometry processing suffer from nonlinearity, forcing users to cope with only locally optimal solutions. Even when one does succeed in reaching global optimality, it is usually impossible to verify. This nonlinearity makes it challenging to leave the realm of piecewise linear geometry, where at least low-level operations such as closest-point queries, injectivity testing, intersection, and collision detection can be solved with confidence, a solid base facilitating development of more complex algorithms.

With our SOS framework, these and other low level operations are readily extended to higher-order surfaces. SOS geometry processing alleviates concerns, for example, that curved surface continuous collision detection might miss collisions due to linearization error, leading to unrealizable states. Similarly, SOS-based closest projection does not suffer from local optima that could affect processes down the line. SOS programming transforms these and other problems on a huge variety of curved patch types into problems where a user can confidently certify a global optimum. Several avenues for further exploration are listed below.

Hardware Despite the flexibility of SOS programming, the runtime cost of solving SDPs can be substantial. Similar to how the availability of high-performance GPUs catalyzed deep learning, we expect that design of specialized SDP hardware would enable the use of SOS programming in many more contexts where runtime was previously too large. At the least, individual kernel problems do not interact and so can be parallelized.

Self-Intersection Barrier (CSB) In the case of two polynomial patches, quantifying their separation distance produces a natural and algebraic barrier function to collision that can be used in methods like [Li+20]. In the case of a single polynomial patch however, the barrier function becomes far less clear since the distance between a patch and itself is always 0. Formulating such a barrier, however, is necessary for methods

like [Li+20] to extend to higher-order surfaces.

Degenerate Instances and Generalized Exact Recovery Our discussion of exact recovery has focused on the case where the optimal moments correspond to a delta measure, i.e., where the moment matrix has rank one. For some problems, optimal measures generically have more than one support point; this is the case for MSS and MEE, for which optimal measures are supported on points where the patch contacts the surface of the bounding sphere or ellipsoid. PD and SI would exhibit similar behavior if not for our symmetry-breaking modification. Other problems, such as CP, have *degenerate instances*, i.e., problem instances for which there are multiple global optima to the polynomial optimization problem. For these instances, optimal solutions to the measure relaxation are mixtures of these global optima.

In such cases, one might still be interested in extracting the support points. While more complicated than in the rank-one case, it is still possible to extract an *atomic measure*—a mixture of deltas—corresponding to a moment matrix when it satisfies the so-called *flat extension* property. This property holds when increasing the degree of the moment relaxation does not increase the rank of the moment matrix. When flat extension holds, one can apply a procedure based on diagonalization of a set of commuting multiplication operators to extract the measure support points [BPT12, §3.5.6]. It would be interesting to test whether our MEE and MSS relaxations satisfy the flat extension property, which would serve as a certificate of generalized exact recovery and allow for extraction of the support points.

These opportunities for further development aside, our broad SOS framework and specific model problems are already beneficial to geometry processing. These problems would otherwise each require their own solutions, including heuristics for number of initial points, density of linearization, tolerance for intersection, gradient step size, and many other nonlinear optimization parameters, all of which must be tuned per surface type. SOS instead provides a single unified framework for common objectives across the most popular surface representations.

5

Conclusion

This thesis tackles several problems across the geometry processing pipeline. In particular, we address methods in quad mesh generation, hex mesh modification, and algebraic collision detection. In this chapter we reflect on the wider applicability of our approaches to problems outside these specific examples.

The first project demonstrates that pairing convex relaxation with the spherical harmonic representation produces crease-aligned cross fields. Convex relaxation was conveniently within reach because even when the non-convex constraint was removed, we could still obtain non-trivial solutions. Furthermore, projection onto the un-relaxed solution only required normalization. In general it is usually not hard to simply remove all non-convex constraints. A non-convex objective can also frequently be convexified by adding new variables with non-convex constraints that are later relaxed away. Computing a projection from the relaxed solution onto the original non-convex feasible space may unfortunately be challenging. It's recommended in [BBV04] to practice disciplined convex programming, where one limits the operations available to them during the problem formulation stage to always arrive at a convex problem.

The use of variable augmentation for hexahedral mesh quality enhancement, falls into a broad class of widely used adaptive remeshing techniques. Often when computing solutions to numerical PDE such as linear elasticity, a geometric solution can be obtained to local optimality on a fixed mesh topology. Many adaptive remeshing

techniques exist that can quantify where a mesh needs more resolution to produce an improved solution [FD09]. Adding that resolution typically involves subdividing mesh elements, and increasing the number of mesh control points which yields an augmented problem. Adaptive remeshing is then alternated with recomputing the solution to the PDE of interest until error is below some threshold. Despite its history, the majority of mesh modifications to a hexahedral mesh do not improve the scaled Jacobian bounds. Global hex mesh subdivision does not change singular structure, while local hex mesh subdivision only creates more singular nodes. Strategic sheet inflation is the first method that can decrease singular node count to prioritize mesh element distortion.

SOS programming is a form of convexification where instead of removing constraints as in the case of relaxation, more are added. Its wide applicability speaks for itself, so instead we'll discuss limitations. The primary limitation is that the entire problem must be algebraically formulated i.e. using polynomials. Trigonometric functions are naturally problematic, however, sometimes re-parameterization techniques can express trigonometric functions algebraically [RKC00; Kav+08; Ami+22]. Still, an exact spiral geometry is out of reach [WS11]. Thankfully, there is no shortage of algebraic problems.

Beyond the problems and methods in this thesis, various creative geometric optimization strategies exist including pre-conditioning [YSC21; NJJ21], competetive gradient descent [Sol+21], and non-convex alternating direction method of multipliers (ADMM) [Mar+20]. Geometric optimization continues to be a very fruitful tool for studying geometry processing.

Bibliography

[AFP00] Luigi Ambrosio, Nicola Fusco, and Diego Pallara. *Functions of bounded variation and free discontinuity problems*. Oxford mathematical monographs. Oxford ; New York: Clarendon Press, 2000. ISBN: 978-0-19-850245-6 (cit. on pp. 21, 24).

[Ali95] Farid Alizadeh. "Interior point methods in semidefinite programming with applications to combinatorial optimization". In: *SIAM Journal on Optimization* 5.1 (1995), pp. 13–51. DOI: 10.1137/0805002 (cit. on p. 81).

[Ami+22] Alexandre Amice, Hongkai Dai, Peter Werner, Annan Zhang, and Russ Tedrake. "Finding and optimizing certified, collision-free regions in configuration space for robot manipulators". In: *Algorithmic Foundations of Robotics XV: Proceedings of the Fifteenth Workshop on the Algorithmic Foundations of Robotics*. Springer. 2022, pp. 328–348 (cit. on p. 112).

[ApS17] MOSEK ApS. *The MOSEK optimization toolbox for MATLAB manual. Version 8.1.* 2017. URL: http://docs.mosek.com/8.1/toolbox/index.html (cit. on pp. 32, 81).

[Bar+87] Robert E. Barnhill, Gerald Farin, M. Jordan, and Bruce R. Piper. "Surface/surface intersection". In: *Computer Aided Geometric Design* 4.1-2 (1987), pp. 3–16. DOI: 10.1016/0167-8396(87)90020-3 (cit. on p. 12).

[BBV04] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004 (cit. on pp. 81, 111).

[BC08] Xavier Bresson and Tony Chan. "Fast dual minimization of the vectorial total variation norm and applications to color image processing". en. In: *Inverse Problems and Imaging* 2.4 (Nov. 2008), pp. 455–484. ISSN: 1930-8337. DOI: 10.3934/ipi.2008.2.455. URL: http://www.aimsciences.org/journals/displayArticles.jsp?paperID=3792 (visited on 01/07/2019) (cit. on p. 21).

[Ben+95] Steven Benzley, Ernest Perry, Karl Merkley, Brett Clark, and Gregory Sjaardema. "A Comparison of All Hexagonal and All Tetrahedral Finite Element Meshes for Elastic and Elasto-Plastic Analysis". In: *International Meshing Roundtable* 17 (Jan. 1995) (cit. on p. 10).

[Bom+13] David Bommes, Marcel Campen, Hans-Christian Ebke, Pierre Alliez, and Leif Kobbelt. “Integer-Grid Maps for Reliable Quad Meshing”. In: *ACM Transactions on Graphics* 32.4 (July 2013). URL: https://hal.inria.fr/hal-00862648 (cit. on p. 18).

[BPT12] Grigoriy Blekherman, Pablo A. Parrilo, and Rekha R. Thomas. *Semidefinite Optimization and Convex Algebraic Geometry*. SIAM, 2012. DOI: 10.1137/1.9781611972290 (cit. on pp. 79–81, 83, 110).

[Bra+18] Christopher Brandt, Leonardo Scandolo, Elmar Eisemann, and Klaus Hildebrandt. “Modeling n -Symmetry Vector Fields using Higher-Order Energies”. en. In: *ACM Transactions on Graphics* 37.2 (Mar. 2018), pp. 1–18. ISSN: 07300301. DOI: 10.1145/3177750. URL: http://dl.acm.org/citation.cfm?doid=3191713.3177750 (visited on 01/07/2019) (cit. on pp. 10, 15, 18, 33, 34, 36, 50).

[Bra+19] Matteo Bracci, Marco Tarini, Nico Pietroni, Marco Livesu, and Paolo Cignoni. “HexaLab.net: An online viewer for hexahedral meshes”. In: *Computer-Aided Design* 110 (2019), pp. 24–36. ISSN: 0010-4485. DOI: https://doi.org/10.1016/j.cad.2018.12.003. URL: https://www.sciencedirect.com/science/article/pii/S0010448518304238 (cit. on pp. 68, 69).

[BS19] Mikhail Bessmeltsev and Justin Solomon. “Vectorization of line drawings via polyvector fields”. In: *ACM Transactions on Graphics (TOG)* 38.1 (2019), p. 9 (cit. on p. 10).

[BZK09] David Bommes, Henrik Zimmer, and Leif Kobbelt. “Mixed-integer quadrangulation”. en. In: *ACM Transactions on Graphics* 28.3 (July 2009), p. 1. ISSN: 07300301. DOI: 10.1145/1531326.1531383. URL: http://portal.acm.org/citation.cfm?doid=1531326.1531383 (visited on 01/07/2019) (cit. on pp. 11, 18, 37, 51).

[Cam+16] Marcel Campen, Moritz Ibing, Hans-Christian Ebke, Denis Zorin, and Leif Kobbelt. “Scale-Invariant Directional Alignment of Surface Parametrizations”. en. In: *Computer Graphics Forum* 35.5 (Aug. 2016), pp. 1–10. ISSN: 01677055. DOI: 10.1111/cgf.12958. URL: http://doi.wiley.com/10.1111/cgf.12958 (visited on 01/10/2019) (cit. on pp. 18, 37, 51).

[CBK15] Marcel Campen, David Bommes, and Leif Kobbelt. “Quantized global parametrization”. en. In: *ACM Transactions on Graphics* 34.6 (Oct. 2015), pp. 1–12. ISSN: 07300301. DOI: 10.1145/2816795.2818140. URL: http://dl.acm.org/citation.cfm?doid=2816795.2818140 (visited on 01/10/2019) (cit. on pp. 18, 37, 51).

[CC19] Etienne Corman and Keenan Crane. “Symmetric Moving Frames”. In: *ACM Trans. Graph.* 38.4 (July 2019). ISSN: 0730-0301. DOI: 10.1145/3306346.3323029. URL: https://doi.org/10.1145/3306346.3323029 (cit. on p. 12).

[CC78] Edwin Catmull and James Clark. “Recursively generated B-spline surfaces on arbitrary topological meshes”. In: *Computer-aided design* 10.6 (1978), pp. 350–355. DOI: 10.1145/280811.280992 (cit. on p. 8).

[CDS10] Keenan Crane, Mathieu Desbrun, and Peter Schröder. “Trivial Connections on Discrete Surfaces”. en. In: *Computer Graphics Forum* 29.5 (Sept. 2010), pp. 1525–1533. ISSN: 01677055. DOI: 10.1111/j.1467-8659.2010.01761.x. URL: http://doi.wiley.com/10.1111/j.1467-8659.2010.01761.x (visited on 01/07/2019) (cit. on p. 11).

[Cha+10] Antonin Chambolle, Vicent Caselles, Matteo Novaga, Daniel Cremers, and Thomas Pock. “An introduction to Total Variation for Image Analysis”. In: 9 (2010). DOI: 10.1515/9783110226157.263 (cit. on p. 29).

[Cig+14] Paolo Cignoni, Nico Pietroni, Luigi Malomo, and Roberto Scopigno. “Field-aligned mesh joinery”. In: *ACM Transactions on Graphics (TOG)* 33.1 (2014), pp. 1–12 (cit. on p. 10).

[CK92] AO Cifuentes and A Kalbag. “A performance study of tetrahedral and hexahedral elements in 3-D finite element structural analysis”. In: *Finite Elements in Analysis and Design* 12.3-4 (1992), pp. 313–318 (cit. on p. 10).

[Di 86] Silvano Di Zenzo. “A note on the gradient of a multi-image”. en. In: *Computer Vision, Graphics, and Image Processing* 33.1 (Jan. 1986), pp. 116–125. ISSN: 0734189X. DOI: 10.1016/0734-189X(86)90223-9. URL: http://linkinghub.elsevier.com/retrieve/pii/0734189X86902239 (visited on 01/07/2019) (cit. on p. 21).

[FD09] Krzysztof Fidkowski and David Darmofal. “Output error estimation and adaptation in computational fluid dynamics: Overview and recent results”. In: *47th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition*. 2009, p. 1303 (cit. on p. 112).

[GDC15] Xifeng Gao, Zhigang Deng, and Guoning Chen. “Hexahedral mesh reparameterization from aligned base-complex”. In: *ACM Transactions on Graphics (TOG)* 34.4 (2015), pp. 1–10 (cit. on p. 73).

[GDT15] Fernando de Goes, Mathieu Desbrun, and Yiying Tong. “Vector Field Processing on Triangle Meshes”. In: *SIGGRAPH Asia 2015 Courses*. SA ’15. Kobe, Japan: ACM, 2015, 17:1–17:48. ISBN: 978-1-4503-3924-7. DOI: 10.1145/2818143.2818167. URL: http://doi.acm.org/10.1145/2818143.2818167 (cit. on p. 10).

[GLK16] Anne Gehre, Isaak Lim, and Leif Kobbelt. “Adapting Feature Curve Networks to a Prescribed Scale”. en. In: *Computer Graphics Forum* 35.2 (May 2016), pp. 319–330. ISSN: 01677055. DOI: 10.1111/cgf.12834. URL: http://doi.wiley.com/10.1111/cgf.12834 (visited on 06/16/2019) (cit. on pp. 34, 49).

[HJ16] Zhiyang Huang and Tao Ju. "Extrinsically smooth direction fields". en. In: *Computers & Graphics* 58 (Aug. 2016), pp. 109–117. ISSN: 00978493. DOI: 10.1016/j.cag.2016.05.015. URL: https://linkinghub.elsevier.com/retrieve/pii/S0097849316300589 (visited on 01/07/2019) (cit. on pp. 11, 23).

[Hua+11] Jin Huang, Yiying Tong, Hongyu Wei, and Hujun Bao. "Boundary aligned smooth 3D cross-frame field". en. In: *Proceedings of the 2011 SIGGRAPH Asia Conference on - SA '11*. Hong Kong, China: ACM Press, 2011, p. 1. ISBN: 978-1-4503-0807-6. DOI: 10.1145/2024156.2024177. URL: http://dl.acm.org/citation.cfm?doid=2024156.2024177 (visited on 01/07/2019) (cit. on pp. 11, 17, 18, 20, 35, 72).

[IBB15] Emmanuel Iarussi, David Bommes, and Adrien Bousseau. "BendFields: Regularized Curvature Fields from Rough Concept Sketches". en. In: *ACM Transactions on Graphics* 34.3 (May 2015), pp. 1–16. ISSN: 07300301. DOI: 10.1145/2710026. URL: http://dl.acm.org/citation.cfm?doid=2774971.2710026 (visited on 01/10/2019) (cit. on p. 10).

[Jak+15] Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. "Instant field-aligned meshes". en. In: *ACM Transactions on Graphics* 34.6 (Oct. 2015), pp. 1–15. ISSN: 07300301. DOI: 10.1145/2816795.2818078. URL: http://dl.acm.org/citation.cfm?doid=2816795.2818078 (visited on 01/07/2019) (cit. on pp. 11, 33, 34, 36, 50).

[Jia+14] Tengfei Jiang, Jin Huang, Yuanzhen Wang, Yiying Tong, and Hujun Bao. "Frame Field Singularity Correction for Automatic Hexahedralization". In: *IEEE Trans. on Visualization & Computer Graphics* 20.8 (Aug. 2014), pp. 1189–1199. ISSN: 1077-2626 (cit. on p. 11).

[Jia+21] Zhongshi Jiang, Ziyi Zhang, Yixin Hu, Teseo Schneider, Denis Zorin, and Daniele Panozzo. "Bijective and Coarse High-Order Tetrahedral Meshes". In: *ACM Transactions on Graphics* (2021). DOI: 10.1145/3450626.3459840 (cit. on p. 79).

[Kav+08] Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. "Geometric Skinning with Approximate Dual Quaternion Blending". In: *ACM Transactions on Graphics* 27.4 (Nov. 2008). ISSN: 0730-0301. DOI: 10.1145/1409625.1409627 (cit. on p. 112).

[Knö+13] Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. "Globally optimal direction fields". en. In: *ACM Transactions on Graphics* 32.4 (July 2013), p. 1. ISSN: 07300301. DOI: 10.1145/2461912.2462005. URL: http://dl.acm.org/citation.cfm?doid=2461912.2462005 (visited on 01/07/2019) (cit. on pp. 11, 15, 18, 23, 27, 33, 34, 36, 50).

[Knö+15] Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. "Stripe patterns on surfaces". en. In: *ACM Transactions on Graphics* 34.4 (July 2015), 39:1–39:11. ISSN: 07300301. DOI: 10.1145/2767000. URL: http://dl.acm.org/citation.cfm?doid=2809654.2767000 (visited on 01/07/2019) (cit. on p. 10).

[KNP07] Felix Kälberer, Matthias Nieser, and Konrad Polthier. "Quadcover-surface parameterization using branched coverings". In: *Computer graphics forum*. Vol. 26. 3. Wiley Online Library. 2007, pp. 375–384 (cit. on p. 11).

[Koc+19] Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. "ABC: A big cad model dataset for geometric deep learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 9601–9611 (cit. on p. 108).

[KP16] K stutis Karčiauskas and Jörg Peters. "Minimal bi-6 G2 completion of bicubic spline surfaces". In: *Computer Aided Geometric Design* 41 (2016), pp. 10–22 (cit. on p. 73).

[KP19] K stutis Karčiauskas and Jörg Peters. "Refinable smooth surfaces for locally quad-dominant meshes with T-gons". In: *Computers & graphics* 82 (2019), pp. 193–202 (cit. on p. 73).

[Las01] Jean B. Lasserre. "Global optimization with polynomials and the problem of moments". In: *SIAM Journal on Optimization* 11.3 (2001), pp. 796–817. DOI: 10.1137/S1052623400366802 (cit. on pp. 92, 94).

[LBK16] Max Lyon, David Bommes, and Leif Kobbelt. "HexEx: Robust Hexahedral Mesh Extraction". In: *ACM Trans. Graph.* 35.4 (July 2016), 123:1–123:11. ISSN: 0730-0301 (cit. on p. 11).

[Li+12] Yufei Li, Yang Liu, Weiwei Xu, Wenping Wang, and Baining Guo. "All-hex meshing using singularity-restricted field". In: *ACM Trans. Graph.* 31.6 (2012), p. 177 (cit. on p. 11).

[Li+20] Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M. Kaufman. "Incremental Potential Contact: Intersection-and Inversion-Free, Large-Deformation Dynamics". In: *ACM Transactions on Graphics* 39.4 (Aug. 2020). ISSN: 0730-0301. DOI: 10.1145/3386569.3392425 (cit. on pp. 109, 110).

[Li+21] Lingxiao Li, Paul Zhang, Dmitriy Smirnov, S Mazdak Abulnaga, and Justin Solomon. "Interactive all-hex meshing via cuboid decomposition". In: *ACM Transactions on Graphics (TOG)* 40.6 (2021), pp. 1–17 (cit. on p. 12).

[Liu+18] Heng Liu, Paul Zhang, Edward Chien, Justin Solomon, and David Bommes. "Singularity-constrained octahedral fields for hexahedral meshing". en. In: *ACM Transactions on Graphics* 37.4 (July 2018), pp. 1–17. ISSN: 07300301. DOI: 10.1145/3197517.3201344. URL: http://dl.acm.org/citation.cfm?doid=3197517.3201344 (visited on 01/07/2019) (cit. on pp. 12, 54, 56, 58).

[Liv+20] Marco Livesu, Nico Pietroni, Enrico Puppo, Alla Sheffer, and Paolo Cignoni. "Loopycuts: Practical feature-preserving block decomposition for strongly hex-dominant meshing". In: *ACM Transactions on Graphics (TOG)* 39.4 (2020), pp. 121–1 (cit. on p. 12).

[Löf04] Johan Löfberg. "YALMIP: A Toolbox for Modeling and Optimization in MATLAB". In: *In Proceedings of the CACSD Conference*. Taipei, Taiwan, 2004. DOI: 10.1109/CACSD.2004.1393890 (cit. on pp. 81, 98).

[Loo87] Charles Loop. "Smooth subdivision surfaces based on triangles". In: *Master's thesis, University of Utah, Department of Mathematics* (1987) (cit. on p. 8).

[LS10] Franck Ledoux and Jason Shepherd. "Topological modifications of hexahedral meshes via sheet operations: a theoretical study". In: *Engineering with Computers* 26.4 (2010), pp. 433–447 (cit. on p. 56).

[Mar+20] Zoë Marschner, David Palmer, Paul Zhang, and Justin Solomon. "Hexahedral Mesh Repair via Sum-of-Squares Relaxation". In: *Computer Graphics Forum*. Vol. 39. 5. Wiley Online Library. 2020, pp. 133–147. DOI: doi.org/10.1111/cgf.14074 (cit. on pp. 78, 81, 87, 96, 112).

[Mar+21] Zoë Marschner, Paul Zhang, David Palmer, and Justin Solomon. "Sum-of-squares geometry processing". In: *ACM Transactions on Graphics (TOG)* 40.6 (2021), pp. 1–13 (cit. on p. 78).

[MC20] Manish Mandad and Marcel Campen. "Bézier guarding: precise higher-order meshing of curved 2D domains". In: *ACM Transactions on Graphics (TOG)* 39.4 (2020), pp. 103–1. DOI: 10.1145/3386569.3392372 (cit. on p. 79).

[Mer79] N David Mermin. "The topological theory of defects in ordered media". In: *Reviews of Modern Physics* 51.3 (1979), p. 591 (cit. on p. 72).

[MPZ14] Ashish Myles, Nico Pietroni, and Denis Zorin. "Robust Field-aligned Global Parametrization". In: *ACM Trans. Graph.* 33.4 (July 2014), 135:1–135:14. ISSN: 0730-0301. DOI: 10.1145/2601097.2601154. URL: http://doi.acm.org/10.1145/2601097.2601154 (cit. on p. 33).

[NJJ21] Baptiste Nicolet, Alec Jacobson, and Wenzel Jakob. "Large steps in inverse rendering of geometry". In: *ACM Transactions on Graphics (TOG)* 40.6 (2021), pp. 1–13 (cit. on p. 112).

[NN94] Yurii Nesterov and Arkadii Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. Society for Industrial and Applied Mathematics, 1994. DOI: 10.1137/1.9781611970791. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611970791 (cit. on p. 81).

[NRP11a] M. Nieser, U. Reitebuch, and K. Polthier. "CubeCover- Parameterization of 3D Volumes". en. In: *Computer Graphics Forum* 30.5 (Aug. 2011), pp. 1397–1406. ISSN: 01677055. DOI: 10.1111/j.1467-8659.2011.02014.x. URL: http://doi.wiley.com/10.1111/j.1467-8659.2011.02014.x (visited on 01/07/2019) (cit. on p. 11).

[NRP11b] M. Nieser, U. Reitebuch, and K. Polthier. "CubeCover–Parameterization of 3D Volumes". en. In: *Computer Graphics Forum* 30.5 (Aug. 2011), pp. 1397–1406. ISSN: 01677055. DOI: 10.1111/j.1467-8659.2011.02014.x. URL: http://doi.wiley.com/10.1111/j.1467-8659.2011.02014.x (visited on 01/07/2019) (cit. on p. 54).

[Par19] Pablo A. Parillo. *Algebraic Techniques and Semidefinite Optimization*. https://learning-modules.mit.edu/materials/index.html?uuid=/course/6/sp19/6.256#materials. 2019 (cit. on p. 81).

[PBS19] David Palmer, David Bommes, and Justin Solomon. "Algebraic Representations for Volumetric Frame Fields". In: *arXiv:1908.05411* (2019) (cit. on pp. 19, 20, 32, 40).

[Plu+21] Kacper Pluta, Michal Edelstein, Amir Vaxman, and Mirela Ben-Chen. "PH-CPF: planar hexagonal meshing using coordinate power fields". In: *ACM Transactions on Graphics (TOG)* 40.4 (2021), pp. 1–19 (cit. on p. 10).

[PPP02] Stephen Prajna, Antonis Papachristodoulou, and Pablo A. Parrilo. "Introducing SOSTOOLS: A general purpose sum of squares programming solver". In: *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.* Vol. 1. IEEE. 2002, pp. 741–746 (cit. on p. 81).

[Put93] Mihai Putinar. "Positive polynomials on compact semi-algebraic sets". In: *Indiana University Mathematics Journal* 42.3 (1993), pp. 969–984 (cit. on p. 83).

[Qua21] Roshan Quadros. *The CUBIT Geometry and Meshing Toolkit*. https://cubit.sandia.gov/. 2021 (cit. on pp. 52, 70).

[Ray+08] Nicolas Ray, Bruno Vallet, Wan Chiu Li, and Bruno Lévy. "N-symmetry Direction Field Design". In: *ACM Trans. Graph.* 27.2 (May 2008), 10:1–10:13. ISSN: 0730-0301. DOI: 10.1145/1356682.1356683. URL: http://doi.acm.org/10.1145/1356682.1356683 (cit. on p. 11).

[RKC00] Stephane Redon, Abderrahmane Kheddar, and Sabine Coquillart. “An algebraic solution to the problem of collision detection for rigid polyhedral objects”. In: *Proceedings of the IEEE International Conference on Robotics and Automation*. Vol. 4. Piscataway, NJ: IEEE, 2000, pp. 3733–3738. DOI: 10.1109/ROBOT.2000.845313 (cit. on p. 112).

[ROF92] Leonid I. Rudin, Stanley Osher, and Emad Fatemi. “Nonlinear total variation based noise removal algorithms”. en. In: *Physica D: Nonlinear Phenomena* 60.1-4 (Nov. 1992), pp. 259–268. ISSN: 01672789. DOI: 10.1016/0167-2789(92)90242-F. URL: http://linkinghub.elsevier.com/retrieve/pii/016727899290242F (visited on 01/12/2019) (cit. on p. 29).

[Ros02] Wulf Rossmann. “Lie groups: an introduction through linear groups”. In: Oxford graduate texts in mathematics (2002) (cit. on p. 22).

[RSL16] Nicolas Ray, Dmitry Sokolov, and Bruno Lévy. “Practical 3D frame field generation”. en. In: *ACM Transactions on Graphics* 35.6 (Nov. 2016), pp. 1–9. ISSN: 07300301. DOI: 10.1145/2980179.2982408. URL: http://dl.acm.org/citation.cfm?doid=2980179.2982408 (visited on 01/07/2019) (cit. on pp. 11, 17, 35).

[Sap96] G. Sapiro. “Vector-valued active contours”. In: *Proceedings CVPR IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. San Francisco, CA, USA: IEEE, 1996, pp. 680–685. ISBN: 978-0-8186-7259-0. DOI: 10.1109/CVPR.1996.517146. URL: http://ieeexplore.ieee.org/document/517146/ (visited on 01/07/2019) (cit. on p. 21).

[SBS20] Dmitriy Smirnov, Mikhail Bessmeltsev, and Justin Solomon. “Learning Manifold Patch-Based Representations of Man-Made Shapes”. In: *International Conference on Learning Representations*. 2020 (cit. on pp. 79, 102, 105).

[SC20] Rohan Sawhney and Keenan Crane. “Monte Carlo Geometry Processing: A Grid-Free Approach to PDE-Based Methods on Volumetric Domains”. In: *ACM Trans. Graph.* 39.4 (2020). DOI: 10.1145/3386569.3392374 (cit. on p. 101).

[Sch+18] Teseo Schneider, Yixin Hu, Jérémie Dumas, Xifeng Gao, Daniele Panozzo, and Denis Zorin. “Decoupling Simulation Accuracy from Mesh Quality”. In: *ACM Transactions on Graphics* 37.6 (Oct. 2018). DOI: 10.1145/3272127.3275067 (cit. on pp. 12, 79).

[Sch+19] Teseo Schneider, Jérémie Dumas, Xifeng Gao, Mario Botsch, Daniele Panozzo, and Denis Zorin. “Poly-Spline Finite-Element Method”. In: *ACM Trans. Graph.* 38.3 (Mar. 2019), 19:1–19:16. ISSN: 0730-0301. DOI: 10.1145/3313797. URL: http://doi.acm.org/10.1145/3313797 (cit. on pp. 12, 79).

[She+16] Zhongwei Shen, Xianzhong Fang, Xinguo Liu, Hujun Bao, and Jin Huang. "Harmonic Functions for Rotational Symmetry Vector Fields". en. In: *Computer Graphics Forum* 35.7 (Oct. 2016), pp. 507–516. ISSN: 01677055. DOI: 10.1111/cgf.13047. URL: http://doi.wiley.com/10.1111/cgf.13047 (visited on 01/07/2019) (cit. on p. 38).

[Smi+20] Dmitriy Smirnov, Matthew Fisher, Vladimir G. Kim, Richard Zhang, and Justin Solomon. "Deep parametric shape predictions using distance fields". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 561–570. DOI: 10.1109/CVPR42600.2020.00064 (cit. on p. 103).

[Sol+21] Yousuf Soliman, Albert Chern, Olga Diamanti, Felix Knöppel, Ulrich Pinkall, and Peter Schröder. "Constrained willmore surfaces". In: *ACM Transactions on Graphics (TOG)* 40.4 (2021), pp. 1–17 (cit. on p. 112).

[Ste+18] Oded Stein, Eitan Grinspun, Max Wardetzky, and Alec Jacobson. "Natural Boundary Conditions for Smoothing in Geometry Processing". In: *ACM Trans. Graph.* 37.2 (May 2018). ISSN: 0730-0301. DOI: 10.1145/3186564. URL: https://doi.org/10.1145/3186564 (cit. on p. 31).

[SVB17] Justin Solomon, Amir Vaxman, and David Bommes. "Boundary Element Octahedral Fields in Volumes". en. In: *ACM Transactions on Graphics* 36.3 (May 2017), pp. 1–16. ISSN: 07300301. DOI: 10.1145/3065254. URL: http://dl.acm.org/citation.cfm?doid=3087678.3065254 (visited on 01/07/2019) (cit. on pp. 11, 17, 35, 49, 72).

[TCL21] Ty Trusty, Honglin Chen, and David I.W. Levin. "The Shape Matching Element Method: Direct Animation of Curved Surface Models". In: *ACM Transactions on Graphics* (2021). DOI: 10.1145/3450626.3459772. URL: https://doi.org/10.1145/3450626.3459772 (cit. on pp. 12, 79, 108).

[Tod16] Michael J Todd. *Minimum-volume ellipsoids: Theory and algorithms*. SIAM, 2016. DOI: 10.1137/1.9781611974386 (cit. on pp. 90, 91).

[TTT01] Kim-Chuan Toh, Michael J. Todd, and Reha H. Tütüncü. "SDPT3—a Matlab software package for semidefinite-quadratic-linear programming, version 3.0". In: *Web page http://www. math. nus. edu. sg/mattohkc/sdpt3. html* (2001). DOI: 10.1080/10556789908805762 (cit. on p. 81).

[Vax+16] Amir Vaxman, Marcel Campen, Olga Diamanti, Daniele Panozzo, David Bommes, Klaus Hildebrandt, and Mirela Ben-Chen. "Directional Field Synthesis, Design, and Processing". en. In: (2016), p. 28 (cit. on p. 10).

[Wei94] Victor I Weingarten. "The controversy over hex or tet meshing." In: *Machine design* 66.8 (1994), pp. 74–76 (cit. on p. 10).

[WHP11] Anna Wawrzinek, Klaus Hildebrandt, and Konrad Polthier. “Koiter’s Thin Shells on Catmull-Clark Limit Surfaces”. In: *Vision, Modeling, and Visualization (2011)*. Ed. by Peter Eisert, Joachim Hornegger, and Konrad Polthier. The Eurographics Association, 2011. ISBN: 978-3-905673-85-2. DOI: 10.2312/PE/VMV/VMV11/113-120 (cit. on p. 12).

[WS11] Charles W Wampler and Andrew J Sommese. “Numerical algebraic geometry and algebraic kinematics”. In: *Acta Numerica* 20 (2011), pp. 469–567 (cit. on p. 112).

[YSC21] Chris Yu, Henrik Schumacher, and Keenan Crane. “Repulsive curves”. In: *ACM Transactions on Graphics (TOG)* 40.2 (2021), pp. 1–21 (cit. on p. 112).

[Zha+20] Paul Zhang, Josh Vekhter, Edward Chien, David Bommes, Etienne Vouga, and Justin Solomon. “Octahedral frames for feature-aligned cross fields”. In: *ACM Transactions on Graphics (TOG)* 39.3 (2020), pp. 1–13 (cit. on pp. 14, 17).

[Zha+23] Paul Zhang, Judy (Hsin-Hui) Chiang, Xinyi (Cynthia) Fan, and Klara Mundilova. “Local Decomposition of Hexahedral Singular Nodes into Singular Curves”. In: *Computer-Aided Design* 158 (2023), p. 103484. ISSN: 0010-4485. DOI: https://doi.org/10.1016/j.cad.2023.103484. URL: https://www.sciencedirect.com/science/article/pii/S0010448523000167 (cit. on p. 52).

[Zhu+20] Junyi Zhu, Lotta-Gili Blumberg, Yunyi Zhu, Martin Nisser, Ethan Levi Carlson, Xin Wen, Kevin Shum, Jessica Ayeley Quaye, and Stefanie Mueller. “CurveBoards: Integrating breadboards into physical objects to prototype function in the context of form”. In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 2020, pp. 1–13 (cit. on p. 10).

[ZJ16] Qingnan Zhou and Alec Jacobson. “Thingi10K: A Dataset of 10,000 3D-Printing Models”. In: *arXiv preprint arXiv:1605.04797* (2016) (cit. on p. 33).