The Dissertation Committee for Zhen Chen
certifies that this is the approved version of the following dissertation:

Complex Wrinkle Simulation and Robust Surface Remeshing

Committee:

Dr. Etienne Vouga, Supervisor

Dr. Qixing Huang

Dr. Alexander Huth

Dr. Danny M. Kaufman

Complex Wrinkle Simulation and Robust Surface Remeshing

by

Zhen Chen

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

May 2024

Acknowledgments

This dissertation is a tribute to the invaluable mentors, dedicated colleagues, cherished friends, and supportive family members who have been my companions on this remarkable journey.

At the outset, I want to express my deepest appreciation to my Ph.D. advisor, Dr. Etienne Vouga. His unwavering support and belief in me laid the foundation for my journey. Etienne has been an indispensable mentor in my research endeavors, programming, technical writing, and notably, in refining my presentation skills. Engaging in numerous enriching discussions with him has been both a privilege and an inspiration. He embodies the essence of a researcher, collaborator, and exceptional teacher. My gratitude towards him is beyond words. I am deeply thankful for the opportunity to join his lab and for the chance to work alongside him over these years.

My journey has been profoundly enriched by collaborations with esteemed scholars: Dr. Danny M. Kaufman, Dr. Mélina Skouras, and Dr. Hsiao-yu Chen. Their exceptional guidance opened new vistas in the realm of physical simulation. I am especially grateful to Danny for facilitating an internship at Adobe, connecting me with a network of remarkable researchers and engineers, making for an enriching summer experience, despite its virtual nature.

I am equally grateful to Dr. Xifeng Gao, Dr. Zherong Pan, and Dr. Kui Wu for their incredible guidance during my internships in Lightspeed Studio, Tencent America. They introduced me to the field of geometry processing, emphasizing the critical importance of robustness in real-time and industrial geometry processing algorithms within the gaming industry. Interning with them was an absolute honor. These collaborations have been crucial in directing my research path and enhancing my intellectual development.

My sincere thanks go to the Department of Computer Science at the University of Texas at Austin for providing an enriching academic environment, resources, and financial support throughout this journey.

I also want to express my gratitude to my colleagues and friends—Josh, Kevin, Yifan, Lemeng, Ye, Yilin, Shujian, Yihao, Minmin, Ruoyu, Jiaming, Jiyang, and Julian—for their insightful discussions and unwavering support.

Lastly, my heartfelt appreciation is extended to my beloved parents. Their constant support and efforts to help me overcome challenges have been the bedrock of my journey. This endeavor would not have reached fruition without their love and encouragement.

Thank you once again to everyone who has played a role in this pivotal chapter of my life, shaping me into the individual I am today. Your invaluable support and presence in my journey have been truly integral. Thank you for being a part of it.

Abstract

Complex Wrinkle Simulation and Robust Surface Remeshing

Zhen Chen, PhD
The University of Texas at Austin, 2024

SUPERVISOR: Dr. Etienne Vouga

The accurate simulation of intricate wrinkles is essential in computer graphics, particularly for creating efficient and realistic animations. In this field, surfaces are typically represented by triangle meshes. Nonetheless, this methodology encounters significant obstacles in accurately depicting fine details like cloth wrinkles: meshes must be of high resolution. This requirement, in turn, results in a decreased computation speed owing to the increased degrees of freedom.

In this study, we introduce a novel method for simulating wrinkles on coarse meshes, complemented by a robust algorithm for transforming high-resolution inputs into these optimized meshes. Our technique leverages a spectral representation of wrinkles, encoding them in terms of amplitude and phase within the frequency domain. This approach effectively surmounts the limitations of conventional Finite Element Methods (FEM). Typically, these methods struggle to maintain high-frequency details on coarse meshes, or they require an extremely high-resolution mesh to capture such details, resulting in decreased solving speed. Our method not only enables the generation of lifelike wrinkle animations but also allows for the customization of wrinkle patterns, including the design and modification of wrinkles.

We first propose a novel wrinkle model that adeptly captures complex, high-frequency wrinkles on coarse meshes, overcoming the resolution constraints of tradi-

tional Finite Element Method frameworks. This model is then extended to facilitate the animation of detailed wrinkle dynamics, including the progression of singularities. Moreover, we introduce an advanced remeshing technique that seamlessly converts high-polygon models—common in natural environments—into visually equivalent low-polygon meshes. This technique proves invaluable not only for preprocessing to generate coarse meshes suitable for our wrinkle simulation approach but also for creating models with multiple levels of detail (LOD), a technique widely utilized in the gaming industry.

Table of Contents

List of Tables 11

List of Figures 12

Chapter 1: Introduction 16

1.1 Publications 18

Chapter 2: Fine Wrinkling on Coarsely Meshed Thin Shells 19

2.1 Introduction 20

2.2 Related Work 25

2.3 Wrinkle Field Modeling 29

2.3.1 Preliminaries 30

2.3.2 Wrinkle Correction from Wrinkle Fields 31

2.3.3 Fast and Slow Variables 33

2.3.4 In-plane Wrinkle Correction Formulae 34

2.3.5 Wrinkle Field Energy 38

2.3.6 Necessity of In-plane Wrinkle Correction 41

2.4 Solving for Static Wrinkled Shape 42

2.4.1 Computing the Base Surface 42

2.4.2 Computing the Wrinkle Fields 44

2.5 Discretization and Solver 46

2.5.1 Relaxed Integrability 47

2.5.2 Optimization 48

2.6 Evaluation and Discussion 49

2.6.1 Draping Behavior 50

2.6.2 Resolution Analysis 52

2.6.3 Meshing Independence 53

2.6.4 Accuracy 55

2.6.5 Performance and Numerical Convergence 62

2.7 Conclusion, Limitations, and Future Work 66

Chapter 3: Complex Wrinkle Field Evolution 79

3.1 Introduction 80

3.1.1 Contributions 82

3.2 Preliminaries and Related Work 84

3.2.1 Wrinkles on Surfaces 84

3.2.2 Prior Work . . . 86
3.3 Complex Wrinkle Field Discretization . . . 89
3.3.1 Complex Wrinkle Field Representation . . . 94
3.4 Interpolation . . . 95
3.4.1 Our Approach . . . 96
3.4.2 Results . . . 99
3.4.3 Comparison to Other Approaches . . . 100
3.5 Wrinkle Upsampling . . . 107
3.5.1 Upsampling ω . . . 108
3.5.2 Upsampling z . . . 109
3.5.3 Results and Comparisons . . . 114
3.6 Wrinkle Design . . . 116
3.6.1 Adding Wrinkles to Surfaces . . . 116
3.6.2 Editing Wrinkles . . . 118
3.6.3 Results . . . 121
3.7 Limitations and Future Work . . . 122
Chapter 4: Robust Low-Poly Meshing for General 3D Models . . . 126
4.1 Introduction . . . 127
4.2 Related Works . . . 130
4.2.1 Low-poly Meshing . . . 130
4.2.2 Iso-surfacing Algorithms . . . 133
4.3 Method . . . 136
4.3.1 Mesh Extraction . . . 137
4.3.2 Mesh Optimization . . . 143
4.3.3 Self-intersection Check Acceleration . . . 149
4.4 Experiments . . . 153
4.4.1 Metrics . . . 154
4.4.2 Comparisons . . . 155
4.4.3 Parameters . . . 160
4.5 Additional Applications . . . 164
4.5.1 Iso-surface Extraction Comparison . . . 164
4.5.2 Cage Generation . . . 165
4.6 Conclusion, Limitations, and Future Works . . . 166
Chapter 5: Conclusion and Future Directions . . . 173

Appendix A: Fine Wrinkling on Coarsely Meshed Thin Shells Appendices . . 176
A.1 Kirchhoff-Love Shells . . . 176
A.2 Tension Field Theory . . . 177
A.3 In-plane Wrinkle Correction Calculation . . . 178
A.4 Derivation of Stretching Term in *TFW* Model . . . 179
A.5 Additional Performance Experiments and Data . . . 181
A.5.1 Per-Iteration Timing Breakdown . . . 181
A.5.2 Residual Plots for Each Example . . . 182
A.5.3 Convergence Discussion . . . 184
A.5.4 Initialization of *StVK* . . . 185
A.5.5 Video Comparisons . . . 186
A.6 Additional Implementation Details . . . 188
A.6.1 Triangulation . . . 188
A.6.2 Initialization . . . 188
A.6.3 Amplitude and Phase Optimization Details . . . 189
A.6.4 Phase Field Extraction . . . 191
A.6.5 Upsampling and Visualization . . . 192
Appendix B: Complex Wrinkle Field Evolution Appendices . . . 194
B.1 Geodesic Discussion . . . 194
B.1.1 Distance Energy Derivation . . . 194
B.1.2 Geodesics Computation . . . 195
B.1.3 Discrete Geodesics . . . 199
B.2 Loop Subdivision Rules for 1-forms . . . 201
B.3 Contangent Vector Computation . . . 201
B.4 Timing . . . 203
B.5 Additional Editing Examples . . . 203
B.6 Extra Comparison Examples . . . 204
B.7 Ablation Study . . . 205
B.7.1 Number of Guide Frames . . . 205
B.7.2 Penalty Coefficient . . . 205
Works Cited . . . 221
Vita . . . 249

List of Tables

2.1 Timing and resolution comparisons between *TFW* and other classical methods 66

4.1 A brief summary of the existing methods by their capabilities of maintaining geometry and topology properties 135
4.2 A speedup summary of self-intersection checks for some of the figures shown in the Chapter 4 150
4.3 Statistics of the results generated for the entire dataset by all comparing low-poly meshing approaches 155
4.4 The statistics for low-poly meshes generated with different screen sizes 161
4.5 The Hausdorff distance statistics 166

A.1 Additional timing information for the *TFW* solver 185
A.2 Additional timing information for the *StVK* solver 186
A.3 Timing for an alternative *StVK* setup when using a problem-specific "flat" initialization 187

B.1 The time and convergence information for all the examples showing in this chapter 203

List of Figures

2.1 Overview of our *TFW* pipeline for predicting the wrinkled equilibrium shape of a thin shell (in this case, a cloth dress draped on a mannequin). 19
2.2 Elastic and tension-field simulation on a coarse dress mesh 22
2.3 2D sketch of the effect that the two in-plane wrinkle correction modes exert on the wrinkle shape 35
2.4 Simulated cylinder wrinkles with/without proposed in-plane correction 41
2.5 Simulation of wrinkles in a highly stretched sheet with *TFW* 55
2.6 Simulation of wrinkles in a highly stretched sheet with *StVK* 56
2.7 Highly stretched sheet comparison 57
2.8 Thin rectangular sheet shearing comparison with FEM solvers 58
2.9 Thin rectangular sheet shearing comparison with the scaling law result 59
2.10 The simulated results of inflated annulus and disk experiments 61
2.11 The simulated tension field result of the inflated disk experiment . . . 62
2.12 The simulated result of inflated teabags 63
2.13 Inflated Teddy bear 63
2.14 Comparison of simulating an asymmetric dress draped over a mannequin 67
2.15 Comparison of simulating pants over a mannequin 68
2.16 Comparison of simulating a symmetric dress draped over a mannequin 69
2.17 A symmetric dress draping over a mannequin in terms of different triangulation, but similar mesh resolution 70
2.18 Simulations of a draped dress with different mesh resolutions using our *TFW* 71
2.19 Draping a square piece of cloth on a sphere 75
2.20 We simulate the draping of a dress with different mesh resolutions, using *StVK* and *ARCSim* 76
2.21 Our results for the twisted cylinder experiments 77
2.22 Comparisons of our *TFW* algorithm and the baseline *StVK* simulations against wall clock time, on a log scale. 78

3.1 Overview of our pipeline for our Complex Wrinkle Field (*CWF*) model 79
3.2 Wrinkles on surfaces are ubiquitous, from sand dunes to the creases and folds on a wrinkly dog face, a bean bag chair and a sequence of wrinkle patterns formed during the twisting of a cloth shirtsleeve . . . 81
3.3 *CWF* interpolation between keyframes for torus and dress example . 84

3.4 Different discretizations of a plane wave associated with $\boldsymbol{z} = \exp(3\pi i x)$ 92
3.5 *CWF* interpolation between two wrinkle patterns on the fertility model 100
3.6 *CWF* interpolation between two wrinkle patterns on the fertility model, where the second keyframe is generated by local modfication of the first one 101
3.7 *CWF* interpolation between two wrinkle patterns on the Stanford bunny, where the second keyframe is generated by local modfication of the first one 102
3.8 Failure of linear interpolation given boundary conditions $\boldsymbol{z}_0 = \exp(ix)$ and $\boldsymbol{z}_1 = \exp(iy)$ 103
3.9 Failure of Knöppel et al. (2015)'s algorithm for wrinkle upsampling . 104
3.10 Interpolation comparisons between *CWF* and several baselines on an example of wrinkles rotating by ninety degrees on the Stanford bunny 106
3.11 Comparison with *C-IPC* Li et al. (2021) 107
3.12 A comparison of our upsampling algorithm to naively applying the Loop subdivision 109
3.13 *CWF* subdivision rules for even and odd vertices. 113
3.14 Successive Loop-upsampled the Stanford bunny 114
3.15 A comparison where we replace Equation (18) in our upsampling algorithm with some alternatives 115
3.16 Comparison between our upsampling algorithm against two baselines from previous work Evgeny and Harders (2019); Chen et al. (2021b) . 116
3.17 Adding wrinkles to the Spot cow 117
3.18 *CWF* interpolation of two wrinkle patterns on the Phantasma model 122
3.19 *CWF* interpolation of another two wrinkle patterns on Spot the cow . 123

4.1 A gallery of *wild* high-poly input meshes and their corresponding low-poly outputs generated by our method 126
4.2 The pipeline of our remeshing algorithm 133
4.3 The iso-surface meshes of a CAD model "block" 138
4.4 voxel size and iso-value relationship 139
4.5 Illustration of our feature-aware iso-surface extraction step for MC33 case 4.1.1 140
4.6 Our iso-surface extraction pipeline 142
4.7 Undesirable visul appearance illustration 145
4.8 The results of different simplification conditions 145
4.9 Without feature alignment step, we will end up with the results with "spikes" 149
4.10 Edge or vertex operation illustration 151

4.11 Comparison with Simplygon 156
4.12 Comparison with academia and open-source solutions 158
4.13 The time statistics for our remeshing method 160
4.14 The re-meshed results w.r.t. different user-specified distance tolerance 161
4.15 The different extracted iso-surface for a fixed offset distance using different voxel sizes 162
4.16 The different results using different flow step fractional ratio 163
4.17 Different feature-line length thresholds lead to different results 163
4.18 The generated shell for a cartoon octopus. 166
4.19 Comparison with Sacht et al. (2015) 167
4.20 Time and memory consumption in terms of screen size for the tree model in Figure 4.2. 168
4.21 Comparison with variants of the pipeline of first mesh repairing and then mesh simplification 169
4.22 The comparison of different iso-surface extraction methods 170
4.23 Our iso-surfacing lookup table 171
4.24 Cube division policies for MC33 cases with more than one components 172

A.1 Breakdown of average time required by each component of *TFW* and *StVK* during one optimization iteration 182
A.2 Comparing residual and wall clock time for *TFW* and *StVK* 183

B.1 Loop subdivision rules for boundary 1-forms 202
B.2 Local rotation of an axial wrinkle pattern on a cylinder 207
B.3 *CWF* interpolation of two wrinkle patterns on the hand 208
B.4 *CWF* interpolation of two wrinkle patterns on the face 209
B.5 *CWF* interpolation of two wrinkle patterns on Spot the cow 210
B.6 *CWF* interpolation of two wrinkle patterns on a symmetric dress . . 211
B.7 Uniformly doubling the wrinkle frequency and halving the amplitude on the dress examples from Chen et al. (2021b) 212
B.8 *CWF* interpolation of two wrinkle patterns on the pants 213
B.9 The results of enlarging a local patch of the phantasma model 214
B.10 The results of rotating wrinkles on a torus by ninety degrees 215
B.11 Interpolation of rotating wrinkle patterns on the phantasma model by ninety degrees 216
B.12 Interpolation of rotating amplitude patterns on the phantasma model by ninety degrees 217

B.13 Interpolation of rotating phase patterns on the phantasma model by ninety degrees 218
B.14 Locally enlarging the wrinkle frequency on the Stanford bunny, and corresponding interpolated wrinkle patterns 219
B.15 Locally enlarging the wrinkle frequency on the Stanford bunny, and corresponding interpolated ampliude and phase patterns 220

Chapter 1: Introduction

Wrinkles are a prevalent phenomenon in everyday life, and understanding their geometry and simulating their deformation have been extensively studied in the field of computer graphics. In a discrete setting, a wrinkle surface is typically represented by piecewise linear elements, such as triangular meshes, where each vertex on the mesh is sampled from the original smooth surface. Numerous methods have been developed for wrinkle simulation based on this discretization. However, traditional finite element method (FEM) systems often struggle with inadequate mesh resolution for accurately representing high-frequency wrinkles. Thus, further exploration is necessary to generate and simulate high-frequency wrinkles effectively.

This thesis investigates recent trends in wrinkle simulation to propose an innovative approach for representing and animating physically plausible wrinkles on coarse geometry. The primary objective is to reduce the degrees of freedom and develop an efficient solver for enhanced performance.

In Chapter 2, we introduce an innovative model and algorithm to capture the high-frequency details of thin shells on coarse meshes, effectively overcoming the resolution constraints encountered in conventional Finite Element Method (FEM) frameworks. Our approach is capable of predicting global, fine-scale wrinkling at frequencies much higher than what the mesh resolution can typically accommodate. It is based on the geometric analysis of elasticity and operates without the need for manual guidance, a corpus of training examples, or the adjustment of ad-hoc parameters. Firstly, we approximate the basic form of the shell utilizing tension field theory, where the material is considered to not resist compression. Subsequently, we enhance this base mesh with wrinkle details, parameterized by an amplitude and phase field solved across the base mesh, collectively defining the wrinkle geometry. Our method is validated through comparisons with both physical experiments and numerical simulations, demonstrating that our algorithm can generate wrinkle patterns remarkably similar to those produced by traditional shell solvers, but with significantly fewer degrees of freedom.

In Chapter 3, we further extend our wrinkle representation to include the intricate dynamics of wrinkle animation—such as rotation, merging, and disappearing—via a concept termed Complex Wrinkle Fields (CWF). This approach utilizes a complex-valued phase function that is almost everywhere-unit across the surface, a frequency one-form, and an amplitude scalar, with a soft compatibility condition that coordinates the frequency and phase. We further propose algorithms for interpolating between two such wrinkle fields, for rendering them as displacements on a refined mesh obtained through Loop subdivision, and for making smooth, localized adjustments to the wrinkle's amplitude, frequency, and direction. These algorithms, for the first time, facilitate the creation and editing of wrinkle animations on triangle meshes that are smooth in space, evolve coherently through time, include singularities along with their complex interactions, and that represent details at frequencies significantly finer than the surface resolution.

In Chapter 4, we introduce a novel algorithm designed to generate a low-resolution version of real-world mesh data, while simultaneously repairing its topological and geometric flaws, including non-manifoldness and self-intersections. This method not only serves as the initial step for the wrinkle models proposed in previous chapters, by providing a reliable coarse mesh but also enables the creation of meshes at multiple levels of detail (LOD). These LOD models facilitate rendering for an approximated view, crucial for achieving a real-time gaming experience, particularly on low-end devices. Furthermore, real-world mesh data frequently exhibit complex topologies and geometries that present obstacles for further processes like surface parameterization and manipulation. Our method effectively addresses these complexities by rectifying the topological and geometric artifacts.

1.1 Publications

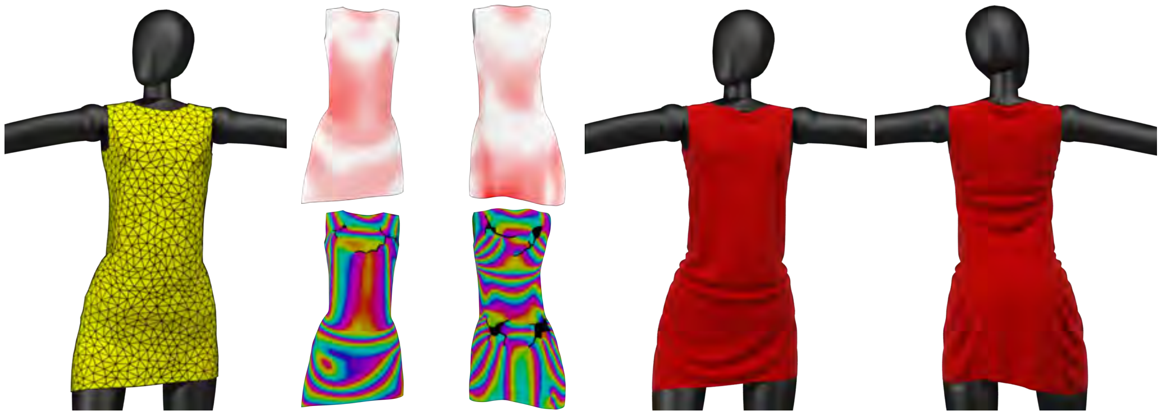
The content of this document can be found in the following publications:

- Chapter 2: Zhen Chen, Hsiao-Yu Chen, Danny M. Kaufman, Mélina Skouras, and Etienne Vouga. 2021. “Fine Wrinkling on Coarsely Meshed Thin Shells”. ACM Trans. Graph. 40, 5, Article 190 (aug 2021), 32 pages.

- Chapter 3: Zhen Chen, Danny M. Kaufman, Mélina Skrous, and Etienne Vouga. “Complex Wrinkle Field Evolution”, ACM Trans. Graph. 42, 4, Article 72 (jul 2023), 19 pages.

- Chapter 4: Zhen Chen, Zherong Pan, Kui Wu, Etienne Vouga, and Xifeng Gao. “Robust Low-Poly Meshing for General 3D Models”. ACM Trans. Graph. 42, 4, Article 119 (jul 2023), 20 pages.

Chapter 2: Fine Wrinkling on Coarsely Meshed Thin Shells[1]



(a) *Tension field base mesh* (b) *Amplitude and phase field* (c) *Recovered wrinkled shape (front and back)*

Figure 2.1: Overview of our pipeline for predicting the wrinkled equilibrium shape of a thin shell (in this case, a cloth dress draped on a mannequin). We approximate the coarse shape of the draped cloth using tension field theory (a), in which material forces do not resist compression. We then augment this *base mesh*, which can be very coarse (around one thousand vertices), with wrinkles. We formulate the elastic energy of the shell in terms of an amplitude (b, *top*) and phase field (b, *bottom*) over the base mesh, which together characterize the geometry of the wrinkles, and solve for these fields globally over the mesh. Our method recovers complex wrinkle patterns with nontrivial geometry and topology (c), including wrinkles with wavelength much smaller than the resolution of the base mesh.

We propose a new model and algorithm to capture the high-definition statics of thin shells via coarse meshes. This model predicts global, fine-scale wrinkling at frequencies much higher than the resolution of the coarse mesh; moreover, it is grounded in the geometric analysis of elasticity, and does not require manual guidance, a corpus of training examples, nor tuning of ad-hoc parameters. We first approximate the coarse shape of the shell using tension field theory, in which material forces do

[1]This chapter is modified from Chen et al. (2021b). For more details, please refer this webpage

not resist compression. We then augment this base mesh with wrinkles, parameterized by an amplitude and phase field that we solve for over the base mesh, which together characterize the geometry of the wrinkles. We validate our approach against both physical experiments and numerical simulations, and show that our algorithm produces wrinkles qualitatively similar to those predicted by traditional shell solvers requiring orders of magnitude more degrees of freedom.

2.1 Introduction

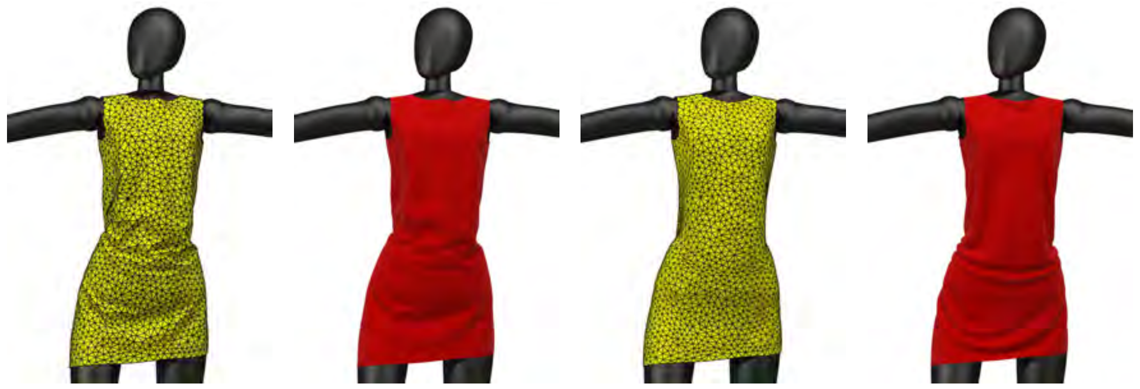
Complex and high-frequency wrinkling patterns give thin-sheet and membrane materials, like cloth and plastic film, their characteristic fine details. Yet the rich diversity of wrinkling behaviors observable in these everyday materials also pose a significant modeling and simulation challenge: failure to represent the material using sufficient degrees of freedom to adequately sample the surface's highest-frequency wrinkle features leads to noisy, aliased results, or else overly-smoothed surfaces whose fine wrinkles are missing altogether. Thus computational meshes with hundreds of thousands of vertices can be required in computer animation or garment modeling applications in order to capture accurate and/or expressive wrinkles.

Towards the efficient and accurate simulation of wrinkling we propose a new model and algorithm to capture the high-definition statics of thin shells via coarse meshes. Our goal is to model the complex and fine wrinkles that arise in the interplay between tension and compression while utilizing just a small number of degrees of freedom. For example, in the dress shown in Figure 2.16 we simulate with $1.4k$ degrees of freedom (compare to a full resolution simulation result using $40k$ vertices in Figure 2.16). The key idea of our approach is to split the kinematics of wrinkled shells into a *base mesh*, representing the coarse envelope of the shell without its fine wrinkles, and a *wrinkle field* parameterized over the base mesh, encoding the amplitude and wavelength of high-frequency wrinkles. In the first half of this chapter, we formulate a reduced-order model of shell statics in terms of these degrees of freedom. In the second, we propose a concrete algorithm for computing the wrinkled static shape of tension-dominated thin materials, such as draped garments or inflated balloons, based on the choice of *tension field theory* for computing the base mesh shape.

Our model predicts global, fine-scale wrinkling at frequencies much higher than the resolution of the base mesh and, in contrast to previous heuristic approaches for wrinkle augmentation, is grounded in the geometric analysis of elasticity, and does not require manual guidance, a corpus of training examples, nor tuning of ad-hoc parameters. We validate our approach against both physical experiments and numerical simulations, show that our algorithm generates both well-known experimental results and simulations with wrinkle quality comparable to those obtained by classical cloth solvers utilizing orders of magnitude more degrees of freedom.

Wrinkle Fields Augmenting a coarse simulation with additional high-frequency detail, via techniques such as normal or displacement mapping (either crafted by artists or learned from data Lähner et al. (2018a)), is a long-standing and powerful strategy. We adopt the approach of several prior works in physics and computer graphics which parameterize wrinkles as spatially-varying amplitude and phase fields. To solve for these wrinkle fields and add fine wrinkles to a base mesh, previous methods explore several ideas based on *local* analyses of its deformation, either by assuming that the base mesh has a very simple geometry, so that wrinkle behavior can be predicted analytically; by restricting to materials like skin where a volumetric substrate drives local wrinkling; or by procedurally modeling wrinkles based on heuristics, user guidance, or data-driven models. While such local models can capture the wrinkling of pinched skin or tight-fitting spandex, where the wrinkle frequency and amplitude are indeed determined by local rather than long-range interactions, they are inapplicable to loose-fitting clothing or draped cloth. In contrast, we develop a principled *global* model of wrinkle fields, grounded in the physics of thin shells.

Problem Scope. We apply our reduced-order wrinkle model to the problem of simulating the static shape of thin shells under a mix of tension and compression, subject to boundary conditions. Here we focus exclusively on this statics problem with potential applications to virtual try-on, garment design, draping, and modeling. We do not consider dynamics, although see Section 2.7 for discussion of how the theory we develop might be applied to dynamics. We solve the statics problem by first computing the base mesh shape, and then solving for the wrinkle orientation, frequency, and amplitude.



(a) Coarse elastic simulation *(b) Corresponding TFW result* *(c) Tension field simulation* *(d) Corresponding TFW result*

Figure 2.2: Augmenting a coarse elastic simulation (a) and tension-field simulation (c) of a dress with wrinkles. Note that using a coarse elastic simulation yields poor results: since a coarse mesh cannot represent fine wrinkling, the simulation produces an *aliased* result with incorrect, coarse wrinkling instead. Adding additional high-frequency wrinkles to this base mesh is not useful. Instead, we propose using a tension-field simulation of the shell as the base mesh; the *TFT* solution is devoid of wrinkles, and becomes a blank canvas on which we can solve for a high-quality wrinkle field.

Choice of Base Mesh Although a coarse-resolution simulation of the shell may be the most obvious choice of base mesh, this choice is flawed (see Figure 2.2). If a shell would wrinkle at a frequency higher than can be resolved by the tessellation, the coarse-resolution simulated shell will buckle, but at an aliased frequency much coarser than the physically-correct wrinkle features. Augmenting such a base mesh with plausible wrinkles would thus require *both* removing the existing, spurious coarse buckling, and then inserting new high-frequency wrinkles. Instead, we propose computing a base mesh free of any wrinkles using *tension field theory* (*TFT*); this *TFT* base mesh then serves as a blank canvas upon which we can optimize for a wrinkle field encoding fine details at the correct frequency and amplitude.

Tension Field Theory The key insight of tension field theory Pipkin (1986); Steigmann (1990) is that tension-dominated thin shell deformation can be understood at two independent scales: the coarse structure and the fine wrinkles. Consider, for instance, manipulating a piece of cloth: the cloth strongly resists *extension*, but allows *compression* with almost no resistance. When compressed, the cloth buckles into a wrinkled shape, at very low energy cost since the bending stiffness of real materials is orders of magnitude weaker than the stretching stiffness. The dominant forces determining the *coarse* structure of cloth that is being tugged or draped are therefore internal tension forces, gravity and other external loads, and bending; the internal compression forces and the buckling and wrinkling they induce play a negligible role in the cloth's overall shape.

Despite their relative unimportance in determining coarse shape, compression forces are the chief source of numerical difficulty when simulating shell statics. Not only does resolving the wrinkles induced by compression require finely-tessellated elements, but the elastic energy of compression is non-convex (since there is symmetry in whether a small, compressed strip of cloth will buckle upwards or downwards; there is also phase-shift symmetry in the wrinkle pattern on a larger cloth patch). In cases where simulating detailed wrinkling is not required, such as when designing inflatable balloons Skouras et al. (2014), there has been great profit in neglecting the compressive forces altogether: the resulting *TFT* simulation minimizes a convex elastic energy which accurately predicts a surface's coarse-scale shape even when the simulation mesh has few degrees of freedom. For tension-dominated problems, such as simulating cloth drapes or deformation of pressurized chambers, *TFT* is thus ideal for generating an approximate coarse base mesh for applying our wrinkle fields (constructed in Section 2.3).

Contributions To summarize, our core contributions include:

- a new formulation of thin shell kinematics that splits degrees of freedom between coarse-scale deformation of the surface, and high-frequency wrinkling, expressed as functions on the coarse surface. We derive an energy model for the shell in terms of these degrees of freedom (Section 2.3);

- we propose an algorithm which uses the above energy model to solve for the static shape of cloth and other thin materials, when subject to loads that induce a mix of tension and compression (Section 2.4). At its heart, this algorithm first solves for the coarse shape using *TFT* simulation, and then, given the coarse mesh, solves a sequence of quadratic programs to compute the wrinkle field parameters;

- we evaluate our model and algorithm on a variety of test cases, including experiments drawn from the physics literature, and simulations of garment drapes and inflatable structures (Section 2.6). We show qualitative agreement between our results and those of both established experiments and full degree-of-freedom shell solvers, even when our method is discretized very coarsely, e.g., using ≈ 1000 degrees of freedom; and

- although low-level performance optimization is not our focus here, we show that our method's advantage of working on much coarser meshes translates into speedups (up to one or two orders of magnitude) when compared to a baseline, optimized Newton-type solver for shell statics (Section 2.6).

We find that coarse base meshes, together with wrinkle fields, are a powerful representation for simulating cloth and other thin materials with complex, highly-detailed, high-frequency wrinkle-like features, and yield striking results in comparison to simulations using classic bending and stretching elements. We hope our model will serve as a foundation for further research into simulation using wrinkle-field kinematics, not only for forward problems, but also for inverse design problems, where the wrinkle amplitude and direction are more natural and semantically meaningful deformation degrees of freedom than traditional vertex displacements.

2.2 Related Work

Thin shell simulation has long been a research focus in both computational mechanics and computer graphics. Considerable effort has focused on improving computational efficiency of generic cloth and shell solvers. This work was pioneered by Baraff and Witkin (1998)'s application of implicit time integration to accelerate cloth simulation. Subsequent research proposes many improvements including implicit-explicit methods Boxerman and Ascher (2004), adaptive remeshing Narain et al. (2012b, 2013); Li et al. (2018); Grinspun et al. (2002), distributed memory parallelism Selle et al. (2009), position-based dynamics Müller et al. (2007), subdivision thin shell element methods Vetter et al. (2014), multi-grid methods Tamstorf et al. (2015), and various approaches to incorporating yarn-level dynamics Kaldor et al. (2008), such as by homogenization Sperl et al. (2020) or enrichment of a triangle mesh by yarn patches Casafranca et al. (2020).

Theoretical Analysis of Wrinkles The interplay of thin shell mechanics and wrinkling has been significantly studied in the physics community. Cerda and Mahadevan (2003) derive a scaling law that relates the wrinkle wavelength to the material parameters of a stretched elastic thin sheet; this experiment was analyzed systematically in follow-on computational work Healey et al. (2013); Li and Healey (2016); Wang et al. (2018). The Cerda and Mahadevan model was later extended to more complex deformations and geometries Paulsen et al. (2016); Aharoni et al. (2017). The work of Paulsen et al. (2016) is particularly notable as it accounts for the role of surface curvature on wrinkling: their extension can be used to predict the wavelength of wrinkles on simple, rotationally-symmetric 3D geometries like cylinders and hemispheres. However, their analysis assumes that the wrinkle direction is known in advance and that wavelength is constant in this direction, which is not true for everyday, complex geometries, for example in draped dresses or pants.

Physically-Inspired Wrinkle Simulation Given the importance of fine wrinkles to the visual quality of cloth simulation, several methods have studied augmenting simulations to better reproduce high-quality wrinkles, either by modifying the solve itself, or adding wrinkles as a post-process. Bergou et al. (2007b) use constrained Lagrangian mechanics to force a high resolution simulation to track the coarse motion of an art-directed low-resolution target surface, thereby enhancing the coarse motion with fine-scale details. Rémillard and Kry (2013b) apply a similar idea to simulation of skin, by using a sparse set of constraints to couple the motion of outer skin layers to the underlying volumetric substrate. In a similar vein, Wrinkle Meshes Müller and Chentanez (2010) computes fine wrinkles by simulating high-resolution patches constrained to an initial, low-resolution simulation. Although both methods generate a high-resolution wrinkled surface whose coarse shape matches a coarse target, these tracking-based methods still require simulating wrinkles on a high-resolution mesh, in contrast to our approach based on coarse wrinkle-field kinematics. Several post-processing methods have been proposed that add dynamic wrinkles based on analyzing the strain tensor after a coarse-scale simulation Gillette et al. (2015b); Rohmer et al. (2010); although real-time, these methods rely on user guidance to choose the proper wrinkle size, rather than inferring the correct wrinkles from the cloth physics.

Evgeny and Harders (2019) predict the wrinkling of human skin, and other materials consisting of a stiff film coupled to a soft substrate. Similar to our approach, Zuenko et al. solve for an amplitude and phase field discretized on a surface mesh. Different from us, however, Zuenko et al. compute wrinkle frequency based on local phenomenological laws drawn from the physics literature that relate wrinkle frequency to the film-to-substrate shear modulus ratio. Their method is thus only applicable to wrinkled skin or spandex, and not to wrinkling of loose-fitting cloth and other shells which are not bonded to any substrate.

In case of no substrate, one way to extend Zuenko et al.'s approach is to use an ad-hoc value of film-to-substrate shear modulus ratio to control the wrinkle frequency; this appears to be how Zuenko et al. simulated Figures 8a,b in their paper, for instance (showing wrinkling on a toroidal balloon and a gold sheet draped on a sphere). As in the techniques above that add dynamics as a postprocess, the downside of this idea is that one now needs to manually tune this parameter, rather than allowing the physics to create the wrinkles automatically. As an alternative, for shells with no substrate, Zuenko et al. also extended their approach by using the scaling law of Vandeparre et al. (2011) instead of film-to-substrate shear modulus ratio to determine wrinkle frequency. The main idea of Vandeparre et al. is to predict the frequency of cloth wrinkles in the interior based on inward propagation from a nearby compressed, clamped boundary. This scaling law gives good results for simulations involving hanging curtains, or other problems where wrinkling is coming from compression at a boundary (see for instance their Figure 8c). Note that in order to be applicable, the Vandeparre et al. scaling law requires (1) that the wrinkling in the interior of the cloth is explainable by wrinkles propagating inward from a nearby boundary, and (2) that the wrinkles at the boundary are caused by compression and clamping at that boundary. Wrinkles due to contact and draping (see the dress in Figure 2.16) in the cloth interior; or shearing of a boundary that is *not* compressed (for instance, see our later experiments: the sheared rectangle, Figure 2.8, and twisted cylinder, Figure 2.21); or strain in the interior that vanishes towards the clamped boundary (like in Cerda and Mahadevan's model problem of a stretched elastic sheet; see Figure 2.5) violate the modeling assumptions of Vandeparre et al. See our comparison with the extended Zuenko et al.'s approach with Vandeparre's law on the stretched sheet and sheared rectangle examples in Figure 2.7 and 2.9 highlighting these issues.

Finally, there has been work on augmenting fluid simulation with high-frequency wave packets Jeschke and Wojtan (2017); these wave packets are "wrinkles" of a sort, but are governed by a very different physics than the shell elasticity considered in this chapter.

Tension Field Theory. As discussed above in our introduction, *TFT* was successfully applied to the design of inflatable structures Skouras et al. (2014), and has been used successfully to explain physical and biological phenomena such as the wrinkling of scar tissue Cerda (2005) and the inflation of parachutes Baginski (2005). The convexity of *TFT* was also exploited in order to optimize the design of skin-tight clothing using sensitivity analysis Montes et al. (2020). Although not explicitly grounded in tension field theory, several computer graphics techniques for simulating cloth Choi and Ko (2002); Jin et al. (2017) adopt a similar idea of neglecting or significantly weakening the elastic forces that resist compression. By reducing the likelihood that small strains induce numerically-challenging out-of-plane buckling, these methods are computationally efficient. Likewise weakening the compression forces offers other advantages as well, such as alleviation of membrane locking.

Data-Driven Approaches A final stream of research on efficiently augmenting simulations with fine wrinkles uses data (gathered from the real world; e.g. via motion capture Lähner et al. (2018a), or collected from offline high-resolution simulations) to learn cloth deformation. To accelerate the latter idea, Kim et al. (2013) constructed a compressible secondary cloth motion graph to sample the dynamic space and reduce storage requirements by a factor of 1000. A common idea, especially useful for adding fine details to T-shirts and other relatively skin-tight garments, is to condition the learned deformation on the pose of an underlying mannequin Wang et al. (2010); Hahn et al. (2014); Santesteban et al. (2019b). This idea can be further applied to transfer of cloth motion from one body shape onto another Guan et al. (2012). Different from the pose-based methods, Kavan et al. (2011) and Seiler et al. (2012) learn a dense upsampling operator to obtain more geometric details on a coarse simulated mesh, and does not assume an underlying mannequin.

Although highly efficient, these methods suffer from the usual issues of data-driven strategies: artifacts appear and quality degrades when the method is applied to simulations that require extrapolation rather than interpolation of existing training data. Methods conditioned on mannequin pose thus cannot be applied to free-floating, environmental cloth.

2.3 Wrinkle Field Modeling

We model the kinematics of wrinkled thin shells in terms of a coarse *base surface*, augmented by wrinkles parameterized by amplitude and phase fields over the base surface. In this section, we discuss the mathematical details of these ideas. The main result of our analysis is the formulation of the shell energy in Equation (2.27), written in terms of the base surface and wrinkle-field degrees of freedom; a discretization of this energy will form the basis of our algorithm in Section 2.4 for computing the static shape of thin shells. Note that for the theory we develop in this section, we do not assume any particular choice of base surface: later in Section 2.4.1 we will discuss our proposed use of *TFT* simulation to compute the base surface. We then cover discretization and implementation details in Section 2.5.

Throughout this section we apply tools and ideas from the physics literature on the geometry of wrinkled sheets Aharoni et al. (2017); Cerda and Mahadevan (2003) and simple curved membranes Paulsen et al. (2016). However, to our knowledge, the general analysis of wrinkle shape and energy on curved shells that we perform here, and the resulting formulas in Equation (2.19) and (2.27), are novel.

2.3.1 Preliminaries

We begin with conventions and notation. We assume that we are modeling a hyperelastic shell of fixed thickness h, and adopt the usual Kirchhoff-Love assumption Ciarlet (2000) that the shell's 3D volume is formed by extruding a *midsurface* $\boldsymbol{r}(u,v) : U \to \mathbb{R}^3$ a distance $h/2$ in both directions along the midsurface normal $\hat{\boldsymbol{n}}(u,v)$, where U is a planar *parameter domain*. For simplicity we assume a St. Venant-Kirchhoff linear constitutive model[2], in which case it can be shown (see e.g. Weischedel (2012)) that the elastic energy of the shell, as a function of the midsurface embedding $\boldsymbol{r}$, is given by the *Koiter energy*

$$E = \int_U \left(\frac{h}{4}W_s + \frac{h^3}{12}W_b\right)\sqrt{\det \boldsymbol{I}_u}\,\mathrm{d}u\mathrm{d}v, \tag{2.1}$$

$$W_s = \left\|\boldsymbol{I}_u^{-1}\boldsymbol{I} - \boldsymbol{id}\right\|_{\mathrm{SV}}^2 \tag{2.2}$$

$$W_b = \left\|\boldsymbol{I}_u^{-1}(\boldsymbol{II} - \boldsymbol{II}_u)\right\|_{\mathrm{SV}}^2 \tag{2.3}$$

where W_s and W_b are stretching and bending energy densities, $\boldsymbol{I}$ and $\boldsymbol{II}$ are the midsurface first and second fundamental forms, respectively, expressed as 2×2 matrices in the parameter coordinates. The tensors $\boldsymbol{I}_u$ and $\boldsymbol{II}_u$ are "rest" fundamental forms encoding the strain-free shell configuration. In the case where the shell is rest-flat and the domain U represents the shell's strain-free shape, $\boldsymbol{I}_u = \boldsymbol{id}$ (the identity matrix) and $\boldsymbol{II}_u = \boldsymbol{0}$. All of our examples are rest-flat, though we will include the rest fundamental forms in the expressions we derive in this section, so that our results extend seamlessly to shells with rest curvature. The norm $\|\cdot\|_{\mathrm{SV}}$ is a quadratic form depending on the material Lamé parameters α and β:

$$\|M\|_{\mathrm{SV}}^2 = \frac{\alpha}{2}\operatorname{tr}^2(M) + \beta\operatorname{tr}(M^2). \tag{2.4}$$

See Appendix A.1 for a more detailed overview of shell theory. Following foundational work in the physics community on wrinkling of sheets Aharoni et al. (2017), our

[2]A linear constitutive model is justified for analyzing the energy of wrinkles, since we expect the residual compressive strain in the wrinkled region (after buckling) to be small. We discuss extending our analysis to other hyperelastic materials in Section 2.7.

fundamental modeling assumption is that the shell midsurface can be decomposed into

$$\boldsymbol{r}(u,v) = \boldsymbol{r}_b(u,v) + \boldsymbol{r}_w(u,v), \tag{2.5}$$

where $\boldsymbol{r}_b$ is a low-frequency *base surface*, from which the shell midsurface is constructed by applying a high-frequency *wrinkle correction* $\boldsymbol{r}_w$. This wrinkle correction can have components in both the normal and tangential directions to the base surface. That is, consider a local frame formed by $\left\{\frac{\partial \boldsymbol{r}_b}{\partial u}, \frac{\partial \boldsymbol{r}_b}{\partial v}, \hat{\boldsymbol{n}}_b\right\}$ at a point on the base surface. The wrinkled shell $\boldsymbol{r}_w(u,v)$ can be expressed as:

$$\begin{aligned}\boldsymbol{r}_w(u,v) &= f_1(u,v)\frac{\partial \boldsymbol{r}_b}{\partial u}(u,v) + f_2(u,v)\frac{\partial \boldsymbol{r}_b}{\partial v}(u,v) + f_3(u,v)\hat{\boldsymbol{n}}_b(u,v)\\ &= \mathrm{d}\boldsymbol{r_b}(u,v)\ \boldsymbol{v}_t(u,v) + f_3(u,v)\hat{\boldsymbol{n}}_b(u,v)\end{aligned} \tag{2.6}$$

where $\boldsymbol{v}_t = (f_1, f_2)$ encodes the tangential displacement due to wrinkling, and f_3 the normal displacement. Note that $\boldsymbol{v}_t$ is a vector field on the *parameter domain* U, which the embedding Jacobian $\mathrm{d}\boldsymbol{r_b}$ maps to a tangent vector on the base surface. We will write $\boldsymbol{I}_b, \boldsymbol{II}_b$ for the first and second fundamental forms of the base surface, respectively.

Notice that we are working with *three* distinct surfaces, each with their own geometry: the (2D) parameterization domain, the (3D) base surface, and the (3D) wrinkled midsurface. We will use subscript u and b to denote quantities associated with the parameterization domain and base surface respectively.

2.3.2 Wrinkle Correction from Wrinkle Fields

As in the analyses of Cerda and others Cerda and Mahadevan (2003); Aharoni et al. (2017); Paulsen et al. (2016), we assume that the wrinkles in the shell have wavelength that can vary spatially over the surface, but that at each point, the wrinkles have a single predominant wavelength and locally coherent wave direction and amplitude. We can thus write the wrinkle correction in terms of the local geometry and deformation of the base surface, together with a non-negative *amplitude* and periodic *phase* field

$$a(u,v) : U \to \mathbb{R}_{\geq 0}, \quad \phi(u,v) : U \to S^1 \cong [0, 2\pi).$$

In particular, we can write f_3 explicitly in terms of amplitude and phase:

$$\boldsymbol{r}_w(u,v) = \mathrm{d}\boldsymbol{r_b}(u,v)\ \boldsymbol{v}_t(u,v) + a(u,v)\cos[\phi(u,v)]\ \hat{\boldsymbol{n}}_b(u,v) \tag{2.7}$$

with base surface normal $\hat{\boldsymbol{n}}_b$, where our remaining task is to determine the in-plane part $\boldsymbol{v}_t$ of the wrinkle correction (which controls the wrinkle shape profile) as well as the wrinkle amplitude and phase fields a and ϕ.

A first observation is that, to a good approximation, surfaces wrinkle in order to compensate for surface area lost to compression. If the strain in the wrinkle direction $\boldsymbol{w}$ is $\epsilon_{\boldsymbol{w}}$, we should expect the arclength of one wrinkle period to match original material length of that period in the shell at rest:

$$\epsilon_{\boldsymbol{w}} \approx \frac{a^2}{2} \|\mathrm{d}\phi\|^2_{\boldsymbol{I}_u^{-1}}. \tag{2.8}$$

This relationship coupling wave amplitude and frequency has been widely exploited, in both physics and computer graphics Rohmer et al. (2010), but it does not tell us the relative magnitude of a and $\mathrm{d}\phi$. We observe that this tradeoff is usually *globally* determined by the boundary conditions, strain, and curvature of the base surface. For instance, if you take a rectangular sheet of paper and shear it by displacing its four corners, the sheet will buckle and form a single, Ω-shaped bump. But if the entire top and bottom boundaries of the sheet are clamped and the sheet is sheared, a high-frequency pattern of wrinkles appears (see Figure 2.8). A second issue is that $\mathrm{d}\phi$ must be integrable, i.e., it must be the derivative of some phase function $\phi : U \to S^1$ which is well-defined, at minimum, on the patches of U where $a > 0$. It is therefore not possible to arbitrarily prescribe $\mathrm{d}\phi$ using Equation (2.8), even if the wrinkle amplitude a could somehow be inferred.

The punchline is that there is no general local rule for choosing a and $\mathrm{d}\phi$, without resorting to user guidance or ad-hoc heuristics. We must instead solve for both terms globally over U.

2.3.3 Fast and Slow Variables

Before dealing with a and ϕ, we begin by analyzing the in-plane correction unknown $\boldsymbol{v}_t$. A tempting idea, which we tried in early unsuccessful experiments, is to solve for $\boldsymbol{v}_t$ along with $\boldsymbol{r}_b$, a, and ϕ, as an additional kinematic degree of freedom. To understand why this approach was misguided, consider that even if we assume no in-plane wrinkle correction ($\boldsymbol{v}_t = \mathbf{0}$), the midsurface shape given by Equations (2.5) and (2.7) is underdetermined. Wrinkles in the midsurface could be represented either by

- adding wrinkles to the base surface $\boldsymbol{r}_b$, and setting $a = 0$;
- using a smooth base surface, but absorbing all midsurface undulation into the amplitude a of the wrinkle field, while keeping $\phi = 0$;
- using a smooth base surface, and a slowly-varying a, with the undulations of the wrinkles induced by variations in the phase ϕ over the surface.

Thinking ahead to when we will want to discretize $\boldsymbol{r}_b$, a, and ϕ on a coarse mesh, it is clear that only the third solution is acceptable, since it is the only one that will not lead to aliasing of the fine wrinkles of $\boldsymbol{r}$ when $\boldsymbol{r}_b$, a, and ϕ are restricted to low frequencies.

Carrying this idea further, we classify variables as *slow* or *fast*. Slow variables change at length scales larger than the wavelength of a single wrinkle, whereas fast variables cannot be approximated as constant, even at the wrinkle scale. Fast variables include ϕ, $\boldsymbol{v}_t$, and $\boldsymbol{r}_b$. We assume that the following variables are slow:

- the base surface fundamental forms $\boldsymbol{I}_b$ and $\boldsymbol{II}_b$; equivalently, the base surface strain and curvature;
- wrinkle amplitude a;
- wrinkle frequency and orientation $\mathrm{d}\phi$;
- the wrinkle shape profile, so that $\boldsymbol{v}_t$ is a superposition of periodic vector fields with slow direction and amplitude.

Notice that the wrinkle frequency $\mathrm{d}\phi$ is slow despite ϕ itself being fast, and that the characterization of a as slow and ϕ as fast breaks the symmetry between a and ϕ both controlling the wrinkle frequency. We will now derive closed-form expressions for $\boldsymbol{v}_t$, based on an analysis of shell statics at the scale of a single wrinkle, baking our assumptions about fast and slow variables into our wrinkle correction model.[3]

2.3.4 In-plane Wrinkle Correction Formulae

Since we assume that the wrinkle shape is a slow variable, $\|\boldsymbol{v}_t\|$ must be a periodic function of ϕ, at the scale of individual wrinkles. Moreover, we can assume without loss of generality that there is no in-plane translation of the wave crests and valleys: $\|\boldsymbol{v}_t\||_{\phi=0} = \|\boldsymbol{v}_t\||_{\phi=\pi} = 0$, since any such translation could be accomplished instead by a phase shift in ϕ. We thus approximate $\boldsymbol{v}_t$ by the first couple of terms in its discrete sine expansion,

$$\boldsymbol{v}_t \approx \boldsymbol{v}_1 \sin\phi + \boldsymbol{v}_2 \sin 2\phi, \tag{2.9}$$

with $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ slow vector fields on U.

[3]When deriving the elastic energy of the wrinkle field (Equation (2.19)), we will also assume that the covariant derivative of wrinkle orientation is negligible. This assumption is justified by the observation that, to good approximation, wrinkles align with the direction of principal tension in the shell, and that these directions do not bend significantly within the material plane (since otherwise the material could further deform to relax the tension).

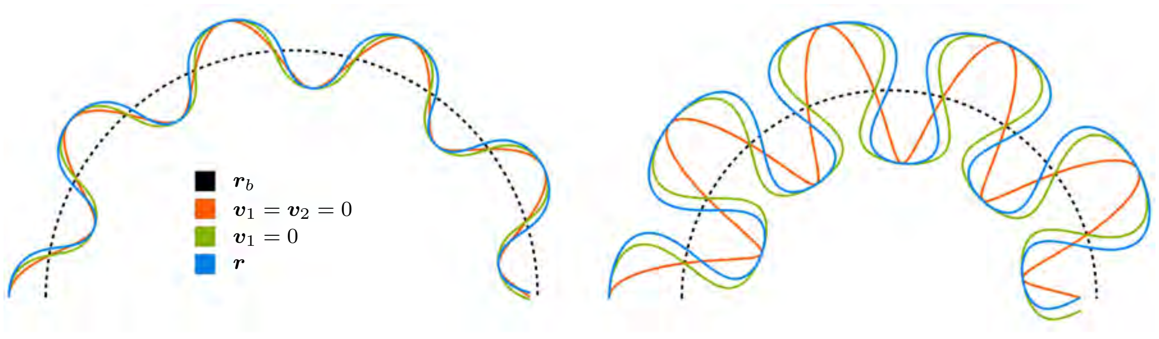


Figure 2.3: 2D sketch of the effect that the two in-plane wrinkle correction modes $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ exert on the wrinkle shape, at small (*left*) and large (*right*) amplitudes. The dotted black curve is the base surface. Purely normal wrinkle displacements (orange) strain the surface significantly near where the wrinkles cross the base surface; the $\boldsymbol{v}_2$ in-plane displacement modifies the wrinkles into a more horseshoe-like shape which equidistributes strain. The $\boldsymbol{v}_1$ term introduces asymmetry between wrinkles above and below the base surface, allowing the wrinkle shape to adapt to curvature of the base surface.

There are geometric reasons for expecting both of these terms to be important (see Figure 2.3). Consider the case of perfectly sinusoidal wrinkles, where $\boldsymbol{v}_t = \mathbf{0}$. These wrinkles induce large variations in strain in the shell over one wavelength, with maximum strain at $\phi = \pi/2, 3\pi/2$ and minimum strain at the wrinkle peaks and valleys. The $\boldsymbol{v}_2$ term allows a redistribution of material within a wrinkle period to equalize strain.

The $\boldsymbol{v}_1$ term accounts for strain variation over one wrinkle period, due to curvature of the underlying base surface. Consider for instance a wrinkled cylinder, where the wrinkling direction $\mathrm{d}\phi$ travels azimuthally around the cylinder (similar to the wrinkles shown in Figure 2.4). Wrinkle peaks are more highly strained than wrinkle valleys, due to the base surface curvature, and the $\boldsymbol{v}_1$ term allows redistribution of material within a wavelength to compensate. Neglecting $\boldsymbol{v}_1$ artificially penalizes coarse wrinkles where the base surface is highly curved in the $\mathrm{d}\phi$ direction, and relatively flat in the perpendicular direction (see also the discussion in Section 2.3.6).

Finding $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ The idea now is to solve for expressions for $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ which minimize the shell's elastic energy density, integrated over a small neighborhood of the surface of size on the order of one wavelength. Since stretching energy dominates bending energy, it is enough to focus on the stretching energy density (2.2), which is quadratic in the midsurface strain.

The directions $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ are slow variables, and so

$$\mathrm{d}\boldsymbol{v}_t \approx (\boldsymbol{v}_1 \cos\phi + 2\boldsymbol{v}_2 \cos 2\phi)\,\mathrm{d}\phi. \tag{2.10}$$

We can use this expression, as well as the definition $\boldsymbol{I} = \mathrm{d}\boldsymbol{r}^T \mathrm{d}\boldsymbol{r}$, to write down a formula for the strain of the midsurface:

$$\begin{aligned}\boldsymbol{I}_u^{-1}\boldsymbol{I} - \boldsymbol{id} \approx\ & \boldsymbol{I}_u^{-1}(\boldsymbol{I}_b - \boldsymbol{I}_u + \frac{1}{2}a^2 \mathrm{d}\phi^T \mathrm{d}\phi) \\ & + \boldsymbol{I}_u^{-1}(-2a\ \boldsymbol{II}_b + [\boldsymbol{I}_b \boldsymbol{v}_1 \mathrm{d}\phi]_T)\cos\phi \\ & + \boldsymbol{I}_u^{-1}(-\frac{a^2}{2}\mathrm{d}\phi^T \mathrm{d}\phi + 2[\boldsymbol{I}_b \boldsymbol{v}_2 \mathrm{d}\phi]_T)\cos 2\phi.\end{aligned} \tag{2.11}$$

Here we use the notation M_T to denote the symmetrization $M + M^T$. All variables in this expression are slow, except for the trigonometric functions in ϕ. Intuitively, strain, and hence elastic energy, is minimized by minimizing each of the trigonometric coefficients; this idea can be formalized by use of a *coarse-graining operator* Aharoni et al. (2017)

$$X^{(\mathrm{cg})} := |\Omega|^{-1} \int_\Omega X dA \tag{2.12}$$

which estimates quantities X averaged over a neighborhood Ω of radius comparable to one wrinkle wavelength. Applying coarse-graining to the thin-shell stretching energy density (2.2) allows us to simplify the energy expression by eliminating oscillatory terms which might be non-zero pointwise, but average to zero (like $\sin\phi$) or a value independent of ϕ (like $\sin^2\phi$) over a wrinkle wavelength:

$$W_s^{(\mathrm{cg})}(\boldsymbol{v}_1,\boldsymbol{v}_2) = \left\|\boldsymbol{I}_u^{-1}\left(\boldsymbol{I}_b - \boldsymbol{I}_u + \frac{1}{2}a^2\mathrm{d}\phi^T\mathrm{d}\phi\right)\right\|_{\mathrm{SV}}^2 \tag{2.13}$$

$$+\frac{1}{2}\left\|\boldsymbol{I}_u^{-1}\left(-2a\ \boldsymbol{II}_b + [\boldsymbol{I}_b\boldsymbol{v}_1\mathrm{d}\phi]_T\right)\right\|_{\mathrm{SV}}^2 \tag{2.14}$$

$$+\frac{1}{2}\left\|\boldsymbol{I}_u^{-1}\left(-\frac{a^2}{2}\mathrm{d}\phi^T\mathrm{d}\phi + 2[\boldsymbol{I}_b\boldsymbol{v}_2\mathrm{d}\phi]_T\right)\right\|_{\mathrm{SV}}^2. \tag{2.15}$$

Notice that $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ appear exclusively in terms (2.14) and (2.15), respectively, and so we can solve for these variables by minimizing each term independently:

$$\boldsymbol{v}_1 = a\boldsymbol{I}_b^{-1}\boldsymbol{v},\quad \boldsymbol{v}_2 = \frac{a^2}{8}\boldsymbol{I}_b^{-1}\mathrm{d}\phi^T \tag{2.16}$$

where

$$\boldsymbol{v} = \left(\frac{\alpha}{\alpha+2\beta}\frac{\mathrm{tr}(\boldsymbol{I}_u^{-1}\boldsymbol{II}_b)}{\|\boldsymbol{w}\|_{\boldsymbol{I}_u}^2} + \frac{2\beta}{\alpha+2\beta}\frac{\boldsymbol{w}^T\boldsymbol{II}_b\boldsymbol{w}}{\|\boldsymbol{w}\|_{\boldsymbol{I}_u}^4}\right)\mathrm{d}\phi^T \tag{2.17}$$

$$+2\frac{\boldsymbol{w}^T\boldsymbol{II}_b\boldsymbol{w}^\perp}{\|\boldsymbol{w}\|_{\boldsymbol{I}_u}^4}\left(\mathrm{d}\phi^\perp\right)^T; \tag{2.18}$$

where to de-clutter notation we define $\boldsymbol{w} = \boldsymbol{I}_u^{-1}d\phi^T$ to be the vector field on U aligned with the wrinkle travel direction, and $\boldsymbol{w}^\perp, d\phi^\perp$ the vector field and one-form orthogonal to $\boldsymbol{w}$ and $d\phi$ under the natural $\boldsymbol{I}_u$ and $\boldsymbol{I}_u^{-1}$ inner products, respectively. The constants α and β are the Lamé parameters. Notice that on a flat sheet, $\boldsymbol{v}_1 = 0$, confirming the intuition that the role of this term is to modulate the wrinkle shape to accommodate curvature on the underlying base surface. See Appendix A.3 for the derivation of these expressions.

To summarize, we now can replace the wrinkle correction term $\boldsymbol{r}_w$ in Equation (2.5) with an explicit formula in terms of $\boldsymbol{r}_b$, a, and ϕ, based on the above derivations of the in-plane part of the correction term:

$$\boldsymbol{r} \approx \boldsymbol{r}_b + \mathrm{d}\boldsymbol{r_b}\boldsymbol{I}_b^{-1}\left(a\sin\phi\,\boldsymbol{v} + \frac{a^2}{8}\sin 2\phi\,\mathrm{d}\phi^T\right) + a\cos\phi\hat{\boldsymbol{n}}_b. \tag{2.19}$$

2.3.5 Wrinkle Field Energy

Equipped with the formula (2.19) giving the geometry of the midsurface as a function of the base surface $\boldsymbol{r}_b$ and wrinkle field a, ϕ, we can now write down an expression for the elastic energy of the wrinkled shell in terms of these kinematics. Note that, unlike in Section 2.3.4, where we were operating at the length scale of individual wrinkles, and could assume that slow variables are approximately constant, the elastic energy of the shell is a *global* quantity and we cannot simply eliminate terms depending on e.g. the amplitude field derivative $\mathrm{d}a$ which, while they might be negligible locally since a is slow, might integrate up into a non-negligible energy contribution on the scale of the entire shell.

Deriving the elastic energy of the wrinkle field is, in principle, simply a matter of plugging in Equation (2.19) into the Koiter energy (2.1) and simplifying by analyzing the scaling of each energy contribution and eliminating negligible terms. Here we summarize the results of this (in practice, rather involved) procedure (see Appendix A.4 for the derivation).

We consider the stretching and bending terms in the energy separately.

Stretching Term As discussed above, the expression in Equation (2.11) cannot be used to measure the strain of the midsurface for the purpose of calculating elastic energy, since that formula neglected terms that are non-negligible on the scale of the whole shell. In Appendix A.4 we derive the following energy expression, by coarse-graining and simplifying the first term in Equation (2.1). We make use of our assumptions in Section 2.3.3 about fast and slow variables:

$$E_s = \int_U \frac{h}{4}\left(W_s^1 + W_s^2 + W_s^3 + W_s^4\right)\sqrt{\det \boldsymbol{I}_u}\, du dv \tag{2.20}$$

$$W_s^1 = \left\| \boldsymbol{I}_u^{-1}\left(\boldsymbol{I}_b - \boldsymbol{I}_u + \frac{1}{2}\mathrm{d}a^T\mathrm{d}a + \frac{1}{2}a^2\mathrm{d}\phi^T\mathrm{d}\phi\right)\right\|_{\mathrm{SV}}^2 \tag{2.21}$$

$$W_s^2 = 4a^2\kappa_\perp^2\frac{\beta(\alpha+\beta)}{\alpha+2\beta}\left(\frac{\left(\boldsymbol{w}^\perp\right)^T\boldsymbol{I}_b\boldsymbol{w}^\perp}{\left(\boldsymbol{w}^\perp\right)^T\boldsymbol{I}_u\boldsymbol{w}^\perp}\right)^2 \tag{2.22}$$

$$W_s^3 = \frac{a^2}{32}\left\|\boldsymbol{I}_u^{-1}(\mathrm{d}\phi^T\mathrm{d}a + \mathrm{d}a^T\mathrm{d}\phi)\right\|_{\mathrm{SV}}^2 \tag{2.23}$$

$$W_s^4 = \frac{1}{8}\left\|\boldsymbol{I}_u^{-1}\mathrm{d}a^T\mathrm{d}a\right\|_{\mathrm{SV}}^2. \tag{2.24}$$

where the vector field parallel to the wrinkle crests $\boldsymbol{w}^\perp$ is defined as in Equation (2.19), and $\kappa_\perp = \left(\boldsymbol{w}^\perp\right)^T \boldsymbol{II}_b\boldsymbol{w}^\perp / \left\|\boldsymbol{w}^\perp\right\|_{\boldsymbol{I}_b}^2$ is the normal curvature of the base surface along the $\boldsymbol{w}^\perp$ direction.

Although this expression is rather involved, each term can be understood, in retrospect, in the context of the geometry of the fine-scale wrinkling:

- In regions of pure tension, the base surface strain tensor is positive-definite. Since $\mathrm{d}a^T\mathrm{d}a$ and $a^2\mathrm{d}\phi^T\mathrm{d}\phi$ are also positive-semidefinite, *any* amount of wrinkling simply drives up the energy contribution W_s^1; this term therefore inhibits wrinkling in regions of pure tension.

- Both $\mathrm{d}a$ and $\mathrm{d}\phi$ play an interchangeable role in W_s^1. However as discussed in Section 2.3.3, we want wrinkle frequency, and *not* wrinkle amplitude, to absorb compression strain. W_s^4 penalizes large variations in winkle amplitude, enforcing that the compression is absorbed by $\mathrm{d}\phi$, not $\mathrm{d}a$ in W_s^1. In other words, W_s^4 enforces that ϕ is the fast variable, and a is slow.

- In regions of mixed tension and compression, assuming that amplitude is constant ($\mathrm{d}a = 0$), W_s^1 is minimized by aligning the wrinkle direction $\mathrm{d}\phi$ with the principal compression direction. Moreover, we therefore recover the intuitive coupling between amplitude, frequency, and strain described in Equation (2.8).

- The W_s^2 term injects dependence on curvature into the determination of wrinkle amplitude and frequency: high-amplitude wrinkles whose peaks and valleys run along directions of high curvature are penalized. Wrinkles where the isolines of ϕ run along directions of low curvature do not suffer this penalty, even if the surface is highly-curved in the wrinkle travel direction $\mathrm{d}\phi$. This behavior matches the expected effect of the $\boldsymbol{v}_2$ on wrinkle shape, as described in Section 2.3.4.

- Finally, W_s^3 penalizes large changes in amplitude along crests of the wrinkle waves. This matches intuition: amplitude is free to change from one wave to the next, but since wrinkle crests align with directions of tension in the shell, we do not expect variations in wrinkle amplitude *along* one wrinkle.

Bending Term Bending of the wrinkled shell at two scales contributes to the total bending energy: 1) bending of the base surface itself, which contributes energy with formula directly analogous to Equation (2.3); 2) bending at the fine scale, due to wrinkling. We estimate this latter term from Equation (2.19) by assuming that the contribution to wrinkle curvature from variations in ϕ dominate any contributions from changes in a or $\hat{\boldsymbol{n}}_b$. In other words, at the fine scale $\boldsymbol{II}_u \approx 0$ and

$$\boldsymbol{I}_u^{-1}\boldsymbol{II} = \boldsymbol{I}_u^{-1}\mathrm{d}^2 r \cdot \hat{\boldsymbol{n}}_b \approx -a\cos\phi \boldsymbol{I}_u^{-1}\mathrm{d}\phi^T\mathrm{d}\phi. \tag{2.25}$$

Coarse-graining this second contribution and adding it to the first yields an expression for bending energy,

$$\begin{gathered} E_b = \frac{h^3}{12}\int_U \left(W_b^1 + W_b^2\right)\sqrt{\det \boldsymbol{I}_u}\,\mathrm{d}u\mathrm{d}v, \\ W_b^1 = \left\|\boldsymbol{I}_u^{-1}(\boldsymbol{II}_b - \boldsymbol{II}_u)\right\|_{\mathrm{SV}}^2, \quad W_b^2 = \frac{a^2}{2}\left\|\boldsymbol{I}_u^{-1}\mathrm{d}\phi^T\mathrm{d}\phi\right\|_{\mathrm{SV}}^2. \end{gathered} \tag{2.26}$$

To sum up, the total elastic energy, in terms of the base surface embedding $\boldsymbol{r}_b$ and the wrinkle field degrees of freedom a and ϕ, is

$$E^{\mathrm{wf}} = E_s + E_b. \tag{2.27}$$

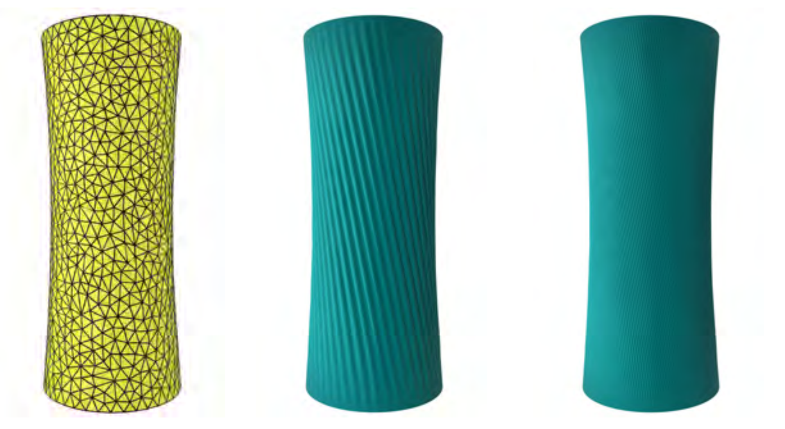

(a) TFT base mesh *(b) Our (TFW) result* *(c) TFW without $\boldsymbol{v}_1$ term*

Figure 2.4: Simulated cylinder wrinkles using our model (see Section 2.4) on a cylinder whose boundaries have been clamped and twisted: (a) the $1.3k$-vertex base mesh, (b) our result with the $\boldsymbol{v}_1$ in-plane correction term included, and (c) our result with this term omitted. With in-plane correction, we get 46 waves, which is consistent with results from traditional shell solvers (see Figure 2.21). Without in-plane correction that accounts for the base surface curvature, our model predicts spuriously high wrinkle frequency (over 180 wrinkles; you may need to zoom in to see them).

Notice that this energy depends only on the wrinkle frequency and direction ($\mathrm{d}\phi$) and not directly on the phase itself (ϕ); as expected since applying a global phase shift to ϕ changes only fast variables, and should not change the coarse-grained energy of the full shell. We will exploit this invariance when we discretize Equation (2.27) in Section 2.5.

2.3.6 Necessity of In-plane Wrinkle Correction

Given the complexity of the stretching energy E_s, one might look for further simplifications. One source of the complexity are the in-plane wrinkle correction terms in Equation (2.19). Dropping one or both of these terms (and in doing so, modifying our assumptions about the shape of the wrinkles at the fine scale) significantly modifies the energy E_s. Without $\boldsymbol{v}_2$, there is no longer an asymmetry (from W_s^4) between frequency induced by $\mathrm{d}\phi$ and by $\mathrm{d}a$; in other words, there is no longer enforcement of a being slow and ϕ being fast. Dropping $\boldsymbol{v}_1$ significantly changes the curvature term W_s^2. Both modifications severely degrade the accuracy of wrinkles we recover using the numerical procedure we next describe in Section 2.4, suggesting that both in-plane correction terms, and all resulting terms in E_s, are essential to the wrinkle model. In Figure 2.4 we perform an ablation study on $\boldsymbol{v}_1$.

2.4 Solving for Static Wrinkled Shape

We now describe an algorithm for using the wrinkle field model of Section 2.3 to optimize for the static equilibrium shape of draped cloth, inflatable structures, and other wrinkled shells.

One could entertain jointly optimizing the wrinkle field elastic energy in Equation (2.27) for both the base surface shape as well as the a and ϕ fields. The challenge with this approach, though, is that our analysis in Section 2.3 assumed that the base surface was smooth, with curvature a *slow* variable. Optimizing jointly for both the base surface and wrinkle fields without enforcing the slow-ness of $\boldsymbol{r}_b$ leads to spurious, aliased solutions similar to Figure 2.2. See Section 2.7 for more discussion of joint optimization. Here we instead propose a simpler scheme, where we first estimate the base surface shape, and then fix $\boldsymbol{r}_b$ and separately solve for the wrinkle field parameters a, ϕ on that base surface.

2.4.1 Computing the Base Surface

As argued in the introduction, a low-resolution FEM simulation of the shell is not a good choice for $\boldsymbol{r}_b$: not only does the coarse simulation lack high-frequency wrinkles, it also already possesses *incorrect* aliased, low-frequency wrinkles which our wrinkle field model is not equipped to correct.

For *tension-dominated* problems, optimizing $\boldsymbol{r}_b$ using tension field theory is a better choice. A thin shell is tension-dominated if, at most points on the surface, the shell is either in pure tension, or in a mix of tension and compression (it is in these latter regions that wrinkles typically form). Examples of tension-dominated shells include garments hung or draped against gravity, chambers under internal pressure, and fabrics or films tugged by external loads or boundary constraints. Shells under primarily pure compression (such as cloth that is balled up on the floor, or axially-crushed coke cans) are not tension-dominated, and out of scope for our approach.

Tension field theory exploits the fact that wrinkles play little role in force transmission, by formulating a simplified description of local stresses as having only tension components, directed along directions of positive principal stress Mansfield (1989); Steigmann (1990); Pipkin (1986, 1994). The theory can be viewed in terms of a modified or *relaxed* variant of the membrane energy density in Equation (2.2) Pipkin (1986, 1994); see Appendix A.2 for an overview. *TFT* optimizes for the coarse envelope of the shell, ignoring local high-frequency wrinkling by allowing the shell to compress without buckling. The *TFT* result is thus ideal as a base surface $\boldsymbol{r}_b$ for wrinkle augmentation, using the theory of wrinkle fields developed in Section 2.3. Since the *TFT* solution expresses only low-frequency geometric features, it remains accurate even for very coarse discretizations of the domain U; moreover the modified stretching energy is *convex* and efficient to minimize Skouras et al. (2014). Figure 2.16, left, shows a *TFT* simulation of a dress draped on a mannequin. Notice the smooth, wrinkle-free shape and coarse tessellation. We present many more *TFT* base meshes in Section 2.6.

Base Surface Bending Model Tension field theory gives a convex replacement for stretching energy, when solving for the base surface shape $\boldsymbol{r}_b$. With an eye towards additional improvements in efficiency and robustness of the base mesh solve, for rest-flat shells we recommend also replacing the elastic bending energy density (Equation (2.3)) with Bergou et al. (2006)'s Quadratic Bending model,

$$W_b^{\mathrm{qb}} = \alpha \langle \Delta_u \boldsymbol{r}_b, \Delta_u \boldsymbol{r}_b \rangle_{\boldsymbol{I}_u}, \tag{2.28}$$

where Δ_u denotes the Laplace-Beltrami operator on U with respect to the metric $\boldsymbol{I}_u$, and α is the first Lamé parameter. We also adopt Wang et al. (2015)'s correction terms to account for the bending energy at the free shell boundaries.

Advantages of quadratic bending over the full shell bending energy include:

- W_b^{qb}, being quadratic in the base surface embedding $\boldsymbol{r}_b$, is convex, so that the total elastic energy of the base mesh (TFT+Quadratic Bending) is convex;
- the energy is very easy to implement.

Quadratic Bending assumes that the shell is deforming isometrically; this assumption is violated in many of our examples, where the base surface compresses significantly. Nevertheless, we did not observe much change in the shape of the base mesh by substituting Quadratic Bending for Equation (2.3). For the dress examples in Figures 2.14 and 2.16, the Hausdorff distance between the base surface computed using full bending, and Quadratic Bending, is 1% of the dress diameter.

2.4.2 Computing the Wrinkle Fields

Once we have found the base surface, we compute a wrinkle field over that surface by minimizing Equation (2.27). Here we describe a couple of additional simplifications and reformulations we recommend to simplify this task.

Approximation of $\boldsymbol{w}$ Notice that in the W_s^2 term in the wrinkle field stretching energy, the *direction* of $\boldsymbol{w}$ is used (both in computing $\kappa_\perp$ and in the formula for W_s^2 itself), but not its *magnitude.* Moreover, as discussed in Section 2.3, the main unknown in solving for the wrinkle field is the tradeoff between amplitude and wrinkle frequency. Wrinkle direction is strongly encouraged to align with the principal compression direction of base surface strain, by the energy term W_s^1. We therefore approximate $\boldsymbol{w}$ as a constant (rather than a function of $d\phi$) given by the solution to the generalized eigenvalue problem

$$(\boldsymbol{I}_b - \boldsymbol{I}_u)\boldsymbol{w} = \lambda \boldsymbol{I}_u \boldsymbol{w} \tag{2.29}$$

with most negative eigenvalue λ. In regions where neither eigenvalue is negative (i.e., in regions of pure tension) we simply ignore the W_s^2 term, under the assumption that $a \approx 0$ in those regions.

Solving for Frequency Instead of Phase In our formulation in Section 2.3, we assume that ϕ is a periodic function. The need for periodicity is evident even in very simple wrinkling scenarios, such as the drape of a square piece of cloth over a sphere (see Figure 2.19): rotationally symmetric wrinkles appear around the circumference of the draped cloth, corresponding to a ϕ which continuously linearly increases as you circulate around the draped portion of the cloth (as shown in Figure 2.19, middle-right). Since Equation (2.27) depends only on the phase field derivative $\mathrm{d}\phi$ and not on the phase itself, we eschew optimizing for ϕ, and instead borrow from the surface parameterization and stripe pattern optimization Knöppel et al. (2015) literature the idea of expressing the wrinkle field elastic energy in terms of the one-form $\omega = \mathrm{d}\phi$. In other words, we solve

$$\begin{aligned} &\arg\min_{a,\omega} E^{\mathrm{wf}}(a, \omega) \\ \text{s.t.} \quad &a \geq 0, \\ &\forall \boldsymbol{p} \in U,\ a(\boldsymbol{p}) = 0 \text{ or } \operatorname{curl} \omega(\boldsymbol{p}) = 0. \end{aligned} \tag{2.30}$$

Here the first constraint enforces that wrinkles cannot have negative amplitude, and the second ensures that the recovered ω can be written, at least locally, as the derivative of a phase field ϕ, everywhere where wrinkles are visible (amplitude is positive). We provide more detail about how to discretize and solve this variational problem in the next section and in Appendix A.6.

2.5 Discretization and Solver

Our pipeline for computing and visualizing the static shape of shells using a *TFT* base mesh and a wrinkle field consists of the following steps:

1. We triangulate U into a coarse simulation mesh $K = (V_u, F, E)$, with vertices $V_u = \{\mathbf{v}_u^1, \mathbf{v}_u^2, \cdots\}$ (Appendix A.6.1).

2. We solve for the base mesh embedding $V_b = \{\mathbf{v}_b^1, \mathbf{v}_b^2, \cdots\}$ by minimizing the *TFT* and Quadratic Bending energies (Section 2.4.1).

3. We represent a as a function on the vertices of K, and ω as a one-form on the mesh edges. We estimate an initial guess for these variables, based on V_u and V_b (Appendix A.6.2).

4. We also locate the faces of K on which the base mesh exhibits pure tension. We collect these faces into a set of *wrinkle-free faces* W (Section 2.5.1).

5. We solve the optimization problem in Equation (2.30) for a and ω, using sequential quadratic programming. We relax the complementarity constraint in this problem by making use of the wrinkle-free faces W (Section 2.5.2 and Appendix A.6.3).

6. We integrate ω to recover the phase field ϕ. Each face maintains a separate value of ϕ for its three vertices (i.e., $\phi \in \mathbb{R}^{3|F|}$) to support the 2π-periodicity of the phase field (Appendix A.6.4).

7. Finally, we visualize the result by upsampling the base mesh and wrinkle fields using Loop subdivision, and displacing the resulting geometry using Equation (2.19) (Appendix A.6.5).

The output of the above *Tension Field + Wrinkle* (*TFW*) pipeline is our prediction of the shell static shape, shown in Figure 2.16, and throughout Section 2.6. In what follows, we discuss the details most critical to understanding and implementing the above pipeline. Further implementation details can be found in Appendix A.6.

2.5.1 Relaxed Integrability

As we wrote in Equation (2.30), when minimizing elastic energy with respect to a and ω, we will want to maintain that $\operatorname{curl} \omega = 0$ everywhere *except* in regions where wrinkles do not exist (amplitude vanishes), so that we can (locally) integrate ω into the phase field ϕ. As has been observed many times by researchers in surface parameterization Kälberer et al. (2007); Bommes et al. (2009), it is absolutely crucial that singularities, where $a = 0$ and $\operatorname{curl} \omega \neq 0$, be allowed to exist. These singularities are topologically required to recover reasonable one-forms ω, even on simple examples. Consider for instance again the example of a square cloth draped on a sphere, shown in Figure 2.19. The obvious solution ω to recover correct wrinkling of the cloth is for ω to circulate (like a whirlpool) around the square's center; clearly the path integral of ω around the square's boundary here is nonzero. But then it is impossible for $\operatorname{curl} \omega = 0$ at every point on the square (doing so would violate the fact that every closed one-form on a simply-connected region is exact).

Ideally, our solver would automatically place these singularities in the optimal location (at the north pole of the sphere, in this case), by enforcing the complementarity constraint in Equation (2.30) on a and $\operatorname{curl} \omega$. However, we are not aware of a simple algorithm for doing so. Instead we make the following observations, leading to a heuristic for placing singularities in reasonable locations a priori:

- in regions under pure tension, the energy term W_s^1 strongly penalizes placing any wrinkles at all. It is reasonable to therefore assume that $a = 0$ in these regions;
- conversely, a singularity where amplitude vanishes is most likely to be located in a region of pure tension, than in a region of mixed tension and compression or pure compression (where W_s^1 penalizes the lack of wrinkling).

Our heuristic, therefore, is to force $a = 0$ on a set of *wrinkle-free faces* W that are under pure tension, and to relax integrability of ω on those faces, allowing the solver to anchor singularities at these faces. To compensate for the coarseness of the base mesh (which often only has thousands or hundreds of triangles), we perform some smoothing before classifying triangles as being under pure tension: we average each triangle's most-negative eigenvalue with that of its three neighbors, and if this average is positive, add the triangle to W. After this procedure, we filter outliers from W by removing any triangle in W whose three neighbors are not members, and adding to W any triangle whose three neighbors are already members.

Figure 2.19, right, shows the results of *TFW* with and without this relaxation of the curl constraint on wrinkle-free faces. Without relaxation (top-right), we do not recover a reasonable phase field, since the global integrability constraint will force ϕ to zig-zag as you travel around the square's circumference, rather than increasing in a smooth gradient.

2.5.2 Optimization

We discretize a using piecewise-linear elements over K, and discretize ω as a one-form on the edges E, in the style of discrete exterior calculus. On triangles where integrability of ω is enforced, we do so via the linear constraint $\mathrm{d}\omega = 0$. We treat first and second fundamental forms $\boldsymbol{I}_u, \boldsymbol{II}_u$ as constant over each triangle; see Appendix A.1 for the relevant formulas. We can then write the energy of Equation (2.30) as a sum of integrals over the triangles not in W; we compute these integrals using three-point quadrature Zhang et al. (2009).

After the above discretization, finding a and ω amounts to minimizing a degree-eight polynomial objective function, subject to linear equality constraints (enforcing integrability) and inequality constraints (enforcing positive amplitudes). We solve this problem via sequential quadratic programming, applying the NASOQ QP solver Cheshmi et al. (2020) to compute the constrained descent directions at each iteration. For more details about the discretization and solver implementation, please see Appendix A.6.3.

2.6 Evaluation and Discussion

Below we consider the behavior of the *TFW* pipeline on a wide range of test examples designed to investigate both fidelity to experimental results as well as comparison to results obtained by large degree-of-freedom, traditional shell simulators. For the latter we utilize three representative simulators:

- As a best-in-class academic library for shell modeling we apply *ARCSim* Narain et al. (2012b, 2013), a widely deployed dynamics cloth simulator. Critically for our comparison, *ARCSim* offers the option of adaptive remeshing, which allows traditional shell simulation to capture fine wrinkle details with lower degree-of-freedom meshes by only refining specific triangles. To apply *ARCSim* for solving statics we apply critically damped time-steps to equilibria. In the following, for each ARCsim example, we will indicate when the adaptive remeshing option is applied or not.

- As a high-performance commercial cloth simulator we also include comparisons with *Marvelous Designer* CLO Virtual Fashion Inc (2020), a widely-used industrial garment-design tool that deploys its own proprietary statics physics solver for predicting garment drape. In the following we will denote this simulator as *MD*.

- Finally, to compare with a baseline, consistent shell model we implement a thin shell statics simulator that solves for the St. Venant-Kirchhoff material model with constant-strain stretching elements and mid-edge bending (Morley) elements Weischedel (2012); Chen et al. (2018a); Grinspun et al. (2003). In the following we will denote this simulator as *StVK*. We made a best effort to optimize this baseline code, while restricting computation to the CPU; 70%

of each Newton iteration is consumed by our chosen third-party linear solver (SuiteSparse's CHOLMOD Chen et al. (2008)) which gives us confidence that our code is free of gross inefficiencies. See Appendix A.5.1 for a detailed timing breakdown.

The full data set of meshes used in our evaluation and the source code of both our *TFW* and *StVK* reference implementations can be found here.

2.6.1 Draping Behavior

We begin by analyzing the qualitative behavior of *TFW* on a series of real-world draping examples exhibiting complex wrinkling geometries and nontrivial wrinkle topology. Notice that unlike data-driven wrinkle-recovery methods, our method is purely model driven. No user guidance nor extra data beyond material parameters and boundary conditions are required. For examples with this level of complexity, exact comparisons do not make sense (many of these drapes likely have multiple metastable states, even before taking differences in how each solver models physics and frictional contact into account). In later sections we will perform exact comparisons on simpler model problems where experimental and/or analytic results are known; here, instead, we show that *TFW* results are comparable qualitatively to those generated by traditional solvers. To set the material parameters, we first choose one of the predefined fabrics given in *ARCSim* (Navy Sparkle Sweat), then derive approximate physical parameters from the provided bending and stretching stiffnesses. We get the following material parameters for *TFW* and *StVK*: density 200kg/m^3, 0.3 Poisson's ratio, 0.1MPa Young's modulus and thickness 1mm. *MD* does not expose these same material parameters, so we instead manually select a material from *MD*'s predefined palette of material types that gives us closest-matching results to *StVK*/*ARCSim*.

Sphere Drape We first drape a square cloth (1m$\times$1m) over a sphere of radius 0.2m. In Figure 2.19 we demonstrate the resulting base surface $\boldsymbol{r}_b$, from the joint *TFT* and Quadratic Bending modeling on a domain with 1.6k vertices, and the resulting *TFW*-generated surface after we visualize the wrinkle field on the base surface. We also show results for *ARCSim* (with adaptive remeshing) and *MD*: for the former, the final adaptive mesh has 32k vertices. For *MD*, we manually search for the coarsest mesh able to resolve the cloth's wrinkles; in this case, 72k vertices. The resulting *TFW* wrinkles are qualitatively similar in frequency and amplitude, especially compared to *MD*, although, as discussed, it is hard to infer ground truth here since we see broken symmetry in both *ARCSim* and *MD* results, where two sides of the front corner have differing shapes, suggesting that there are many metastable solutions. One noticeable difference is that *TFW* does not "puff out" as much as the other two simulations; and the wrinkles do not collapse and flatten under their own weight in the manner that is seen in the *ARCSim* and *MD* results. See Section 2.7 for more discussion of these limitations.

Garments We next consider garment drapes. We pose two dress patterns (from the Berkeley Garment Library) and a pair of pants on mannequins. Figures 2.14, 2.15, and 2.16 detail results and relevant mesh resolutions for all solvers. These examples all demonstrate complex geometry and topology; notice that our results recover wrinkle fields with widely-varying frequency and direction, and that despite the coarse base mesh employed, *TFW* generates results qualitatively close to the three traditional, high-degree-of-freedom cloth solvers. The last column in Figures 2.14, 2.15, and 2.16 present final results when employing *ARCSim* with *adaptive remeshing* enabled. Here these adapted-mesh examples give a sense as to why traditional simulations generally succeed in recovering wrinkle details only when the simulation mesh is at quite high resolution (over 40k vertices were needed for these three garment examples). Our *TFW* model generates comparable wrinkle patterns with just a few thousands vertices per example (3.4k, 1.4k and 4k for the two dresses and pants respectively).

2.6.2 Resolution Analysis

We consider the sensitivity of traditional shell simulators to their mesh resolution, and contrast with the *TFW* pipeline's low-resolution requirements for its base mesh. To do so we study the asymmetric dress as well as a new example, a twisted cylinder (see Figure 2.21): here a cylindrical (but rest-flat) shell of radius 1m, height 5m, and thickness 0.1mm, clamped at the top and bottom boundaries, is twisted at the top boundary by $10°$ while keeping the distance between the top and bottom boundaries constant. This example uses 0.44 Poisson's ratio and 10MPa Young's modulus. The cylinder example is ideally suited for a convergence analysis, as the number of wrinkles around the cylinder is objective, discrete, and easily-counted. We can thus use the number of wrinkles as a proxy for how rapidly each simulation method converges under refinement.

We first test *TFW* on irregular meshes for these examples with increasing resolutions. For the dress, we start from a base mesh with only 382 vertices and monotonically increase the mesh resolution. We observe that our model converges to a consistent shape at $\approx 1.4k$ vertices (see Figure 2.18). Applying the same test on the twisted cylinder yields an even more impressive result, where a consistent shape emerges at ≈ 180 vertices (see Figure 2.21).

We next probe the behavior of the traditional solvers by performing the following experiments: (1) we run *StVK*, for irregular meshes of increasing resolution; (2) we do the same for *ARCsim*, with adaptive remeshing disabled in order to force use of a mesh with given resolution; and (3) we run *ARCSim* with adaptive remeshing enabled. See Figures 2.20 and 2.21 for results. Clear aliasing of high frequencies are evident, as expected, at coarse resolutions. For the twisted cylinder, it is difficult to say how close *StVK* and *ARCSim* are to converging, as the wrinkle number keeps changing for them even for meshes with over $96k$ vertices. Similarly, adaptive remeshing also has trouble determining a final, consistent state. Here we show the result of adaptive meshing after waiting two days (8,000 simulation steps). This example illustrates that adaptive meshing is not a silver bullet for effectively resolving wrinkle features: adaptivity does not provide appreciably smaller meshes in examples like this when fine wrinkles cover a large portion of the shell surface. Moreover, this example illustrates how local refinement can introduce artifacts in the final shape: earlier decisions about where to refine biases how the shell later deforms; see the unequal wrinkle spacing in Figure 2.21, left column.

Unlike the cylinder example, a careful convergence analysis is not possible for the dress example, as we observe that for all methods, the same code and the same mesh with different initializations can converge to different solutions. We can determine that the traditional simulators all appear to roughly converge for this example with meshes in the range of about 20–40k vertices for each solver. We also observe that *ARCSim*, both with and without remeshing, yields qualitatively different solutions for the dress example. Despite the lack of quantitative certainly, we can confidently conclude, however, that the resolution required to reproduce wrinkles with frequency and fidelity qualitatively equal to those predicted by our method generally requires approximately an order of magnitude higher resolution than *TFW* for all three simulators (*ARCSim*, *StVK*, and *MD*) for both the cylinder and dress examples.

2.6.3 Meshing Independence

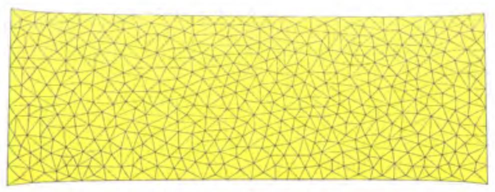
We next probe mesh-dependence of *TFW* by testing four differently generated simulation meshes for the same dress-drape example. We consider meshes generated by: (a) direct Delaunay triangulation (via the Triangle library Shewchuk (1996)); (b) upsampling from a lower-resolution Delaunay mesh, where we use a modified Loop subdivision to keep vertices along seams of the garment stitch pattern unchanged; (c) downsampling from a finer Delaunay mesh, where we also keep the seam vertices unchanged; and (d) extracting a mesh from *ARCSim* after allowing it to take one step with its adaptive remeshing enabled. We run our *TFW* pipeline for each of the four simulation meshes. Figure 2.17 demonstrates that, despite these variations in triangulation, the final wrinkled shapes are consistent.

However, as the tension field theory provides no penalty on compression, a given amount of compression applied to region of a surface may not equidistribute within that region, so that compressive strain may end up concentrated in only a narrow strip of triangles. In practice we do observe some inconsistency ($\approx$15%) in our model (see the third and fourth columns of Figure 2.21) for regular, structured meshes, due to this phenomenon. For example, while for the irregular Delaunay mesh (1.3k vertices) we get 46 waves, for the regular mesh generated by diagonalizing a regular $N \times K$ quad mesh, where N refers to the number of uniform azimuthal samples, and K the number of uniform axial samples, we get varying wave numbers: for the ones diagonalized to align with the twist direction, we have 39 and 38 waves, and for the ones with opposite diagonalization, this number is 40, where we tested with $N = 80, K = 12$, and $N = 120$ and $K = 9$.

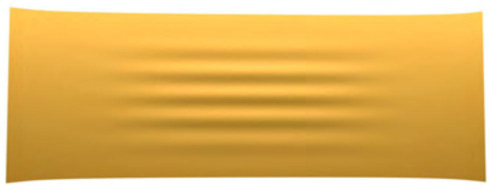
In turn, we observe that traditional solvers can also suffer from artifacts on regular meshes aligned unfavorably with the wrinkles. First consider the *StVK* simulator. The last three columns in Figure 2.21 illustrate this effect, where we test the *StVK* simulator with mesh resolutions ranging from 6k vertices and then doubling resolution until 96k. The seventh column in Figure 2.21 shows the results for irregular Delaunay meshes, where the wrinkle frequency increases from 10 to 19. For the regular mesh diagonalized along the twist direction, the number of wrinkles oscillates (the eighth column in Figure 2.21), whereas for the meshes diagonalized in the opposite way, this number increases from 12 to 20 (the ninth column in Figure 2.21). For such problems, we observe that even *ARCSim* has convergence issues, ultimately producing an irregular wrinkle pattern with 69k vertices; see the first two columns in Figure 2.21.

2.6.4 Accuracy

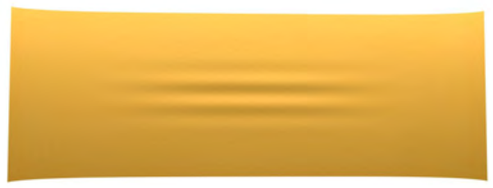
To further investigate accuracy of the *TFW* model we now consider examples where the wrinkling behavior is known either through analysis or experiment.



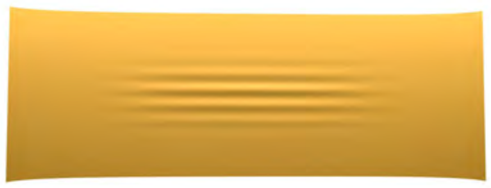
(a) TFT base mesh



(b) TFW



(c) StVK with 260k vertices



(d) ARCSim

Figure 2.5: Simulation of wrinkles in a highly stretched sheet Wang et al. (2018). (a) *TFT* base mesh with 522 vertices. (b) *TFW* predicts correct wrinkling to within 20% of theoretical values with the given base mesh. (c) *StVK* will also predict qualitatively-correct wrinkles, however only starting with meshes at a resolution of 260k-vertices and higher. (d) *ARCSim*'s result, with adaptive remeshing. The final mesh has 13k vertices.

Stretched Sheets In a pioneering experiment, Cerda and Mahadevan (2003) studied wrinkling in a thin sheet whose left and right boundaries are clamped and then pulled apart. The sheet compresses in the perpendicular (vertical) direction due to Poisson's ratio and horizontal wrinkles appear. They provide scaling laws for determining the wrinkle frequency and amplitude. A range of successive works have extended the detailed study of this problem analytically and numerically with varying elasticity models Healey et al. (2013); Li and Healey (2016); Wang et al. (2018).

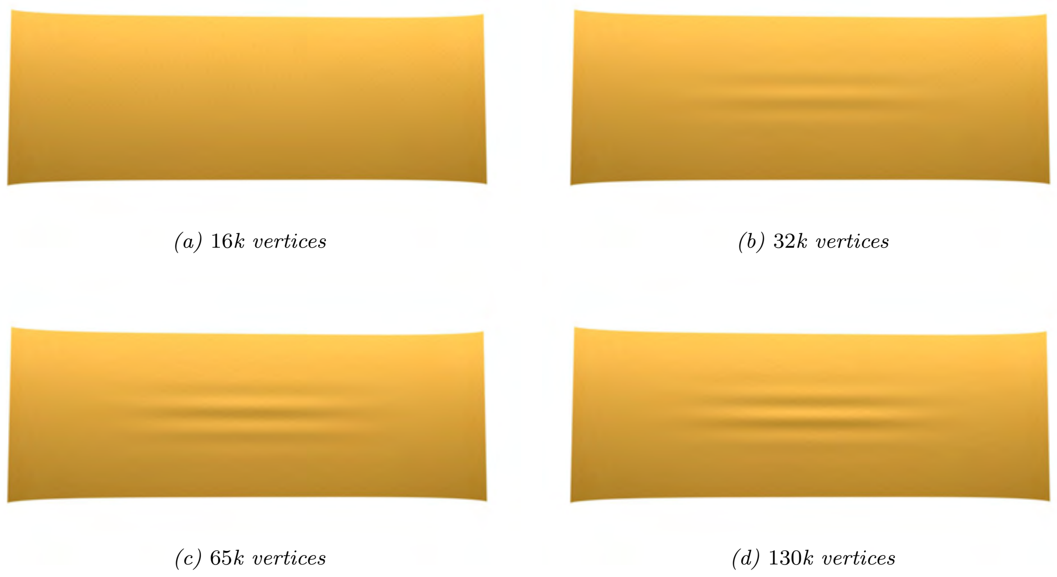
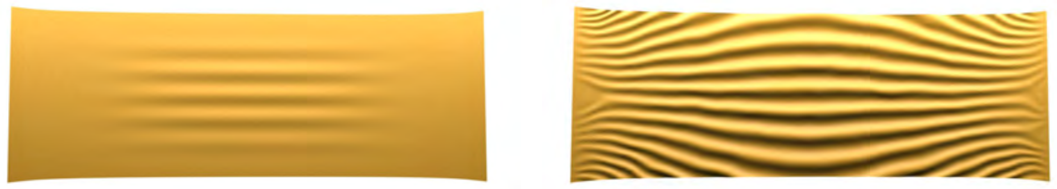

(a) 16*k vertices* *(b)* 32*k vertices* *(c)* 65*k vertices* *(d)* 130*k vertices*

Figure 2.6: Simulation of wrinkles in a highly stretched sheet Wang et al. (2018) using the *StVK* model on a sequence of Delaunay meshes. (a) No waves appear in the mesh with 16k vertices. (b) 3 waves with 0.17mm peak amplitude appear when mesh resolution increases to 32k. (c) A 65k-vertex mesh yields 3 waves but 0.27mm peak amplitude. (d) Mesh with 130k vertices ends up with 3 waves and 0.31mm peak amplitude.

We simulate this model problem with a rectangular sheet of size $0.25m \times 0.1m$, with thickness 0.1mm, Poisson ratio $\nu = 0.5$ and Young's modulus $Y = 1MPa$. *TFW* produces wrinkles with 5 wave periods and a peak amplitude $0.42mm$ on a base mesh of 522 vertices. These values are within 20% of the results determined by Wang et al. (2018) ($0.35mm$ for peak amplitude). In comparison, *ARCSim* with adaptive remeshing enabled, generates a mesh of $13k$ vertices, producing 5 wrinkles and peak amplitude of $0.34mm$. See Figure 2.5 for visualization of these results.

Despite the seeming simplicity of this example set-up, traditional simulation methods struggle to correctly predict the wrinkling behavior here and so it provides a simple "unit-test" for accuracy. *StVK* can produce wrinkles, but success is sensitive to resolution. We test *StVK* on a sequence of Delaunay meshes of the rectangle,



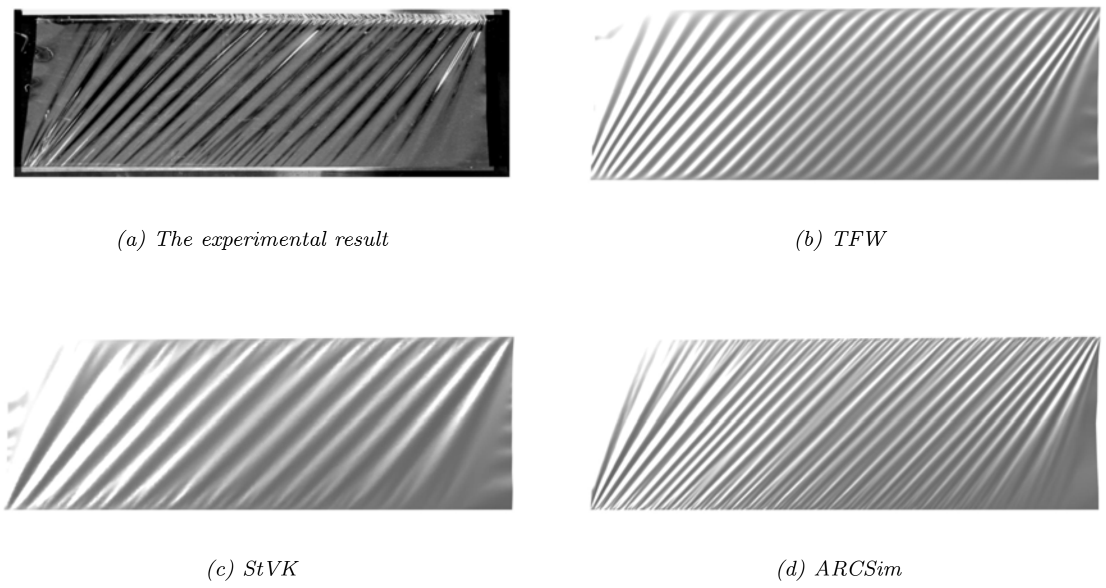
(a) TFW *(b) Vandeparre et al. (2011)*

Figure 2.7: Comparison of (a) our method and (b) wrinkles generated using the scaling law of Vandeparre et al. (2011) for the model problem of a highly stretched sheet Wang et al. (2018). In examples where the wrinkles are not caused by wrinkles propagating inward from a clamped boundary, the analysis of Vandeparre et al. does not apply.

doubling the number of vertices each time. Wrinkles first appear at $32k$ vertices, but the predicted pattern still does not converge to a consistent result even as we extend $StVK$ to a $260k$ vertex mesh. For this highest-resolution mesh $StVK$ generated a solution with 3 wave periods and peak amplitude $0.33mm$ (Figure 2.6 and Figure 2.5(c)). Given that the amplitude of the wrinkles is quite small compared to the dimension of the overall structure, membrane locking Chapelle and Bathe (2011) is likely responsible for artificially stiffening the material, especially on "coarse" meshes.

We also use this model problem to probe the scaling law proposed by Vandeparre et al. (2011), and suggested by Evgeny and Harders (2019) for use in cases where a wrinkling shell is not bonded to a volumetric substrate. As shown in Figure 2.7, although the Vandeparre et al. scaling law is suitable for predicting frequency cascades in wrinkles that propagate inward from clamped boundaries, it is not suitable for predicting more general wrinkling patterns, even in simple cases like the stretched sheet experiment.

Sheared Rectangles Wong and Pellegrino (2006) study the wrinkle profile of a sheared rectangle whose top and bottom boundaries are clamped and sheared in the horizontal direction while its left and right boundaries are left free. We reproduce



(a) The experimental result

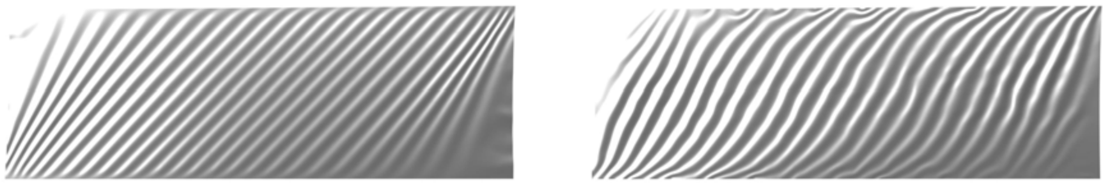
(b) TFW

(c) StVK

(d) ARCSim

Figure 2.8: A thin rectangular sheet sheared in the horizontal direction. (a) The experiment result from Wong and Pellegrino (2006) yields 19 wrinkles, (b) our method (*TFW*) produces 25 wrinkles, (c) *StVK* generates 13 wrinkles, and (d) *ARCSim* gives 26 waves.

their experimental set-up in simulation, where *TFW* generates 25 wrinkles with $1k$ vertices. As in the previous experiments, we search for the minimum-resolution of the simulation for which *ARCSim* and *StVK* does not exhibit significantly degraded wrinkling; under this methodology *ARCSim* gives 26 wrinkles using an $18k$-vertex mesh, and *StVK* produces 13 wrinkles on a $20k$-vertex mesh. Wong and Pellegrino report 19 wrinkles in the actual experiment. Visually, as presented in Figure 2.8, *TFW* shows consistency with the real world example, where both the simulated wavelength and wrinkle direction are well-aligned with the experimental result. Finally, notice from the experimental photograph that our assumption of a single dominant wavelength is valid over most of the wrinkled surface, and that our model recovers these dominant wrinkles; however there is also a thin boundary layer near the clamped edges of the sheet where a superimposed second frequency of wrinkles can be seen. Our model



(a) TFW *(b) Vandeparre et al. (2011)*

Figure 2.9: Comparison of wrinkles generated using (a) our method and (b) the scaling law of Vandeparre et al. (2011) for shearing a rectangular sheet Wong and Pellegrino (2006). Again, in this example, the interior wrinkles are not caused by boundary wrinkles that propagate inward from a compressed, clamped boundary (here the boundary is sheared inextensibly), so the analysis of Vandeparre et al. does not apply. You can see unnatural wrinkles near the top and bottom boundaries in (b).

cannot currently resolve these secondary wrinkles; see Section 2.7 for discussion of how our model might be extended to the case of multiple superimposed wrinkles.

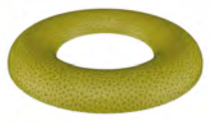
For this problem, we also compute the wrinkles using the scaling law proposed by Vandeparre et al. (2011), which is suggested by Evgeny and Harders (2019) as an extension of their method for shells with no substrate. Similar to the stretched sheet experiment, we observe unnatural behavior near the boundaries (see Figure 2.9).

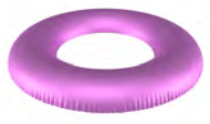
Inflated Structures Inflatable structures exhibit a wide range of complex wrinkling behaviors that have been modeled with direct application of tension-field simulations Skouras et al. (2014). Here we consider *TFW*'s behavior on a range of inflated examples by augmenting our *TFT* model with Skouras et al. (2014) pressure force. In each of the following three examples we inflate a balloon design formed by sewing together two copies (panels) of a planar domain along their boundaries: an annulus, disk, and rectangle.

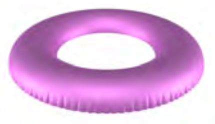
Sewing two annuli (Poisson's ratio $\nu = 0.44$, Young's modulus $Y = 10MPa$) together (inner radius 0.04m and outer radius 0.1m) yields a torus. We observe fine wrinkling around the outer equator of the torus when simulating *TFW* with a $2k$ vertex base mesh. We perform the same experiment with *StVK*, observing that wrinkles are not fully resolved until we reach a mesh of $40k$ vertices (Figure 2.10).

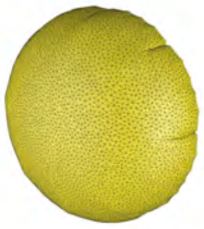
Two disks (radius 0.1m) sewn along their boundary yields the classic Mylar balloon, with wrinkles around the equator. Here we use the same material parameters as in the previous example. Interestingly, for this example, inspecting a real-world balloon reveals features at two scales: a small number of coarse *creases* appear equally spaced around the equator, with fine *wrinkles* in between. Both features can also be seen in the *StVK* simulation of the Mylar balloon, and in previous simulations of this problem using adaptive subdivision finite elements Vetter et al. (2014). Here we find that *StVK* requires a simulation mesh of at least $40k$ vertices to resolve these features. *TFW* also produces a result with both scales of features, despite its wrinkling model assuming only a single wrinkle frequency (Figure 2.10). This behavior is surprising at first until we observe that the creases already begin in the *TFT* base mesh. The *TFW* wrinkle model then appropriately augments these creases with the fine wrinkles. We are currently uncertain when and why these sharp creases appear in the base mesh, but do verify that this is not an artifact due to the choice of our bending model: these sharp creases appear even if there is zero bending energy (Figure 2.11). We do observe that resolving the sharp creases requires a relatively high base mesh resolution relative to correctly predicting the compression field and fine wrinkling; in this case we get sharp creases if our base mesh has at least $5k$ vertices. There has been relatively little work in the physics literature studying *TFT* in this regime; better understanding of creasing behavior in *TFT* base meshes and likewise its corresponding implications for *TFW* wrinkling remains promising future work.

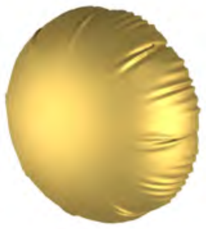
Two rectangular patches (width 1.4m, height 0.74m), with thickness 1mm, Poisson's ratio 0.3 and Young's modulus $0.1MPa$, when inflated yield the classic "teabag" shape. Understanding the coarse shape and wrinkling of this teabag is a classical problem in mathematics Paulsen (1994); Pak and Schlenker (2010). *TFW* with a $1k$-vertex mesh predicts fine wrinkles around the boundary of the teabag,

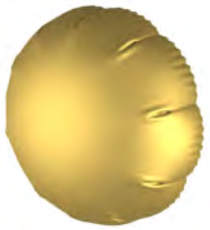

(a) TFT base mesh


(b) TFW


(c) StVK


(c) TFT base mesh


(d) TFW


(e) StVK

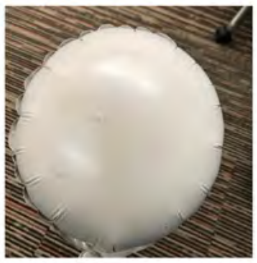

(f) Real balloon

Figure 2.10: The simulated results of inflated annulus and disk experiments. *Top*: from left to right, a $2k$ base mesh, the *TFW* solution, and the *StVK* solution, on a mesh with $40k$ vertices. *Middle left:* the simulated *TFT* base mesh for the disk balloon, with $5k$ vertices. *Middle right:* the wrinkled mesh produced by *TFW*. *Bottom left:* simulated result on $40k$-vertex mesh. *Bottom right:* the real-world balloon.

but here we do not additionally recover the coarse creases evident in a $30k$ *StVK* simulation (Figure 2.12). Again, as in the "Mylar balloon" discussion above, this behavior is closely related to how well the base mesh resolves creasing. Here we only start to see creases in the base *TFT* mesh at finer resolutions ($\approx 30k$ vertices).

Lastly, we simulate wrinkles on the Teddy bear example from Skouras et al. (2014). This bear has more complex inflated geometry—it is sewn from 17 planar patches. *TFW* generates wrinkles in a $2k$-vertex base mesh that are qualitatively similar to those that appear in a *StVK* simulation starting at around $98k$ vertices (Figure 2.13).

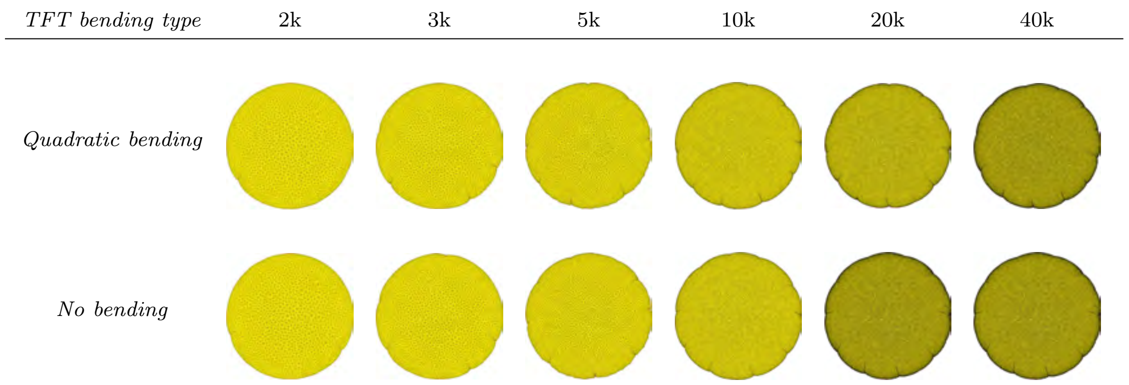
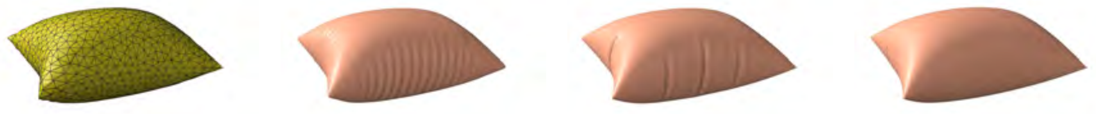


Figure 2.11: The simulated tension field result of the inflated disk experiment. From left to right the resolutions are 2k, 3k, 5k, 10k, 20k, 40k. *Top row*: simulated *TFT* model with quadratic bending. Numbers of sharp creases are 1, 4, 8, 10, 11, 12. *Bottom row*: the simulated *TFT* model without bending term; the corresponding wrinkle numbers are 2, 5, 8, 10, 10, 10.

2.6.5 Performance and Numerical Convergence

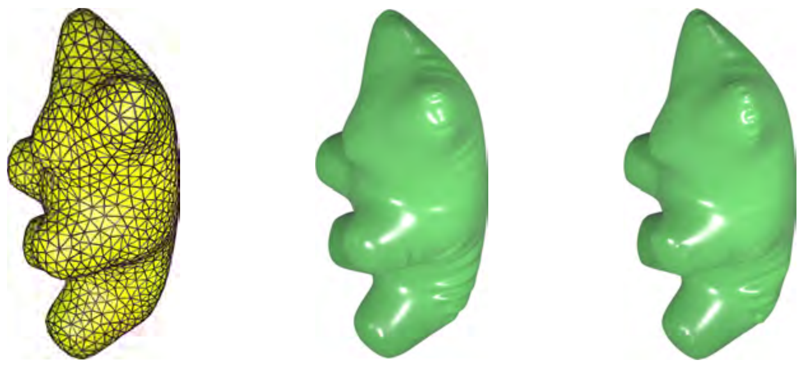
In this work we have focused on accurately capturing the wrinkling features of thin shell elastica with as few computational degrees of freedom as possible. In so doing we have derived the *TFW* model which successfully obtains fine wrinkling behavior over a wide range of examples, with generally an order-of-magnitude less degrees-of-freedom than standard shell methods; our method generally requires an order of magnitude less computation time as well. We summarize the resolution and timing of our experiments in Table 2.1. We ran our experiments on a desktop with a 8-core Intel Core i9-9900K CPU, clocked at 3.6 GHz and 128 GB of memory.

Resolution Comparisons We compare *TFW*, *StVK*, and *ARCSim* in terms of the lowest-resolution mesh required in order to resolve the wrinkles without significant degradation. For *TFW* and *StVK*, we begin with a lowest-resolution Delaunay mesh, and solve for the static wrinkled shape; we double the resolution of the mesh and repeat the experiment until the wrinkle pattern converges to a consistent shape, or



(a) TFT base mesh *(b) TFW* *(c) StVK* *(d) MD*

Figure 2.12: The simulated result of inflated teabags: (a) the *TFT* base mesh with 936 vertices; (b) shape recovered using *TFW*; (c) *StVK* simulation on a 30k-vertex mesh, and (d) *Marvelous Designer* simulation with 30k vertices. Among all these results, *StVK* achieves the most natural shape (sharp creases + small wrinkles). *TFW* will need a much higher resolution to capture these sharp creases in the *TFT* base mesh.



(a) TFT base mesh *(b) TFW* *(c) StVK*

Figure 2.13: Inflated Teddy bear: (a) the 2k-vertex *TFT* base mesh, (b) corresponding *TFW* result, and (c) simulated result with *StVK* (98k vertices). We can see *TFW* predicts a similar wrinkle patterns as *StVK*, but requires 50 times fewer vertices.

until the simulation takes more than a week to terminate (in which case we halt the experiment). The resolution in the table is the lowest at which the result is the consistent wrinkled shape. We indicate with a "−" superscript the simulations where the coarsest-resolution simulation is already consistent (so that potentially even lower-resolution consistent simulations are possible) and with a "+" superscript those simulations that became too computationally expensive before yielding consistent results.

For *ARCSim*, we enable adaptive remeshing and list the final mesh resolution in Table 2.1. *ARCSim* requires specification of additional parameters, including most notably a minimum triangle size, which significantly affects the quality of the solved static shape. We set frame time to $0.04s$, frame steps to 8, and end time to $40s$; for the minimum triangle size, we try $0.01m$, then $0.005m$ and finally $0.001m$, accepting the first result in which the wrinkles are not aliased. Note that we do not report *ARCSim* results for the inflated structures, since *ARCSim* does not implement a pressure force. For the other examples, we disable the *ARCSim* "popfilter" module, as well as the "collision" module for examples without collisions.

Timing Comparisons Computation of the *TFW* base mesh is now reduced via our choice of *TFT* and quadratic bending to an entirely convex problem on a coarse mesh. Similarly, computation of the wrinkle field is then likewise a small, sparse optimization of our discretized wrinkle energy subject to sparse curl-constraints and simple bound constraints on the amplitude degrees of freedom. In our experiment, the reduction in mesh resolution directly translates into reduction in computational cost, when comparing *TFW* to traditional solvers. That said, apples-to-apples comparisons are not straightforward: unlike for the resolution experiments above, timing comparisons depend significantly on the low-level implementation details of both our method and the baselines. Moreover, it is not clear how to determine a termination condition that is consistent across all methods: gradients with respect to vertex positions, and with respect to wrinkle field variables a and ω, have different units and cannot be compared.

Despite the above difficulties, we provide some timing data in Table 2.1. For each experiment that does not involve collision response[4], we run *TFW* and *StVK* for 1000 SQP iterations, or until the residual infinity norm is below 10^{-6}, whichever comes first. For both methods, 1000 iterations is far more than necessary to reach a visually-stable static shape. We visually inspect the intermediate configurations and

[4]We do not perform timing comparisons for the examples involving contact, since our (very naive) contact solver is very slow and dominates the cost of both the *TFW* and *StVK* optimizations.

select the earliest iteration where the solution matches the visually-stable shape; the wall-clock time of this iteration is listed as the *stable time* in Table 2.1. We use the *TFT* base mesh as the initial guess for both *TFW* and *StVK* (and the timings in the table do not include this preprocessing step); the time needed to compute the initial guess is negligible compared to the subsequent solve times. See Appendix A.5.1 for timing numbers.

Although inherently subjective, this methodology allows us to give some sense of the relative performance of *TFW* versus baselines, and we observe that the manually-selected visually-stable frame matches a "dogleg" in each example's stationarity residual plots. See Appendices A.5.2 and A.5.3 for additional data and residual plots for all examples listed in the table. We observe that *TFW* offers a speedup ranging from 1.28x (for the torus) to 345.1x (for the stretched sheet experiment) compared to *StVK*. For all the experiments, *TFW* succeeds in reaching a visually-stable wrinkled shape within one minute.

Since commerical shell solvers like ABAQUS emphasize accuracy and generality over performance, and since we are not aware of any established computer graphics software for shell statics, we used our own implementation of Morley shell elements (based on the implementation hints provided by Grinspun et al. (2003) and Weischedel (2012); see Appendix A.1) as the *StVK* baseline. We made best-effort optimizations to improve performance of both the *TFW* and *StVK* algorithms; in both cases the majority of the time spent each iteration is in the third-party solver (NASOQ Cheshmi et al. (2020)) in the case of *TFW*, and SuiteSparse's parallel implementation of supernodal sparse Cholesky decomposition Chen et al. (2008) in the case of *StVK*). See Appendix A.5.1 for more information including data about the timing breakdown within each optimization iteration.

Finally, in Figure 2.22, we visualize the progress of the *TFW* and *StVK* solvers on a wall-clock time axis, to give an equal-effort comparison of the two methods. On the timeline (which is in log scale) we indicate the range of times during which each simulation has not yet reached a visually-stable state in red. The visually-stable time (as listed in Table 2.1) corresponds to the transition on the timeline from red to blue. In cases where we terminate a simulation early, due to having reached very low stationarity residual (10^{-6}), the termination time corresponds to the transition from blue to green. We show representative stills of each simulation; each still was sampled at the wall clock time of its left edge. Please see Appendix A.5.3 for raw data and discussion of the blue-to-green transition, and the result videos showing a, ω (for *TFW*) and vertex displacement (for *StVK*) versus wall clock time.

Models	*StVK*			*ARCSim*	*TFW*		
	#verts	#iter	stable time (s)	#verts	#verts	#iter	stable time (s)
sphere drape	—	—	—	$32k$	$1.6k^{-}$	30	11.08
symmetric dress	$40k$	—	—	$8k$	$1.4k$	20	8.18
asymmetric dress	$52k^{-}$	—	—	$13k$	$3.4k$	30	23.19
pants	$60k^{-}$	—	—	$15.8k$	$4k$	25	38.94
stretched sheet	$260k^{+}$	13	886.80	$11k$	522^{-}	13	2.57
sheared rectangle	$20k^{+}$	220	673.11	$13k$	$1k^{-}$	50	9.63
torus	$40k$	7	57.29	—	$2k$	55	44.81
balloon	$40k$	10	76.52	—	$5k$	30	52.33
teabag	$30k$	24	148.37	—	936^{-}	13	1.98
teddy	$98k$	10	251.91	—	$2k^{-}$	75	26.29
twisted cylinder	$96k^{+}$	120	1528.68	$68k$	688	400	50.94

Table 2.1: Timing and resolution information for the examples we show in this chapter. Reported mesh resolutions are generally the coarsest-possible that do not exhibit degradation of the wrinkle shape; see main text for details of the methodology. A superscript $-$ means that the lowest-resolution mesh we tried produced good results (so that the true minimum-required resolution might be lower); a superscript $+$ indicates that the highest-resolution mesh we tried (listed in the table) failed to produce acceptable results after one week of simulation; we did not continue to probe higher resolutions for these examples. We did not simulate the inflatable structures using *ARCSim* as the simulator does not include a pressure model. For examples that do not involve contact, we also report timing information for *TFW* and *StVK*. We list the iteration number and wall-clock time for when each simulation converges to a visually-stable result (see main text for methodology).

2.7 Conclusion, Limitations, and Future Work

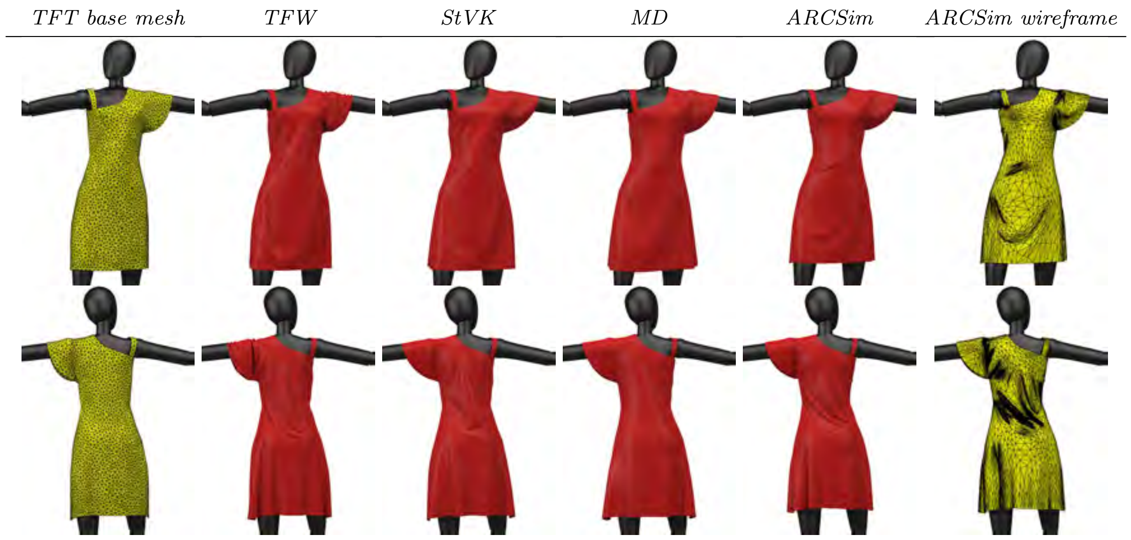


Figure 2.14: An asymmetric dress draped over a mannequin. The first row shows the front view of the dress after simulation and the second row shows the back view of the same dress. From left to right: the 3.4k-vertex base mesh, simulated using *TFT*; the simulated wrinkled shell using *TFW* on that base mesh; the result from *StVK* on a Delaunay mesh with 50k vertices; the result from *MD* on a 50k-vertex mesh; the result from *ARCSim*, with adaptive remeshing enabled; a wireframe rendering of that same *ARCSim* result (the mesh has 13k vertices). For *StVK* and *MD*, we tuned mesh resolution to be about as coarse as possible without causing significant degradation in the wrinkle pattern.

We have introduced *TFW*, a new model and algorithm for high-fidelity modeling of wrinkling thin shells suitable for low-resolution computational meshes. The key insight is that by decoupling the wrinkled shape into slow and fast variables, and deriving an approximate elastic energy in terms of the slow amplitude and frequency variables, high-resolution, physically-principled detail can be added to a wrinkle-free coarse mesh without aliasing. We have demonstrated *TFW*'s ability to generate realistic and often predictive results across a wide range of challenging examples with an order-of-magnitude less degree of freedoms and speedups of between 1.28x and 345.1x compared to traditional finite elements.

Although our solver implementation in Section 2.5 serves as proof of concept for applying wrinkle fields as a new approach to static simulation of fine wrinkling,

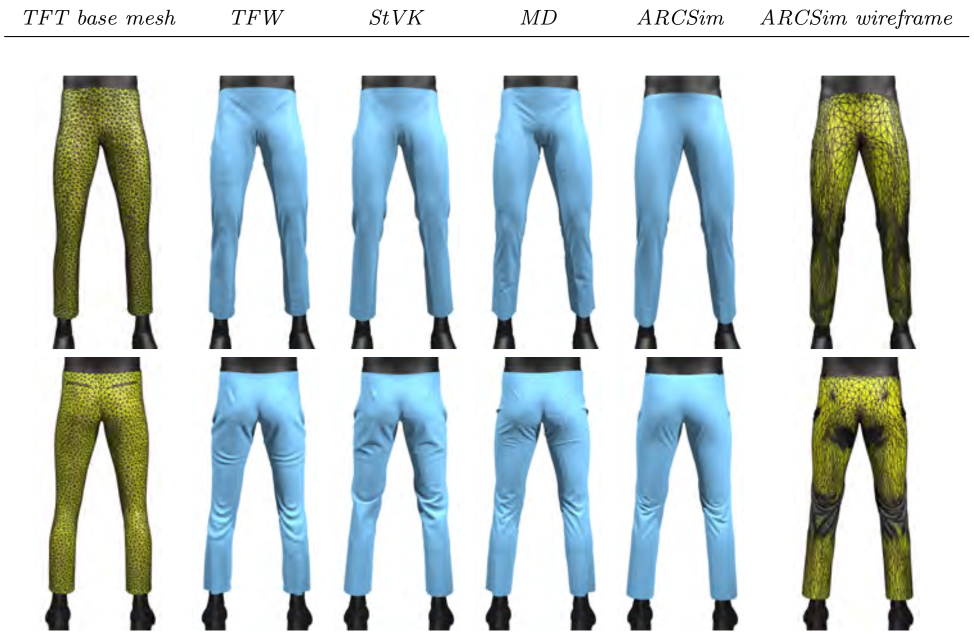


Figure 2.15: A mannequin wearing simulated pants. The first row shows the front view of the pants after simulation and the second row shows the back view of the same pants. From left to right: a 4k-vertex base mesh, simulated using *TFT*; the simulated wrinkled shell using *TFW* on that base mesh; the result from *StVK* on a Delaunay mesh with 60k vertices; the result from *MD* on a 60k-vertex mesh; the result from *ARCSim*, with adaptive remeshing enabled; a wireframe rendering of that same *ARCSim* result (the mesh has 15k vertices). For *StVK* and *MD*, we tuned mesh resolution to be about as coarse as possible without causing significant degradation in the wrinkle pattern.

many avenues of future work remain before a wrinkle-field approach is industry-ready as a practical replacement for the current, triangle-element-based approach:

Performance. This chapter largely focused on foundational theory, rather than low-level performance optimization (nevertheless, in the experiments of Section 2.6.5 we were able to achieve significantly faster performance with *TFW* compared to baseline *StVK*, due to the vastly coarser mesh needed by *TFW*). To compete with commercial GPU-based cloth solvers, *TFW* would need additional performance improvements. The performance bottleneck for *TFW* (see Appendix A.5.1 for a breakdown)

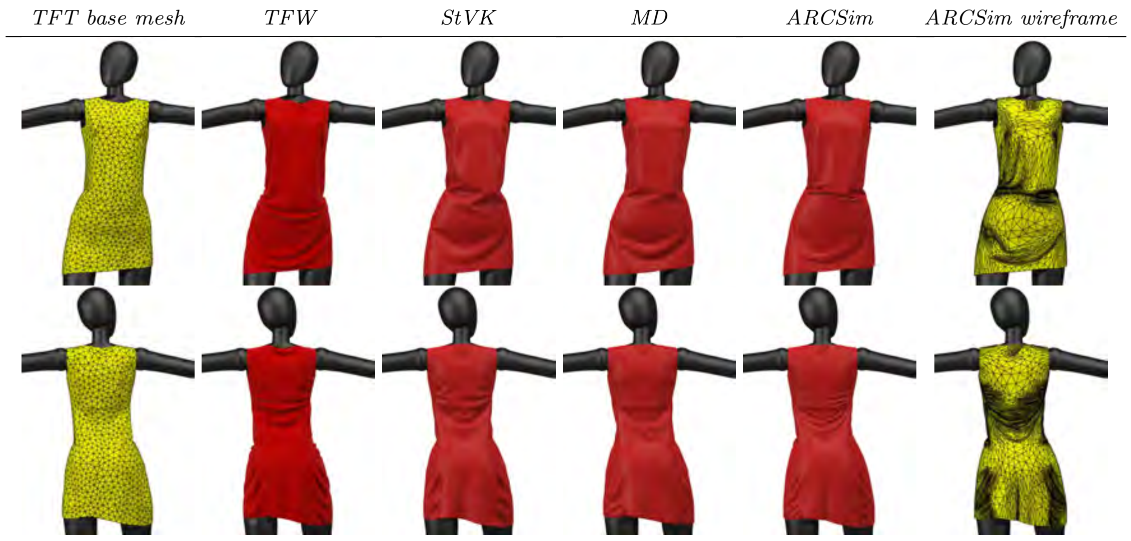


Figure 2.16: A symmetric dress draped over a mannequin. The first row shows the front view of the dress after simulation and the second row shows the back view of the same dress. From left to right: a 1.4k-vertex base mesh, simulated using *TFT*; the simulated wrinkled shell using *TFW* on that base mesh; the result from *StVK* on a Delaunay mesh with 40k vertices; the result from *MD* on a 40k-vertex mesh; the result from *ARCSim*, with adaptive remeshing enabled; a wireframe rendering of that same *ARCSim* result (the mesh has 7.8k vertices). For *StVK* and *MD*, we tuned mesh resolution to be about as coarse as possible without causing significant degradation in the wrinkle pattern.

is the SQP solve for amplitude and frequency (Section 2.5.2); at a mininum, a high-performance implementation of *TFW* would need to use a parallelized QP solver on the GPU instead of the CPU-based NASOQ library. Other potential optimizations include starting from a better initial guess for amplitude and phase (based on additional analysis of the physics of wrinkling, or provided by a data-driven approach), simplifying the discretization of the reduced-order elastic energy (for example, by replacing three-point quadrature of the integrals in Equation (2.30) with simpler expression derived using Discrete Exterior Calculus), or replacing our SQP strategy for minimizing elastic energy with a different optimization strategy.

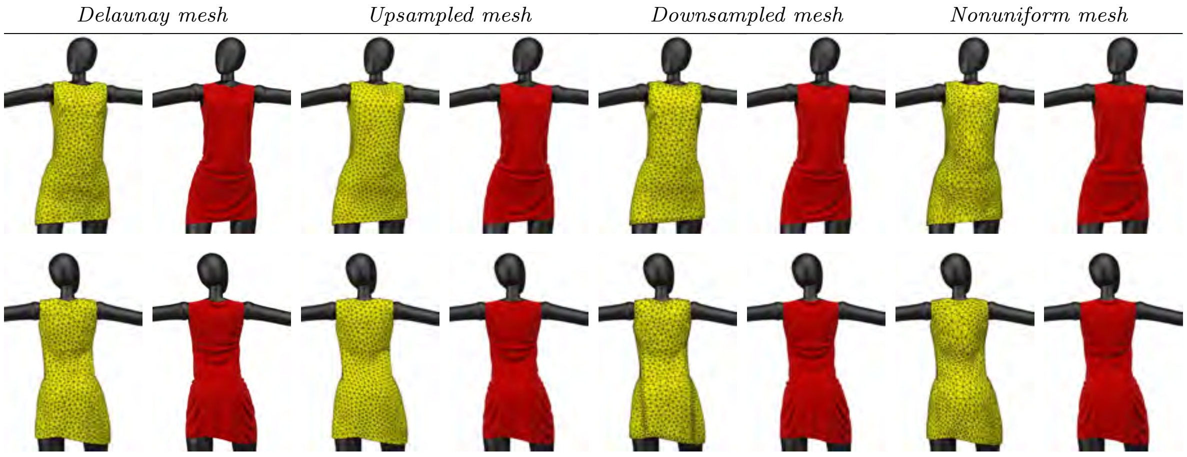


Figure 2.17: A symmetric dress draping over a mannequin in terms of different triangulation, but similar mesh resolution (1.7k $\pm$ 0.3k vertices). The first row shows the front view of the dress after simulation and the second row shows the back view of the the same dress. From left to right: *TFT* simulated result of a 1406-vertex Delaunay mesh. The corresponding wrinkled shape using our wrinkled-field model on that base mesh; *TFT* simulated result of a mesh with 1399 vertices. This mesh is generated by Loop upsampling a lower resolution Delaunay mesh. The corresponding wrinkled shape using our wrinkled-field model on that base mesh; *TFT* simulated results of a mesh with 1959 vertices, downsampled from a higher resolution Delaunay mesh (keeping the stitching boundary unchanged). The corresponding wrinkled shape using our wrinkled-field model on that base mesh; *TFT* results of the mesh extracted after one-step arcsim remeshing process with 1981 vertices. The corresponding wrinkled shape using our wrinkled-field model on that base mesh. Our wrinkled-field model yields a consistent final result with respected to these different triangulation.

Two-way Base Surface-Wrinkle Field Coupling. We derived an approximated elastic energy E^{wf} (Equation (2.27)) that involves both a base surface $\boldsymbol{r}_b$ and wrinkle field variables a, ω. In this chapter, we solve first for the base surface using tension field theory (*TFT*), and then fix this surface and optimize independently for the wrinkle variables. A natural question is whether the *TFT* base mesh is truly optimal in terms of minimizing E^{wf}. As we can see from Fig. 2.19, in some cases it is clear that the *TFT* solution does not "puff out" enough: the wrinkling of the draped cloth corregated the surface, which in turn penalizes bending of the cloth perpendicular to the wrinkles, near where the cloth breaks contact with the sphere, so that the

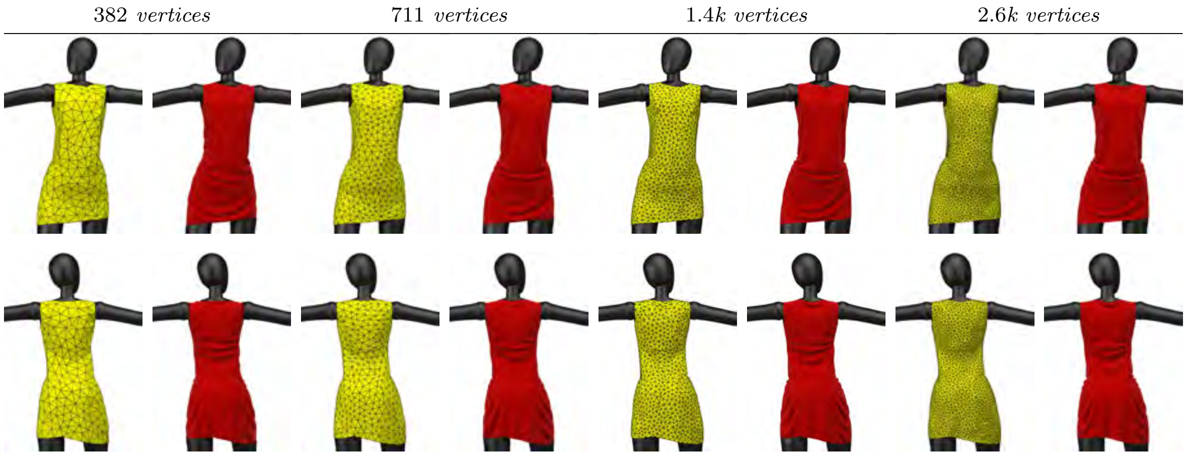


Figure 2.18: Simulations of a draped dress with different mesh resolutions using our *TFW* model. The wireframe figures to the left of each image pair show the base mesh; the red meshes are the final results. Notice that *TFW* yields a consistent wrinkle pattern when the resolution reaches around 1.4k vertices.

draped cloth has a more conical than cylindrical coarse shape. As discussed earlier, jointly optimizing E^{wf} for both $\boldsymbol{r}_b$ and a, ω does not work, since E^{wf} assumes that the base surface strain and curvature are slow variables, which is no longer necessarily true if $\boldsymbol{r}_b$ is allowed to vary arbitrarily. One idea might be to parameterize $\boldsymbol{r}_b$ using differential coordinates, a low-resolution subdivision surface, or some other space of deformations that ensures the base surface strain and curvatures stay slow. Another potential approach to two-way coupling would be to alternate solving for $\boldsymbol{r}_b$ and the wrinkle field, where an "effective" rest curvature is computed for $\boldsymbol{r}_b$ based on the current wrinkle field at each iteration.

Other Base Surface Limitations. Even when using *TFT* to compute the base surface, sometimes the base surface mesh can have large variations in strain, or defects such as inverted or collapsed triangles, which can cause *TFW* to struggle. For example, the relatively slow performance of *TFW* on the inflated torus example is due to high noise in the amount of compression in the corresponding base mesh. The root cause of this noise is that *TFT* has multiple possible solutions in regions of compression. For example, if a square piece of cloth is compressed in the horizontal direction, both the deformation where the entire cloth has equal compression strain (desirable), and the deformation where only a thin vertical column of the cloth compresses while the rest of the cloth translates isometrically (undesirable), are minimizers of the *TFT* energy. The quality of the *TFT* base surface might be improved by adding some regularization terms to the *TFT* energy, or by incorporating two-way coupling of the base surface with the wrinkle patterns, as discussed above.

Phase Ambiguity. Our approach computes wrinkle amplitude and frequency, and then solves for phase ϕ as a post-process, with $d\phi = \omega$. This recovery procedure can determine phase only up to a global phase shift, $\phi \to \phi + k$; this shift cannot be determined from the wrinkle field. For statics problems the shift is unimportant, as it affects the precise wrinkled geometry, but not coarse-scale features of the wrinkled surface such as wrinkle orientation, amplitude, and frequency. The phase shift ambiguity does mean that the wrinkled surface $\boldsymbol{r}_w$ cannot be used to measure convergence of the wrinkle field optimizations, in experiments such as we did in Section 2.6.5. Any use of wrinkle fields for dynamics would also need to account for this phase ambiguity across time, as otherwise, sudden changes in the global phase shift would be perceived as "popping" in animations.

Dynamics. This chapter considers only shell statics. Extending *TFW* to dynamics is a natural follow-up direction; the simplest approach would be to animate the base mesh and then add wrinkles quasi-statically. The main new feature needed for such quasi-static simulation is a method for solving for a temporally-coherent global phase shift in ϕ at each time step (see previous point) so that wrinkles appear to change smoothly over time. More interesting, and more challenging, would be to equip the wrinkles themselves with inertia, and implement two-way coupling of the wrinkles and base mesh (see above).

Collisions. We currently consider only collisions of the *TFT* base mesh against the environment. There are many interesting directions for future work improving collision handling of wrinkled surfaces, such as taking into account collisions when solving for the wrinkle field (so that wrinkles in contact with other objects have flattened shape), detecting and resolving self-contact of the wrinkled surface (where $\boldsymbol{r}_w$ self-collides but $\boldsymbol{r}_b$ does not), anisotropic friction models that account for the wrinkling direction and amplitude when a wrinkled surface slides against another object, etc. Most of these future directions will first require research into handling two-way coupling of the base surface and the wrinkle field (see above).

More General Constitutive Models. Real-world textiles are woven or knitted, and obey macroscopic constitutive laws that are far more complex than the *StVK* isotropic material we assume in our derivations. Theoretically, there is no obstruction to extending our derivation in Section 2.3 to other constitutive laws. For orthotropic or anisotropic linear materials, the St. Venant-Kirchhoff material norm $\|\cdot\|_{\mathrm{SV}}$ would need to be replaced by a different quadratic norm, new expressions for the in-plane correction terms (Equation (2.16); see also Appendix A.3 and Equation (A.6)) would need to be derived for the new norm, and the in-plane energy term (Equation (2.22)) updated accordingly. In our derivation we exploited symmetries of the STVK material norm to simplify these calculations, and for other constitutive models, the corresponding equations may be less pleasant. Nonlinear materials would require more extensive rederivation, while still following the roadmap we sketch in Section 2.3. However, we suspect general nonlinear constitutive models would produce results that differ little from those of their linearizations, since *all* hyperelastic material models reduce to an (isotropic or anisotropic) linear material in the small-strain limit, and note that in regions of wrinkling, the strain in the compression direction is typically small since most of it has been relieved by buckling.

More General Wrinkling Model. Several of our modeling decisions in Section 2.3 might be revisited and extended in future work: for instance, we assume a single predominant wrinkling frequency at each point on the shell, and while this assumption appears to hold in physical experiments, we do know that near boundary layers, real-world shells often exhibit a second, finer frequency of wrinkling (see the sheared rectangle experiment in Figure 2.8 for instance). One could also explore more general expressions for the wrinkling waveform, for instance by adding additional frequencies to f_t, or changing the normal displacement from a cosine wave to other shapes such as sinc waves Evgeny and Harders (2019).

Injecting Noise and Asymmetry. In some examples, such as the sphere drape (Figure 2.19) and the inflated structures, the *TFW* wrinkles can look "too symmetric" compared to wrinkles in real materials, where imperfections and defects break symmetry. Real wrinkles often also collapse under gravity and fold on themselves (noticeable for the sphere drape, in particular), phenomena we do not attempt to model in this chapter. In future work, post-processing could be done on the wrinkle field to emulate these effects, while maintaining the correct wrinkle frequency and amplitude.

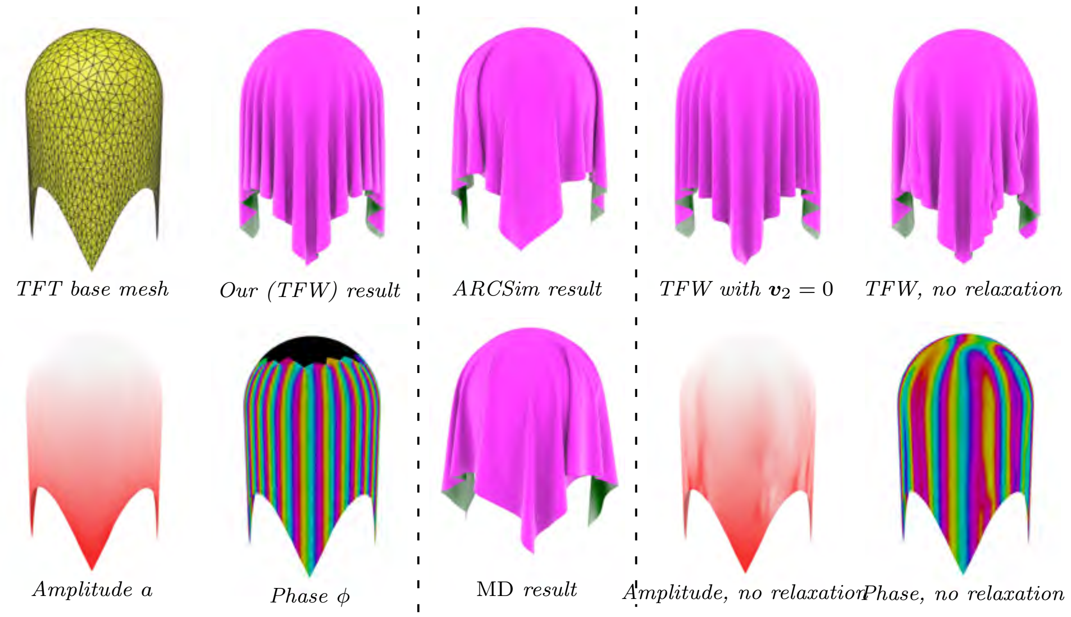


Figure 2.19: Draping a square piece of cloth on a sphere. *Left*: the base surface $\boldsymbol{r}_b$, after optimization using *TFT* and Quadratic Bending, on a domain with $1.6k$ vertices, the surface after we optimize and visualize a wrinkle field on this base surface, and a visualization of the computed amplitude and phase fields a, ϕ over $\boldsymbol{r}_b$. Black regions are where wrinkles don't exist and the phase is undefined. *Middle*: comparison to simulation results from two standard cloth solvers, *ARCSim* and *MD* (Marvelous Designer). To capture all detailed wrinkles, a high resolution simulation mesh was needed (here we use $72k$ vertices). *Right*: Some ablation experiments on this example. Here we show the unnatural undulations produced when visualizing the same amplitude and phase fields as in the bottom-left subfigures, but applying only normal displacement ($\boldsymbol{v}_2 = 0$). We also show the disastrous consequences of failing to relax the integrability constraints on ω on pure-tension faces. The recovered amplitude a and phase ϕ are not the expected smooth, periodic solution since without relaxing integrability near the north pole of the sphere, it is mathematically impossible for an ω to circulates around the circumference of the draped portion of the cloth.

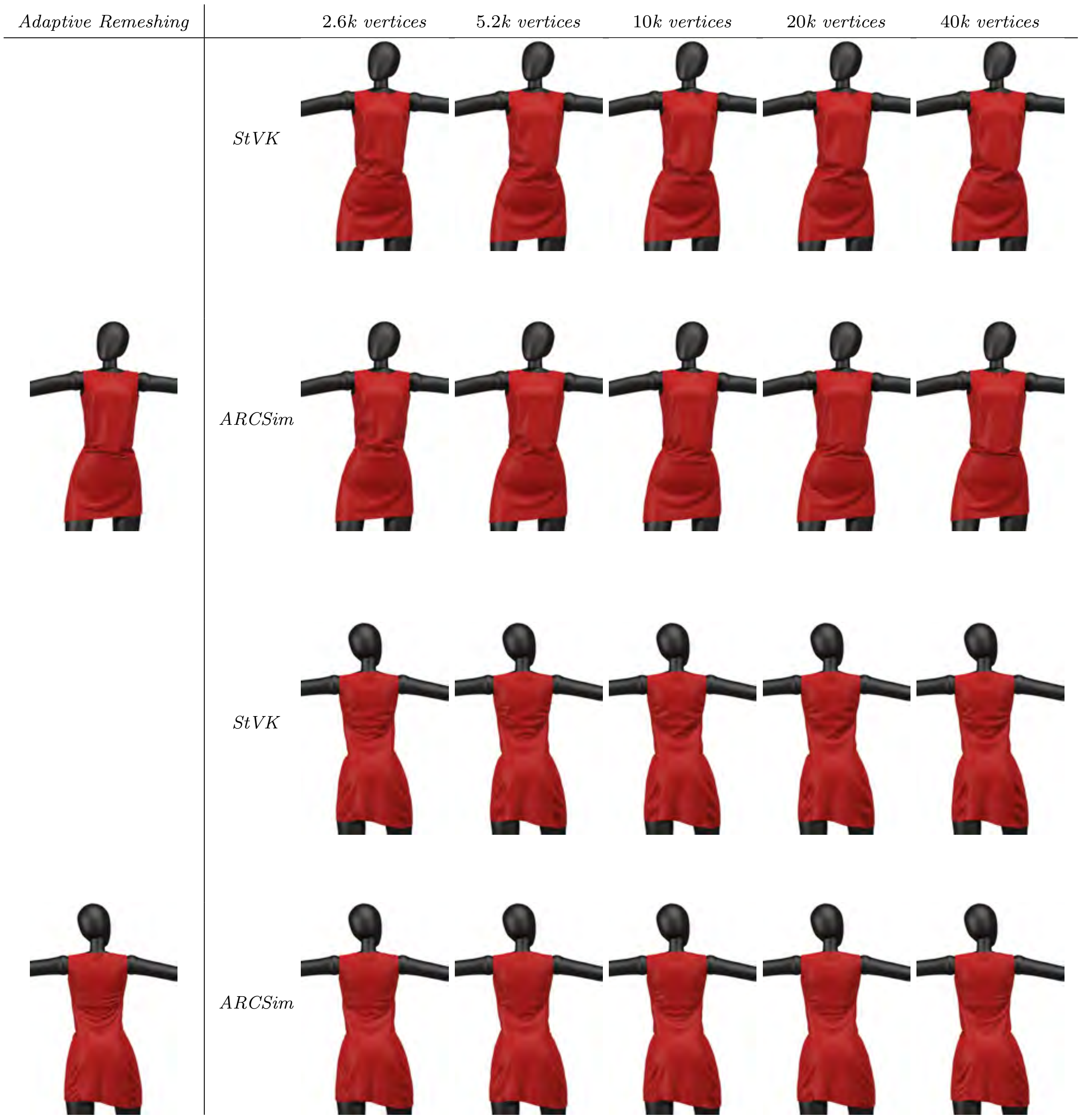


Figure 2.20: We simulate the draping of a dress with different mesh resolutions, using *StVK* and *ARCSim* (with (*left column*) and without (*second and fourth row*) remeshing). The heading of each column indicates the mesh resolution of that column. The simulations appear to converge at about 20–40k vertices for each solver.

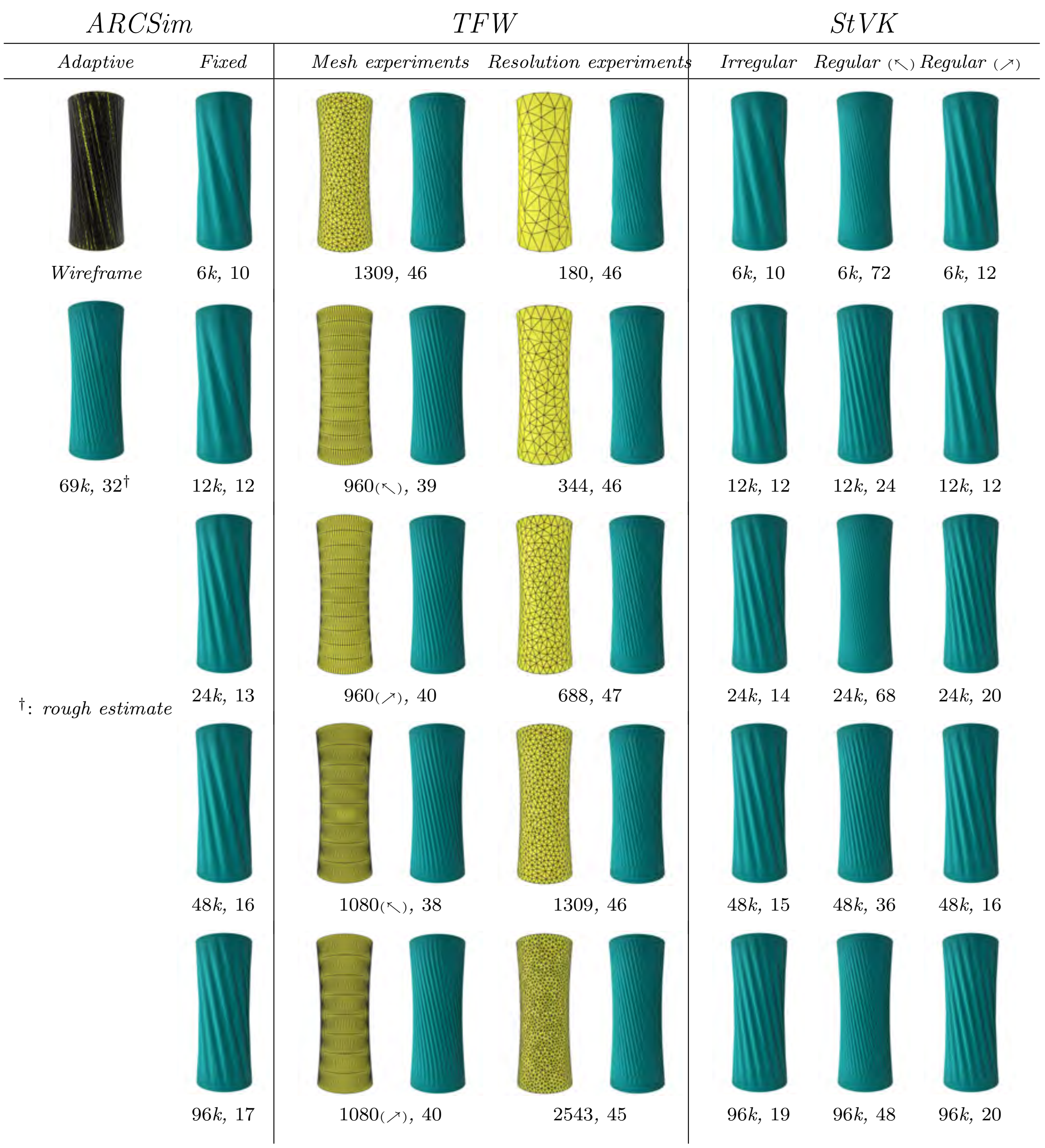


Figure 2.21: Our results for the twisted cylinder experiments. Numbers below figures are (resolution, # wrinkles); for the adaptive *ARCSim* simulation, wrinkle count is a rough estimate as the irregularity of the wrinkles precludes an exact count. The symbols (↖) and (↗) indicate use of a regular mesh with edges aligned along and against the wrinkles, respectively.

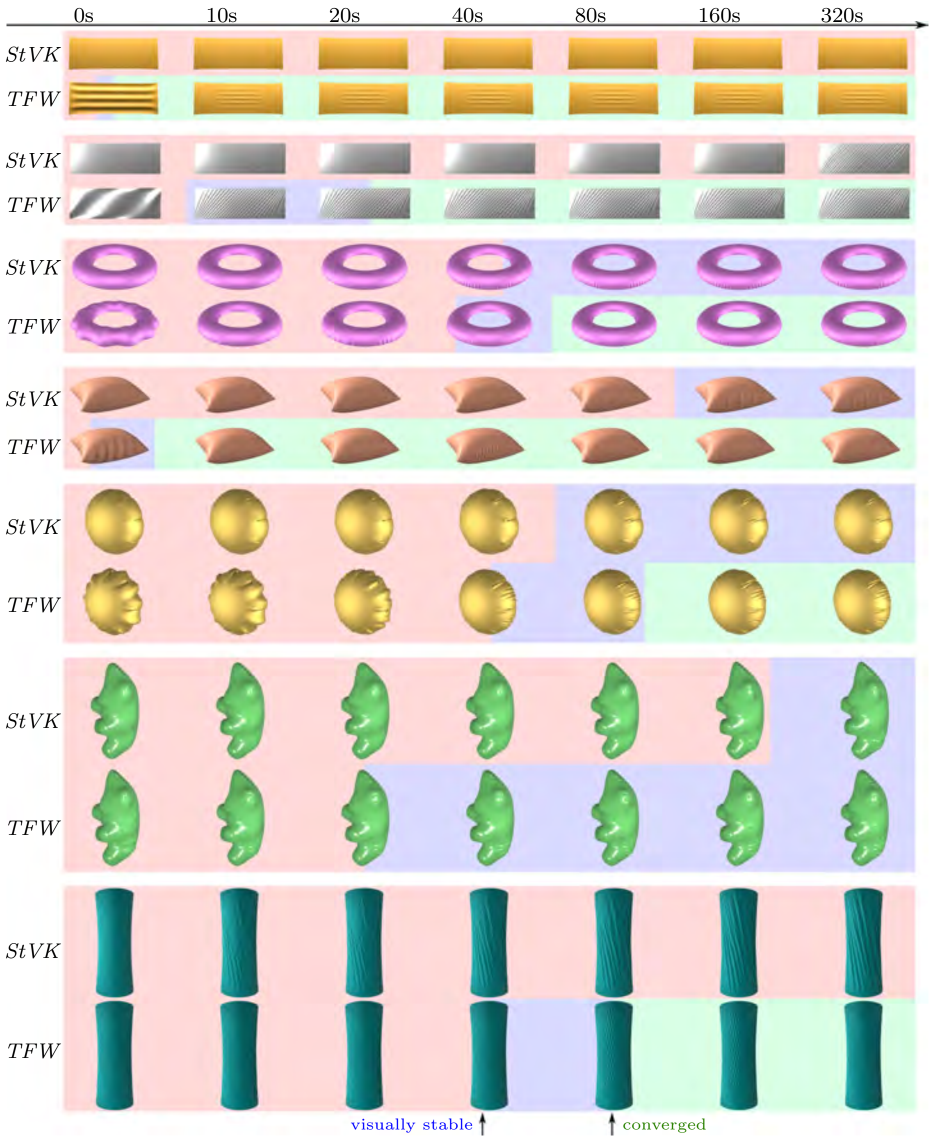


Figure 2.22: Comparisons of our *TFW* algorithm and the baseline *StVK* simulations against wall clock time, on a log scale. Each rendered result is sampled at time corresponding to the image's left edge. Background color indicates current state of each simulation: light red indicates the simulation has not yet reached a visually-stable state, light blue indicates the simulation has become visually-stable, and light green means the solver has terminated due to having small stationarity residual. The boundary between colors correspond to the transitions between these three states, on the wall-clock-time axis. Note that the transitions between red and blue are exactly the times listed in Table 2.1.

Chapter 3: Complex Wrinkle Field Evolution[1]

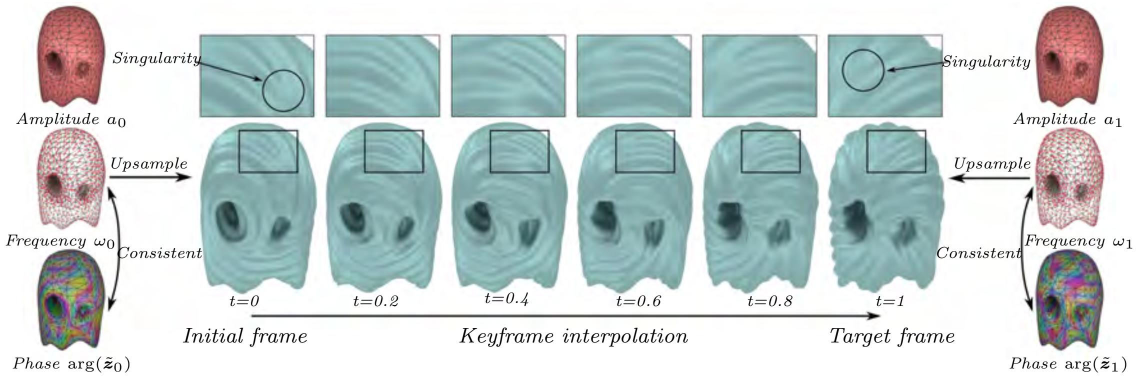


Figure 3.1: We propose Complex Wrinkle Fields (*CWF*s), a new discrete wrinkle model that enables the resolution of highly detailed wrinkle patterns on coarse base-mesh geometry. The *CWF* representation consists of a positive number a per vertex encoding the wrinkle amplitude, a one-form ω per edge to model wrinkle frequency, and a complex number $\tilde{z}$ per vertex to represent wrinkle phase, coupled via a weak variational consistency condition ensuring that $\tilde{z}$ can capture singularities while also being as compatible with ω as possible (Section 3.3.1). We equip the *CWF* representation with a novel temporal interpolation algorithm (Section 3.4) and a spatial up-sampling method (Section 3.5) that together allow for smooth interpolation between wrinkle patterns represented on surfaces by *CWF*s (leftmost and rightmost column), and base-mesh-independent rendering of arbitrarily high-resolution wrinkle patterns. Together these contributions make it possible to smoothly evolve wrinkle patterns between two prescribed keyframes (middle columns) with automatic merging, splitting, and reconnection of wrinkles as necessary via smooth sliding of singularities across the surface (zoomed-in figures in middle columns). Please check the **Interpolation Results** extra video (00:38–00:53; note this is an additional video separate from the main supplemental video) for the corresponding wrinkle animation.

We propose a new approach for representing wrinkles, designed to capture complex and detailed wrinkle behavior on coarse triangle meshes, called Complex Wrinkle Fields. Complex Wrinkle Fields consist of an almost-everywhere-unit complex-valued

[1]This chapter is modified from Chen et al. (2023a). Please refer this webpage for details. All the videos mentioned in this Chapter can be found here

phase function over the surface; a frequency one-form; and an amplitude scalar, with a soft compatibility condition coupling the frequency and phase. We develop algorithms for interpolating between two such wrinkle fields, for visualizing them as displacements of a Loop-subdivided refinement of the base mesh, and for making smooth local edits to the wrinkle amplitude, frequency, and/or orientation. These algorithms make it possible, for the first time, to create and edit animations of wrinkles on triangle meshes that are smooth in space, evolve smoothly through time, include singularities along with their complex interactions, and that represent frequencies far finer than the surface resolution.

3.1 Introduction

Across widely ranging spatial and temporal scales, wrinkles on surfaces are a fundamental geometric structure. We encounter these structures in our daily interactions with the moving folds and creases in cloth, skin and films, and likewise, we regularly observe them in natural phenomena such as the slow evolution of sand dunes and the rapid rippling of shallow water. Wrinkles on surfaces thus critically enrich otherwise coarse geometric structures with important visual details, while manufactured wrinkle patterns enable the design of complex structures and material behaviors Lähner et al. (2018b); Evgeny and Harders (2019); Chen et al. (2021b).

The real-world examples illustrated in Figure 3.2 demonstrate the characteristic and highly complex features of surface wrinkling behaviors in the wild. Locally, wrinkles are generally characterized by a dominant frequency and amplitude, both of which vary inhomogeneously *but smoothly* over the surface. At the same time *singularities* punctuate wrinkle patterns (see e.g. circled regions in Figure 3.2). These singularities are the branch points of the phase field where multiple peaks and troughs join together. Near these singularities wrinkles are fundamentally *non-bandlimited*: as we approach a singularity, circulating an arbitrarily small distance around it results in an arbitrarily large shift in phase. Figure 3.2, bottom row, shows several frames of

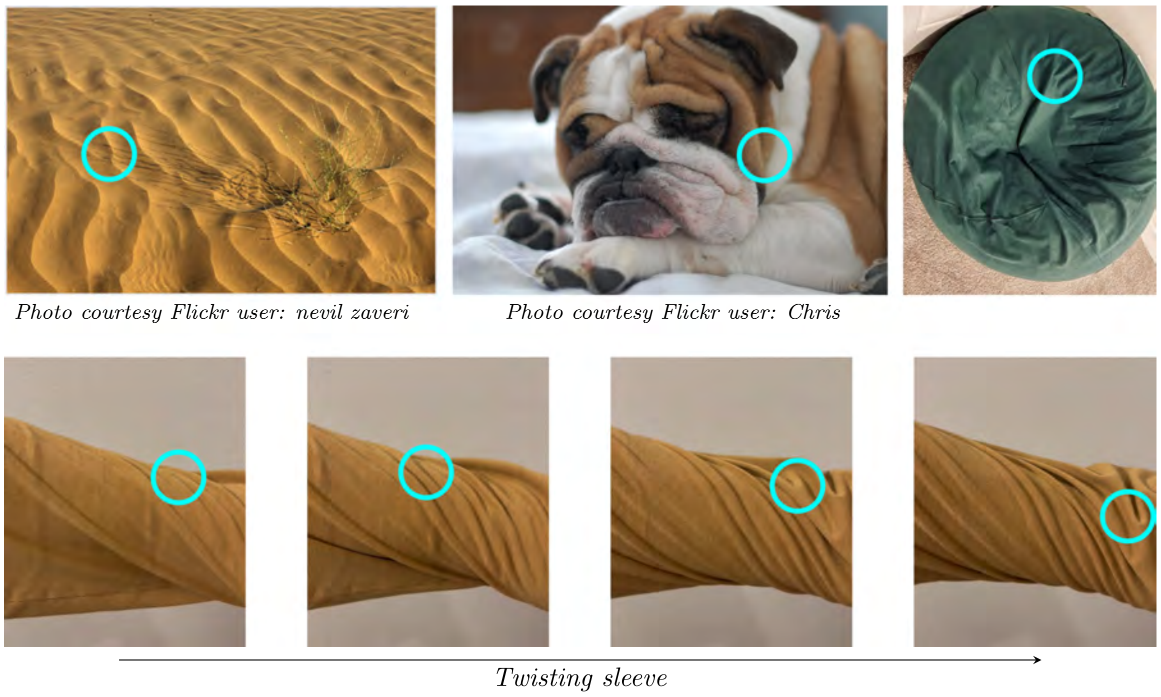


Photo courtesy Flickr user: nevil zaveri
Photo courtesy Flickr user: Chris
Twisting sleeve

Figure 3.2: Wrinkles on surfaces are ubiquitous, from sand dunes (*top-left*) to the creases and folds on a wrinkly dog face (*top-middle*) and a bean bag chair (*top right*). Notice that several key features are evident in all of these examples: frequency and amplitude that vary smoothly over the surface, punctuated by singularities (we circle some examples in aqua) where frequency diverges, amplitude vanishes, and multiple wave crests and troughs commingle in a Y-like pattern. In the bottom row, we show a sequence of wrinkle patterns formed during the twisting of a cloth shirtsleeve, where the singularities appear, merge and slide during the pattern's evolution.

wrinkle evolution during twisting of a sleeve of a cloth. Singularities appear, merge, and slide over the surface as wrinkles shift, split, and reconnect during their evolution.

In the smooth setting (see Section 3.2.1) wrinkles have an appealing and compact representation as amplitude and phase signals-on-surface; on the other hand the above characteristic features pose significant modeling challenges when it comes to discretizing and evolving wrinkles on triangle meshes. In this work we address these challenges.

3.1.1 Contributions

First, because the evolving singularities and high-frequency wrinkles in fine wrinkle patterns are generally not captured at practical mesh resolutions, we construct a discretization of wrinkles called the Complex Wrinkle Field (CWF) that captures in-element singularities and multiple sub-element wrinkling periods (Section 3.3). The CWF representation enables mesh-independent resolution of fine wrinkles.

Second, to enable smooth design, editing, control, animation, and evolution between wrinkle patterns we construct a mechanics-based algorithm for continuous and temporally-coherent interpolation between *arbitrary* wrinkle patterns on surfaces (Section 3.4). Our algorithm takes as input two CWF endpoints ("keyframes") and solves a boundary value problem approximating a geodesic path in the space of inextensible shells. The results are complex, shifting wrinkle patterns, where singularities evolve automatically to support the necessary branching and merging of wrinkles needed to interpolate between the keyframed patterns.

Third, to complement CWF's ability to model sub-element wrinkle resolution, we derive a new subdivision method that maps CWFs on triangle meshes to amplitude and phase on vertices generated by Loop iterates (Section 3.5). This method resolves arbitrarily fine wrinkled geometry, including singularities, while avoiding artifacts seen in existing baselines (see Section 3.5.3) and so provides high-quality upsampling of wrinkles for rendering and other downstream applications.

We extensively evaluate each of these contributions relative to state-of-the-art alternatives, and then demonstrate their joint application to animating wrinkle evolution on triangle surfaces, and to smooth, user-in-the-loop wrinkle editing (including both local and global manipulations). Across these applications we show that our contributions account for evolving singularities while generating smooth and intuitive interpolants between wrinkle patterns (Section 3.6). For example, in Figure 3.3 (bottom row), we show that, with proposed techniques, we can generate wrinkles with the double frequency and half of amplitude of the simulated results obtained from

Chen et al. (2021b)— effectively replacing thick cloth with thinner, silkier material without re-simulation. Furthermore, we can swipe through a smooth interpolation of the wrinkle patterns to select the desired look.

Despite the prominence of evolving wrinkles in the natural world, we emphasize that to date there is **no** model or method that supports temporal wrinkle interpolation on surfaces (in the sense of computing a path of spatially- and temporally-continuous wrinkle patterns between prescribed keyframes). Our *CWF* representation and algorithms are the first designed from the ground up to address the challenges imposed by evolving singularities during wrinkle interpolation. These challenges are generally unavoidable, even when the underlying surface and the prescribed keyframes are simple. For example, consider the torus interpolation in Figure 3.3 (top row) and in the supplementary **Interpolation Results** video at 00:02–00:19. Although neither keyframe contains singularities, it is topologically impossible to follow a smooth path between them without introducing (and then annihilating) singularities along the way.

In summary, Complex Wrinkle Fields (*CWF*) make it possible, for the first time, across all wrinkle inputs, to create, edit, and animate detailed wrinkles on triangle meshes that are smooth in space and evolve smoothly through time; while, at the same time, implicitly including the complex interactions of singularities, and resolving wrinkle frequencies far finer than the surface mesh resolution. Raw data and a reference implementation of our algorithms, can be found in this webpage. The latest version of the code can also be found on GitHub[2].

[2]https://github.com/zhenchen-jay/Complex-Wrinkle-Field.git

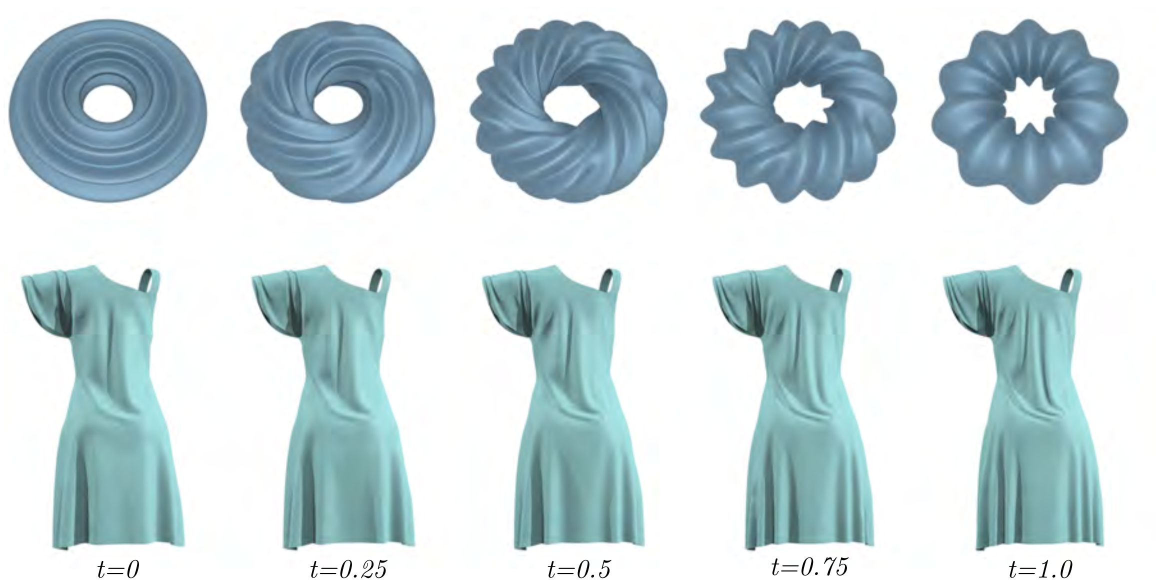


Figure 3.3: Two examples of *CWF* interpolation between keyframes specified at $t = 0$ and 1. *Top row:* a torus, with wrinkles keyframed to rotate by ninety degrees; *bottom row:* a dress example from Chen et al. (2021b) (our TFW Section 2), where we emulate replacing the cloth with a thinner material by uniformly doubling the wrinkle frequency and halving the amplitude. For the rotating torus example, although neither keyframe contains singularities, it is topologically impossible to follow a smooth path between them without introducing (and then annihilating) singularities along the way. Here our *CWF* interpolation allows wrinkles to break apart before reconnecting again. The dress demonstrates the possibility of using our algorithms to reasonably edit physical wrinkles and generate corresponding smooth animation. Videos of these two examples can be found in the **Interpolation Results** supplementary video at 00:02–00:19 (for the torus) and 03:47–04:03 (for the dress).

3.2 Preliminaries and Related Work

3.2.1 Wrinkles on Surfaces

Smooth Setting A useful Knöppel et al. (2013, 2015) representation of wrinkles on smooth manifolds $\mathcal{M}$ is by a single complex smooth function $\boldsymbol{z} : \mathcal{M} \to \mathbb{C}$, where the wrinkle amplitude a is encoded as $|\boldsymbol{z}|$, and the associated phase as $\theta = \arg(\boldsymbol{z})$. Here we assume wrinkles are normal displacements of $\mathcal{M}$ with magnitude $a \cos\theta = \Re(\boldsymbol{z})$. The smoothness of $\boldsymbol{z}$ guarantees a smooth non-negative amplitude, and a smooth associated phase, except at a set of singular points where $\boldsymbol{z} = 0$ and along branch cuts between them. At these singular points the phase is undefined and the amplitude vanishes. Away from these singularities, the derivative of phase yields the (one-form-valued) wave frequency $\omega = d\theta$. It is important to note that the frequency field singularities are not the usual ones studied in geometry processing where $\omega = 0$; rather, here $\|\omega\| \to \infty$ as one approaches a singularity.

Discrete Setting Frequency and amplitude are *the* semantically-meaningful parameters for encoding the wrinkling phenomena discussed in Section 3.1 and shown in Figure 3.2. Our goal then is to translate the above spectral representation of wrinkles from the smooth setting to triangle meshes $\mathcal{T} = \{\mathbf{v}, \mathbf{e}, \mathbf{f}\}$. As we motivated in the last section, a practical discretization of $\boldsymbol{z}$ on a triangulation by a set of discrete parameters $\boldsymbol{u}$ should satisfy three basic desiderata:

1. The parameters $\boldsymbol{u}$ must allow for singularities; i.e., it should be possible to reproduce wrinkle patterns with branch points as illustrated in Figure 3.2.

2. High frequency wrinkling should be representable *independent* of the resolution of the underlying triangle mesh, and in particular, wrinkles should be allowed much more than a single period within a triangle. This decoupling (the main motivation for a spectral representation of wrinkles) is needed both because fine wrinkle patterns are common in the real world (as in the skin and cloth images in Figure 3.2) and because high frequencies are unavoidable in the vicinity of singularities.

3. We must be able to render a smoothly wrinkled, high-resolution surface for all valid $\boldsymbol{u}$, *and* this surface geometry must change continuously for continuous changes in $\boldsymbol{u}$. This requirement represents a minimum foundation for doing temporally-coherent wrinkle interpolation.

Surprisingly, given how straightforward it is to represent wrinkles in the smooth setting, the most direct strategies for discretizing $\boldsymbol{z}$ (e.g., storing a complex number z_i on each vertex) do not meet the above requirements, as we discuss shortly; constructing the right discrete representation to support temporal interpolation is a challenge in its own right. After reviewing related work we will present our discrete *CWF* model in Section 3.3, and compare with some natural alternative choices of discretization and demonstrate their shortcomings.

3.2.2 Prior Work

Spectral Representations of Wrinkles on Surfaces Prior work has recognized the significant challenges in representing wrinkles on triangle meshes, and has offered a range of partial solutions. A common approach Chen et al. (2021b); Aharoni et al. (2017); Paulsen et al. (2016) is to design for the wave frequency without regard to phase, using vector-field design tools from geometry processing; when phase is required (e.g., for rendering), it is recovered as a post-process by solving for the phase whose gradient most closely matches the frequency: either via least-squares optimization, or using techniques from the parameterization literature Ling et al. (2015); Diamanti et al. (2015); Ray et al. (2006); Zhang et al. (2010). Knöppel et al. (2015) propose a particularly powerful variation of this idea: their algorithm solves for a stripe pattern (phase field) over arbitrary surfaces given a user-provided vector field, where singularities are placed at the centers of the triangles as needed to produce topological dislocations in the pattern. Noma et al. (2022) build on this technique by allowing users to interactively edit the location of singularities in stripe patterns. Beyond designing stripe patterns, Knöppel et al.'s algorithm has been applied for real-time surface parameterization Lichtenberg et al. (2018) and continuous fiber design Boddeti et al. (2020).

While these "frequency-first" strategies can produce beautiful, high-frequency wrinkle patterns on surfaces, they all suffer from a key limitation: smooth change in frequency does not yield a smooth change in phase. These strategies are therefore unsuitable for smooth animation of wrinkles due to temporal incoherence: they will produce a temporally discontinuous phase sequence, even when the input frequency sequence undergoes a smooth temporal evolution. For example, in the **main** supplementary video (02:59–03:44) and the **Comparisons** video (00:05–00:59), we show an animation of rotating waves on a torus, where we apply the method described by Chen et al. (2021b) and Knöppel et al. (2015) to every frame. Although the individual frames are spatially smooth, they are not temporally coherent.

Vector Field Design Over decades, many methods have been developed for vector field design. Vaxman et al. (2016) and de Goes et al. (2016b) summarize the classical ways to design and discretize vector fields on triangle meshes. In general, there are several ways to represent discrete vector fields. One popular approach specifies each vector with respect to a local Cartesian or polar coordinate system Knöppel et al. (2013); Diamanti et al. (2014). This representation is agnostic to singularities and well-suited for methods that automatically place them Bommes et al. (2009); Panozzo et al. (2014); Ray et al. (2009); Diamanti et al. (2015); Jakob et al. (2015). However, enforcing global integrability of the vector field in this representation is difficult and requires non-linear constraints or integer variables. Another approach represents vector fields implicitly in terms rotation angles on dual edges Li et al. (2006); Ray et al. (2008); Crane et al. (2010). This representation is particularly useful for parameterizing vector fields while maintaining full control over singularity placement and index Fisher et al. (2007); Zhang et al. (2006); Solomon and Vaxman (2019) but integrability is even more elusive: *two* levels of integrability constraints are required (one for integrating the angles to well-defined vectors and another for integrating the vectors to a well-defined scalar field). As we discuss in Section 3.3, neither of these representations are suited for representing temporally-coherent, high-frequency wrinkle fields with moving singularities on discrete triangle meshes.

Vector Field Subdivision Subdivision operators have been designed that extend Loop subdivision to one-forms on triangle mesh edges de Goes et al. (2016a); Wang et al. (2006). We will use this scheme as a part of our approach (Section 3.5.1).

Custers and Vaxman (2020) designed a subdivsion scheme for tangent directional fields on triangle faces. However, these methods do not cover the corresponding subdivision of amplitude and phase, which are essential for upsampling wrinkles.

Vector Field Temporal Interpolation Temporally interpolating vector fields has long been studied in geometry processing. Zavala-Hidalgo et al. (2003) interpolate high-frequency vector wind fields by decomposing these fields into a basis of empirically-derived orthogonal functions, and then interpolating eigenmodes in time. Chen et al. (2012) propose a framework for designing time-varying vector fields by solving a spatial-temporal Poisson problem; their method implicitly generates bifurcations and singularities in the vector field. Sato et al. (2018) propose an algorithm that takes two entire time sequences of velocity fields and produces time sequences that blend between them. Solomon and Vaxman (2019) use optimal transport theory to match the singularities in two input vector fields, which allows for their direct interpolation. However, vector field interpolation, on its own, does not solve the problem of interpolating wrinkles, where the phase and amplitude must also be interpolated, and, as discussed above, there is no obvious way to recover temporally-coherent phase from frequency.

Spectral Wrinkle Evolution There are several methods for simulating spectrally-represented wrinkles forwards through time from an initial state: in a line of work, Jeschke, Wojtan, and collaborators Jeschke and Wojtan (2015, 2017); Jeschke et al. (2018) propose several approaches to efficiently evolve water waves based on Airy wave theory Airy (1842), which uses a sum of sinusoidal functions to linearly approximate the motion of surface waves on the body of water. Evgeny and Harders (2019) simulate wrinkle motion on 3D surfaces via reaction-diffusion Turk (1991); Witkin and Kass (1991). Although these approaches generate animations of wrinkles given initial conditions (initial value problem), they do not address the wrinkle keyframe interpolation problem, which is completely different (boundary value problem).

Shell Geodesics Heeren et al. (2012) propose a way to find the geodesic between two thin shells based on the minimization of strain dissipation. This idea is further explored for time discretization of geodesic calculus on certain Riemannian manifolds Rumpf and Wirth (2014), understanding the geometry of the space of shells Heeren et al. (2014), or finding geodesics between two general elastic shapes Ezuz et al. (2019); Sassen et al. (2020). Von-Tycowicz et al. (2015) adopt a similar idea and design a special subspace optimization problem to interpolate between elastic body poses in real time. Our wrinkle interpolation algorithm follows in the footsteps of these methods.

Surface Augmentation Methods have also been proposed to augment coarse surfaces with fine wrinkles as an alternative to expensive, fine-scale physical simulation. Indirect techniques have been proposed for upsampling cloth Bergou et al. (2007a); Wang (2021) and skin Rémillard and Kry (2013a) by simulating a higher-resolution mesh that is constrained to stay close to an existing, coarse simulation. Rohmer et al. (2010) and Gillette et al. (2015a) add fine wrinkle detail to coarse cloth simulations by tracing compression direction fields in a temporal coherent manner. Cutler et al. (2005) allow users to design wrinkle patterns on a set of reference poses, then add weighted combinations of the designed wrinkles to each frame of simulation based on local stresses. Data-driven methods for enriching coarse meshes with fine detail, for example, by representing wrinkles as normal maps Lähner et al. (2018b) or displacements Chen et al. (2018b, 2021a); Santesteban et al. (2019a); Zhang et al. (2021), are likewise an active and rapidly evolving area of research but require significant preprocessing and generally depend on a corpus of data.

3.3 Complex Wrinkle Field Discretization

In the smooth setting, a single complex function $\boldsymbol{z}$ suffices to represent a wrinkle pattern. Unfortunately, $\boldsymbol{z}$ admits no straightforward discretization that safisfies the properties listed in Section 3.2.1. In this section, we introduce our complex wrinkle field (*CWF*) representation, the design decisions that motivate it, and some of the pitfalls with alternative discretizations. In the following sections we will then discuss how to both (a) temporally interpolate between *CWF*s and (b) upsample (and so render) *CWF*s while preserving these properties.

High-level Summary There are three key design decisions that motivate *CWF*s:

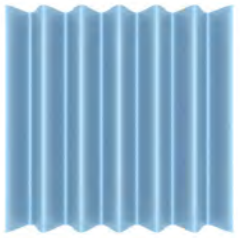
- To represent high-frequency wrinkles on a mesh, unlike in the smooth setting, a phase-based representation is insufficient. The discretization must additionally track frequency.
- For the wrinkle representation to capture singularities, the frequency field cannot be globally integrable in the usual sense of being the derivative of a real-valued function, *and* cannot even be locally integrable near singularities. We therefore eschew integrability entirely and allow arbitrary frequencies.
- Since frequency is not integrable, we cannot and should not exactly enforce that frequency is the derivative of phase. But we also do not want phase and frequency to be incompatible. We therefore require *soft* compatibility of phase and frequency, in a way that is well-defined at singularities and enforces $\mathrm{d}\arg(\boldsymbol{z}) \approx \omega$ away from singularities.

A *CWF* therefore consists of three sets of degrees of freedom: (1) a one-form ω_{jk} on the edges of $\mathcal{T}$ (representing the wrinkle pattern frequency); (2) a real scalar a_j per mesh vertex (representing the wrinkle amplitude); and (3) a complex number $\tilde{\boldsymbol{z}}_j$ per vertex (encoding the wrinkle phase). There are no constraints on the ω_{jk} and a_j, but $\tilde{\boldsymbol{z}}$ is required to be approximately unit and to satisfy a *weak compatibility* condition with respect to ω. The reader immediately interested in the formal definition of a *CWF* may skip to Section 3.3.1; in what follows we motivate and justify each of the above decisions.

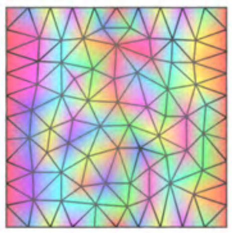
Need for Both Frequency and Phase The most direct way to discretize $\boldsymbol{z}$ is as a discrete function on the mesh vertices. The strength of this approach is that it uniquely pins down the wrinkle amplitude $|\boldsymbol{z}|$ and wrinkle phase $\arg(\boldsymbol{z})$ at each vertex. However, $\boldsymbol{z}$ alone is unable to represent wrinkles with high frequencies, as illustrated in Figure 3.4: at most one wrinkle period per edge is possible, since the phase change $\arg(\boldsymbol{z}_j) - \arg(\boldsymbol{z}_i)$ across the edge is bounded by 2π. By contrast, a frequency one-form ω can encode arbitrarily high frequencies independent of mesh resolution; however, a discrete representation based on frequency alone is unsuitable for applications like ours requiring temporal coherence, since wrinkle phase is underdetermined given frequency alone.

A mixed representation that tracks both phase and frequency simultaneously combines their advantages, and eliminates their disadvantages. This observation motivates a discrete representation that includes both a real-valued one-form ω_{ij} on the mesh edges, and a complex number $\boldsymbol{z}_j$ per mesh vertex. (In the *CWF* representation, we further decompose $\boldsymbol{z}_j$ into parts as $\boldsymbol{z}_j = a_j \tilde{\boldsymbol{z}}_j$ with $a_j, \tilde{\boldsymbol{z}}_j$ as our DOFs rather than $\boldsymbol{z}_j$ itself, for reasons we cover below.)

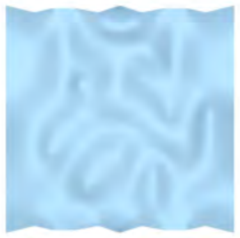
No Integrability Constraints on Frequency In the smooth setting, frequency is *defined* as the derivative of phase. A natural question when discretizing frequency is whether to require it, by analogy, to be (discretely) integrable. Several important subtleties make doing so impractical. Note that, in the smooth setting, ω is not globally integrable in the usual sense: ω is the exterior derivative of a section of an S^1-bundle over $\mathcal{M}$-with-singularities-removed. The line integral of ω around a singularity does not necessarily vanish and is instead a multiple of 2π. Requiring $\oint \omega = 0$ around every closed curve in $\mathcal{M}$ would constrain the wrinkle pattern to have *no* singularities. Similarly, in the discrete setting, if a triangle $jk\ell$ contains a singularity, the discrete curl on that triangle $\omega_{jk} + \omega_{k\ell} + \omega_{\ell j}$ does not vanish. Rather than selectively enforcing integrability of ω on only some triangles (which would require explicitly tracking which triangles contain singularities of which index using



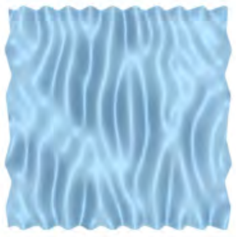
(a) Smooth wrinkles



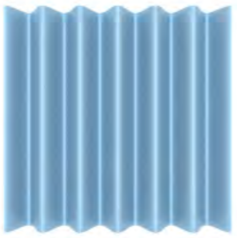
(b) Coarse $\arg \boldsymbol{z}$



(c) Upsampled coarse $\boldsymbol{z}$



(d) Incompatible CWF



(e) Compatible CWF

Figure 3.4: We show different discretizations of a plane wave associated with $\boldsymbol{z} = \exp(3\pi i x)$. (a): the exact wrinkle pattern $\boldsymbol{z}$, visualized as a normal displacement with magnitude $\Re(\boldsymbol{z})$. (b): we sample $\boldsymbol{z}$ onto a coarse mesh and plot $\arg(\boldsymbol{z}_j)$; the discrete phase is badly aliased. (c): attempting to visualize the wrinkles by displacing each vertex by $\Re(\boldsymbol{z}_j)$ yields a surface that bears no resemblance to the expected wrinkle pattern in subfigure (a). (d): the result when we apply our subdivision algorithm from Section 3.5 to visualize a *CWF* with $\boldsymbol{z}_j = \exp(3\pi i x_j)$ and the unrelated, incompatible $\omega_{jk} = (0, 3\pi) \cdot (\boldsymbol{v}_k - \boldsymbol{v}_j)$. Although the rendered wrinkled mesh is smooth, it is riddled with mesh-dependent singularities, and ω has little semantic relationship to the waves in the resulting pattern. We insist on soft compatibility between ω and $\tilde{\boldsymbol{z}}$ to avoid such artifacts. (e): this time, we use $\omega_{jk} = (3\pi, 0) \cdot (\boldsymbol{v}_k - \boldsymbol{v}_j)$, a compatible frequency that gives us a valid *CWF*. When rendered, this discrete *CWF* is indistinguishable from the exact pattern.

integer variables, etc.) we instead follow in the footsteps of previous work Knöppel et al. (2015); Evgeny and Harders (2019) and allow arbitrary ω. The singularities in the wrinkle pattern are then implicitly determined by where ω fails to be integrable.

Frequency-Phase Consistency Since we do not require ω to be integrable, ω cannot in general be exactly the derivative of phase. Recall, however, that the motivation for including frequency in our discrete representation is so that it can supply the information missing from $\boldsymbol{z}_j$ (alone) about the jump in phase across a mesh edge.

For this purpose, frequency is only useful if, far away from singularities, we *do* have $\omega \approx \mathrm{d}\arg(\boldsymbol{z})$. (See Figure 3.4d for an example where frequency and phase are totally uncorrelated.) We therefore require the phase variables in a *CWF* to be "as compatible as possible" with the frequency, (a) given that frequency is not necessarily integrable and (b) accounting for the necessary presence of singularities.

In the smooth setting, one formulation of the above *soft* compatibility condition is to require $\boldsymbol{z}$ to be compatible in a least-squares sense by minimizing an energy similar to

$$\int_{\mathcal{M}} \|\mathrm{d}\arg(\boldsymbol{z}) - \omega\|^2 \, \mathrm{d}A. \tag{3.1}$$

We cannot use this energy as-is, as it is formally undefined near singularities and at branch cuts. To avoid the discontinuity in $\arg(\boldsymbol{z})$ at branch cuts, we rewrite the constraint in terms of $\tilde{\boldsymbol{z}} = \boldsymbol{z}/|\boldsymbol{z}|$ Knöppel et al. (2015),

$$\tilde{\boldsymbol{z}} = \arg\min_{\tilde{\boldsymbol{z}}} \int_{\mathcal{M}} \|(\mathrm{d} - i\omega)\tilde{\boldsymbol{z}}\|^2 \, \mathrm{d}A \quad s.t.\ |\tilde{\boldsymbol{z}}| = 1, \tag{3.2}$$

which has the added benefit of eliminating some of the nullspace from Equation (3.1) (which is invariant under rescalings of $\boldsymbol{z}$).

Singularity Handling Equation (3.2) remains ill-posed at singularities (where neither $\tilde{\boldsymbol{z}}$ nor ω are well-defined) and ill-conditioned close to the singularities, where $\|\omega\| \to \infty$.

There are many potential strategies for relaxing Equation (3.2) to regularize the behavior at singularities. Our approach is to relax the unit-norm constraint, so as to allow $\tilde{\boldsymbol{z}}$ to vanish at singularities, using a Ginzburg-Landau-type term Kohn (2006); Viertel and Osting (2019):

$$\tilde{\boldsymbol{z}} \in \mathrm{Opt}_{\omega} = \arg\min_{\tilde{\boldsymbol{z}}} \int_{\mathcal{M}} \left[\|(\mathrm{d} - i\omega)\tilde{\boldsymbol{z}}\|^2 + \left(|\tilde{\boldsymbol{z}}|^2 - 1\right)^2 \right] \mathrm{d}A. \tag{3.3}$$

For later reference, we define:

$$\begin{aligned} E_{\text{compat}}(\tilde{\boldsymbol{z}}, \omega) &= \int_{\mathcal{M}} \|(\mathrm{d} - i\omega)\tilde{\boldsymbol{z}}\|^2 \, \mathrm{d}A, \\ E_{\text{unit}}(\tilde{\boldsymbol{z}}) &= \int_{\mathcal{M}} \left(|\tilde{\boldsymbol{z}}|^2 - 1\right)^2 \, \mathrm{d}A. \end{aligned} \tag{3.4}$$

We next discretize this variational characterization of $\tilde{\boldsymbol{z}}$ to define the final, discrete soft compatibility conditions we impose to our *CWF* representation.

3.3.1 Complex Wrinkle Field Representation

Motivated by the above discussion, we define *CWF* to consist of the following degrees of freedom:

1. a (not necessarily integrable) frequency one-form, encoded as a real number ω_{jk} on each directed edge of the mesh (with $\omega_{jk} = -\omega_{kj}$);
2. a real number a_j and complex number $\tilde{\boldsymbol{z}}_j$ on each vertex of the base triangle mesh $\mathcal{T}$, which together encode the approximate amplitude and direction of $\boldsymbol{z}_j = a_j \tilde{\boldsymbol{z}}_j$ discretizing the smooth setting's amplitude-phase field $\boldsymbol{z}$;
3. with the $\boldsymbol{z}_j$ constrained to be *softly compatible* with the frequency one-form, $\mathrm{d}\arg(\boldsymbol{z}) \approx \omega$, via discretization of Equation (3.3) on the base triangle mesh.

To discretize the first, least-squares, compatability term of Equation (3.3), we apply the formulation proposed by Knöppel et al. (2015), as it is designed to be robust in the case that ω_{jk} is larger than 2π on an edge jk,

$$\hat{E}_{\text{compat}}(\tilde{\boldsymbol{z}}, \omega) = \sum_{\text{edges } k\ell} A_{k\ell} \left|\tilde{\boldsymbol{z}}_k \exp(i\omega_{k\ell}) - \tilde{\boldsymbol{z}}_\ell\right|^2, \tag{3.5}$$

and directly apply piecewise-constant discretization for the second, unit-length penalty term,

$$\hat{E}_{\text{unit}}(\tilde{\boldsymbol{z}}) = \sum_{\text{vertices } j} A_j \left(|\tilde{\boldsymbol{z}}_j|^2 - 1\right)^2, \tag{3.6}$$

where the $A_{k\ell}$ and A_j are edge and vertex barycentric area weights, respectively. We then require

$$\tilde{\boldsymbol{z}} \in \hat{\mathrm{Opt}}_\omega = \arg\min_{\tilde{\boldsymbol{z}}} \left[\hat{E}_{\mathrm{compat}}(\tilde{\boldsymbol{z}}, \omega) + \hat{E}_{\mathrm{unit}}(\tilde{\boldsymbol{z}})\right], \tag{3.7}$$

where $\hat{\mathrm{Opt}}_\omega$ defines our discretized consistency conditions.

Remarks Note that in Equations (3.4) and (3.7) we have made an implicit choice to weight the compatibility and unit-norm terms equally. For regions away from the singularities, $E_{\mathrm{compat}} \approx 0$, so that Equation (3.4) gives us unit $\tilde{\boldsymbol{z}}$ for any positive scaling of E_{unit}. For regions near the singularities, $E_{\mathrm{compat}} \to \infty$ unless $\tilde{\boldsymbol{z}} \to 0$ so that, for any scaling of E_{unit}, the least-squares energy E_{compat} is the dominant term, which ensures that $\tilde{\boldsymbol{z}} \approx 0$. These observations show that Opt_ω is largely invariant to the relative scaling of the two constraint terms, and we make the simplest choice to weight them equally. Note also that Opt_ω is a nontrivial subspace and contains more than a single element. At minimum, if $\tilde{\boldsymbol{z}} \in \mathrm{Opt}_\omega$, then so is its global phase shift $\exp(i\theta)\tilde{\boldsymbol{z}}$ for every $\theta \in [0, 2\pi)$.

3.4 Interpolation

In this section we present an algorithm for *temporal interpolation* of *CWF*s: given two *CWF*s as boundary conditions, $(\omega^0, a^0, \tilde{\boldsymbol{z}}^0)$ and $(\omega^1, a^1, \tilde{\boldsymbol{z}}^1)$, we seek a smooth interpolant $\gamma(t) = (\omega[t], a[t], \tilde{\boldsymbol{z}}[t])$ through the space of *CWF*s which matches the prescribed boundary conditions at $t = 0, 1$ and which approximates the behavior of real-world wrinkled materials deforming over time.

The challenge with computing such an interpolation is that while there are many smooth choices of interpolant $\gamma(t)$, they are often qualitatively unacceptable. For physically-based wrinkle evolution, wrinkles should *slide* over the surface rather than disappear in one location and then reappear at a target location, and singularities should slide smoothly over the surface. Otherwise plausible-seeming interpolants that are not specifically designed to promote physical fairness of the wrinkle motion regularly produce obvious visual artifacts that are unacceptable for many applications such as animation or editing. For many examples please see Section 3.4.3 and the **Interpolation Results** video.

3.4.1 Our Approach

To compute physics-inspired, smooth paths between boundary conditions in the space of *CWF*s, we begin with the closely-related problem of computing geodesics in the space of thin shells (see e.g. the work of Heeren et al. (2012)). For our application, the key idea from this work boils down to defining a metric on shell space that measures the length of a path between configurations in terms of the integrated norm of strain-rate along that path. A shortest geodesic in this space is then, effectively, the path that requires doing the least work on the system when deforming the shell along that path.

To compute comparable wrinkling geodesics for *CWF*s we measure a bending strain[3] given in terms of the combined geometry of the underlying wrinkle-free base surface $\mathcal{M}$ and the wrinkle displacements Chen et al. (2021b),

$$\begin{aligned}\epsilon &= \boldsymbol{I}^{-1}\left(\boldsymbol{II} - \bar{\boldsymbol{II}} - a\cos\theta\omega^T\omega\right) && (3.8)\\ &= \boldsymbol{I}^{-1}\left(\boldsymbol{II} - \bar{\boldsymbol{II}} - \Re(\boldsymbol{z})[\mathrm{d}\arg\boldsymbol{z}]^T[\mathrm{d}\arg\boldsymbol{z}]\right), && (3.9)\end{aligned}$$

where $\boldsymbol{I}$ and $\boldsymbol{II}$ are the first and second fundamental forms of the base surface and $\bar{\boldsymbol{II}}$ is the second fundamental form of the shell's assumed rest state. Following Heeren et al. (2012), we can then define our distance on paths $\gamma(t)$ by

$$d(\gamma) = \int_0^1 \int_{\mathcal{M}} \|\dot{\epsilon}[\boldsymbol{z}(t)]\|^2 \mathrm{d}A\,\mathrm{d}t, \tag{3.10}$$

[3]We make the simplest assumption that the stretching strains in the wrinkled surface represented by a *CWF* are negligible. This decision is justified as compressive stresses should be mostly absorbed by wrinkling. That said, unless both *CWF* keyframes of an interpolation represent isometric deformations of the same underlying surface, there must be some amount of stretching involved and so it should be interesting to consider inclusion of stretching measures as well in future work.

where, assuming a hyperelastic homogeneous and isotropic material, we apply the StVK material norm for $\|\cdot\|$; please see Appendix B.1 for details.

Computing Geodesics Geodesics are then constrained minimizers γ of Equation (3.10) satisfying $\tilde{\boldsymbol{z}}(t) \in \mathrm{Opt}_{\omega(t)}$. However, computing these geodesics by directly minimizing Equation (3.10) is difficult. In Appendix B.1.2 we derive, after a sequence of approximations and additional simplifying assumptions, an approximate solution to Equation (3.10) given by

$$a(t) = (1-t)a^0 + ta^1 \tag{3.11}$$

$$\tilde{\boldsymbol{z}}(t) = \arg\min \; E_{\text{smooth}}(\tilde{\boldsymbol{z}}) + cE_{\text{opt}}(\tilde{\boldsymbol{z}}, \omega) \tag{3.12}$$

and an explicit formula for $\omega(t)$ given in Equation (B.34) (Appendix B.1.2), where

$$E_{\text{smooth}}(\tilde{\boldsymbol{z}}) = \frac{1}{2}\int_{\mathcal{M}} g_{\text{bd}} \int_0^1 \left|\dot{\tilde{\boldsymbol{z}}}\right|^2 \mathrm{d}t\mathrm{d}A \tag{3.13}$$

$$g_{\text{bd}} = \frac{(a^0)^2 \|\omega^0\|^4 + (a^1)^2 \|\omega^1\|^4}{2} \tag{3.14}$$

approximates Equation (3.10) after a and ω are substituted in and simplified—effectively giving a weighted Dirichlet energy in spacetime promoting smoothness of wrinkle phase; and

$$E_{\text{opt}}(\tilde{\boldsymbol{z}}, \omega) = \int_0^1 \left[E_{\text{compat}}(\tilde{\boldsymbol{z}}[t], \omega[t]) + E_{\text{unit}}(\tilde{\boldsymbol{z}}[t])\right] \mathrm{d}t \tag{3.15}$$

provides a penalty term, with stiffness c, for the constraint $\tilde{\boldsymbol{z}} \in \mathrm{Opt}_\omega$ that $\tilde{\boldsymbol{z}}$ and ω are approximately compatible. We use $c = 10^3 g_{\text{ave}}$ in all of our examples, where

$$g_{\text{ave}} = \int_0^1 \int_{\mathcal{M}} a^2 \|\omega\|^4 \,\mathrm{d}A\,\mathrm{d}t \tag{3.16}$$

is the spacetime averaged amplitude squared times frequency quartic, in order to make the cE_{opt} unit consistent. If c is too low, the interpolant $\gamma(t)$ fails to satisfy the *CWF* compatibility constraint, yielding noisy wrinkle pattern evolution and many extra singularities. If c is too high, the optimization problem (3.12) becomes numerically unstable and does not converge to a smooth solution. Please see Appendix B.7.2 for some experiments probing the effect of the choice of c on our results and the suitability of our empirically selected value.

Discretization Given two *CWF* boundary conditions $(\omega^0, a^0, \tilde{\boldsymbol{z}}^0)$ and $(\omega^1, a^1, \tilde{\boldsymbol{z}}^1)$ and a desired number of intermediate frames N, we discretize time into N steps of size $\delta t = 1/N$ and minimize a discretization of Equation (3.12) over the base triangle mesh $\mathcal{T}$ with a globalized Newton solve. Please see Appendix B.1.3 for additional details, including the full formulas for all our discrete analogues of the above energy terms used in our final solve. Within our Newton implementation we use SuiteSparse's parallel implementation of supernodal sparse Cholesky decomposition Chen et al. (2008) as our linear solver, and terminate when converged to an objective gradient norm smaller than 10^{-6}.

Linear Substepping The cost of computing N interpolating frames using our algorithm is dominated by the cost of solving Equation (3.12), which scales roughly linearly in N. When many frames are needed, for example when rendering videos, this computation is wasteful, since consecutive frames will be almost identical. For large N, we suggest using our algorithm to compute $N' = N/k$ guide frames instead, and then linearly interpolating the *CWF* variables for k steps between each guide frame. As a rule of thumb, linear interpolation is an adequate approximation to a geodesic path between guide frames if the distance traveled by wrinkles during that time does not exceed the average mesh edge length. We use $N = 200$, with $N' = 50$ and $k = 4$ in all of our videos, and provide timing information for the examples in this chapter in Table B.1. We further provide some experiments showing the effect of the choice of N' on the interpolant quality in Appendix B.7.1.

Handling Incompatible Boundary Conditions The above algorithm assumes that provided boundary conditions are valid *CWF*s. If applied to boundary conditions that severely violate *CWF* soft compatibility, our interpolant will move abruptly near $t = 0$ and 1 (since Equation (3.15) will enforce the soft compatibility condition everywhere except at the infeasible, prescribed endpoints). Since in applications these boundary conditions will often be provided by the user (as keyframes for animation, e.g.) we propose pre- and post-processing steps that allows use of our algorithm even when the boundary conditions have incompatible frequency and phase.

In particular, notice that the discrete compatibility term in Equation (3.5) implies the constraint per edge

$$\tilde{\boldsymbol{z}}_k \exp(i\omega_{k\ell}) = \tilde{\boldsymbol{z}}_\ell. \tag{3.17}$$

This allows us to decompose any frequency one-form ω into a consistent, constraint-satisfying portion, ω_{com}, and a residual $\delta\omega$; that is, there is a unique one-form $\delta\omega$, with $-\pi \leq (\delta\omega)_{jk} < \pi$ on each edge jk, such that $\omega_{\text{com}} = \omega - \delta\omega$ and $\tilde{\boldsymbol{z}}$ exactly satisfy Equation (3.17) on every edge. To handle non-*CWF* boundary conditions, we decompose the frequencies ω^0 and ω^1 into their consistent and inconsistent parts, and apply interpolation to the *CWF*s boundary conditions $(\omega^0_{\text{com}}, a^0, \tilde{\boldsymbol{z}}^0)$ and $(\omega^1_{\text{com}}, a^1, \tilde{\boldsymbol{z}}^1)$ instead. We then interpolate $\delta\omega$ using the same explicit interpolation applied for $\omega(t)$ given in Equation (B.34) (Appendix B.1.2), and add this frequency component back into the final interpolant: $\gamma(t) = (\omega(t) + \delta\omega(t), a(t), \tilde{\boldsymbol{z}}(t))$.

3.4.2 Results

We demonstrate that our interpolation algorithm yields high-quality wrinkle evolution given a variety of keyframes pairs. Even more interpolation examples, with keyframes created by local edits to wrinkle patterns, are discussed in Section 3.6. Important note: it is difficult to assess the quality of the interpolant, and especially temporal coherence, from a sparse set of stills. Please see the **Comparisons** supplemental video for animations of these examples.

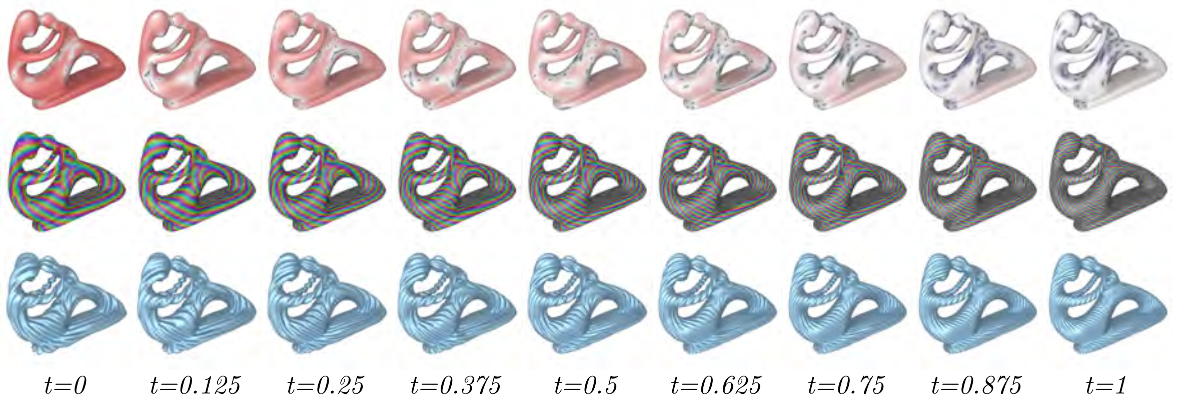


Figure 3.5: *CWF* interpolation between two wrinkle patterns on the fertility model, where the second keyframe (last column) has triple the frequency and one-third the amplitude of the first keyframe (first column). We show, from top to bottom, the amplitude, phase, and rendered wrinkles for several intermediate frames during interpolation. On the amplitude plot, blue indicates $|\boldsymbol{z}| = 0$ (singularities) and red indicates larger amplitude. For the animation, please check the **Interpolation Results** video in the supplementary material at timestamp 01:46–02:01.

Change in Wrinkle Frequency Increasing the frequency of a wrinkle pattern on a compact surface like the fertility model (Figures 3.5 and 3.6), though conceptually simple, involves a complex path through the space of wrinkle fields since it is impossible to have "fractional" numbers of wrinkles. New wrinkles must be born as the frequency increases, with singularities appearing in pairs and sliding over the surface to "unzip" a new wrinkle.

Change in Wrinkle Direction We also show several examples of interpolation between wrinkle patterns that have been globally or locally rotated by ninety degrees relative to each other in Figures 3.1 and 3.7. The amplitude and phase plots show singularities appearing, splitting, and merging in a complex dance in both cases.

3.4.3 Comparison to Other Approaches

We compare our method against some baselines.

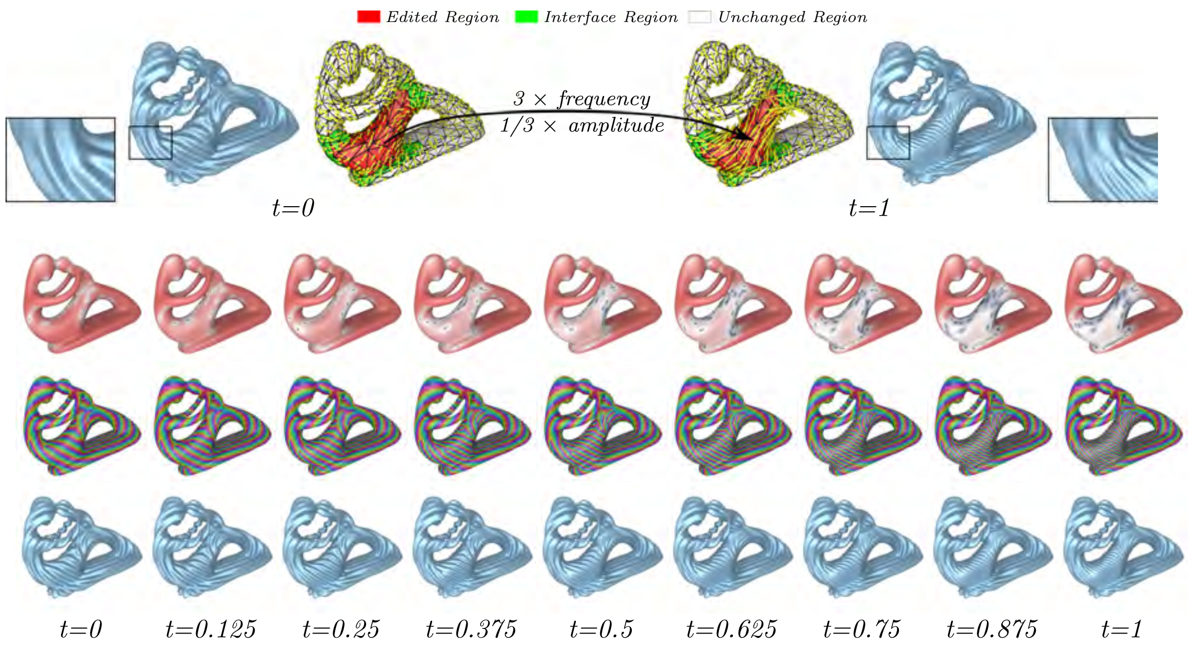


Figure 3.6: *CWF* interpolation of two wrinkle patterns on the fertility model, where the second keyframe's wrinkles are triple the frequency of the first keyframe, and their amplitude shrinks by one-third on a **local** patch of surface (the red region on the mesh up top). We design an interface region (the green region; see Section 3.6) to ensure a smooth transition between the red edited region and the unchanged white region. In the supplementary **Interpolation Results** video at timestamp 02:02–02:20, we show the corresponding wrinkle animation for a better temporal visualization.

Linear Interpolation It is natural to try to temporally interpolate between *CWF* boundary conditions by linearly interpolating the constituent variables:

$$a(t) = (1-t)a^0 + ta^1;\ \omega(t) = (1-t)\omega^0 + t\omega^1;\ \tilde{\boldsymbol{z}}(t) = (1-t)\tilde{\boldsymbol{z}}^0 + t\tilde{\boldsymbol{z}}^1.$$

Such an interpolant lacks physical meaning and moreover does not stay within the space of *CWF*s, since there is no reason to expect $\omega(t)$ and $\tilde{\boldsymbol{z}}(t)$ to stay approximately compatible during the interpolation. See Figure 3.10 and Figures B.9, B.10 in Appendix B.6 for some examples of artifacts that arise when using this linear scheme. Note that replacing linear interpolation of $\tilde{\boldsymbol{z}}$ and ω with a more complicated non-linear interpolant would not help resolve this incompatibility issue.

For comparison on a more didactic example, consider the simple case of a plane

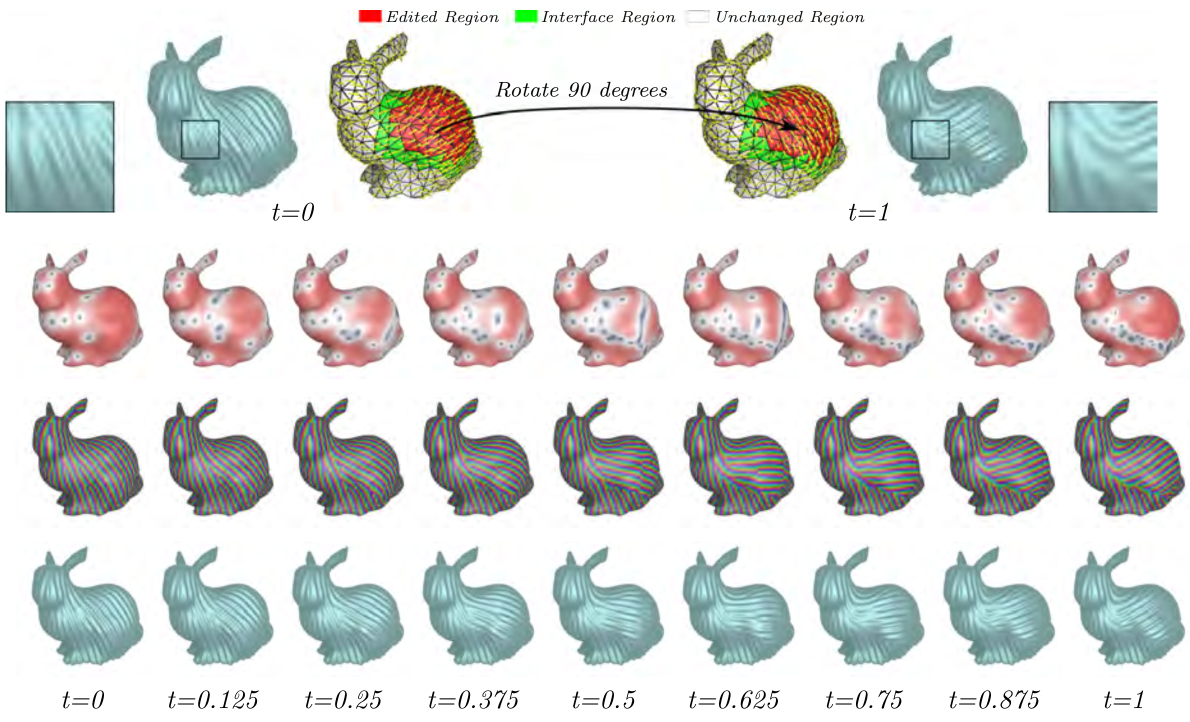


Figure 3.7: *CWF* interpolation of two wrinkle patterns on the Stanford bunny, where the user has **locally** rotated the frequency field by ninety degree (red region of the top mesh). Top row: the user-specified keyframes. Second row: the upsampled amplitudes $|\boldsymbol{z}| = a|\tilde{\boldsymbol{z}}|$ of the *CWF*, ranging from blue at singularities ($|\boldsymbol{z}| = 0$) to red where the amplitude attains its maximum. Third row: the upsampled phase of the *CWF*. Bottom row: the rendered wrinkle patterns. We see nontrivial wave mergers via singularities sliding. Please check the **Interpolation Results** video in the supplementary material at timestamp 01:29–01:45 for the corresponding animation.

wave rotating by ninety degrees, $z^0 = \exp(ix)$ and $z^1 = \exp(iy)$. On a sufficiently small patch centered at the origin, we expect interpolation to rotate the plane wave, with an interpolant resembling $z(t) = \exp(i[x\cos\frac{\pi t}{2} + y\sin\frac{\pi t}{2}])$. In Figure 3.8 we see that when using our proposed interpolation algorithm, the interpolant is indeed rigid rotation of the wrinkles. On the other hand, linear interpolation *shears* the wrinkles, which midway through the interpolation degenerate along a singular line.

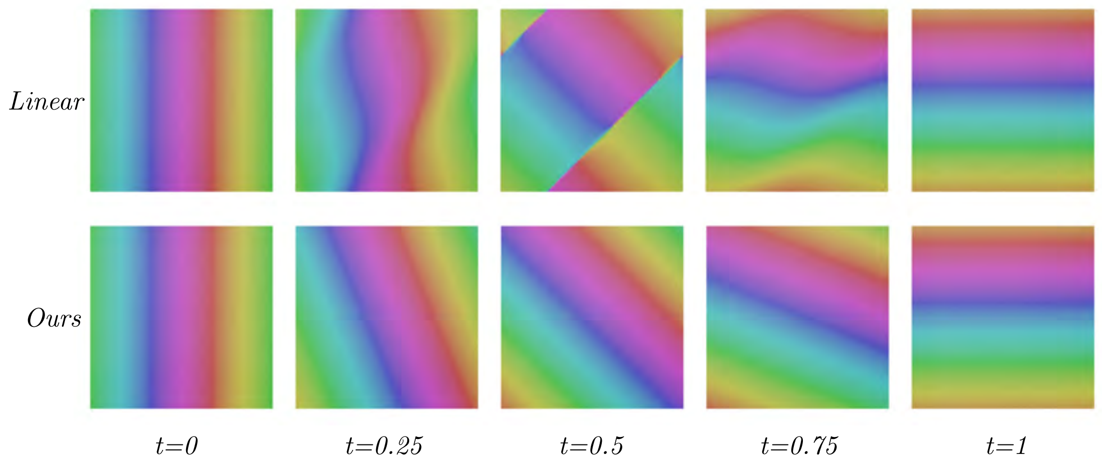


Figure 3.8: Failure of linear interpolation given boundary conditions $\boldsymbol{z}_0 = \exp(ix)$ and $\boldsymbol{z}_1 = \exp(iy)$. We show the phase patterns at several intermediate frames using linear interpolation (top row) and our proposed interpolant (bottom row). Notice that linear interpolation introduces severe artifacts in the wrinkle pattern: wrinkles shear and degenerate along a sharp line midway through the interpolation.

Knöppel et al. (2015) In their paper, Knöppel et al. propose a method to extract vertex phase information from a provided frequency one-form by solving an eigenvalue problem. They also propose a *spatial* interpolation scheme for extending phase into triangles by placing singularities at their centers as needed. While well-suited for generating individual, static geometries, Knöppel et al.'s method is not designed to give temporally coherent motion when run on a sequence of frames with smoothly-changing frequency. Indeed, if we apply their method to $\omega(t)$ (computed using Equation (B.34) in Appendix B.1.2) to generate a phase interpolant $\boldsymbol{z}(t)$, we get beautiful individual frames, as expected (Figure 3.10) but see that the frames are discontinuous in time (see the **main** supplemental video at timestamp 03:44–04:04).

Moreover, Knöppel et al. focus exclusively on phase fields and do not discuss wrinkles with amplitude. We make a good-faith attempt to incorporate amplitude in the method (See Figure 3.9) by linearly interpolating the keyframe vertex amplitudes in time and then barycentrically blending them into each triangle. The resulting artifacts are due to a mismatch between where Knöppel et al. places phase singularities (at centers of triangles) and where amplitude vanishes (which can only be at vertices).

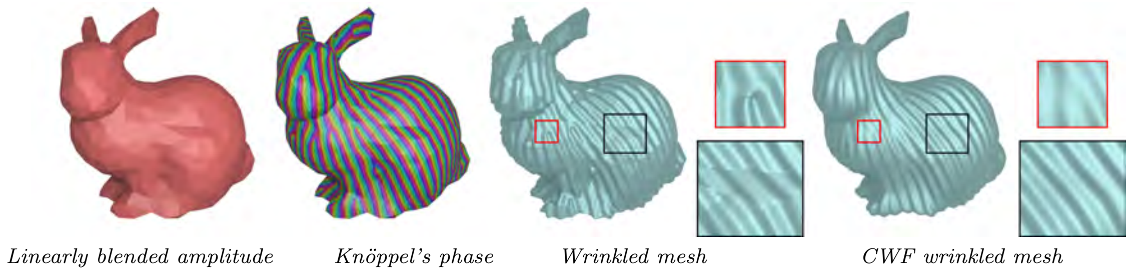


Figure 3.9: Knöppel et al. (2015)'s algorithm is designed only for stripe patterns (phase fields), not wrinkles with amplitude. Naively combining the phase field produced by Knöppel et al's method with linearly-interpolated amplitude (through barycentric blending from triangle corners) yields artifacts near wrinkle singularities (the red zoomed-in region). In this case, the coarse mesh has a constant amplitude on each vertex. Moreover, without our *CWF* subdivision algorithm, meshing artifacts are apparent in the rendered result (the black zoomed-in region).

Chen et al. (2021b) a.k.a. Chapter 2 In our *TFW* model, we also describe a method for rendering high-resolution wrinkled surfaces given frequency and amplitude fields on the surface (Appendix A.6.5). To compare against this work, we again use Equations ((3.12)), Equation (B.34) (Appendix B.1.2) to compute $a(t)$ and $\omega(t)$ and apply *TFW* to each frame of the interpolation. See Figure 3.10 and the **Comparisons** supplemental video for the results. Unlike Knöppel et al. (2015), *TFW* model does consider wrinkle amplitude, and so can produce smooth wrinkled surface geometry. However, their approach again suffers from temporal incoherence. Moreover, we observe their method also can fail to capture the sliding of singularities over the surface, instead producing degenerate, noisy phase fields; see e.g., Figure 3.10 around $t = 0.5$.

Keyframes from Physical Simulation Finally, we compare the behavior of our interpolant to the results of physically simulated wrinkle evolution. We start by simulating the twisting of a 1-meter-high (diameter 0.25 meters, thickness 0.318 mm, Young's Modulus 0.821 MPa, density 472.6 kg/m^3, and Poisson's ratio 0.243), 88k-vertex cylinder mesh, by 80 degrees with C-IPC's Li et al. (2021) open-source implementation of shell finite elements. The cylinder's top and bottom boundaries are pinned to rotate (time-stepped at 0.04 seconds/frame) in opposite directions at 10 degrees/s without gravity or external forces. To avoid simulating the transient dynamics before wrinkles appear, we initialize the cylinder slightly twisted (by 8 degrees in both direction) and run the simulation for 80 frames (so that cylinder ends are twisted by 40 degrees in both directions at the last frame). During this motion, wrinkles rotate and merge; see Figure 3.11, top, and the **Comparisons** supplemental video.

To compare these simulated results with corresponding intermediate frames computed using *CWF* interpolation, we take the first and last frame of the simulation as interpolation inputs. Specifically, we decimate the cylinder mesh to 688 vertices and then apply a tension-field-theory-based static solver Skouras et al. (2014), implemented by Chen et al. (2021b), to compute a wrinkle-free base mesh for every frame. We then manually set the frequency at the two endpoint keyframes by counting the number of wrinkles in the corresponding frames of the C-IPC simulation, and scale the wrinkle amplitude to preserve surface area of the cylinder. In Figure 3.11, bottom, we see that the *CWF* interpolated frames then well-align with the input keyframes and, in-between, obtain qualitatively similar merging and deformation to the simulation results. That said, these interpolated frames do have expected and notable differences from the simulated results: deformation paths differ (e.g., with wrinkles merging at different times) and, of course, the *CWF* model (as it considers only a single primary frequency) does not capture the secondary wrinkles observable in the simulation results.

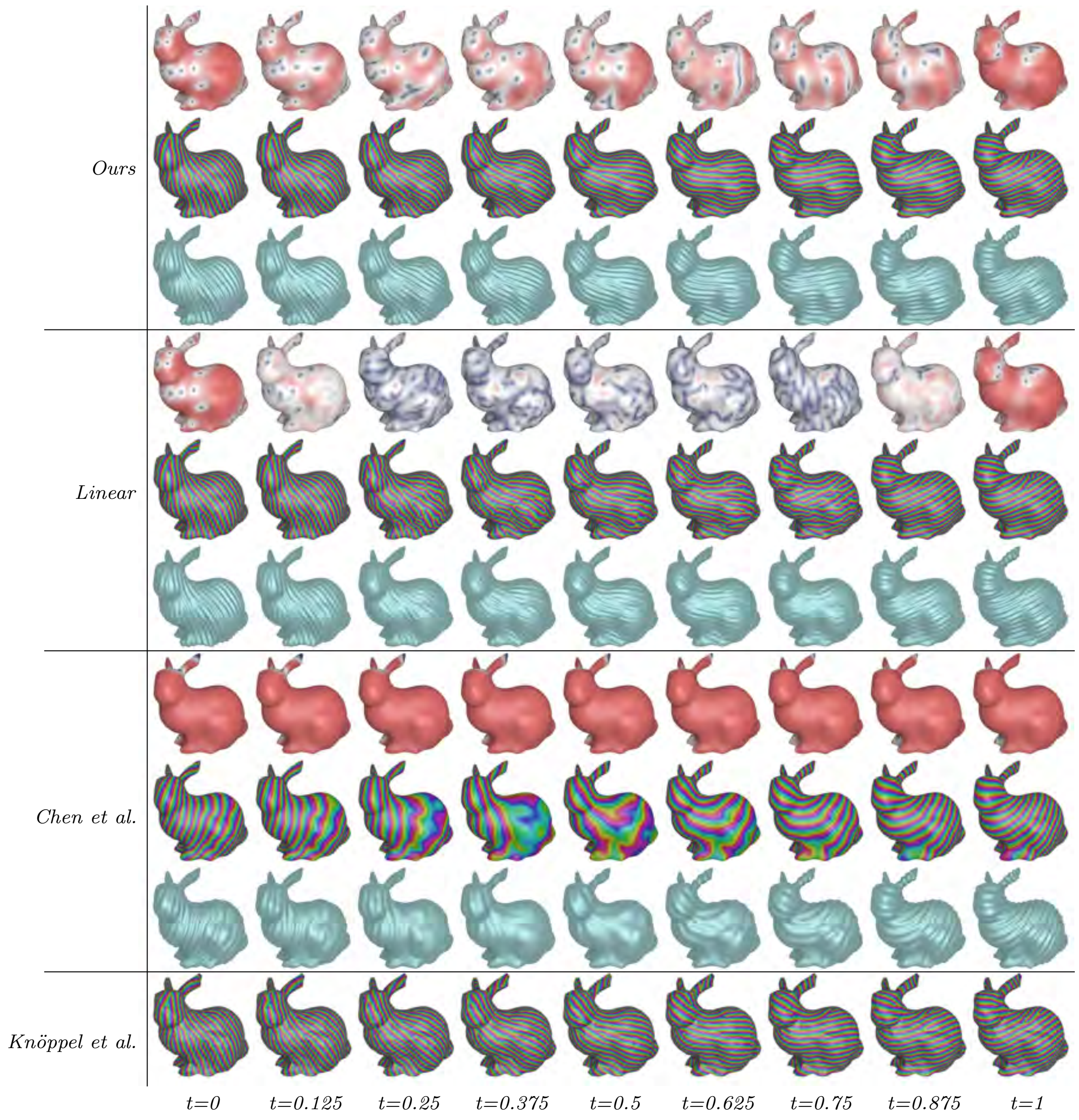


Figure 3.10: We compare our interpolation algorithm to several baselines on an example of wrinkles rotating by ninety degrees on the Stanford bunny. Linear interpolation leaves the space of *CWF*s and consequently produces severe artifacts. Both Knöppel et al. (2015) and Chen et al. (2021b) produce temporally incoherent results, though these discontinuities are hard to see in static frames and much more obvious in the **main** supplementary video (at 03:44–04:04) and the **Comparisons** supplementary video (at 01:00–01:56). Additionally, Chen et al.'s phase field becomes noisy midway through the interpolation.

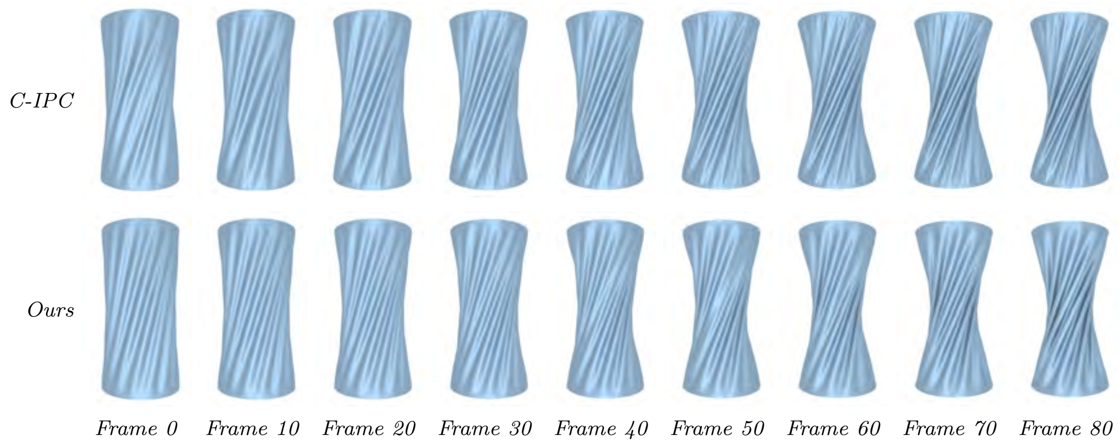


Figure 3.11: We compare *CWF* interpolation to the results of physically simulated wrinkle evolution. We simulate the twisting of a cylinder by 64 degrees (from 16 to 80 degrees) with *C-IPC* Li et al. (2021) *(top)* and compare its results with *CWF* interpolation using the the first and last simulated frame (*bottom*) as boundary conditions. *CWF* smoothly interpolates the given keyframes with qualitatively-similar wrinkle merging and deformation (though it does not capture secondary wrinkles that emerge during simulation). See the **Comparisons** supplemental video (at 02:57–03:13) for an animated comparison.

3.5 Wrinkle Upsampling

Representing fine wrinkle patterns on coarse meshes is only useful if there is a way to actually see them. Given a triangle mesh $\mathcal{T}$ and a *CWF* on that mesh, rendering the wrinkled surface requires both upsampling $\mathcal{T}$ itself (yielding a smooth canvas on which to place wrinkles, free of meshing artifacts) and turning the *CWF* into a displacement map that can be applied to the upsampled mesh.

To that end, we propose the following upsampling strategy:

- For upsampling $\mathcal{T}$, we focus exclusively on Loop subdivision Loop (1987), as it is the most commonly-used triangle mesh subdivision scheme.
- In the remainder of this section, we will describe a rule for mapping per-vertex complex numbers $\boldsymbol{z}$ and per-edge frequencies ω from $\mathcal{T}^k$, the input mesh after k levels of Loop refinement, to $\mathcal{T}^{k+1}$. The quantities on $\mathcal{T}^0$ can be read directly

from the *CWF* ($\boldsymbol{z}_i = a_i \tilde{\boldsymbol{z}}_i$ on the vertices), and then upsampled to any desired level of refinement.

- To materialize the wrinkled mesh at subdivision level k, we apply the displacement $|\boldsymbol{z}_i| \hat{\boldsymbol{n}}_i \cos(\arg \boldsymbol{z}_i)$ to vertex $\mathbf{v}_i$, where $\hat{\boldsymbol{n}}_i$ is the vertex normal.

We use this approach to render all smooth wrinkled surfaces in this chapter (we apply 4 or 5 levels of subdivision, depending on the frequency of wrinkles in each example). We have not analyzed this process to prove that the limit surfaces belong to some smoothness class, though in practice all of our results are visually smooth, including near singularities. Although the above procedure is designed to render *CWF*s, it can also be used to visualize $\boldsymbol{z}, \omega$ pairs that do not obey approximate compatibility. We have observed that the resulting upsampled wrinkles are also smooth, although they include extraneous singularities not reflected in $\boldsymbol{z}$ or ω (see Figure 3.4d).

Preliminaries We now describe the upsampling maps $\boldsymbol{z}^k \to \boldsymbol{z}^{k+1}$ and $\omega^k \to \omega^{k+1}$. To avoid cumbersome superscripts we will write $\mathcal{T}$ for $\mathcal{T}^k$ and $\mathcal{T}'$ for $\mathcal{T}^{k+1}$, and similarly use primes to denote quantities on $\mathcal{T}^{k+1}$. We write $\mathbf{v}$ for the vertices of $\mathcal{T}$ and S_0 for the Loop subdivision mask, so that $\mathbf{v}'_j = [S_0 \mathbf{v}]_j$.

3.5.1 Upsampling ω

Research in geometry processing Wang et al. (2006); de Goes et al. (2016a) has established a subdivision mask S_1 on one-forms that is compatible with the Loop subdivision mask and the discrete differential operator, in the sense that for any function f on $\mathcal{T}$, $S_1 \mathrm{d} f = \mathrm{d} S_0 f$. On closed meshes, we use this operator directly, and set $\omega' = S_1 \omega$. On meshes with boundary, we modified the standard Loop formula and de Goes et al.'s formula slightly to keep boundary corners sharp (see Appendix B.2 for details).

3.5.2 Upsampling z

There are two obvious ideas for how to upsample the $\boldsymbol{z}$ values: either by subdividing $\boldsymbol{z}$'s real and imaginary parts separately, or its magnitude and phase. Neither approach works, for essentially the same reasons discussed in Section 3.3 as motivating the use of frequencies in addition to phase in the *CWF* representation: in regions where the frequency is high, phase values at vertices undersample the wrinkles. Computing $\boldsymbol{z}'$ by averaging these aliased samples yields an equally-aliased $\boldsymbol{z}'$. We show a comparison between our approach (described below) and these two naive baselines in Figure 3.12.

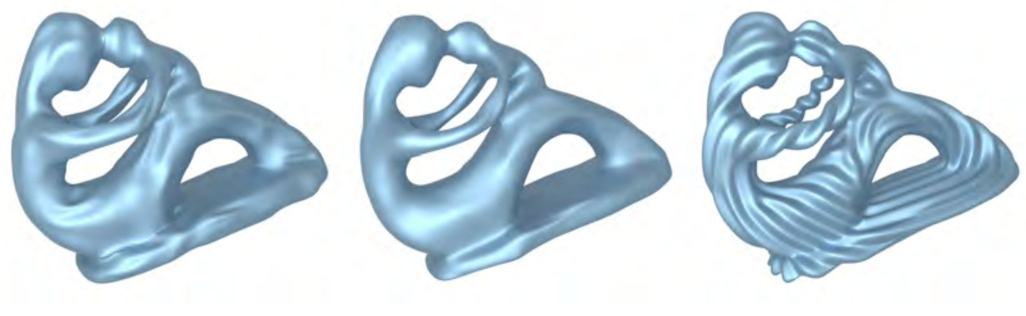

Subdivision of $|\boldsymbol{z}|$, $\arg \boldsymbol{z}$
Subdivision of $\Re(\boldsymbol{z})$, $\Im(\boldsymbol{z})$
Our upsampling

Figure 3.12: A comparison of our $\boldsymbol{z} \to \boldsymbol{z}'$ upsampling algorithm (right) to naively applying the Loop subdivision mask to $\boldsymbol{z}$'s amplitude and phase (left), and to $\boldsymbol{z}$'s real and imaginary parts (middle). Neither baseline preserves the frequency of the wrinkles in the original *CWF*.

3.5.2.1 Our Approach

To upsample $\boldsymbol{z}$ without aliasing the wrinkles, we need to incorporate frequency information from ω into the upsampling procedure. We build up to the final formulas for computing $\boldsymbol{z}'$ on the vertices of $\mathcal{T}'$ by first examining some simpler special cases. We will use the following assumption as our guiding principle for deriving our subdivision rules: in a local patch of the surface, the *phase* of $\boldsymbol{z}$ might change quickly, but the *frequency* does not. This assumption allows to extrapolate and interpolate frequency values away from the edges on which they are defined.

Upsampling on a Line Segment Let us first consider the case of a line segment with vertices $\mathbf{v}_0$ and $\mathbf{v}_1$, whose $\boldsymbol{z}$-values are $\boldsymbol{z}_0$ and $\boldsymbol{z}_1$, and whose edge frequency is ω. Suppose we need to estimate $\boldsymbol{z}$ at a point $\boldsymbol{P} = (1-\alpha)\mathbf{v}_0 + \alpha\mathbf{v}_1$. Since we assume frequency changes slowly, we can extrapolate $\boldsymbol{z}$ at $\boldsymbol{P}$ from $\boldsymbol{z}_0$ by $\boldsymbol{z}(\boldsymbol{P}) \approx \boldsymbol{z}_0 \exp(i\alpha\omega)$; similarly from $\boldsymbol{z}_1$ by $\boldsymbol{z}_1 \exp(i(1-\alpha)\omega)$. These formula do not necessarily agree unless $|\boldsymbol{z}_0| = |\boldsymbol{z}_1|$ and ω and $\boldsymbol{z}$ are exactly compatible. We reconcile the two votes by barycentrically blending them:

$$\boldsymbol{z}(\boldsymbol{P}) = (1-\alpha)\boldsymbol{z}_0 \exp(i\alpha\omega) + \alpha\boldsymbol{z}_1 \exp(i(1-\alpha)\omega). \tag{3.18}$$

Notice that $\boldsymbol{z}(\boldsymbol{P})$ interpolates $\boldsymbol{z}_0$ and $\boldsymbol{z}_1$.

Upsampling in a Triangle We use a similar strategy for evaluating $\boldsymbol{z}$ at any point $\boldsymbol{P} = \sum_{j=0}^{3} \alpha_j \mathbf{v}_j$ within a triangle $\{\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2\}$, whose corresponding $\boldsymbol{z}$ values are $\boldsymbol{z}_0$, $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$, and whose edges have frequencies ω_{01}, ω_{12}, and ω_{20}. Similar to the line segment case, we can estimate $\boldsymbol{z}(\boldsymbol{P})$ by extrapolating from a triangle corner $\mathbf{v}_j$ by $\boldsymbol{z}(\boldsymbol{P}) \approx \boldsymbol{z}_j \exp(i\omega_{j\to\boldsymbol{P}})$, where

$$\omega_{j\to\boldsymbol{P}} := \alpha_{j+1}\omega_{j(j+1)} + \alpha_{j+2}\omega_{j(j+2)}$$

is the frequency on the segment $\mathbf{v}_j\boldsymbol{P}$, as measured at vertex $\mathbf{v}_j$ using $\omega_{j(j+1)}$ and $\omega_{j(j+2)}$ as a basis for evaluating ω in any direction. Barycentrically blending this formula from three corners gives us:

$$\boldsymbol{z}(\boldsymbol{P}) = \sum_{j=1}^{3} \alpha_j \boldsymbol{z}_j \exp\left[i\left(\alpha_{j+1}\omega_{j(j+1)} + \alpha_{j+2}\omega_{j(j+2)}\right)\right]. \tag{3.19}$$

This triangle interpolant is similar to the one proposed by Evgeny and Harders (2019) with one key difference: they formulate Equation (3.19) purely for phase and propose barycentrically interpolating amplitude separately. The Zuenko and

Harders approach is unsuitable for triangles containing singularities, since upsampled amplitude does not neccessarily vanish at the singularites, creating visual artifacts (Figure 3.16).

3.5.2.2 Loop Subdivision Rules for $\boldsymbol{z}$

We now generalize Equation (3.19) to compute $\boldsymbol{z}'_i$ on the Loop-upsampled vertex $\mathbf{v}'_i$ of $\mathcal{T}'$. We must consider two cases: $\mathbf{v}'_i$ might be a repositioned vertex already present in $\mathcal{T}$ (an *even* vertex), or an entirely new vertex that came from splitting an edge of $\mathcal{T}$ (an *odd* vertex). We separately describe how to compute $\boldsymbol{z}'$ in each case; see Figure 3.13 for a visual summary of all the rules.

Odd Rule For an odd vertex, the standard Loop mask is

$$\mathbf{v}_j^{'} = \frac{1}{8}(\mathbf{v}_0 + \mathbf{v}_1) + \frac{3}{8}(\mathbf{v}_2 + \mathbf{v}_3)$$

on a "diamond" with common edge $\mathbf{v}_2\mathbf{v}_3$. This mask can be rewitten

$$\begin{aligned} \mathbf{v}_j^{'} &= \frac{1}{2}\boldsymbol{P}_0 + \frac{1}{2}\boldsymbol{P}_1 \\ \boldsymbol{P}_0 &= \frac{1}{4}\mathbf{v}_0 + \frac{3}{8}\mathbf{v}_2 + \frac{3}{8}\mathbf{v}_3 \\ \boldsymbol{P}_1 &= \frac{1}{4}\mathbf{v}_1 + \frac{3}{8}\mathbf{v}_2 + \frac{3}{8}\mathbf{v}_3. \end{aligned} \tag{3.20}$$

Notice that these formula combine barycentric sampling within a triangle (to compute $\boldsymbol{P}_0$ and $\boldsymbol{P}_1$) followed by sampling the midpoint of a line segment (to compute $\mathbf{v}'_j$.) We can convert this interpolation into two single triangle interpolations at $\boldsymbol{P}_0$, w.r.t. triangle $\{\mathbf{v}_0, \mathbf{v}_2, \mathbf{v}_3\}$ and $\boldsymbol{P}_1$, w.r.t. triangle $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$, together with line segment interpolation at $\mathbf{v}'_j$, w.r.t. $\{\boldsymbol{P}_0,\ \boldsymbol{P}_1\}$. We can transform these rules for computing $\mathbf{v}'_j$ into rules for computing $\boldsymbol{z}'_j$ by applying the formulas from the special cases discussed above: we use Equation (3.19) to compute $\boldsymbol{z}$ at $\boldsymbol{P}_0$ and $\boldsymbol{P}_1$. Next, we would like to use Equation (3.18) to compute $\boldsymbol{z}'_j$. There are two obstacles: first, we need the frequency evaluated on the line segment $\boldsymbol{P}_0\boldsymbol{P}_1$. Second, this line segment generally does not lie on $\mathcal{T}$, but rather cuts through ambient space, whereas frequencies are intrinsic to $\mathcal{T}$'s cotangent space.

We therefore modify the approach of Equation (3.18) to account for these differences. First, we sample ω at $\boldsymbol{P}_0$ by interpolating the values of ω at the edges of triangle $\mathbf{v}_0\mathbf{v}_2\mathbf{v}_3$ into the interior (see Appendix B.3 for more details; in particular note that $\omega(P_0)$ is a cotangent vector and not a scalar). We use this frequency to extrapolate $\boldsymbol{z}(P_0)$ to $\mathbf{v}'_j$ along the *intrinsic* edge from $\boldsymbol{P}_0$ to $\boldsymbol{P}_1$:

$$\boldsymbol{z}_{\boldsymbol{P}_0\to\mathbf{v}'_j} = \boldsymbol{z}(\boldsymbol{P}_0)\exp\left(i\omega(P_0)(\boldsymbol{P}_1^* - \boldsymbol{P}_0)/2\right), \tag{3.21}$$

where $\boldsymbol{P}_1^*$ is the location of $\boldsymbol{P}_1$ after rigidly unfolding triangle $\mathbf{v}_1\mathbf{v}_2\mathbf{v}_3$ along the common edge $\mathbf{v}_2\mathbf{v}_3$ to lie in triangle $\mathbf{v}_0\mathbf{v}_2\mathbf{v}_3$'s tangent plane. We compute $\boldsymbol{z}_{\boldsymbol{P}_1\to\mathbf{v}'_j}$ analogously.

Then the final $\boldsymbol{z}'_j$ is the average of the corresponding contributions from $\boldsymbol{P}_0$ and $\boldsymbol{P}_1$:

$$\boldsymbol{z}'_j = \frac{1}{2}(\boldsymbol{z}_{\boldsymbol{P}_0\to\mathbf{v}'_j} + \boldsymbol{z}_{\boldsymbol{P}_1\to\mathbf{v}'_j}). \tag{3.22}$$

Even Rule Next, we consider an even vertex $\mathbf{v}'_0$ corresponding to $\mathbf{v}_0$ on $\mathcal{T}$, with neighbors $\mathbf{v}_1, \dots, \mathbf{v}_n$. $\mathbf{v}'_0$ has position

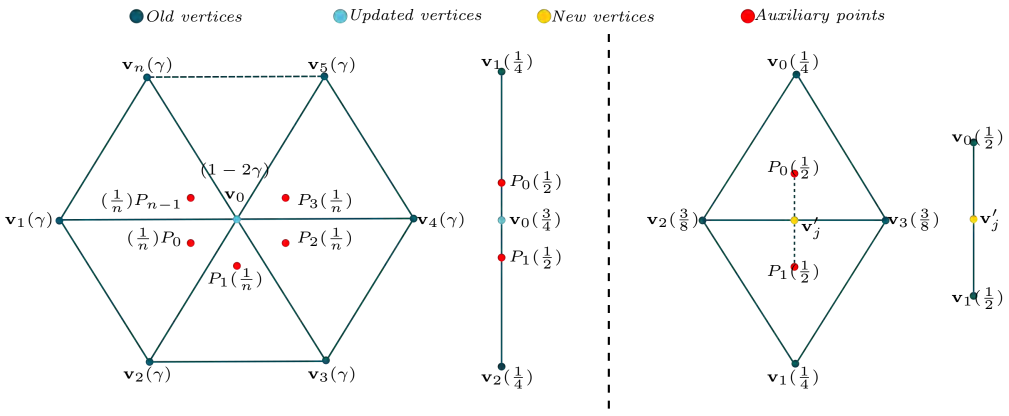
$$\mathbf{v}'_0 = (1-\alpha n)\mathbf{v}_0 + \sum_{j=1}^{n} \alpha \mathbf{v}_j, \tag{3.23}$$

where α is the traditional even-neighbor Loop weight and depends on the valence n of $\mathbf{v}_0$ Loop (1987). As in the odd case, we can refactor Equation (3.23) as a convex combination of points linearly interpolated within a single triangle:

$$\mathbf{v}'_0 = \sum_{j=0}^{n-1} \frac{1}{n}\boldsymbol{P}_j, \quad \boldsymbol{P}_j = (1-2\gamma)\mathbf{v}_0 + \gamma\mathbf{v}_{j+1} + \gamma\mathbf{v}_{j+2}, \tag{3.24}$$

for $\gamma = \frac{n\alpha}{2}$. We now follow the same recipe as in the odd case: we compute $\boldsymbol{z}(\boldsymbol{P}_j)$ using Equation (3.19), and write $\boldsymbol{z}'_0$ as the average of contributions extrapolated from each $\boldsymbol{P}_j$.

Mesh Boundaries For even vertices $\mathbf{v}_0$ on the mesh boundary with neighbors $\mathbf{v}_1$ and $\mathbf{v}_2$, we can once again write the Loop subdivision mask in terms of an average of linear interpolations along mesh edges, and apply the same recipe as in the even and odd interior cases above. In this case $\mathbf{v}'_0 = (\boldsymbol{P}_0 + \boldsymbol{P}_1)/2$, where $\boldsymbol{P}_0 = \frac{1}{4}\mathbf{v}_1 + \frac{3}{4}\mathbf{v}_0$ and $\boldsymbol{P}_1 = \frac{1}{4}\mathbf{v}_2 + \frac{3}{4}\mathbf{v}_0$. The odd boundary case involves sampling $\boldsymbol{z}$ on a mesh edge and so Equation (3.18) can be applied directly.



(a) Even rules *(b) Odd rules*

Figure 3.13: $\boldsymbol{z}$ subdivision rules for even and odd vertices. We show the rules for both interior and boundary vertices. First, sample ω and $\boldsymbol{z}$ (using Equation (3.19)) at the red points $\boldsymbol{P}_i$, using the barycentric weights specified in parentheses next to the vertices. Finally, extrapolate $\boldsymbol{z}(\boldsymbol{P}_i)$ to the upsampled vertex (blue or yellow) using an analogous approach to Equation (3.21), and average these extrapolations using the weights in parentheses next to the $\boldsymbol{P}_i$.

Remark Notice that the upsampling strategy explained above has the following useful property: it exactly reproduces plane waves, in the sense that if $\mathcal{T}$ is a piece of the plane, ω is the differential of a linear function $\theta(\boldsymbol{P}) = \boldsymbol{v} \cdot \boldsymbol{P}$, and $\boldsymbol{z}(= \tilde{\boldsymbol{z}})$ is consistent with ω, then under refinement the *CWF* $(\omega, 1, \tilde{\boldsymbol{z}})$ converges to (a phase shift of) the smooth plane wave $\psi(\boldsymbol{P}) = \cos(\boldsymbol{v} \cdot \boldsymbol{P})$.

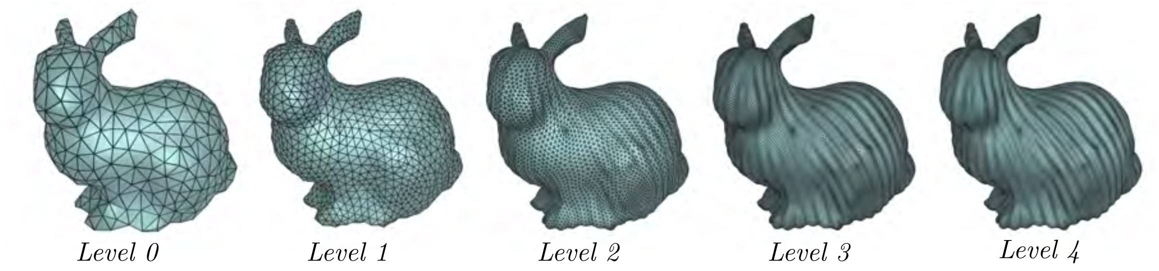


Figure 3.14: We successively Loop-upsample the Stanford bunny, alongside the $\boldsymbol{z}$ and ω of a CWF on the bunny, and render the resulting displaced surface. After several rounds of subdivision, the wrinkle pattern appears and is stable under additional iterations of subdivision.

3.5.3 Results and Comparisons

We show successive upsampling of a CWF on the Stanford bunny in Figure 3.14. Wrinkles appear as soon as the subdivided mesh has high enough resolution to resolve their frequency. See all other figures of rendered wrinkles in this chapter for more examples of our subdivision algorithm at work.

Triangle Interpolation Alternatives Several alternative formulas have been proposed for interpolating phase into a triangle from samples at its corners; in Figure 3.15 we compare our choice of Equation (3.19) to several potential alternatives: the side-vertex scheme used in Jeschke and Wojtan (2015)'s work, and the nine-parameter Clough-Tocher cubic scheme Farin (1986). Notice that away from the singularities, all of these three approaches achieve reasonable results; however the side-vertex and Clough-Tocher cubic scheme fail to resolve neighborhoods of singularities. The reason for this failure is that both schemes express phase at interior points in terms of a rational or polynomial function of phase at the triangle vertices. It is mathematically impossible for such functions to produce singularities (where phase must have a branch point where it's undefined).

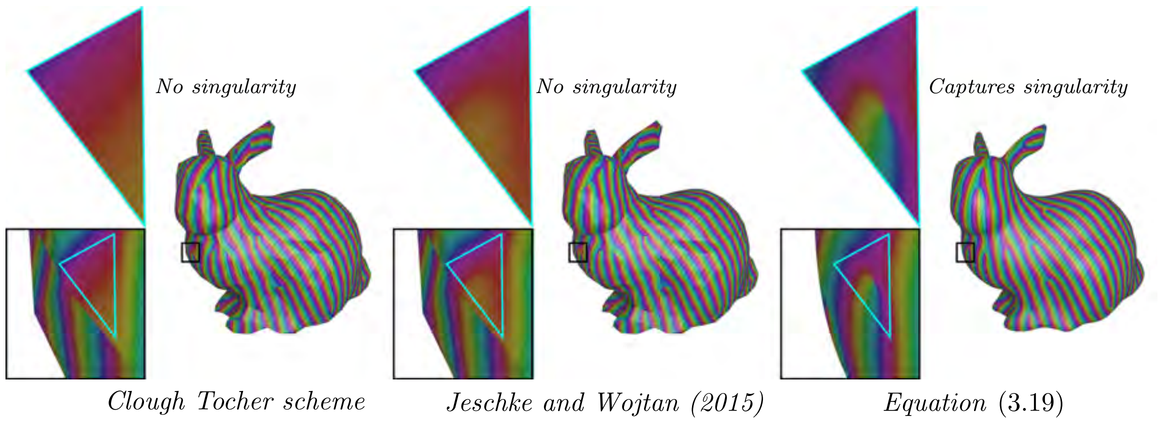


Figure 3.15: A comparison where we replace Equation (18) in our upsampling algorithm with some alternatives, for a *CWF* on the Stanford bunny. From left to right: the nine-parameter Clough Tocher cubic scheme Farin (1986), the side-vertex scheme used by Jeschke and Wojtan (2015), and Equation (3.19). Only Equation (3.19) successfully resolves the phase behavior near singularities (zoomed-in regions).

Other Upsampling Methods In Figure 3.16, we compare our upsampling algorithm against two baselines from the literature Evgeny and Harders (2019); Chen et al. (2021b). Zuenko and Harders render fine wrinkles on coarse meshes by first linearly subdividing the mesh, interpolating phase onto the fine mesh using a variant of Equation (3.19) and amplitude using barycentric interpolation, and then smoothing the result with SPHERIGON Volino and Thalmann (1998) as a post-process to remove some of the artifacts from the wrinkled mesh geometry. Since they separately upsample wrinkle amplitude and phase, there is no guarantee that amplitude vanishes at phase singularities, leading to noticable noise (in the red and black zoomed-in regions, for instance). Moreover the use of linear subdivision on $\mathcal{T}$ means that the edges of the coarse mesh remain noticeable in the final rendered image, despite the post-processing.

Like in our approach, Chen et al. (2021b) use Loop subdivision to upsample $\mathcal{T}$, as well as the wrinkle amplitude and phase, but their method requires that the input frequency and phase are exactly compatible. This restriction makes their approach unsuitable for rendering *CWF*s. In Figure 3.16 we make a best-effort comparison by projecting ω to the closest globally-integrable frequency using mixed-integer programming, but doing so severely distorts the wrinkle pattern, and many singularities are missing from the final rendered image.

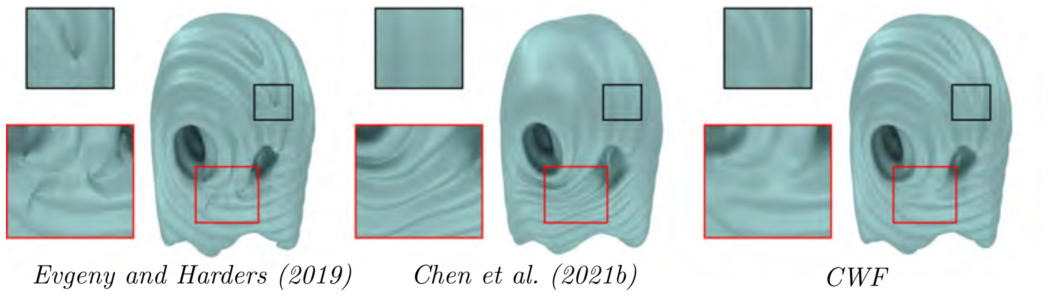


Figure 3.16: We compare our upsampling algorithm against two baselines from previous work Evgeny and Harders (2019); Chen et al. (2021b). Zuenko and Harders perform linear subdivision of $\mathcal{T}$ and separately upsample amplitude and phase in a way that produces noticeable meshing artifacts and discontinuities near the singularities (see zoomed-in regions). Chen et al. (2021b)'s upsampling scheme only works for curl-free frequency fields. To use their method, we project ω onto the space of globally integrable one-forms, but doing so distorts the wrinkle pattern, which is missing many singularities.

3.6 Wrinkle Design

In this section, we demonstrate applications of our *CWF* algorithms for user-based design and editing of wrinkled surfaces via interpolation. We first develop tools for the creation and editing of *CWF* keyframes and then show their application. Please also see our **main** and **Interpolation Results** videos for more details and results.

3.6.1 Adding Wrinkles to Surfaces

For adding wrinkles to a surface a user first provides a mesh $\mathcal{T} = \{\mathbf{v}, \mathbf{e}, \mathbf{f}\}$ and selects k vertices as *source points* s (green points in Figure 3.17a) and at each of those points a desired wrinkle frequency vector $\boldsymbol{v}_s$ (the yellow arrow). A user then

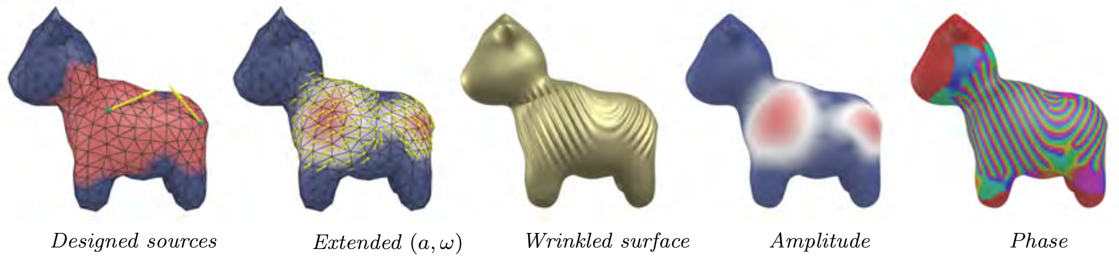


Figure 3.17: Adding wrinkles to the Spot cow. From left to right: The user specifies desired wrinkle directions (yellow vectors) and regions of influence for each chosen direction (red regions). The user also assigns an amplitude to each region. We extend a and ω from the user input to the entire mesh. Finally, we solve for $\tilde{\boldsymbol{z}}$ and apply our upsampling algorithm to materialize high-resolution wrinkled geometry. We show the amplitude and phase of the final wrinkle pattern, after upsampling, on the right.

also specifies a region of influence $\text{in}_s \subset \mathbf{v}$ for each source point (the white and red regions in Figure 3.17a)[4] and a target amplitude a_s for the wrinkles within the region of influence.

Let S be the set of all source points $\{s_i\}_{i=1}^{k}$. We call the set of all vertices in any region of influence the *influenced vertices* $\text{in}_V = \bigcup \text{in}_{s_i}$. The influenced faces in_F are those with at least one influenced vertex, and likewise for the influenced edges in_E.

From this input we solve for a wrinkle pattern on $\mathcal{T}$. The process can be divided into two parts: (1) input extension, where we extend the user-provided amplitude and frequency samples to an a and ω on the entire mesh, and (2) wrinkle synthesis, where we compute $\tilde{\boldsymbol{z}}$ from a and ω and use the upsampling technique introduced in Section 3.5 to export the final wrinkled geometry.

We first apply the vector heat method Sharp et al. (2019) to extend the user-provided frequencies $\boldsymbol{v}_{s_i}$ to a vertex-based vector field $\boldsymbol{v}_i$ on $\mathbf{v}$. Then for any influenced

[4]For simplicity we compute the region of influence as the n-ring neighborhood of the chosen points, for a user-specified n; one could also compute a geodesic disk.

edge $\mathbf{e}_{ij} \in \text{in}_E$ we set

$$\omega_{ij} = \frac{1}{2}(\boldsymbol{v}_i \cdot \mathbf{e}_{ij} + \boldsymbol{v}_j \cdot \mathbf{e}_{ij}),$$

and for any edge that is not influenced, we set $\omega_{ij} = 0$.

We compute per-vertex amplitudes by solving

$$\arg\min_a E(a) \quad \text{s.t.} \quad \begin{cases} a_i = 0, & \mathbf{v}_i \notin \text{in}_V \\ a_{s_i} = c_{s_i}, & \forall s_i \in \boldsymbol{S} \end{cases} \tag{3.25}$$

where

$$E(a) = \frac{1}{2}a^T L a + \sum_{i=1}^{k} \sum_{\mathbf{v}_j \in \text{in}_{s_i}} (a_j - a_{s_i})^2 A_j + \sum_{F_{\ell mn} \in \text{in}_F} \frac{a_\ell^2 + a_m^2 + a_n^2}{3} (\omega_{\ell m} + \omega_{mn} + \omega_{n\ell})^2$$

and L is the positive semi-definite cotangent Laplacian matrix, A_j is the vertex barycentric area, and $\omega_{\ell m} + \omega_{mn} + \omega_{n\ell}$ is the discrete curl. The first term promotes smoothness of a; the second term measures agreement with the user-specified amplitudes in the affected regions, and the final term couples the a to the ω to ensure that amplitude goes to 0 at the singularities of ω. Finally, for the wrinkle synthesis step, we solve the eigenvalue problem proposed by Knöppel et al. (2015) to determine $\tilde{\boldsymbol{z}}$ and then directly apply the subdivision algorithm from Section 3.5 to $(\omega, a, \tilde{\boldsymbol{z}})$.

Remarks The procedure above is used to create local wrinkle patterns. If global patterns are desired instead (such as the wrinkle patterns on the bunny in Figure 3.7), we first compute ω using Knöppel et al. (2013)'s algorithm, and globally scale the frequency and set a global constant amplitude to achieve the desired wrinkle characteristics. Then we find the consistent $\tilde{\boldsymbol{z}}$ values on the vertices and upsample the result using the wrinkle field synthesis algorithm described above.

3.6.2 Editing Wrinkles

Next, we describe a tool for editing wrinkles, given an existing mesh $\mathcal{T}$ and CWF $(\tilde{\boldsymbol{z}}^0, a^0, \omega^0)$. There are four steps in the editing pipeline:

- The user picks a region of the mesh to edit (red in Figure 3.7).
- The user edits the wrinkles in that region. Options we implemented are: rotating the frequency, changing the frequency magnitude, and changing the wrinkle amplitude.
- We blend the new wrinkle pattern in the edited region with the existing one inside an *interface* region around the chosen region (green region in Figure 3.7). Wrinkles away from the edited region and its interface do not change.
- We interpolate between the original *CWF* and the edited one, using our interpolation algorithm.

The third step requires some elaboration. Let $\mathbf{f}_{\text{edit}} \subseteq \mathbf{f}$ be the set of faces where the user has chosen to make edits, and let ω^1 and a^1 be the requested frequency and amplitude in that region. We need to construct a new $\tilde{\boldsymbol{z}}^1$, which satisfies (1) $\tilde{\boldsymbol{z}}^1 = \tilde{\boldsymbol{z}}^0$ far away from $\mathbf{f}_{\text{edit}}$; and (2) $\tilde{\boldsymbol{z}}^1$ is consistent with ω^1 within $\mathbf{f}_{\text{edit}}$. To avoid a discontinuity in frequency and amplitude on the edited surface, we create an *interface* region $\mathbf{f}_{\text{inter}}$ of user-specified size around $\mathbf{f}_{\text{edit}}$. The remaining mesh faces will not change; we call them $\mathbf{f}_{\text{fixed}}$.

Let $\mathbf{e}_{\text{edit}}$ be the edges of the triangles in $\mathbf{f}_{\text{edit}}$, and $\mathbf{e}_{\text{fixed}}$ and $\mathbf{v}_{\text{fixed}}$ the edges and vertices of $\mathbf{f}_{\text{fixed}}$, respectively. The above requirements can be written as follows:

- $\tilde{\boldsymbol{z}}^1_j = \tilde{\boldsymbol{z}}^0_j$, for any $\mathbf{v}_j \in \mathbf{v}_{\text{fixed}}$.
- $|\tilde{\boldsymbol{z}}^1_j| = 1$, for any $\mathbf{v}_j \in \mathbf{v} \setminus \mathbf{v}_{\text{fixed}}$.
- $\tilde{\boldsymbol{z}}^1_k \exp(i\omega_{kj}) - \tilde{\boldsymbol{z}}^1_j = 0$ for any edge $\mathbf{e}_{jk} \in \mathbf{e}_{\text{edit}}$.
- The interface frequency and amplitude fields are smooth.

We first interpolate ω^1 into the interface region. To get a frequency field that is as smooth as possible, we minimize the Dirichlet energy of that field with Dirichlet boundary conditions:

$$\begin{aligned} &\arg\min_{\omega} \sum_{\mathbf{f}_{\ell mn}} (\omega_{\ell m} + \omega_{mn} + \omega_{n\ell})^2 + \sum_{\mathbf{v}_m} \left(\sum_{\mathbf{e}_{mn}} c_{mn}\omega_{mn} \right)^2 \\ &\text{s.t} \quad \begin{cases} \omega_{jk} = \omega_{jk}^0, & \mathbf{e}_{jk} \in \mathbf{e}_{\text{fixed}} \\ \omega_{jk} = \omega_{jk}^1, & \mathbf{e}_{jk} \in \mathbf{e}_{\text{edit}}, \end{cases} \end{aligned} \tag{3.26}$$

where c_{mn} is the cotangent weight w.r.t. edge $\mathbf{e}_{mn}$, and the first and second term are the discrete curl and divergence respectively.

Once we have the frequency field, the amplitude field is the optimal solution of the following modified Dirichlet energy:

$$\begin{aligned} &\arg\min_{a} \frac{1}{2} a^T L a + \sum_{F_{\ell mn}} \frac{a_\ell^2 + a_m^2 + a_n^2}{3} (\omega_{\ell m} + \omega_{mn} + \omega_{n\ell})^2 \\ &\text{s.t} \quad \begin{cases} a_j = a_j^0, & \mathbf{v}_j \in \mathbf{v}_{\text{fixed}} \\ a_j = a_j^1, & \mathbf{v}_j \in \mathbf{v}_{\text{edit}}, \end{cases} \end{aligned} \tag{3.27}$$

which asks for an amplitude that is as smooth as possible while ensuring (via the first term) that the amplitude vanishes at singularities. Here L is the positive semi-definite cotangent Laplacian.

Finally, we compute an approximately-consistent $\tilde{\boldsymbol{z}}$ inside the interface and edited regions by solving an optimization problem generalizing the strategy from Knöppel et al. (2015):

$$\min_{\tilde{\boldsymbol{u}}, s} E_{\text{compat}}(P\tilde{\boldsymbol{u}} + s\boldsymbol{u}^0, \omega) \quad \text{s.t.} \quad \left\| P\tilde{\boldsymbol{u}} + s\boldsymbol{u}^0 \right\|_{M_0}^2 = 1 \tag{3.28}$$

where $\tilde{\boldsymbol{u}}$ is a vector of size $|\mathbf{v}| - |\mathbf{v}_{\text{fixed}}|$ representing the unknown values of $\tilde{\boldsymbol{z}}$ in the non-fixed region, $\boldsymbol{u}_j^0 = \tilde{\boldsymbol{z}}_j^0$ for any $\mathbf{v}_j \in \mathbf{v}_{\text{fixed}}$ (with other entries zero), M_0 is the mesh barycentric mass matrix, and P is the inclusion matrix that extends $\tilde{\boldsymbol{u}}$ to the whole domain by inserting zeros for vertices in the fixed region. The single scalar value s allows for rescaling (but no other changes) within the fixed region. The inclusion of s

in the optimization is required since the more straightforward restriction of Knöppel et al. (2015)'s method to the non-fixed region,

$$\min_{\tilde{\boldsymbol{u}}} E_{\text{compat}}(P\tilde{\boldsymbol{u}} + \boldsymbol{u}^0, \omega) \quad \text{s.t.} \quad \left\|P\tilde{\boldsymbol{u}} + \boldsymbol{u}^0\right\|^2_{M_0} = 1, \tag{3.29}$$

enforces an average scaling of $\tilde{\boldsymbol{u}}$ within the non-fixed region separate from, and generally inconsistent with, the scaling of $\boldsymbol{u}^0$ in the fixed region.To transform Equation (3.28) so that it is once again an eigenvalue problem, we make the substitutions $\boldsymbol{v} = [\tilde{\boldsymbol{u}}, s]$ and $M_0' = P^T M_0 P$. From $\tilde{\boldsymbol{u}}$ we compute $\boldsymbol{u} = P\tilde{\boldsymbol{u}}/s + \boldsymbol{u}^0$ and then set $\tilde{\boldsymbol{z}}_j^1 = \boldsymbol{u}_j/|\boldsymbol{u}_j|$. Notice that $(\omega^1, a^1, \tilde{\boldsymbol{z}}^1)$ is not necessarily a valid CWF since $\tilde{\boldsymbol{z}}^1$ may not satisfy the soft compatibility condition exactly (and it is not generally possible to satisfy them without changing the $\tilde{\boldsymbol{z}}_j$ within the fixed region). We can render the wrinkle field nonetheless, and use it as a keyframe for wrinkle evolution by projecting it to a CWF using the pre-processing described in Section 3.4.

3.6.3 Results

We perform a variety of local edits to wrinkles on surfaces and visualize the wrinkle evolution during the editing process. Figures 3.6 and 3.18 illustrate local frequency increase, with a corresponding decrease in amplitude in the former example. In Figure 3.7 a user demands a local rotation of the wrinkles by 90 degrees, while maintaining the frequency magnitude and amplitude. We also support different types of edits on different patches. In Figure 3.19, we enlarge the frequency of the patch on Spot's body by 2.5 times, while at the same time rotating the direction of wrinkles on the head by 90 degrees. These examples show that we can interpolate between extreme frequency or direction differences, and the singularities nevertheless slide smoothly over the surface without temporal discontinuities or other artifacts seen in results from previous work.

Please see the **Interpolation Results** supplemental video, and Appendix B.5, for more examples of editing wrinkles, including some examples of modifying the output of wrinkles computed via physical simulation.

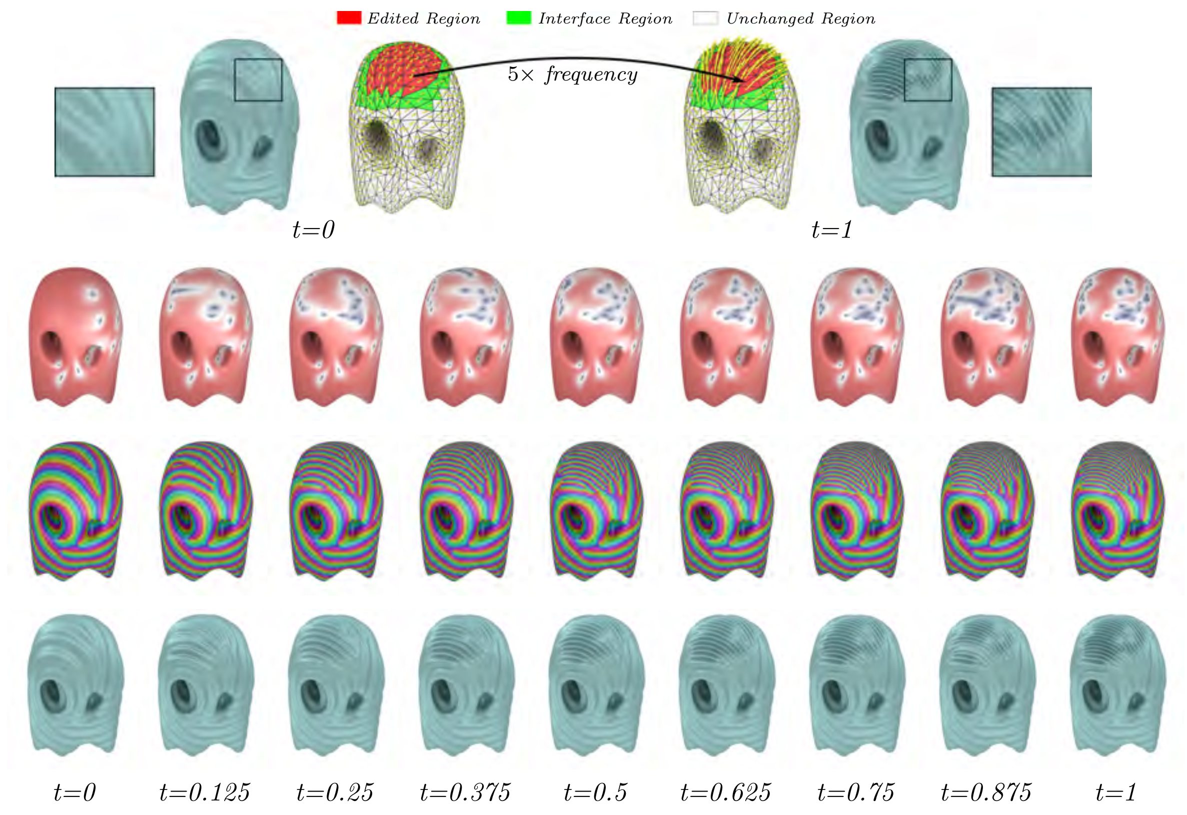


Figure 3.18: *CWF* interpolation of two wrinkle patterns on the Phantasma model, where the user has **locally** increased wrinkle frequency by 5×. Our algorithm successfully generates a smooth path between these challenging keyframes. For the corresponding wrinkle animation, please watch the **Interpolation Results** video at timestamp 00:54–01:11 in the supplementary materials.

3.7 Limitations and Future Work

In this chapter we made several design decisions that could be fruitfully revisited in future work in order to generalize *CWF*s and our interpolation or upsampling algorithms, such as restricting wrinkles to waves with a single (but spatially-varying) frequency, rather than a superposition of frequencies Rémillard and Kry (2013a); and limiting refinement to Loop subdivision. The comparison to wrinkled materials (e.g., the simulated cloth in Figure 3.11) indicates significant potential value in extending this work to model additional, secondary, higher-frequency wrinkle fields superimposed upon the first. Another particularly interesting direction of future work is to study the use of CWFs for solving PDEs involving waves other than the interpolation boundary value problem: for example, we believe that it's likely that cloth dynamics would benefit from CWF kinematics as it would allow simulation of wrinkles that have much higher frequency than the base mesh. Finally, complex wrinkle fields could be used to model skin wrinkles or fingerprints; though as argued by Evgeny and Harders (2019), high-quality skin wrinkles would likely require extending CWFs to waves with non-sinusoidal profile.

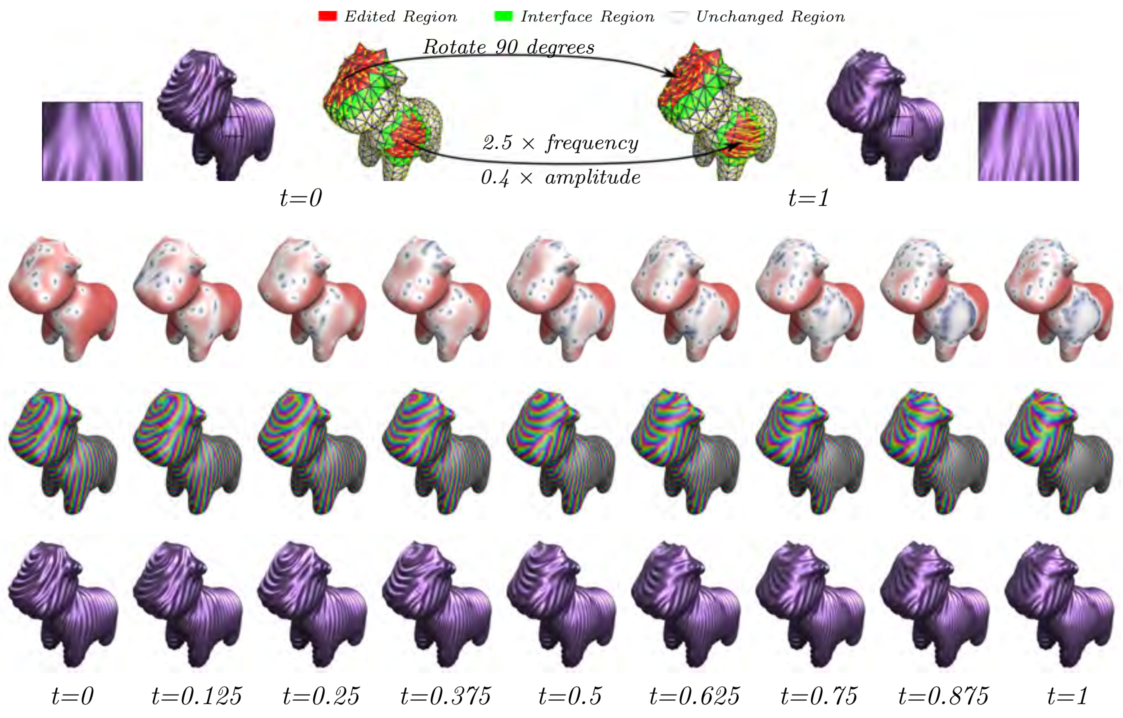


Figure 3.19: CWF interpolation of two wrinkle patterns on Spot. Please refer to the **Interpolation Results** video in the supplementary materials for the corresponding wrinkle animation (timestamp 02:21-02:37).

Interpolation Performance Interpolating CWFs is currently far from real-time (please refer the time table in the supplemental document for more details); our implementation took 4 minutes to compute keyframe interpolation on average, with 32 seconds minimum and 12 minutes maximum among all the examples. One problem is that our optimization problem is not a convex optimization: although the other two terms (E_{smooth} and E_{compat}) in Equation (3.12) are quadratic, E_{unit} is nonlinear and not convex, and this nonlinearity is enough to cause Newton's method difficulty. Performance could be improved by refactoring the optimization problem (including perhaps by improving the initial guess) or parallelizing the linear solver on the GPU. Note that the size of the linear system (in terms of number of non-zeroes) grows linearly in the number of requested frames N. To improve the efficiency, we proposed the use of guide frames (Section 3.4.1). We study the performance vs. quality tradeoff of the choice of number of guide frames in Appendix B.7.1; careful tuning of this parameter could achieve a bettter tradeoff than our (conservative) use of $N' = 50$.

Preventing Collisions When amplitudes are large, it is possible for neighboring wrinkle periods to collide, or for wrinkles to "poke through" distant portions of the base mesh. Our work currently does not detect or prevent such collisions in any way.

Alternative Incompatibility Norm When defining Opt_ω, we choose to use the $L2$ norm to measure the incompatibllity between ω and $\tilde{\boldsymbol{z}}$ (Equation (3.2)). Given that singularies are usually sparsely distributed on a wrinkled surface, a promising alternative is the $L1$ norm, which promotes such sparsity, but is more difficult to optimize.

More General Wrinkle Parameterization Several of our editing examples take as input wrinkle geometry computed by spectral wrinkle simulators Chen et al. (2021b); Evgeny and Harders (2019). In these cases we could directly convert the output of the simulators into *CWF* variables. For more general wrinkled surfaces (computed by a finite element simulator Narain et al. (2012a), for instance), a missing step is how to decompose a high-resolution, *wrinkled* mesh into the coarse but *smooth* base mesh $\mathcal{T}$ and the *CWF* variables. It's unclear how to best perform this decomposition, especially when wrinkles are complex, aliased, and contain many singularities. In addition to optimization-based approaches, another promising avenue is training a neural network to perform the decomposition.

Improved Wrinkle Physics Our interpolation is physics-inspired, but due to simplifying assumptions in the derivation such as neglecting the inertia of wrinkles and the effect of base mesh stretching on the wrinkle evolution, our interpolant is not guaranteed to exhibit fully realistic wrinkle dynamics. One possible improvement is to rederive the interpolation formula using the full reduced-order elastic energy derived by Chen et al. (2021b) instead of assuming inextensible shells. Additionally, the current model interpolates the CWF variables completely independently of any movement or deformation of the base mesh. A scheme that jointly optimizes for a coupled base mesh and CWF interpolant could result in wrinkle evolution with higher physical realism. Along similar lines, our current model does not consider inhomogeneities or special features of the base mesh, such as hems or seams, which significantly affect fabric wrinkling in the real world. A possible direction for further exploration is to extend our work by imposing spatial boundary conditions on the CWF variables at the locations of stiff seams.

Chapter 4: Robust Low-Poly Meshing for General 3D Models[1]

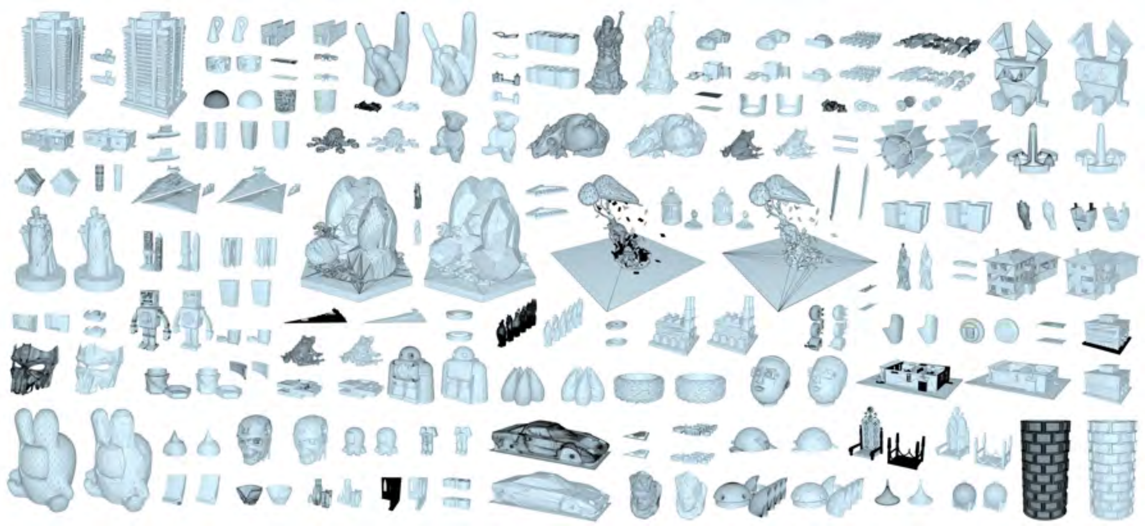


Figure 4.1: A gallery of *wild* high-poly input meshes and their corresponding low-poly outputs generated by our method, where the low-polys are manifold, watertight, and self-intersection free, and have a small visual difference from their high-poly counterparts.

We propose a robust re-meshing approach that can automatically generate visual-preserving low-poly meshes for any high-poly models found in the wild. Our method can be seamlessly integrated into current mesh-based 3D asset production pipelines. Given an input high-poly, our method proceeds in two stages: 1) Robustly extracting an offset surface mesh that is feature-preserving, and guaranteed to be watertight, manifold, and self-intersection free; 2) Progressively simplifying and flowing the offset mesh to bring it close to the input. The simplicity and the visual-preservation of the generated low-poly is controlled by a user-required target screen size of the input: decreasing the screen size reduces the element count of the low-poly

[1]This chapter is modified from Chen et al. (2023b). Please refer this webpage for details.

but enlarges its visual difference from the input. We have evaluated our method on a subset of the Thingi10K dataset that contains models created by practitioners in different domains, with varying topological and geometric complexities. Compared to state-of-the-art approaches and widely used software, our method demonstrates its superiority in terms of the element count, visual preservation, geometry, and topology guarantees of the generated low-polys.

4.1 Introduction

Mesh is a ubiquitously employed representation of 3D models for digital games. While a mesh with a large number of polygons (high-poly) is required to express visually appealing details, rendering its low-poly approximation at distant views is a typical solution to achieve real-time gaming experience, especially on low-end devices. High-polys, no matter whether they are manually created through modeling software or automatically converted from CSG and implicit functions, often have complex topology and geometries, such as numerous components, high genus, non-manifoldness, self-intersections, degenerate elements, gaps, inconsistent orientations, etc. These complexities can pose significant challenges to the design of automatic low-poly mesh generation algorithms.

Over past decades, two typical ways are developed to obtain low-poly textured models: automatic *mesh reduction* that preserves original textures Hoppe (1999); Liu et al. (2017); manual mesh modeling followed by UV-generation and texture baking that creates new textures. Mesh reduction usually removes triangles through iterative application of local operations Garland and Heckbert (1997) or element clustering Cohen-Steiner et al. (2004), which relies on existing mesh vertices and connectivity. As a result, this method is only suitable for generating the medium-levels of LOD, while introducing serious artifacts when generating low-polys for meshes with excessive topology complexity as illustrated in Figure 4.12 and Figure 4.21. In the second pipeline, while UV-generation and texture baking can be done via semi-automatic tools (e.g. Maya and Marmoset), manual meshing is the most labor(cost)-intensive step. Therefore, we address this most urgent and challenging problem, aka, low-poly meshing.

Many automatic *re-meshing* approaches exist to represent the original mesh with a proxy and simplify the proxy mesh via a row of different techniques, e.g., polygonal mesh construction by plane fitting and mixed-integer optimization Nan and Wonka (2017), cage mesh generation through voxel dilation and mesh simplification Calderon and Boubekeur (2017), shape abstraction by feature simplification Mehra et al. (2009), extremely low-poly meshing using visual-hull boolean operations Gao et al. (2022), mesh simplification through differentiable rendering Hasselgren et al. (2021), enclosing mesh generation through alpha-wrapping with an offset Portaneri et al. (2022), and learning based approaches Chen et al. (2020, 2022b). However, these re-meshing approaches either rely on heavy user interactions, need careful parameter tweaking, or work for a limited type of model. Commercial software also has low-poly mesh functions, but generates unsatisfactory results in many cases. For example, in Figure 4.11, and Table 4.3, we show the generated results using different low-poly construction modules in Simplyon AB (2022). It generates meshes with either triangle intersections, or non-satisfactory visual appearances. It remains to be a challenging task to automatically and robustly generate low-poly meshes for general 3D models used in the industry.

In practice, artists still manually craft low-poly meshes to ensure that they have a small number of triangles and preserve the visual appearance of the original mesh as much as possible. This often involves multiple iterations of manual adjustments, which incurs intensive labor work and prolonged project periods and remains to be a bottleneck for the current fast-changing game industry. Robust and automatic approaches that can generate satisfactory low-polys are in high demand.

From an input mesh with arbitrary topology and geometry properties, our goal is to generate its low-poly counterpart that is visually indistinguishable from faraway views. The visual appearance of a 3D shape can be evaluated by its silhouette and surface normal, while the simplicity of a low-poly mesh is typically measured by the number of triangles. We propose a new approach to generate low-polys that is both simple and visual-preserving, with the additional guarantees of being manifold, watertight, and self-intersection free. These additional properties are essential for artists to conveniently perform UV-generation and texture baking on the bare low-poly mesh.

Our method combines the idea of mesh-reduction and re-meshing. During the first stage, we re-mesh the high-poly to a "clean" proxy mesh and remove all the topological complexities therein. We then aggressively reduce the element count of the proxy mesh, while geometrically deforming the proxy to maintain visual preservation, leading to our low-poly output.

Our method specifically requires two inputs: a high-poly mesh and a parameter n_p, which represents the screen-space size of the high-poly mesh when rasterized onto a 2D screen. In practical terms, let l_p denote the 2D screen's pixel length, and l represent the diagonal length of the high-poly mesh's bounding box. The parameter n_p can be calculated as l/l_p, signifying the maximum number of pixels that the high-poly mesh's diagonal could occupy across all potential rendering views. During the first stage of our method, we build an unsigned distance field for the input and introduce a novel offset surface extraction method to extract a d-isosurface with $d = l/n_p$. Our algorithm not only guarantees the offset mesh is manifold, watertight, and self-intersection-free, but also recovers the normal approximated sharp features. During our second stage, we alternate among three steps, i.e. mesh simplification, mesh flow process, and feature alignment, to reduce the element count of the extracted mesh, while bringing the mesh close to the input and maintaining the aforementioned guarantees. All three steps contain only local operations, such as edge collapse and vertex optimization. Therefore, any local operation that violates a guarantee can be easily rolled back.

By construction, our algorithm is robust and automatic. The effectiveness of our approach is demonstrated by comparing it with state-of-the-art methods and popularly used software on a subset of the Thingi10K dataset Zhou and Jacobson (2016) containing 3D models with varying complexities. All the data shown in this chapter and the executable program can be found here[2].

4.2 Related Works

We first review low-poly mesh generation methods and then summarize the methods for iso-surface extraction.

4.2.1 Low-poly Meshing

Obtaining a low-poly mesh has been a research focus in computer graphics for several decades. Early works use various mesh reduction techniques that directly operate on the original inputs through iterative local element removal. Examples involve geometric error-guided techniques Hoppe (1996); Garland and Heckbert (1997); Lescoat et al. (2020), structure-preserving-constrained technique Salinas et al. (2015), volume-preserving technique Lindstrom and Turk (1998), and image-driven technique Lindstrom and Turk (2000), to name just a few. Clustering-based approaches Cohen-Steiner et al. (2004); Li and Nan (2021) provide another direction for reducing the element count. An inclusive survey is given in Khan et al. (2022). While these approaches are well recognized in game production pipelines, they are better suited for reducing the mesh size of the original models to a medium level, e.g. reducing the number of faces by 20% - 80%. Unfortunately, for 3D graphics applications running on lower-end devices, often a much coarser low-poly mesh is desired. Such extremely low-poly meshes require topologic and geometric simplifications that are far beyond the capabilities of these mesh reduction techniques. Unlike mesh reduc-

[2]https://robust-low-poly-meshing.github.io/

tion techniques, a parallel effort aims at re-meshing, i.e., completely reconstructing a new mesh mimicking the original one. These methods vary drastically in their techniques and we classify them by their main feature into *voxelization base re-meshing*, *primitive fitting*, *visual-driven*, and *learning-based*.

Voxelization Based Re-meshing Both Mehra et al. (2009) and Calderon and Boubekeur (2017) rely on a voxelization of the raw inputs to obtain a clean voxel surface. While Mehra et al. (2009) requires feature-guided re-triangulation, deformation, and curve-network cleaning to generate shape abstractions for architectural objects, Calderon and Boubekeur (2017) assumes the input meshes come with clear separation of the inside and outside space and heavily depends on user interactions to generate the final low-polys. Recently, Wu et al. (2022) combine voxelization-based remeshing with patch-based simplification to generate occluders for building models to pre-cull unseen meshes before online rendering.

Primitive Fitting Various primitives can be composed to fit an object. For example, methods in Chauve et al. (2010); Kelly et al. (2017); Fang et al. (2018); Nan and Wonka (2017); Fang and Lafarge (2020); Bauchet and Lafarge (2020) first compute a set of planes to approximate patch features detected in point clouds or 3D shapes, and then select a faithful subset of the intersecting planes to obtain the desired meshes. However, the key challenges of this type of method are: 1) properly computing a suitable set of candidate planes is already a hard problem by itself; 2) the complexity of the resulting mesh is highly unpredictable, requiring many trial-and-error to find a possible good set of parameters. Works using other primitives Mehra et al. (2009); Huang et al. (2014); Yang and Chen (2021); Wei et al. (2022), such as boxes, convex shapes, curves, etc., are also promising directions, but none of them has been specifically dedicated for generating low-polys.

Visual-driven Approaches Differentiable rendering Laine et al. (2020) rises as a hot topic that enables the continuous optimization of scene elements through the guidance of rendered image losses. However, most of them Luan et al. (2021); Hasselgren et al. (2021); Nicolet et al. (2021) require an initial mesh that is typically a uniformly discretized sphere. The key obstacle to generating low-polys via differentiable rendering is that the mesh reduction cannot be modeled as a differentiable optimization process. Although the analysis-by-synthesis type of optimizations Luan et al. (2021) could be employed, the Laplacian regularization term used by most differentiable rendering techniques can guide the mesh far from the groundtruth, especially in a low-poly setting. A visual hull-based approach Gao et al. (2022) has been recently proposed to generate extremely low-polys for building models, however, it not only creates sharp creases for organic shapes, but also makes it hard to control the target element number.

Learning-based Methods A conventional 3D mesh reconstruction pipeline is composed of three steps: plane detection, intersection, and selection, while learning-based methods enable alternative pipelines. As an example, by converting it to a BSP-net, Chen et al. (2020) demonstrates that low-polys can be extracted from images. However, this method shares the common shortcomings of learning approaches: a large dataset is required for the network training, and the learned model works only for meshes of a similar type. It further requires the voxelizations of the dataset to have well-defined in/out segmentation. Furthermore, the generated meshes inherit the issues of polyfit-like Nan and Wonka (2017) approaches: it creates sharp creases that are not present in the high-poly, and parameter tuning is difficult. By embedding a neural net of marching tetrahedral into the differentiable rendering framework, Munkberg et al. (2022) can optimize the meshes and materials simultaneously. As demonstrated in their work, by controlling the rendered image resolution, it can generate 3D models in a LOD manner. But these extended features brought by learning approaches are beyond the scope of our investigation.

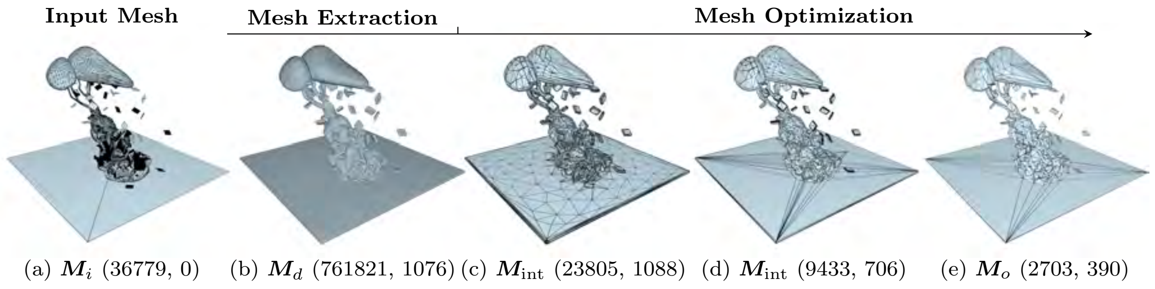

Input Mesh
Mesh Extraction
Mesh Optimization
(a) $\boldsymbol{M}_i$ (36779, 0)
(b) $\boldsymbol{M}_d$ (761821, 1076)
(c) $\boldsymbol{M}_{\text{int}}$ (23805, 1088)
(d) $\boldsymbol{M}_{\text{int}}$ (9433, 706)
(e) $\boldsymbol{M}_o$ (2703, 390)

Figure 4.2: The pipeline of our algorithm. The notation $(\bullet, \bullet)$ represents (number of faces, light field distance to the input mesh), where the latter is a measure of visual similarity between two 3D shapes, introduced by Chen et al. (2003). (a): The input high-poly surface, which is not necessarily manifold, orientable, or self-intersection free. In this example, the input surface $\boldsymbol{M_i}$ is not 2-manifold, with 114 non-2-manifold edges, has 751 components and is not orientable (the back faces are rendered in black). It has over 36k faces, among which 23345 faces are self-intersected. (b): The extracted iso-surface $\boldsymbol{M}_d$ with $d = \frac{l}{200}$, where l is the bounding box diagonal size of $\boldsymbol{M}_i$ (Section 4.3.1). It is an orientable, water-tight, self-intersection free mesh with 19-components and 761k faces. (c-e) Our mesh optimization step (Section 4.3.2), during which we apply mesh simplification, flow, and alignment. While the simplification step may result in a slight increase in LFD, the subsequent flow and alignment steps enhance visual similarity. Consequently, the overall optimization step progressively reduces the light field distance and simplifies the mesh, and the intermediate meshes denoted as $\boldsymbol{M}_{\text{int}}$. The output $\boldsymbol{M}_o$ has only 2703 faces. Our approach resolves the existing topologic (non-manifoldness) and geometric issues (self-intersection), and approximates the high-poly with 7.3% faces.

4.2.2 Iso-surfacing Algorithms

The marching cubes (MC) algorithm was proposed concurrently by Lorensen and Cline (1987) and Wyvill et al. (1986) for reconstructing iso-surfaces from discrete signed distance fields. Several follow-up works were proposed to solve the tessellation ambiguities in each cube Dürst (1988); Nielson and Hamann (1991); Matveyev (1994); Chernyaev (1995); Nielson (2003, 2004). One of the best methods is the marching cubes 33 (MC33) Chernyaev (1995), which enumerates all possible topologic cases based on trilinear interpolation in the cube. The follow-ups resolve non-manifold edges in MC33 Lopes and Brodlie (2003); Custodio et al. (2013). MC33 was correctly implemented by Vega et al. (2019) after resolving the defective issues of the previous implementations Lewiner et al. (2003); Custodio et al. (2013). However, none of these methods is able to recover sharp features.

To capture sharp features of the iso-surface, Kobbelt et al. (2001) first introduced an extended marching cubes method (EMC) to insert additional feature points, given that the normals of some intersecting points are provided. Dual contouring (DC) Ju et al. (2002) adapted this idea with Hermite data (the gradient of the implicit surface function). They proposed to insert one dual feature point inside a cube and then connect the dual points to form an iso-surface. DC does not need to perform the edge-flip operations required by Kobbelt et al. (2001), but often generates non-manifold surfaces with many self-intersections. The non-manifold issue was later addressed in Ju and Udeshi (2006), while the self-intersection issue was addressed in Schaefer et al. (2007). However, none of these two methods solves both the non-manifold and self-intersection problems simultaneously. Dual Marching Cubes (DMC) Schaefer and Warren (2004) considers that the dual grid aligns with features of the implicit function, and extracts the iso-surface from the dual grid. DMC can preserve sharp features without excessive grid subdivisions as required by DC. Unfortunately, DMC still does not guarantee the generated mesh is self-intersection-free. Manson and Schaefer (2010) avoided self-intersections by subdividing each cube into multiple tetrahedra, and then applying marching tetrahedra (MT) to extract the iso-surface Doi and Koide (1991). This approach solves the self-intersection problems in the DMC approach, but the division of multiple tetrahedra, together with the employed octree structure, makes the algorithm either generate an overly dense mesh or require trial-and-error for suitable octree depth parameter settings. A survey about these approaches can be partially found in de Araújo et al. (2015). Recently, Portaneri et al. (2022) proposed an algorithm to generate watertight and orientable surfaces that strictly enclose the input. Their output is obtained by refining and carving the 3D Delaunay triangulation of the offset surface, however, still without the feature-preserving property.

There are also several learning-based approaches for iso-surface extraction.

Deep marching cubes Liao et al. (2018) and deep marching tetrahedra Shen et al. (2021) learn differentiable MC and MT results. However, none of them can capture the sharp features of the initial surface. Neural marching cubes Chen and Zhang (2021) and Neural dual contouring Chen et al. (2022a) train the network to capture the sharp features without requiring extra Hermite information. However, the former generates self-intersected meshes, and the latter leads to non-manifold results. In Table 4.1, we summarize these methods and show their strength and weakness in terms of topologic and geometric properties: manifoldness, self-intersection-free, and sharp feature preservation.

Method	Manifold	Free of Self-Intersection	Preserve Features
Lorensen and Cline (1987)	$\surd$	$\surd$	$\times$
Wyvill et al. (1986)	$\surd$	$\surd$	$\times$
Chernyaev (1995)	$\surd$[3]	$\surd$	$\times$
Doi and Koide (1991)	$\surd$	$\surd$	$\times$
Kobbelt et al. (2001)	$\surd$	$\times$	$\surd$
Ju et al. (2002)	$\times$	$\times$	$\surd$
Ju and Udeshi (2006)	$\surd$	$\times$	$\surd$
Schaefer et al. (2007)	$\times$	$\surd$	$\surd$
Manson and Schaefer (2010)	$\surd$	$\surd$	$\surd$
Portaneri et al. (2022)	$\surd$	$\surd$	$\times$
Liao et al. (2018)	$\surd$	$\times$	$\times$
Shen et al. (2021)	$\surd$	$\times$	$\times$
Chen et al. (2021a)	$\surd$	$\times$	$\surd$
Chen et al. (2022a)	$\times$	$\surd$	$\surd$

Table 4.1: A brief summary of the existing methods by their capabilities of maintaining geometry and topology properties. A more detailed survey can be found in de Araújo et al. (2015).

[3]Although the initial paper results in non-manifold edges, this artifact was fixed by the follow-up works Lopes and Brodlie (2003); Custodio et al. (2013)

4.3 Method

Given the input of a polygonal mesh $\boldsymbol{M}_i$, a maximum number of screen size n_p (i.e. the number of pixels covered by the diagonal length of the input's bounding box), and an optionally user-specified target number of triangles n_F, we seek to generate a triangle mesh $\boldsymbol{M}_o$ with the following properties:

Pro.I The number of triangles in $\boldsymbol{M}_o$ is either minimized or equals to n_F if provided as a parameter;

Pro.II $\boldsymbol{M}_o$ is indistinguishable from $\boldsymbol{M}_i$ when rendered from a faraway view (a view where the diagonal length of the bounding box of $\boldsymbol{M}_i$ is less than n_p pixels);

Pro.III $\boldsymbol{M}_o$ is both topologically and geometrically clean, i.e., water-tight, manifold, and intersection-free.

These three properties of $\boldsymbol{M}_o$ ensure rendering quality and enable any downstream geometric processing on it to have high computational efficiency, requiring no mesh repairing steps. The level of visual preservation in our second property is measured by Silhouette difference and the normal difference between $\boldsymbol{M}_i$ and $\boldsymbol{M}_o$. A similar normal indicates $\boldsymbol{M}_o$ preserves the sharp features of $\boldsymbol{M}_i$ as much as possible.

We follow several principles to design our approach: 1) We make no assumptions on the topologic or geometric properties of the input, allowing our approach to handle any models created in the wild; 2) We adopt an interior-point optimization-like strategy to realize the topology and geometry properties of Pro.III one by one: once a property is satisfied, it will be maintained for the rest of the steps; 3) We value robustness with the highest priority, so that our approach can process any inputs created by different domains of applications. Under guaranteed robustness, we further attempt to improve the computational efficacy to the greatest extent possible.

Overview We tackle this problem in two main stages (Figure 4.2), namely, *mesh extraction* (Section 4.3.1), and *mesh optimization* (Section 4.3.2). During the mesh extraction stage, we first compute an unsigned distance field for $\boldsymbol{M}_i$, then introduce a novel iso-surface mesh extraction approach for a positive offset distance d ($d = l/n_p$ as mentioned in Section 3.1), and finally remove all invisible disconnected components from the extracted iso-surface to obtain a mesh $\boldsymbol{M}_d$. Our generated $\boldsymbol{M}_d$ optimally recovers the sharp features implied by the $d-$iso-surface of the distance field, and guarantees watertightness, manifoldness, and free of self-intersections. The purpose of this stage is to generate a "clean" proxy mesh $\boldsymbol{M}_d$ of the input $\boldsymbol{M}_i$ that possibly has "dirty" topologic and geometric configurations. Our second mesh optimization stage involves a while-loop of three sequential steps: simplification, flow, and alignment. The simplification step aims to reduce the number of triangles of $\boldsymbol{M}_d$ by performing one pass of quadric edge-collapse decimation for the entire mesh; the flow step aims to pull $\boldsymbol{M}_d$ close to $\boldsymbol{M}_i$ via a per-vertex distance minimization; and the alignment step aims to optimize the surface normal of $\boldsymbol{M}_d$ so that the sharp features are maintained, which is achieved through local surface patch optimization. When the while loop stops, we output the final mesh $\boldsymbol{M}_o$. Since all three steps contain only local operations, the guarantees of $\boldsymbol{M}_d$ achieved during the first stage can be easily maintained by rolling back or skipping any operations that violate a guarantee. In the following sections, we provide technical details for each stage.

4.3.1 Mesh Extraction

Given $\boldsymbol{M}_i$ and d, our goal is to extract an $d-$iso-surface mesh $\boldsymbol{M}_d$ that is watertight, manifold, feature-preserving, and self-intersection-free. Ensuring all these properties simultaneously is a challenging task. As an example, simply applying the well-known algorithm of MC33 cannot capture sharp features of the iso-surface as shown in Figure 4.3. We tackle the mesh extraction problem by re-meshing the extracted local surface patches from templates of MC33. We selectively insert additional points to refine these local surface patches. Our key technique lies in the proposed

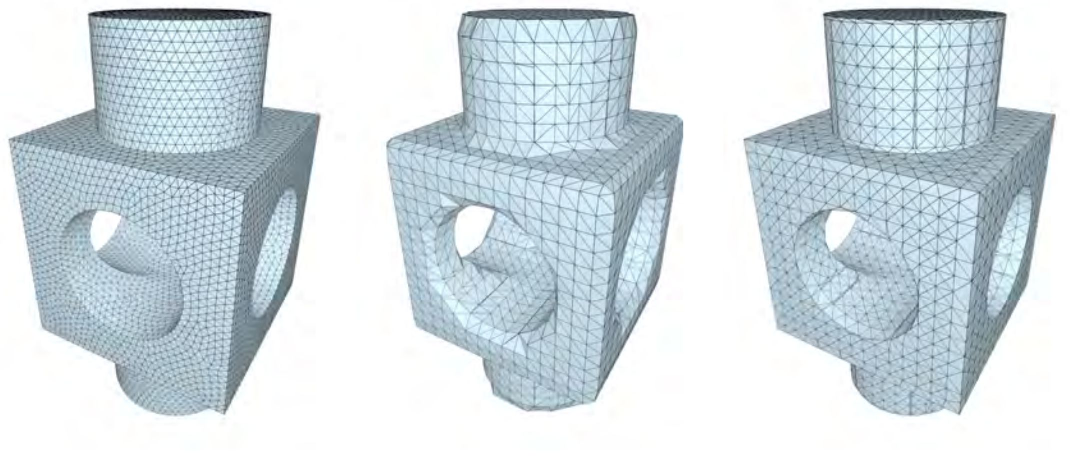

(a) Input (b) MC33 (c) Ours

Figure 4.3: The iso-surface meshes of a CAD model "block". Compared with classic MC33 algorithm Chernyaev (1995), our iso-surfacing approach achieves a better visual similarity by recovering sharp features.

mesh refinement technique that 1) guarantees topologically watertight and manifold properties and 2) captures geometric sharp features without causing self-intersection. We first compute a proper *discretization* of an unsigned distance function defined for $\boldsymbol{M}_i$, then analyze the connectivity changes when inserting new points to the MC33 templates to maintain the *topology guarantees* of the resulting mesh. After that, we focus on the *geometry fidelity* of the resulting mesh, i.e. feature-preserving and self-intersection-free. For the extracted mesh $\boldsymbol{M}_d$, we finally remove invisible components.

Discretization An unsigned distance field is defined as a function:

$$f(\boldsymbol{p}) := \min_{\boldsymbol{q}\in\boldsymbol{M}_i} \|\boldsymbol{p}-\boldsymbol{q}\|, \tag{4.1}$$

where $\boldsymbol{p} \in \mathcal{R}^3$. The implicit function of d-iso-surface is $f(\boldsymbol{p}) = d$. Since the explicit representation of the d-iso-surface is intractable, we follow the general pipeline of prior iso-surfacing approaches that first voxelize the ambient space around $\boldsymbol{M}_i$, and then approximate the solution through extracting local surface patches within each voxel. Since the local patches are typically simple, the voxel size plays an important role in the to-be-extracted mesh. A coarse voxel size can miss important solutions, such as the one illustrated in Figure 4.4, where no d-iso-surface could be extracted if a grid size large than $2d$ is used, while an excessively small voxel size will result in a dense grid that is time-consuming to compute (Figure 4.15). By default, we choose the edge length of a voxel to be $d/\sqrt{3}$ to avoid geometric feature losses as illustrated in the Figure 4.4.

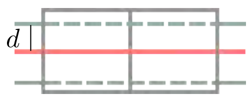


Figure 4.4: The d-iso-surface (green lines) of the input mesh (red line) cannot be captured if the voxel size is larger than $2d$.

Topologic Guarantees For each voxel, we employ existing templates to decide the iso-contours Lorensen and Cline (1987); Chernyaev (1995); Custodio et al. (2013), and then insert an additional point for each contour. The templates of either the original MC Lorensen and Cline (1987) or MC33 Chernyaev (1995); Custodio et al. (2013) can be used to generate the iso-contours since they both ensure the extracted mesh is watertight and manifold. We choose MC33 in this work since it covers more linear interpolation cases and resolves the ambiguity in MC, thus extracting iso-surface meshes with generally smaller genus Chernyaev (1995); Custodio et al. (2013). As illustrated in Figure 4.23, we insert one vertex per iso-contour surface if it is homemorphic to a disk, where the iso-contour surfaces of cases 4.1.2, 6.1.2, 7.4.2, 10.1.2 and 12.1.2, and one iso-contour of case 13.5.2 are excluded since they have two boundaries. This scheme ensures that the additional vertices neither bring any non-manifold configurations nor create holes in the resulting mesh.

Feature Vertex Insertion We use one vertex per local patch with a disk-topology to provide more degrees of freedom to capture sharp features. Its position can be computed by minimizing the distance to patch vertices along the normal directions:

$$\arg\min_{\boldsymbol{x}} \sum_i \left(\boldsymbol{n}_{\boldsymbol{p}_i} \cdot (\boldsymbol{x} - \boldsymbol{p}_i)\right)^2, \tag{4.2}$$

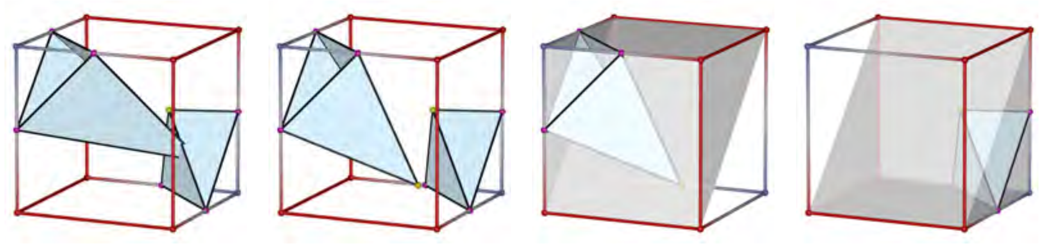

(a) Without constraints (b) With constraints (c) Polyhedron$_0$ (d) Polyhedron$_1$

Figure 4.5: Illustration of our feature-aware iso-surface extraction step for MC33 case 4.1.1. The cube vertices with iso-value smaller than d is marked as blue, while the red vertices are the opposite. The iso-points are marked as pink, and the feature points are marked as yellow. Without forcing the feature points within their belonging polyhedra (c, d), it is easy to obtain a mesh with self-intersections (a).

where $\boldsymbol{x}$ is the desired position, $\boldsymbol{p}_i$ and $\boldsymbol{n}_{\boldsymbol{p}_i}$ denote an iso-contour vertex and its normal, respectively, and the summation goes through all patch vertices. However, without any constraints, $\boldsymbol{x}$ may be arbitrarily positioned and cause surface intersections in the extracted mesh. This issue is deteriorated by the template cases with multiple iso-contours. Figure 4.5 illustrates such as example using MC33 case 4.1.1.

We propose a simple approach to recover feature vertices without mesh intersections. Given a voxel, we subdivide it into convex polyhedra within which the feature vertices are constrained. As illustrated in Figure 4.24, the subdivision is performed according to the number and the different configurations of the iso-contours with a disk-topology. For example, for cases with only a single disk-topology iso-contour, such as case 1, no subdivision is involved and the polyhedron is the voxel itself, and for those with multiple iso-contours, such as case 4.1.1, convex and planar polygons are needed to partition the voxel into non-overlapping polyhedra. If we constrain the position of the inserted vertex stays inside its belonging polyhedron, the extracted mesh is guaranteed to incur no self-intersections.

Accordingly, for each inserted vertex, we obtain its coordinate x by solving a linear constrained quadratic programming:

$$\begin{aligned} &\arg\min_{\boldsymbol{x}} \sum_i \left(\boldsymbol{n}_{\boldsymbol{p}_i} \cdot (\boldsymbol{x} - \boldsymbol{p}_i)\right)^2 \\ &\quad s.t. \quad \boldsymbol{n}_s \cdot (\boldsymbol{x} - c_f) < 0, \ \forall \boldsymbol{n}_s \in N_s \end{aligned} \tag{4.3}$$

where N_s is the set of face normals of the corresponding polyhedron of x, and c_f is the center of the corresponding polyhedron face. To compute $\boldsymbol{n}_{\boldsymbol{p}_i}$, if $\boldsymbol{p}_i \in \boldsymbol{M}_i$, we simply use the mesh normal, otherwise, we first solve for any $\boldsymbol{p}_i^\star \in \arg\min_{\boldsymbol{p}\in\boldsymbol{M}_i} \|\boldsymbol{p}_i - \boldsymbol{p}\|$ and then let $\boldsymbol{n}_{\boldsymbol{p}_i} := (\boldsymbol{p}_i - \boldsymbol{p}_i^\star)/\|\boldsymbol{p}_i - \boldsymbol{p}_i^\star\|$.

Feature Extraction The previous step recovers vertices on sharp features of the d-iso-surface. But their connections may not align well with the sharp edges, the highlighted region in Figure 4.6b demonstrates this issue. Furthermore, since the sharp features exist in a small fraction of voxels, we aim at a minimal increase in the additional feature edges and vertices by inserting only those feature vertices on sharp features and using the original MC33 templates as much as possible. However, we do not know the sharp features of the d-iso-surface as prior. Therefore, we introduce a posterior approach to recover the necessary feature curves, which involves two phases: *Feature Edge Adjustment*, and *Feature Filtering*. The first phase generates an iso-surface mesh by considering all inserted feature vertices as sharp features. With this iso-surface mesh, we can use existing automatic feature identification approaches to obtain sharp features. The second phase generates the actual iso-surface mesh $\boldsymbol{M}_d$ by blending the iso-contour patches containing the detected feature vertices with those original MC33 patches that do not contain any sharp features.

Feature Edge Adjustment During the first phase, we insert a feature vertex for every disk-topologic iso-contour patch. We then perform an edge-flip operation for every mesh edge if the flipped edge connects two inserted feature vertices (see Algorithm 2 for more details). To ensure the self-intersection-free guarantee, we skip those edge-flips that may cause self-intersections. This phase can already produce an

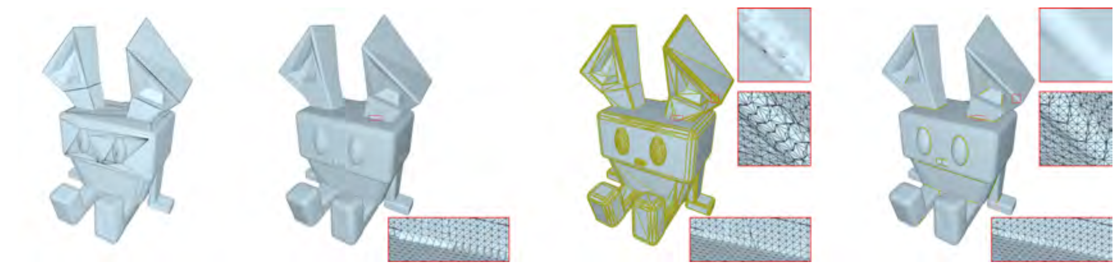

(a) Input mesh $\boldsymbol{M}_i$ (b) Unflipped feature mesh (c) Flipped feature mesh (e) Final iso-surface $\boldsymbol{M}_d$

Figure 4.6: Our iso-surface extraction pipeline. We mark the initial feature points in the third figure and the remaining ones after applying feature graph filter in the last figure (the yellow points). Our post process successfully resolves the sawtooth artifacts around the ear of the character (the top two zoomed-in figures in the third and last columns, with the top ones are rendered without wireframes), but still keeps the major sharp features, for example, the bottom zoomed-in figures.

iso-surface mesh that satisfies the desired topologic and geometric properties. However, as mentioned earlier, this iso-surface mesh contains more elements than desired and those unnecessary "fake" sharp features are noisy and not visual-appealing (see the top two zoomed-in figures in Figure 4.6c).

Feature Filtering During the second phase, we first extract a feature graph from the resulting mesh of the first phase. The feature graph is composed of a set of feature curves where each curve is a sequence of mesh edges with its dihedral angle smaller than θ_0 (see Gao et al. (2019) for details). We then mark a feature curve as valid if it is composed of more than l_0 mesh edges. The valid feature curves are considered to contain "real" sharp features to recover. After that, for each iso-contour patch, we keep those inserted feature vertices if they are on the valid feature curves, otherwise, we use their original template. This step removes a lot of noisy, "fake" feature edges. Finally, we perform the edge flip algorithm Algorithm 2 once more to extract the d-iso-surface mesh $\boldsymbol{M}_d$.

Our feature recovery algorithm performs well for the models with various features that can be represented by piecewise line segments, e.g. sharp curves in Figure 4.3 and the eye and beak contours in Figure 4.12.

Algorithm 1 Iso-surface Extraction

```
    Input: $\boldsymbol{M}_i$, $d$, $\theta_0$, $l_0$
    Output: $\boldsymbol{M}_d$
 1: $G \leftarrow$ gridDiscretization($\boldsymbol{M}_i, d$)                ▷ generate the grids
 2: Compute $f(\boldsymbol{p})$ for all grid points $\boldsymbol{p}$ in $G$                ▷ Equation 4.1
 3: for each $cube \in G$ do                ▷ iso-surface extraction
 4:     Lookup for the template case                ▷ Chernyaev (1995), Figure 4.23
 5:     for each Disk-topologic patch in cube case do
 6:         Compute the iso-points on cube edges
 7:         Form the quadratic program                ▷ Equation 4.3, Figure 4.24
 8:         Solve for feature vertices                ▷ Figure 4.23
 9:     end for
10: end for
11: $\boldsymbol{M}_d \leftarrow$ edgeFlip($\boldsymbol{M}_d$) to connect feature vertices                ▷ Algorithm 2
12: $\boldsymbol{M}_d \leftarrow$ featureFilter($\boldsymbol{M}_d, \theta, l_0$)
13: $\boldsymbol{M}_d \leftarrow$ removeInterior($\boldsymbol{M}_d$)
```

Interior Removal Since we use an unsigned distance function, our final extracted iso-surface $\boldsymbol{M}_d$ may have interior components, which are totally invisible. Given the generated mesh $\boldsymbol{M}_d$ are watertight and free of self-intersection, we can apply the in-and-out test and remove the components which is purely inside of any of the others.

4.3.2 Mesh Optimization

Starting from a mesh $\boldsymbol{M}_d$ that is watertight, manifold, feature preserving, and self-intersection-free, we now introduce an iterative mesh optimization approach to obtain a final $\boldsymbol{M}_o$ that satisfies our three desired properties, i.e., Pro.I-III. As shown in Algorithm 3, our optimization involves a maximum of N iterations of three sequential steps: *simplification*, *flow*, and *alignment*. We stop the iterations until either the Hausdorff distance between the simplified meshes of two consecutive loops (relative change) is smaller than ϵ, the loop number reaches N, or the target face number reduces below n_F, where the first condition has the highest priority by default.

Algorithm 2 Edge Flip

```
    Input: $\boldsymbol{M}_d$
    Output: $\boldsymbol{M}_d$                                  ▷ mesh after edge-flips
 1: $Q \leftarrow \{\}$
 2: BVHTree T
 3: T.build($\boldsymbol{M}_d$)                                 ▷ Karras (2012)
 4: for each edge $e \in \boldsymbol{M}_d$ do
 5:     if $e$.oppVs are feature vertices then
 6:         $Q$.push($e$)              ▷ opposite vertices are feature vertices
 7:     end if
 8: end for
 9: while $Q \neq \{\}$ do
10:     $e \leftarrow Q$.top()
11:     if $e$ was not flipped before then
12:         if isIntersectionFree($\boldsymbol{M}_d$, T, $e$) then
13:             flipEdge($\boldsymbol{M}_d$, $e$)
14:             T.refit($\boldsymbol{M}_d$)                ▷ update BVH Karras (2012)
15:         end if
16:     end if
17: end while
```

Each of the three steps involves only local operations. To ensure our optimization proceeds towards the generation of $\boldsymbol{M}_o$ with the desired properties, we perform the following checks for the meshes before and after applying a local operation:

1. Topology consistency: the updated mesh is manifold, watertight, and has the same genus and the number of components as the mesh before applying the local operation;

2. Self-intersection-free: the updated mesh is free of intersections.

Mesh Simplification This step aims to achieve the first property of $\boldsymbol{M}_o$, i.e., $\boldsymbol{M}_o$ contains as few triangles as possible. We perform an entire pass of the standard edge-collapse operation for all edges of $\boldsymbol{M}_o$ to reduce as many faces as possible or match the target face number n_F, where the coordinate of the newly generated vertices are determined by the quadratic edge metric (QEM) Garland and Heckbert (1997) weighted by virtual planes for each edge to avoid the degeneracy in planar regions. Importantly, the topologic and geometric validity of $\boldsymbol{M}_o$ is maintained during the simplification process by skipping those edge-collapse operations that may violate the aforementioned checks. Moreover, to ensure we get closer to $\boldsymbol{M}_i$, we also skip the collapse operations which increase the distance between affected local triangle patches and $\boldsymbol{M}_i$. For efficiency concerns, we only compute the one-sided distance from the local patch to the input mesh. This one-sided check may result in the acceptance of unexpected collapses, as illustrated in Figure 4.7. To overcome this, we further skip the operations leading to a Hausdorff distance larger than d, where the two involving meshes are the ones before and after the local operation and the Hausdorff distance is computed approximately by sampling points on the local triangle patches as in Cignoni et al. (1998). We show the comparison in Figure 4.8. Without the guarantee of a distance decrease, we will lose some important information. Without the guarantee of a small Hausdorff distance, we may end up with larger silhouette difference and normal difference, that is, worse visual similarity.

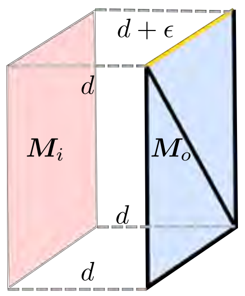


Figure 4.7: Collapsing the yellow edge reduces the distance from $\boldsymbol{M}_o$ to $\boldsymbol{M}_i$ ($d+\epsilon \to d$), but leads to an undesirable visual appearance.

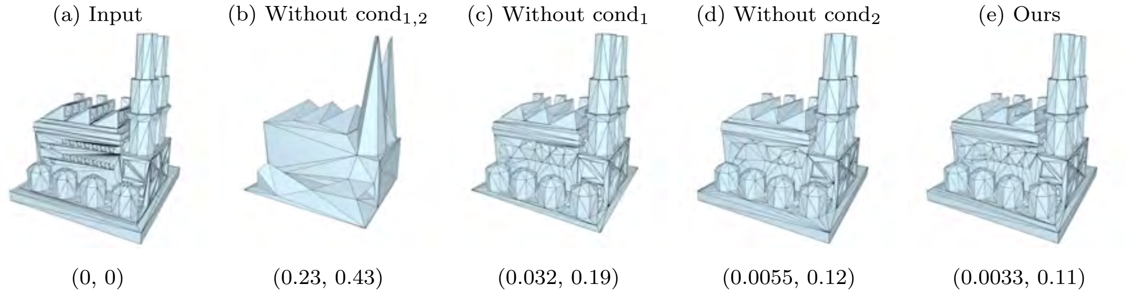

(a) Input
(b) Without $\text{cond}_{1,2}$
(c) Without cond_1
(d) Without cond_2
(e) Ours
(0, 0)
(0.23, 0.43)
(0.032, 0.19)
(0.0055, 0.12)
(0.0033, 0.11)

Figure 4.8: The results of different simplification conditions. $(\bullet, \bullet)$ denotes (silhouette difference, normal difference). cond_1: skip the collapse which increases the vertex-surface distance to $\boldsymbol{M}_i$; cond_2: skip the collapse which results a large Hausdorff distance. After applying the both conditions, we achieve a better visual score.

Mesh Flow Our mesh flow step brings $\boldsymbol{M}_o$ geometrically close to $\boldsymbol{M}_i$ and reduces the silhouette visual differences between the two meshes. The detailed algorithm is provided in Algorithm 3 (Line 7-11).

Algorithm 3 Mesh Optimization Process

```
    Input: $\boldsymbol{M}_i$, $\boldsymbol{M}_d$, $d$, $n_F$ $N$, $r$, $\epsilon$
    Output: $\boldsymbol{M}_o$
 1: $\boldsymbol{M}_o \leftarrow \boldsymbol{M}_d$
 2: $l \leftarrow$ bboxSize($\boldsymbol{M}_i$)                          ▷ bounding box diagonal size
 3: for $i = 0$ to $N$ do
 4:     $\boldsymbol{M}' \leftarrow \boldsymbol{M}_o$
 5:     $\boldsymbol{M}_o \leftarrow$ meshSimplification($\boldsymbol{M}_i, \boldsymbol{M}_o, d, n_F$)   ▷ Algorithm 5
 6:     $\widetilde{\boldsymbol{M}} \leftarrow \boldsymbol{M}_o$
 7:     for each vertex $\boldsymbol{v} \in \boldsymbol{M}_o$ do                 ▷ mesh flow step
 8:         $\boldsymbol{v}^* \leftarrow \mathbf{argmin}_{\boldsymbol{u} \in \boldsymbol{M}_i} \|\boldsymbol{u} - \boldsymbol{v}\|$
 9:         $d_{\boldsymbol{v}} \leftarrow r(\boldsymbol{v}^* - \boldsymbol{v})$                      ▷ successive flow, $r < 1$
10:         $\boldsymbol{v} \leftarrow$ localUpdate($\boldsymbol{M}_o, \boldsymbol{v}, d_{\boldsymbol{v}}$)            ▷ Algorithm 4
11:     end for
12:     for each vertex $\boldsymbol{v} \in \boldsymbol{M}_o$ do                 ▷ feature alignment step
13:         $\boldsymbol{v}_{\text{opt}} \leftarrow$ featureAlignment($\widetilde{\boldsymbol{M}}, \boldsymbol{M}_o, \boldsymbol{v}$)   ▷ Algorithm 6
14:         $d_{\boldsymbol{v}} \leftarrow \boldsymbol{v}_{\text{opt}} - \boldsymbol{v}$
15:         $\boldsymbol{v} \leftarrow$ localUpdate($\boldsymbol{M}_o, \boldsymbol{v}, d_{\boldsymbol{v}}$)            ▷ Algorithm 4
16:     end for
17:     if Hausdorff($\boldsymbol{M}_o, \boldsymbol{M}'$) $< \epsilon \cdot l$ then
18:         Break                                      ▷ update is small enough
19:     end if
20: end for
```

When actually applying the mesh flow process, for each vertex $\boldsymbol{v}$ in $\boldsymbol{M}_o$, we find its Euclidean-distance-wise closest point $\boldsymbol{v}^*$ of $\boldsymbol{M}_i$ and successively push $\boldsymbol{v}$ to $\boldsymbol{v}^*$ along the vector $d_{\boldsymbol{v}} = \boldsymbol{v}^* - \boldsymbol{v}$. Instead of updating $\boldsymbol{v}$ to $\boldsymbol{v}^*$ directly, we deform $\boldsymbol{v}$ towards $\boldsymbol{v}^*$ based on a constant fractional ratio r of the vector, which allows more moving space for the entire mesh and reduces the chance of optimization stuck when $\boldsymbol{M}_o$ is still far from $\boldsymbol{M}_i$. We also apply a simple line search for the self-intersection-free check to find the maximum step size during the local deformation (Algorithm 4).

Algorithm 4 Local Update

```
    Input: $\boldsymbol{M}$, $\boldsymbol{v}$, $d_{\boldsymbol{v}}$
    Output: updated $\boldsymbol{v}$
    Note: $\boldsymbol{v}$ is a vertex of $\boldsymbol{M}$.
1: $\alpha \leftarrow 1$
2: while $\boldsymbol{v} + \alpha d_{\boldsymbol{v}}$ leads to self-intersections do
3:     $\alpha \leftarrow \alpha/2$
4: end while
5: return $\boldsymbol{v} + 0.95\alpha d_{\boldsymbol{v}}$          ▷ Using 0.95 to avoid numerical error
```

Feature Alignment The previous mesh flow can stretch the mesh unanimously, breaking features and creating dirty inputs for subsequent mesh simplification and flow procedure (see Figure 4.9 for an example). We thus introduce a feature alignment step. For each vertex $\boldsymbol{v}$, we seek an optimized position $\boldsymbol{v}_{\text{opt}}$ by minimizing the shape difference between the local surface of $\boldsymbol{v}_{\text{opt}}$ and that of $\boldsymbol{v}$ before mesh flow:

$$E(\boldsymbol{v}) := \sum_{f \in \boldsymbol{N}^1(\boldsymbol{v})} \left\| \frac{\boldsymbol{n}_f}{\|\boldsymbol{n}_f\|} - \frac{\tilde{\boldsymbol{n}}_f}{\|\tilde{\boldsymbol{n}}_f\|} \right\|^2, \tag{4.4}$$

where we use the normal disagreement to approximate the shape difference (Line 6 in Algorithm 6), $\boldsymbol{N}^1(\boldsymbol{v})$ is the faces within the 1-ring neighbor of $\boldsymbol{v}$, and $\boldsymbol{n}_f$, $\tilde{\boldsymbol{n}}_f$ are the unnormalized face normal of the current mesh and the one before the flow respectively. The summation takes over all faces within the 1-ring neighborhood of vertex $\boldsymbol{v}$. This face normal difference summation approximates the vertex normal difference. Notice that Equation 4.4 is a nonlinear function, which can be solved by the classical Newton's Method. In order to improve the efficiency, we instead treat the $\|\boldsymbol{n}_f\|$ as constant (equal to the value at the beginning of the alignment step, denoted as c_n) and solve a quadratic approximation of Equation 4.4:

$$E(\boldsymbol{v}) := \sum_{f \in N^1(\boldsymbol{v})} \left\| \frac{\boldsymbol{n}_f}{c_n} - \frac{\tilde{\boldsymbol{n}}_f}{\|\tilde{\boldsymbol{n}}_f\|} \right\|^2. \tag{4.5}$$

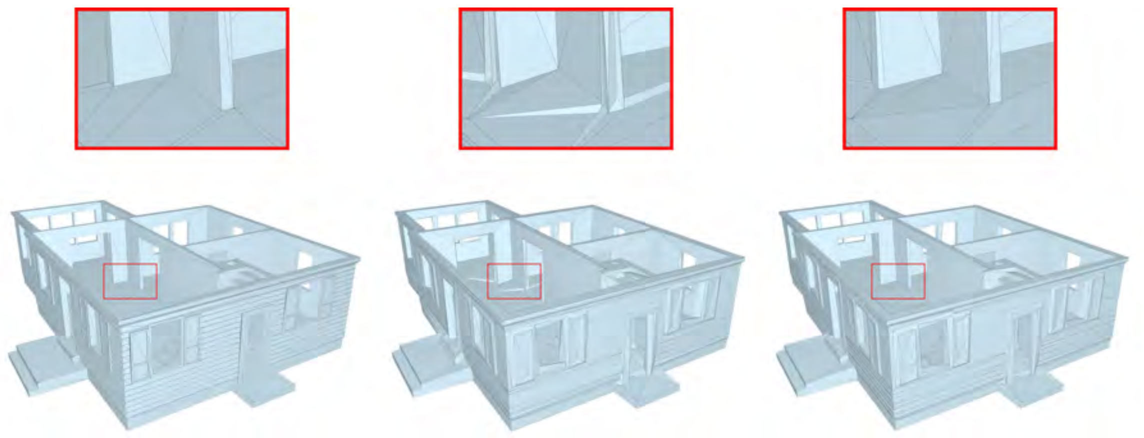
Once we obtain the corresponding $\boldsymbol{v}_{\text{opt}}$ that minimizes Equation 4.5, we update $\boldsymbol{v}$ to be $\boldsymbol{v}_{\text{opt}}$ with the line search Algorithm 4 to prevent self-intersections. Given this local

Algorithm 5 Mesh Simplification

```
    Input: $\boldsymbol{M}_i$, $\boldsymbol{M}_o$, $d$, $n_F$
    Output: simplified mesh $\boldsymbol{M}_o$
    Notes: $\boldsymbol{M}_i$ is the reference mesh, $n_F$ is optional
 1: Form priority queue $Q$                                  ▷ Garland and Heckbert (1997)
 2: BVHTree T
 3: T.build($\boldsymbol{M}_o$)                                           ▷ Karras (2012)
 4: while $Q \neq \{\}$ do
 5:     $e \leftarrow Q$.top()
 6:     if $e$ has been visited before then
 7:         continue                                          ▷ has been collapsed
 8:     end if
 9:     if topologyConsistencyCheck($\boldsymbol{M}_o, \mathbf{e}$) failed then
10:         continue                                        ▷ Cignoni et al. (2008)
11:     end if
12:     if not isIntersectionFree($\boldsymbol{M}_o, \mathrm{T}, \mathbf{e}$) then
13:         continue                          ▷ collapse will introduce intersections
14:     end if
15:     $\boldsymbol{M}_\mathbf{e} \leftarrow$ sub-mesh of $\boldsymbol{M}_o$ adjacent to $\mathbf{e}$
16:     $\boldsymbol{M}'_\mathbf{e} \leftarrow \boldsymbol{M}_\mathbf{e}$ after collapse
17:     if dist($\boldsymbol{M}_\mathbf{e} \rightarrow \boldsymbol{M}_i$) < dist($\boldsymbol{M}'_\mathbf{e} \rightarrow \boldsymbol{M}_i$) then
18:         continue                        ▷ collapse increases the distance to $\boldsymbol{M}_i$
19:     end if
20:     if dist($\boldsymbol{M}_\mathbf{e} \rightarrow \boldsymbol{M}'_\mathbf{e}$) > $d$ or dist($\boldsymbol{M}'_\mathbf{e} \rightarrow \boldsymbol{M}_\mathbf{e}$) > $d$ then
21:         continue                                  ▷ distance update is too large
22:     end if
23:     collapseEdge($\boldsymbol{M}_o, \mathbf{e}$)                         ▷ satisfy all desired properties
24:     T.refit($\boldsymbol{M}_o$)                                   ▷ update BVH Karras (2012)
25:     if $n_F$ is given and $\boldsymbol{M}_o$.faceNumber $\leq n_F$ then
26:         break
27:     end if
28: end while
29: return $\boldsymbol{M}_o$
```

Algorithm 6 Feature Alignment

```
    **Input:** $\widetilde{\boldsymbol{M}}$, $\boldsymbol{M}_o$, $\boldsymbol{v}$
    **Output:** updated $\boldsymbol{v}$ which preserves features
    **Require:** $\widetilde{\boldsymbol{M}}$ and $\boldsymbol{M}_o$ has the same mesh connectivity
1: for each $f \in N^1(\boldsymbol{v})$ do                                ▷ loop over adjacent faces
2:     $\boldsymbol{n}_f \leftarrow e_0 \times e_1$                        ▷ unnormalized face normal of $\boldsymbol{M}_o$
3:     $c_n \leftarrow \|\boldsymbol{n}_f\|$                               ▷ get the initial norm
4:     $\tilde{\boldsymbol{n}}_f \leftarrow \tilde{e}_0 \times \tilde{e}_1$  ▷ unnormalized face normal of $\widetilde{\boldsymbol{M}}$
5: end for
6: Fix $c_n$, get $\boldsymbol{v}_{\text{opt}}$ by minimizing  Equation 4.5   ▷ quadratic program
7: return $\boldsymbol{v}_{\text{opt}}$
```



(a) Input (b) Without feature alignment (c) Ours

Figure 4.9: Without feature alignment step, we will end up with the results with “spikes” (see zoomed-in region for details).

operation only slightly updates the mesh, our quadratic approximation leads to small errors, but in turn, significantly boosts performance (turning a non-convex problem to an unconstrained quadratic program).

4.3.3 Self-intersection Check Acceleration

Starting from an intersection-free 3D triangle mesh, our low-poly re-meshing pipeline could introduce intersections when performing the edge flips during mesh extraction, the edge collapses during mesh simplification, and the vertex optimization

Figures	T_{n}(s)	T'_{n}(s)	$T'_{\mathrm{n}}/T_{\mathrm{n}}$	$T^{\star}_{\mathrm{n}}$(s)	$T^{\star}_{\mathrm{n}}/T_{\mathrm{n}}$	T_{t}(s)	T'_{t}(s)	$T'_{\mathrm{t}}/T_{\mathrm{t}}$	$T^{\star}_{\mathrm{t}}$(s)	$T^{\star}_{\mathrm{t}}/T_{\mathrm{t}}$
Figure 4.2	23.80	2587.09	108.71	111.59	4.69	119.31	2689.11	22.54	213.63	1.79
Figure 4.6	16.15	2847.69	176.31	119.82	7.42	115.86	2955.23	25.51	224.50	1.94
Figure 4.11(d)^0	21.98	2653.21	120.70	158.18	7.20	120.35	2769.56	23.01	266.17	2.21
Figure 4.12(g)^0	13.51	2409.66	178.40	70.66	5.23	93.06	2497.08	26.83	156.08	1.68
Figure 4.12(g)^1	10.24	1377.05	134.51	43.98	4.30	57.25	1421.92	24.84	87.36	1.53
Figure 4.21(j)^0	58.93	7958.25	135.05	552.99	9.38	360.12	8291.89	23.03	885.49	2.46
Figure 4.21(j)^1	5.92	819.00	138.37	39.56	6.68	29.79	846.21	28.41	65.04	2.18
Figure 4.14(d)	13.92	2284.73	164.08	106.71	7.66	99.87	2382.30	23.86	200.90	2.01
Figure 4.22(a)^2	15.65	2283.99	145.93	96.42	6.16	98.12	2376.82	24.22	187.28	1.91
Average	20.01	2802.30	144.67	144.43	4.69	121.52	2914.46	24.69	254.05	1.79

Table 4.2: A speedup summary of self-intersection checks (used in edge flip, edge collapse, and vertex optimization steps) for some of the figures shown in this chapter. The upper index of the figures indicates the corresponding row of that figure. T_{n}: the time cost of the neighboring triangle intersection check only using surface normal test. T'_{n}: the same time information with full normal cone test (surface normal + contour test). $T^{\star}_{\mathrm{n}}$: the same time information but applying parallel triangle pairs intersection check. $T_{\mathrm{t}}, T'_{\mathrm{t}}, T^{\star}_{\mathrm{t}}$: the total time information of the whole intersection-check process (neighboring triangle intersection check + BVH check), corresponding to $T_{\mathrm{n}}, T'_{\mathrm{n}}, T^{\star}_{\mathrm{n}}$.

during the mesh flow and the feature alignment steps. Note that we say a mesh has intersections when any of its two triangles overlaps, or any of its two non-adjacent triangles touch or intersect.

Before discussing our accelerated check of self-intersections, we first introduce the necessary notations. For a vertex $\boldsymbol{v}$, we denote $\boldsymbol{N}^i(\boldsymbol{v})$ as the set of all triangles that are bounded within its i-th ring neighborhood. For example, for the bottom left image in Figure 4.10, $\boldsymbol{N}^1(\boldsymbol{v})$ is the red region, and $\boldsymbol{N}^2(\boldsymbol{v})$ is the union of red and green region[4]. We further denote $\boldsymbol{M}_{\mathbf{e}}$ as the local neighborhood related to a certain local operation, where $\boldsymbol{M}_{\mathbf{e}}$ endows different definitions. For edge flip, we define $\boldsymbol{M}_{\mathbf{e}} = \{\mathbf{f}_1, \mathbf{f}_2\}$ where $\mathbf{f}_1$ and $\mathbf{f}_2$ are the two neighboring triangles of $\mathbf{e}$. For edge collapse, $\boldsymbol{M}_{\mathbf{e}} = \boldsymbol{N}^1(\boldsymbol{v}')$ where $\boldsymbol{v}'$ is the newly created vertex. For vertex optimization (otherwise known as smoothing), we let $\boldsymbol{M}_e = \boldsymbol{N}^1(\boldsymbol{v})$. Moreover, we let $\boldsymbol{M}_s$ be the sub-mesh formed by all faces that share at least one vertex with $\boldsymbol{M}_{\mathbf{e}}$ but not in $\boldsymbol{M}_{\mathbf{e}}$ (the green regions in Figure 4.10). Finally, we let $\boldsymbol{M}_1 = \boldsymbol{M}_s \bigcup \boldsymbol{M}_{\mathbf{e}}$. The rest of the

[4]These sets of triangles are called "topological neighborhoods", introduced in Attene (2010)

mesh are denoted as $\boldsymbol{M}_r$ (the blue regions in Figure 4.10).

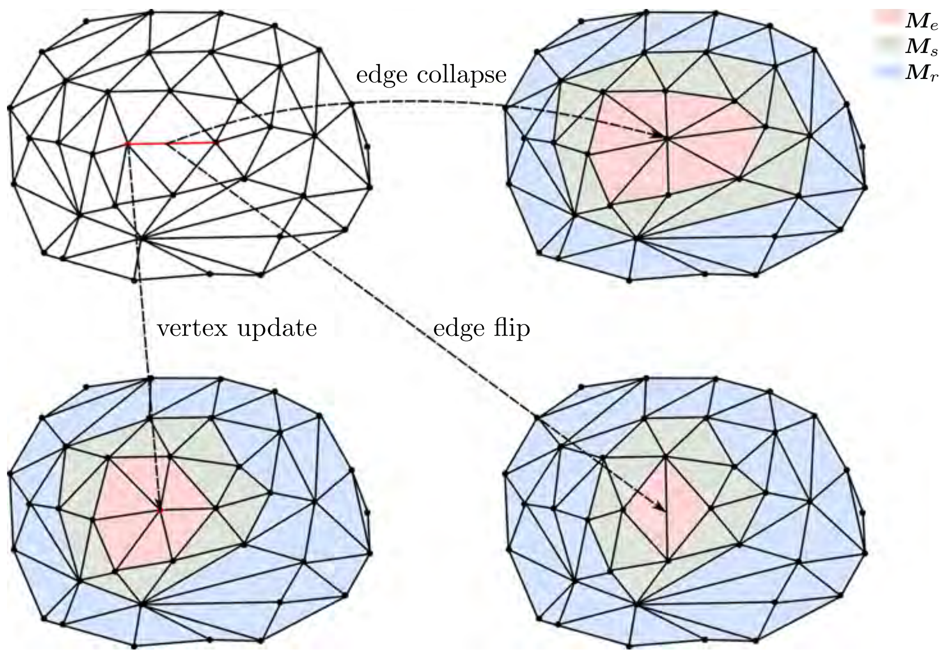


Figure 4.10: Edge or vertex operation illustration. $\boldsymbol{M}_1 = \boldsymbol{M}_e \bigcup \boldsymbol{M}_s$.

We note that a mesh-reduction operation does not introduce self-intersection iff the following two conditions hold:

1. $\boldsymbol{M}_\mathbf{e}$ does not intersect $\boldsymbol{M}_r$;
2. $\boldsymbol{M}_1$ is self-intersection free.

Here we skip the intersection check within $\boldsymbol{M}_s$, $\boldsymbol{M}_r$ and between $\boldsymbol{M}_s$ and $\boldsymbol{M}_r$, because $\boldsymbol{M}_s$ and $\boldsymbol{M}_r$ are the unchanged sub-mesh of the mesh before the local operation, which is free of self-intersections. In general, the two conditions above can be check by conventional triangle-triangle intersection test. However, checking the first condition above is computationally inefficient, especially when $\boldsymbol{M}_r$ contains lots of triangles. Given $\boldsymbol{M}_\mathbf{e}$ does not share any vertex or edge with $\boldsymbol{M}_r$, this part can be handled by standard BVH-based collision detection. The detailed algorithm is given in Algorithm 7.

Algorithm 7 BVH Meshes Intersection Check

```
    Input: $\boldsymbol{M}_{\mathbf{e}}$, $\boldsymbol{M}_r$, T (BVH tree of $\boldsymbol{M}_r$)
    Output: whether $\boldsymbol{M}_{\mathbf{e}}$ intersects with $\boldsymbol{M}_r$
    Notes: all faces $\boldsymbol{M}_{\mathbf{e}}$ do not share vertices with the faces in $\boldsymbol{M}_r$
1: for each face $\mathbf{f} \in \boldsymbol{M}_{\mathbf{e}}$ do
2:     $\mathbf{f}_1 \leftarrow$T.closestFace($\mathbf{f}$)                ▷ get the closest face
3:     if triTriIntersection($\mathbf{f}, \mathbf{f}_1$) then
4:         return true                                   ▷ does intersect
5:     end if
6: end for
7: return false                                          ▷ does not intersect
```

Unfortunately, for the second case, all the faces in $\boldsymbol{M}_{\mathbf{e}}$ share at least one vertex with the faces in $\boldsymbol{M}_s$. The BVH-based acceleration is no longer efficient, as the shared features always lead to failure in BVH culling. In this scenario, the naive approach involves $|\boldsymbol{M}_e| \cdot |\boldsymbol{M}_s|$ pairs of triangle-triangle intersection check. Although $|\boldsymbol{M}_{\mathbf{e}}|$ and $|\boldsymbol{M}_s|$ are usually small for one local operation, the three edge flip, edge collapse, and vertex optimization operations will typically be executed for a massive number of times during the entire re-meshing pipeline. In practice, we find that this $\boldsymbol{M}_1$ intersection-free check takes $\sim 50\%$ of the computational time of the whole intersection check process. Avoiding unnecessary triangle-triangle intersection checks, which is expensive to compute, will lead to a dramatic speedup. To this end, we note that $\boldsymbol{M}_1$ is open, and to check whether a mesh with boundaries has self-intersection, Volino and Thalmann (1994) introduce a theory providing a sufficient condition: Let $\boldsymbol{M}$ be a continuous surface, bounded by $\partial\boldsymbol{M}$, $\boldsymbol{M}$ is self-intersection free if there exists a vector $\boldsymbol{n}$, such that:

1. *Surface Normal Test*: For every point $\boldsymbol{p} \in \boldsymbol{M}$, $\boldsymbol{n}_{\boldsymbol{p}} \cdot \boldsymbol{n} > 0$, where $\boldsymbol{n}_{\boldsymbol{p}}$ is the surface normal at $\boldsymbol{p}$;

2. *Contour Test*: The projection of the contour $\partial\boldsymbol{M}$ along the $\boldsymbol{n}$ is not self-intersected.

They also provide a discrete version for triangle meshes:

1. *Surface Normal Test*: The angle of the normal cone formed by all triangle face normals is less than $\frac{\pi}{2}$;
2. *Contour Test*: The projection of the mesh boundary $\partial \boldsymbol{M}$ along the normal cone axis is not self-intersected.

For the first test, one can use the tight normal cone merging algorithm mentioned by Han et al. (2021), and for the second test, Wang et al. (2017) proposed a side-sign based unprojected contour test.

Surface normal test only need $|\boldsymbol{M}_1|$ times normal cone expansion Han et al. (2021). As shown in Table 4.2, only applying *Surface Normal Test* results in $\sim$145$\times$ speedup compared with the full normal cone test, and 4.69$\times$ speed up compared with parallel triangle-triangle pair check. Moreover, this normal cone test acceleration speeds up the whole self-intersection check process by 24.69$\times$ and 1.79$\times$ compared with the full normal cone test and parallel triangle-triangle pair check, respectively. Surface normal test alone in practice is enough to generate a surface without self-intersection. We perform only the surface normal test during the self-intersection check, and if it fails, we apply direct triangle-triangle pair checks. Although the surface normal test alone is not sufficient to ensure free of self-intersection of $\boldsymbol{M}_1$, in practice, we find our final output $\boldsymbol{M}_o$ is always self-intersection free. We also perform a self-intersection check of $\boldsymbol{M}_o$. If $\boldsymbol{M}_o$ intersects itself, we remove the surface normal test filter, and re-run the algorithm with direct triangle-triangle pair checks.

4.4 Experiments

We implement our algorithm in C++, using Eigen for linear algebra routines, CGAL Brönnimann et al. (2022) for exact triangle-triangle intersection check, libigl Jacobson et al. (2018) for basic geometry processing routines. We use the fast winding number Barill et al. (2018) for interior components identification. We implement the bottom-up BVH traversal algorithm mentioned in Karras (2012) to refit the

BVH for self-intersection check, and use Metro Cignoni et al. (1998) for Hausdorff distance computation. Unless particularly mentioned, we set $n_p = 200$, $\theta_0 = 120°$, $l_0 = 4$, $N = 50$, $r = \frac{1}{8}$, and $\epsilon = 10^{-4}$ by default and run our experiments on a workstation with a 32-cores Intel processor clocked at 3.7Ghz and 256Gb of memory, and we use TBB for parallelization.

Dataset We test our algorithm on a subset of Thingi10K Zhou and Jacobson (2016), where we randomly choose 100 models while filtering out those with the number of triangles smaller than 5000. For this dataset, the average number of faces and disconnected components are 120k and 10. The average number of non-manifold edges and self-intersected triangle pairs are 2197 and 6729, respectively.

4.4.1 Metrics

We evaluate the generated low-poly meshes from several aspects, including the number of contained triangles, topology (watertightness and manifoldness) and geometry (self-intersection-free) guarantees, and the visual preservation of the input.

Similarity Metrics For visual similarity measurement, we employ the following metrics:

1. Hausdorff distance (HD), used to measure the geometrical distance between two 3D shapes;

2. Light field distance (LFD) Chen et al. (2003), which measures the visual similarity between two 3D shapes;

3. Silhouette and normal differences Gao et al. (2022), denoted as SD and ND respectively;

4. Peak signal-to-noise ratio (PSNR) computed by rendering the high- and low-poly meshes with 48 camera views and averaging the PSNR of the 48 pairs of images.

Among all these metrics, a smaller HD, LFD, SD, or ND indicates a better visual similarity, while for PSNR the higher the better.

Methods	#V	#F	#C	r_f	r_m	r_w	r_s	PSNR		LFD		SD		ND		HD	
								ave	sd	ave	sd	ave	sd	ave	sd	ave	sd
Simplygon^1	703	1631	8	9.0%	37.0%	34.0%	100%	24.79	2.73	844.94	1806.07	0.013	0.016	0.049	0.030	0.024	0.016
Simplygon^2	763	1631	2	62%	93.0%	93.0%	100%	25.05	2.41	347.50	188.68	0.0048	0.0043	0.040	0.064	0.022	0.020
Blender	842	1803	12	11.0%	19.0%	14.0%	100%	23.68	3.51	1220.74	1637.12	0.030	0.058	0.071	0.092	0.040	0.052
QEM	707	1629	9	5.0%	12.0%	10.0%	100%	25.18	2.87	748.98	1059.45	0.012	0.021	0.041	0.049	0.031	0.023
Gao et al.	458	912	2	60.7%	91.0%	91.0%	89.0%†	22.55	2.63	1254.85	3978.50	0.020	0.055	0.063	0.059	0.078	0.073
KSR	727	1471	7	2.1%	2.1%	2.1%	96.0%†	22.96	3.31	3108.90	8138.43	0.058	0.12	0.089	0.13	0.029	0.020
PolyFit	69	55	5	0%	88.9%	0.0%	54.0%*	17.31	1.41	6173.04	25941.63	0.29	0.15	0.51	0.16	0.29	0.156
Ours^P	214	592	1	100%	100%	100%	100%	18.67	2.19	3696.22	2700.15	0.14	0.14	0.24	0.17	0.15	0.11
TetWild	753	1611	5	63.0%	32.0%	32.0%	100%	24.26	2.85	1932.26	7011.41	0.029	0.094	0.062	0.11	0.050	0.083
fTetWild	773	1643	4	73.7%	38.9%	38.9%	95.0%*	24.21	2.77	2195.83	9562.01	0.037	0.13	0.069	0.14	0.059	0.12
ManifoldPlus	747	1610	3	24.0%	66.0%	64.0%	100%	25.14	2.49	559.84	1674.21	0.0060	0.0070	0.042	0.070	0.026	0.020
AlphaWrapping^1	804	1631	1	93.0%	100%	100%	100%	23.18	2.28	667.40	369.27	0.018	0.0085	0.059	0.06	0.037	0.024
AlphaWrapping^2	743	1510	1	100%	100%	100%	100%	25.06	2.66	327.32	184.00	0.0046	0.0058	0.042	0.068	0.032	0.024
Ours^Q	760	1631	2	58.0%	100%	100%	100%	22.90	2.23	716.04	413.22	0.020	0.0090	0.060	0.067	0.029	0.020
Ours	760	1631	2	100%	100%	100%	100%	25.21	2.49	310.30	169.58	0.0045	0.0045	0.037	0.067	0.022	0.020

Table 4.3: Statistics of the results generated for the entire dataset by all comparing low-poly meshing approaches, including the number of vertices (#V), the number of triangles (#F), the number of components (#C), the ratios between the number of meshes being self-intersection-free (r_f), manifold (r_m), watertight (r_w), successfully generated (r_s) and the 100 models in the dataset, and the average (ave) and standard deviation (sd) of the four visual preservation metrics, i.e., PSNR, LFD, SD, ND, and HD. We treat a case as a failure if the algorithm terminated with an exception (marked as $\star$), or reaches the timeout threshold (1h, marked as $\dagger$).

4.4.2 Comparisons

To demonstrate the effectiveness of our approach, we compare against ten competing methods, including two modules of the state-of-the-art commercial solution—Simplygon AB (2022), denoted as Simplygon^1 and Simplygon^2, four academic approaches, and four baselines by combining mesh repairing and simplification. Since only Simplygon^2 cannot exactly control the element count of the generated mesh, we compare all methods by matching the element counts of their results to those generated by Simplygon^2 with 200 as its parameter value.

Comparison with Commercial Software Simplygon AB (2022) can automatically generate simplified meshes and is popularly used by game studios. We compare our approach with both its mesh reduction (Simplygon^1) and re-meshing (Simplygon^2) modules. As shown in the Simplygon^1 and Simplygon^2 rows of Table 4.3, Simplygon can robustly process all meshes in the dataset, while Simplygon^2 generates better results than Simplygon^1 from basically all aspects but still introduces self-intersections and non-manifoldness for some models. In comparison, our approach not only guarantees the outputs are topologically clean and free of surface intersections, but also preserves the visual appearance much better, e.g., with 63.2% and 10.7% higher LFD, on average over the tested dataset than Simplygon^{1,2} respectively. Figure 4.11 illustrates their visual comparisons on two models.

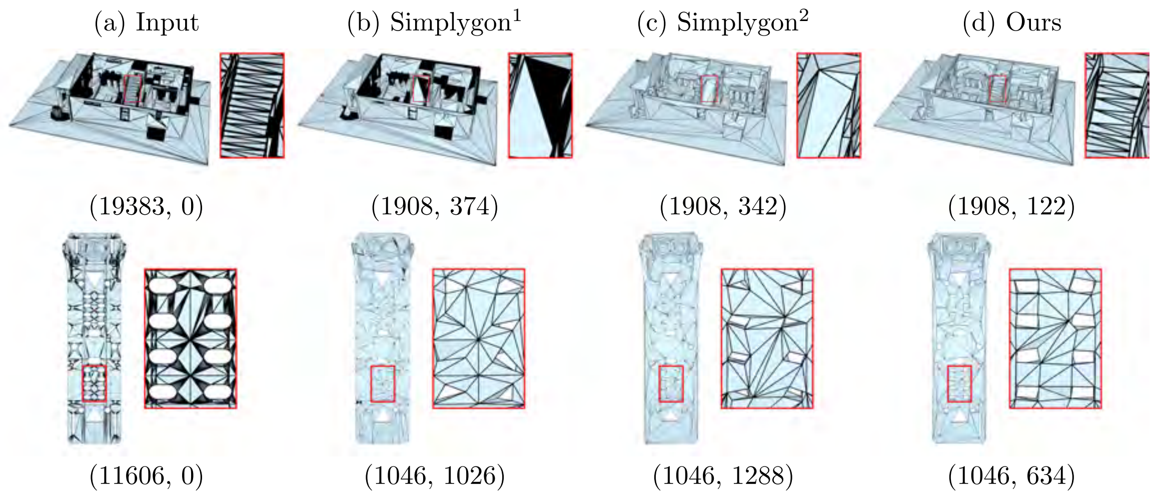


Figure 4.11: Comparison with Simplygon. (•, •) denotes (face number, light field distance). Notice that, some input meshes may have inconsistent face orientations, such as the mesh shown in the top row where back faces are rendered in black. From the zoomed-in regions in the top row, only our method keeps the features of the stairs. From the bottom zoomed-in row, our approach has the best match of the input.

Comparison with Academic Approaches We compare our algorithm with three state-of-the-art low-poly mesh generation methods, i.e., PolyFit Nan and Wonka

(2017), KSR Bauchet and Lafarge (2020) and Gao et al. (2022), and two typically used mesh simplification approaches, i.e., QEM module in MeshLab Cignoni et al. (2008) and the Blender decimation modifier. For PolyFit and KSR, we use the uniform sampling filter in MeshLab Cignoni et al. (2008) to simple 1M points on the input mesh. We use the built-in PolyFit API in CGAL with default parameters for final mesh generation, For the KSR method, in accordance with the authors' suggestion, we utilize the plane-extraction approach proposed by Yu and Lafarge (2022) and subsequently employ KSR for surface reconstruction. For all of these, we use the executable program provided in the authors' website[5] with default parameters. Note that PolyFit often generates meshes with much fewer triangles than the target value. In this case, we further simplify our algorithm to match the triangle numbers of their outputs, which are denoted as Ours^P. In contrast, KSR generates more triangles than the target value. In this case, we apply a post QEM step to simplify its output to the target triangle number. For QEM Cignoni et al. (2008), we first try to match the target triangle number with the topology preservation option turned on. We then turn it off if the simplification cannot reduce the element count to the desired value. As shown in Table 4.3, PolyFit fails to generate results for 46 out of 100 models due to the failure of planar feature detection, which is a challenge by itself; KSR and Gao et al. (2022)'s approach fail to provide any results for 4 and 13 out of 100 models respectively, within the computing time limit of 1h; QEM Cignoni et al. (2008) and Blender generate considerably worse results in terms of topology and geometry guarantees. As shown in Table 4.3, our approach not only has geometrical and topological guarantees, but also achieves the best visual similarity scores, with an LFD 95.0%, 90.0%, 58.6%, 74.5%, and 75.3% smaller than those generated by PolyFit, KSR, QEM, Blender, and Gao et al. (2022), respectively. This behaves similarly to the other metrics. We also demonstrate some visual results in Figure 4.12.

[5]GoCopp and KSR in https://team.inria.fr/titane/software/

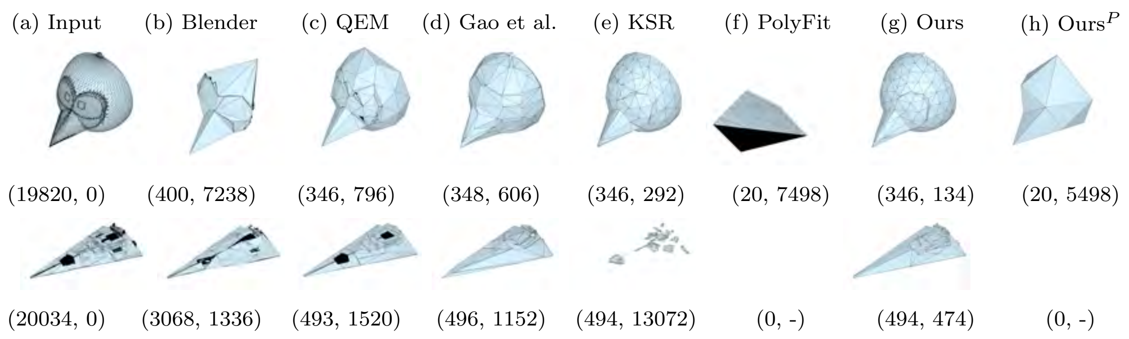


Figure 4.12: Comparison with academia and open-source solutions, where (•, •) denotes (face number, light field distance). The inverted faces are rendered as black. Note that, even after re-orientation using MeshLab Cignoni et al. (2008), inverted faces appears in the results of PolyFit. Besides, PolyFit also fails in the second example.

Comparison with Alternative Pipeline One alternative approach for low-poly meshing is to first repair the input surface to get a high-quality surface mesh through the various mesh repair methods Hu et al. (2018, 2020); Huang et al. (2020); Portaneri et al. (2022); Diazzi and Attene (2021), then apply a mesh simplification step (for example QEM Cignoni et al. (2008)) to reduce the element count to a specific number. We also show the comparison between our approach and four variants of this two-step process, i.e., TetWild Hu et al. (2018) + QEM, fTetWild Hu et al. (2020) + QEM, ManifoldPlus Huang et al. (2020) + QEM, AlphaWrapping Diazzi and Attene (2021) + QEM (AlphaWrapping[1]). For QEM, we first turn on the topology and normal preservation options, and set the target face number as the one from Simplygon[2]. If the QEM fails to simplify the mesh under these conditions, we turn off topology and normal options and simplify the mesh again. It is worth noting that removing the interior of other mesh repairing results did not affect the results since most of these approaches, such as TetWild Hu et al. (2018), and AlphaWrapping Portaneri et al. (2022) (AlphaWrapping), have either implicitly or directly removed the interior. However, ManifoldPlus Huang et al. (2020) produced inconsistent face orientations, requiring more complex interior removal approaches that we are not aware of. As demonstrated in Table 4.3 and Figure 4.21, the main drawback of this idea is that, although mesh-repairing approaches can fix the mesh to some extent, the follow-up simplification step will break the desired properties especially when the desired element count is small. For example, although MainfoldPlus Huang et al. (2020) and AlphaWrapping Portaneri et al. (2022) generate manifold and watertight mesh, respectively, the following simplification step breaks these guarantees. Notice that some mesh repairing methods also introduce issues in the meshes. For example, there are lots of self-intersections in MainifoldPlus's output of the first example of Figure 4.21. AlphaWrapping Portaneri et al. (2022) always generates a self-intersection-free surface, but it does not capture the sharp features in the input mesh, which leads to undesired visual appearances after simplification. TetWild Hu et al. (2018) and fTetWild Hu et al. (2020) can generate meshes with non-manifold configurations, which could be further repaired to be manifold at the cost of surface intersections Attene et al. (2009).

Additionally, we experiment with replacing parts of our algorithm with alternative methods. For instance, combining our mesh extraction with QEM (Ours^Q) leads to significantly inferior results compared to our original approach. Furthermore, when combining our mesh optimization with other mesh repair methods, such as AlphaWrapping (denoted as AlphaWrapping^2), the results exhibit comparable LFD values but worse HD outcomes.

To sum up, comparing to these baseline variants, our method ensures the generated mesh is topologically clean, geometrically self-intersection-free, and visually appearance preserving.

Timings We take about 7 minutes on average to finish the re-meshing of the entire tested dataset while the others take less than 2 minutes, except for Gao et al. (2022) (over 10 minutes) and the KSR method (over 15 minutes). In Figure 4.13, we further analyze the time costs of different stages of our approach: the edge flip step in the iso-surface extraction step(t_e); the other iso-surface steps(t_i); interior removal step(t_r);

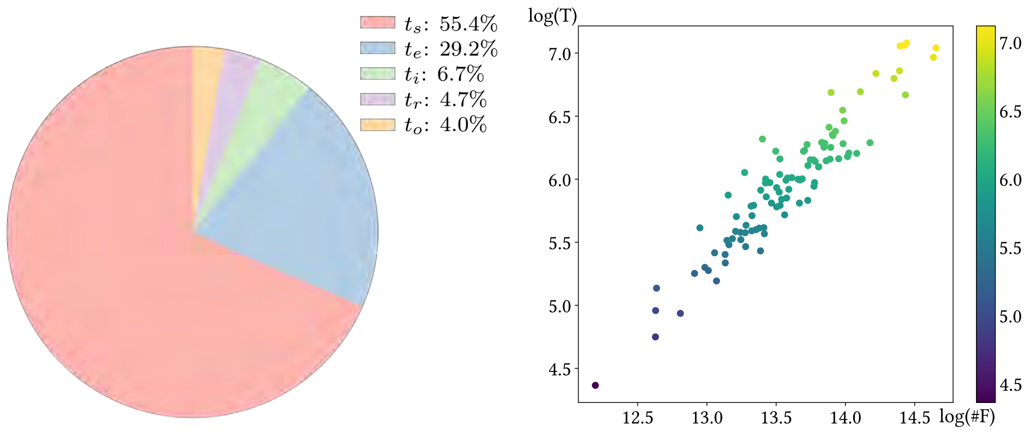


Figure 4.13: The time statistics for our method. *left:* the pie chat of time costed the different stages of our algorithm. All the symbols are defined in Section 4.4.2. *right:* the log-log (base e) plot of our time cost T (in seconds) and the face number ($\#F$) of extracted iso-surface. We find that the most time consuming parts (over 80%) are the edge-flip step (t_e) in iso-surface generation step and the mesh simplification (t_s) during the mesh optimization, where massive self-intersection checks are applied to ensure the desired intersection-free properties. This explains the strong positive correlation between the time consumption and the face number in extracted iso-surface. Indeed, the more faces you have, the more edge flip and collapse operations will be applied.

mesh simplification step(t_s); and the others, like flow, alignment, and I/O(t_o). We find that the most time-consuming part is the edge-flip (in iso-surface extraction) and mesh simplification (in mesh optimization) with self-intersection checks involved. It turns out that these two parts take over 80% of our entire process (See Figure 4.13), of which self-intersection check takes over 70% of the time. For this reason, the more faces the extracted iso-surface, the more edge flip and collapse operations will be conducted, ultimately leading to a higher cost. The right image of Figure 4.13 also reveals this positive correlation.

4.4.3 Parameters

Screen Size n_p We conduct a performance analysis in terms of user-specified screen size n_p and the corresponding iso-values d ($d = l/n_p$ as mentioned in Section 3.1). In Table 4.4, we report the average face number, timing and the visual metrics for

3 different choices of n_p. We notice that increasing n_p improves the visual similarity between our output and the input high-poly mesh, but at the same time, it will cost more time and end up with a larger number of faces. Figure 4.14 also provides an illustration.

n_p	#F	Time(s)		PSNR		LFD		SD		ND		HD	
		ave	sd	ave	sd	ave	sd	ave	sd	ave	sd	ave	sd
50	139	30.40	8.82	20.67	2.26	1298.14	956.31	0.030	0.015	0.093	0.070	0.047	0.027
100	408	99.92	39.13	22.45	2.33	656.88	320.81	0.014	0.0086	0.062	0.067	0.033	0.022
200	1193	440.12	234.35	24.40	2.48	367.14	190.88	0.0065	0.0058	0.042	0.066	0.022	0.020

Table 4.4: The statistics for low-poly meshes generated with different screen sizes (n_p). Increasing the screen size results in a better re-meshing result, but leads to a larger face number and a slower solving speed.

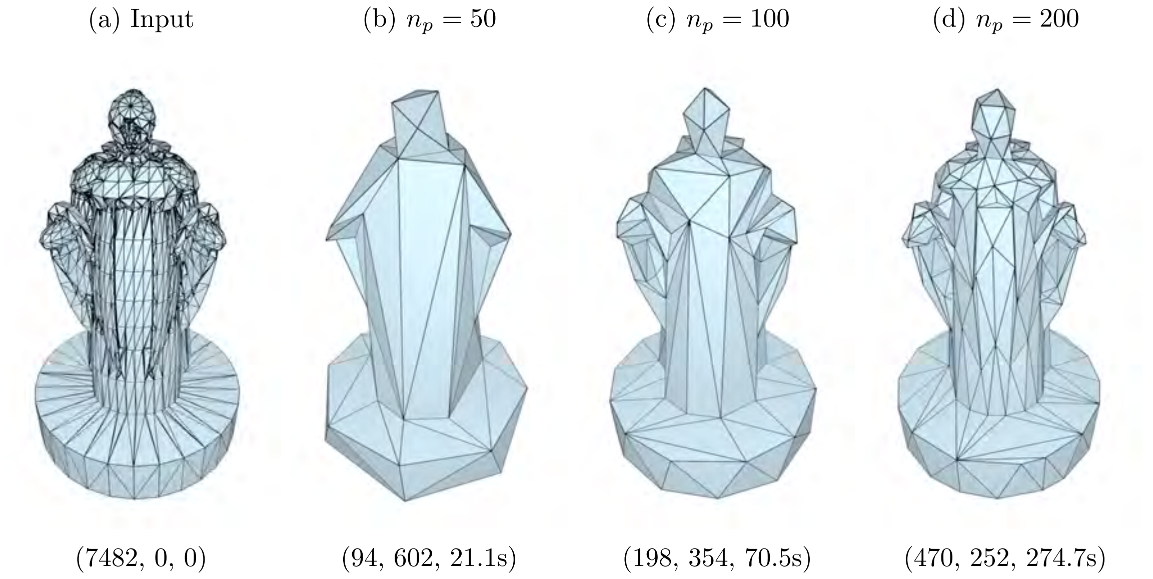


Figure 4.14: The re-meshed results w.r.t. different user-specified distance tolerance, where l is the diagonal length of the bounding box of input mesh. $(\bullet, \bullet, \bullet)$ denotes (face number, light field distance, time cost). The smaller the tolerance is, the better re-meshed result we will get, but at the same, the computational cost grows.

Voxel Size Given a screen size n_p, different voxel size will lead to different results. In Figure 4.15, we compared the extracted iso-surface results using different voxel sizes for a fixed iso-value $d = l/n_p$, where l is the diagonal length of the bounding box and $n_p = 200$. As we argued in Section 4.3.1, too large voxels may lead to the missing parts of the extracted iso-surface (second left image), while too small voxels will slow down the extraction (rightmost image). To achieve a trade-off between efficiency and performance, we set the diagonal length of voxels to be equal to our offset distance.

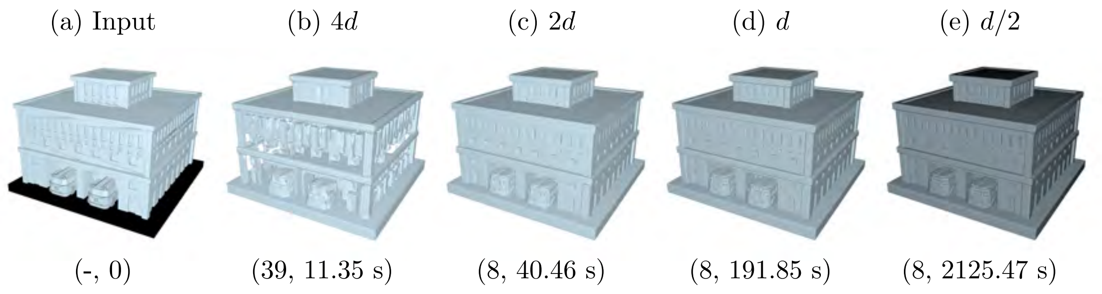


Figure 4.15: The different extracted iso-surface for a fixed offset distance using different voxel sizes. $(\bullet, \bullet)$ denotes (#genus, time cost). The "-" means that the input mesh is non-manifold. The black bottom in the first figure is due to inversed face orientation. As observed, a larger voxel size (e.g., 4d) produces a high-genus surface. Reducing the voxel size captures finer details but increases the computational cost.

Flow Step Fractional Ratio r As we state before, during the geometric flowing process, we multiply the flow direction by a fraction r to allow more moving space for the entire mesh and to achieve a better-optimized result. In practice, we find that a smaller step size will lead to a better visual similarity (a smaller *light field distance*) between the output mesh $\boldsymbol{M}_o$ and the input mesh $\boldsymbol{M}_i$, but at the cost of a larger number of triangles. From the test shown in Figure 4.16, we empirically choose $\frac{1}{8}$ as the default value to achieve a good balance between a low element count and a high visual similarity to the input.

Feature Curve Length l_0 In Figure 4.17, we show different iso-surface results based on different choices of feature-line length. Increasing this threshold does gradually solve the "saw-tooth" issue of the initially extracted iso-surface (Figure 4.17b). At the same time, it will blur some sharp features. In practice, we find $l_0 = 4$ is a

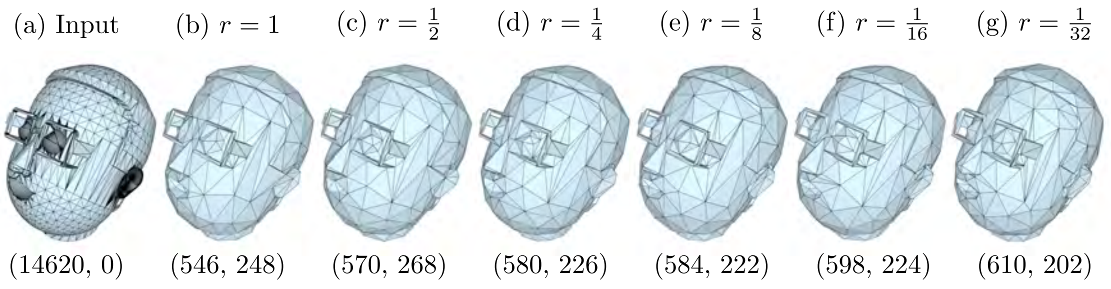


Figure 4.16: The different results using different flow step fractional ratio r. $(\bullet, \bullet)$ denotes (#faces, LFD). As we can see, decreasing r will lead to a smaller LFD, but in turns, it will produce an output with larger number of faces.

choice of ideal trade-off. A better understanding of choosing these parameters needs further exploration, we leave this as future work.

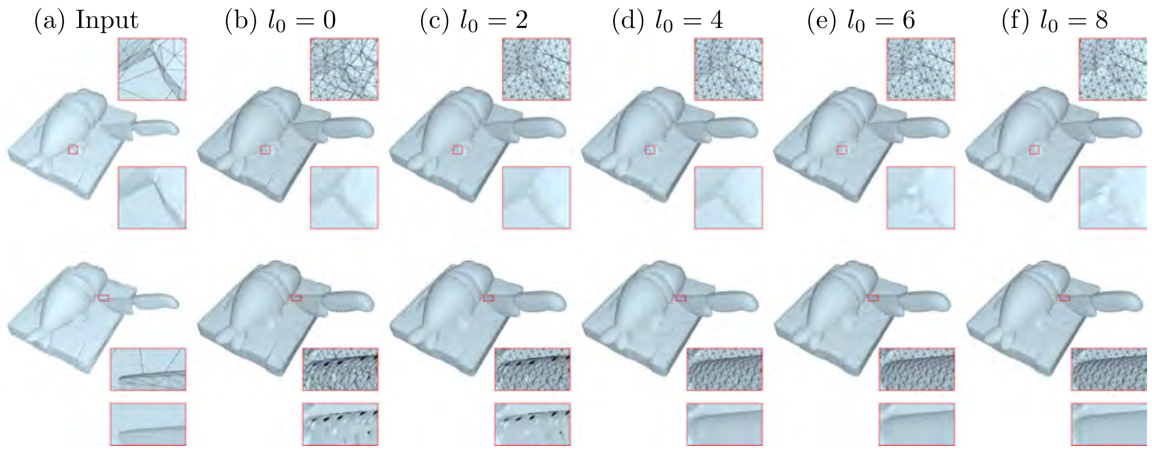


Figure 4.17: Different feature-line length threshold l_0 leads to different results, where we also show the zoomed-in regions without the wireframes for a better visualization. The red-framed top and bottom rows are the same models with different rendering. As we can see, larger threshold does smooth out the geometry (the black regions disappears), but at the same time, some of sharp features are blurred. $l_0 = 4$ achieves a trade-off between these considerations, thus we choose this as our default parameter.

4.5 Additional Applications

4.5.1 Iso-surface Extraction Comparison

The mesh extraction step of our algorithm can be independently useful, where many competing algorithms have been proposed in the past as shown in Table 4.1. We show the advantage of our mesh extraction algorithm by comparing our approach with: 1) MC33 Chernyaev (1995), 2) EMC Kobbelt et al. (2001), 3) DC Ju et al. (2002), and 4) Manson and Schaefer (2010)'s approach. The first three serve as the baselines, and the last one meets all the desired properties listed in Table 4.1. In order to apply these algorithms to any input mesh $\boldsymbol{M}_i$, we convert the input mesh $\boldsymbol{M}_i$ to be an implicit function by Equation 4.1, and the corresponding Hermite data (Section 4.3) for DC. We modified the EMC algorithm provided in Mario Botsch (2015), adapt the Vega et al. (2019)'s implementation of MC33, use the embedded DC function in libigl Jacobson et al. (2018), and choose the Manson and Schaefer (2010)'s own implementation[6] to generate the corresponding results. In Figure 4.22, we show the extracted iso-surface in terms of the different iso-values: $l/50$, $l/100$, $l/200$ and $l/400$, where l is the diagonal length of the bounding box of $\boldsymbol{M}_i$. Among all of these examples, we use the same grid resolution as ours, except for Manson and Schaefer (2010)'s approach, where the default octree settings are used. One thing to point out is that although all approaches generate reasonable results, prior works suffer from several drawbacks: MC33 can generate a closed and self-intersection-free manifold surface, but cannot capture the sharp creases especially when the grid resolution is low (see the zoom-in of Figure 4.22); EMC and DC recover the sharp features, but they either may lead to self-intersections or have no guarantees of the manifoldness and self-intersection-free properties (see Section 4.2.2 for more detailed discussion); Manson and Schaefer (2010)'s approach may generate iso-surface with an undesired high genus (circled regions in Figure 4.22).

[6]`http://josiahmanson.com/research/iso_simplicial/`

4.5.2 Cage Generation

Our re-meshing scheme can be easily adapted to generate cages for the input mesh, without any requirements for it to be manifold, watertight, or self-intersection-free. The cage mesh has to fully enclose but not penetrate a 3D model. We can easily achieve this by adding a penetration check during our mesh optimization step. More specifically, every time we update a vertex position in the flow and alignment steps, we simply modify Algorithm 4 by adding one more intersection check between the current mesh and the input. We reject any edge collapse that leads to the intersection with the input. These additional checks can be handled efficiently by classical BVH-based collision detection Karras (2012). In Figure 4.18, we show the cage generated by our algorithm. Unlike the automatic caging algorithm Sacht et al. (2015) requiring the input to be watertight, self-intersection-free, and manifold, our algorithm makes no assumptions of the input mesh. For example, the input model in Figure 4.18 has 32 non-manifold edges and 264 intersecting triangle pairs.

To compare with Sacht et al. (2015) more thoroughly, we run both approaches to generate cages for a dataset Gao et al. (2019) containing 93 meshes with clean topology and geometry that is required by Sacht et al. (2015)'s approach. We use the author-provided code to generate a cage with their E_{varap} energy (see Section 3.2 of Sacht et al. (2015) for details). We run both methods by setting the number of triangles of the final cage to be 2000 and the computing time limit of 1h. As shown in Table 4.5, Sacht et al. (2015)'s solution returns run time error for 20 models and fails to produce any results within the time limit for 5 models. At the same time, our approach successfully generates a tighter cage (smaller Hausdorff distance) for the entire dataset. Notice that, some cages generated by Sacht et al. (2015) have really bad artifacts, for example, the "spikes" shown in Figure 4.19. We also reported the updated statistics after manually removing these models in the last two rows of Table 4.5.

Methods	r_s	$\text{HD}_{c\to i}$		$\text{HD}_{i\to c}$		HD	
		ave	sd	ave	sd	ave	sd
Ours	100%	0.21	0.58	0.20	0.53	0.22	0.59
Sacht et al.	73.1%	1.11	7.17	38.03	306.89	38.03	306.89
Ours*	100%	0.14	0.36	0.13	0.32	0.15	0.37
Sacht et al.*	74.7%	0.15	0.33	0.19	0.45	0.19	0.45

Table 4.5: The Hausdorff distance statistics. $\text{HD}_{c\to i}$ is the Hausdorff distance from generated cage to the input mesh, $\text{HD}_{i\to c}$ is the distance from the opposite direction, and $\text{HD} = \max(\text{HD}_{c\to i}, \text{HD}_{i\to c})$. r_s is the successful ratio. Sacht et al. (2015) failed to produce the results for 20 out of 93 models due to the run time error, for 5 out of 93 models since exceeding the time threshold.

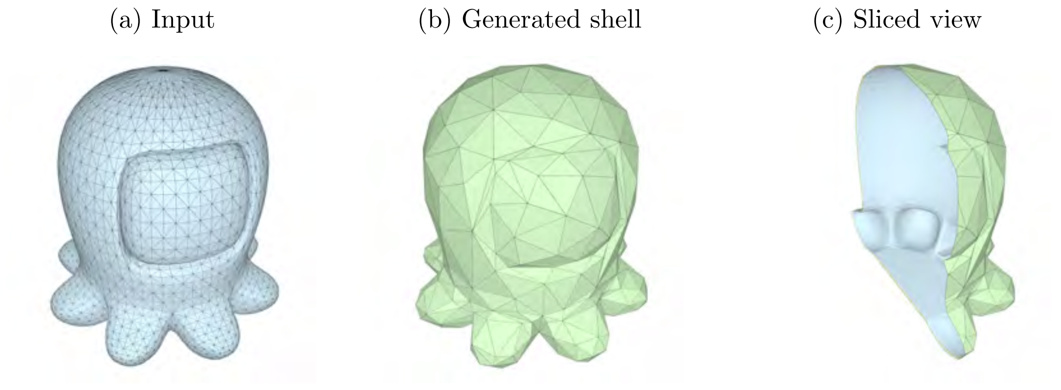


Figure 4.18: The generated shell for a cartoon octopus.

4.6 Conclusion, Limitations, and Future Works

In this chapter, we propose a robust approach to generate low-poly representations of any input mesh. Our approach can be decomposed into two independently useful stages: 1) the iso-surface extraction stage (re-meshing), where we extract a water-tight, feature-preserving, and self-intersection-free iso-surface of the input mesh with any user-specific iso-value; 2) a mesh optimization stage, where we alternatively re-mesh and flow the extracted the surface to meet the desired properties: low-resolution and visually close to the input mesh. Although we currently cannot guarantee the deviation bound, our algorithm effectively adheres to it, with a Hausdorff distance (HD) of $4.4d$ for the dataset, where d represents the offset distance.

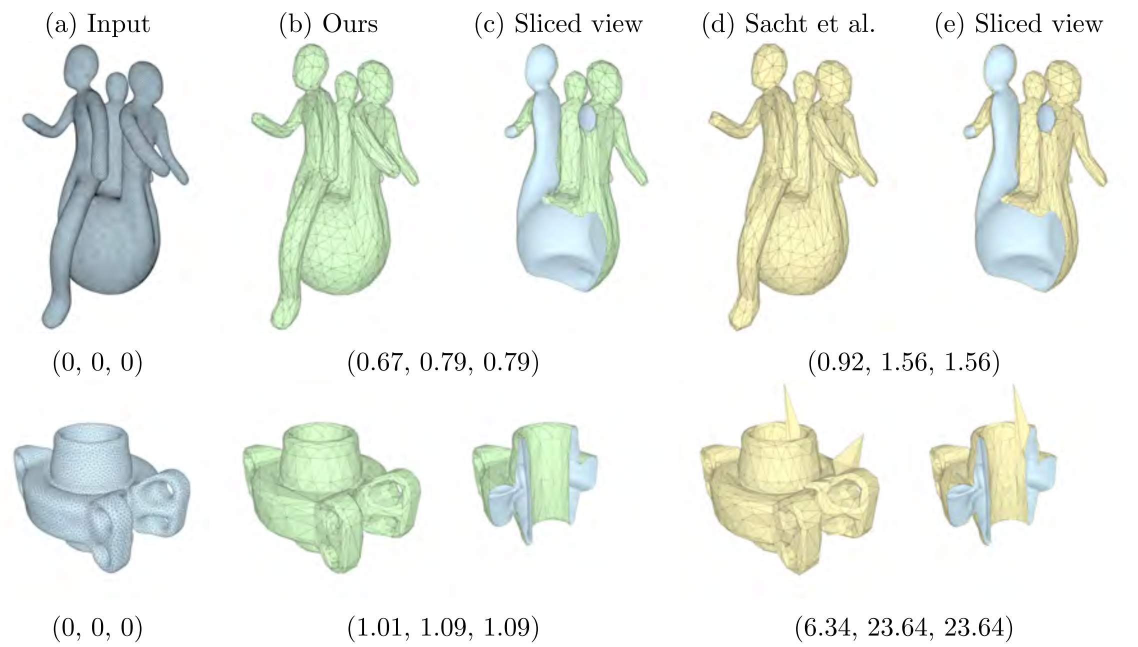


Figure 4.19: Comparison with Sacht et al. (2015). (•, •, •) denotes the Hausdorff distance from cage to input, from input to cage, and between input and cage, respectively. Even if the input mesh is water-tight, Sacht et al. (2015) may end up with bad cage shape (bottom row).

Scalability Figure 4.20 illustrates the relationship between screen size and the time and memory requirements for the tree model shown in Figure 4.2. While our approach successfully produces results for larger screen sizes, it does not demonstrate optimal scalability in terms of memory and time efficiency. The primary reason for memory consumption is the dense grid generation, which accounts for over 70% of memory usage. We believe that using a sparse grid implementation will alleviate this issue, and we plan to explore this as a future engineering improvement. Regarding efficiency, as shown in Table 4.4, our approach spends the majority of its time (over 80%) on simplification and edge flip steps during iso-surface extraction. These steps involve numerous intersection checks, and a parallel implementation could significantly accelerate the process. Additionally, our current iso-surface extraction relies on a CPU-based algorithm, and we aim to develop a GPU-based version in future work.

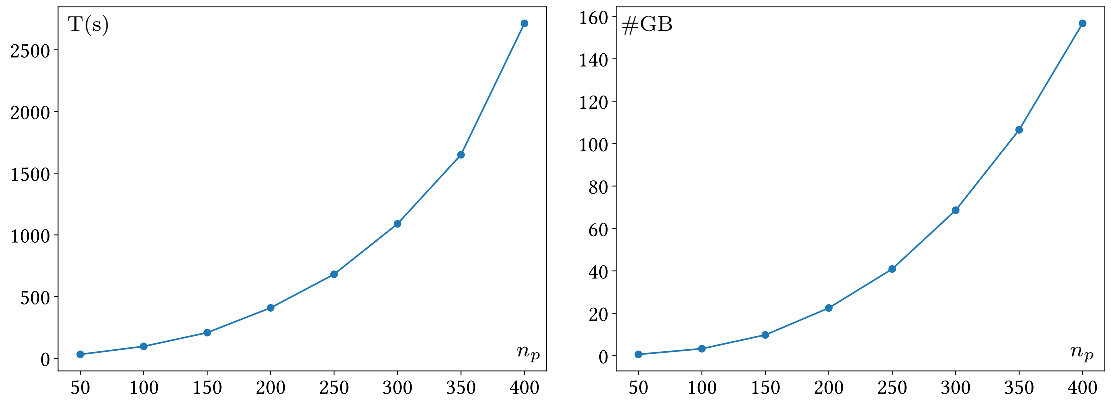


Figure 4.20: Time and memory consumption in terms of screen size for the tree model in Figure 4.2.

Manifoldness, Self-intersection-freeness, Watertightness Our approach guarantees to produces intersection-free, manifold, and watertight outputs. Ensuring intersection-free and manifold properties facilitates easier UV unwrapping and minimizes visible appearance artifacts during texture baking. However, when dealing with an open input mesh, watertightness might not be essential, where a post mesh segmentation process could be employed to remove the redundant faces to carve out the open region.

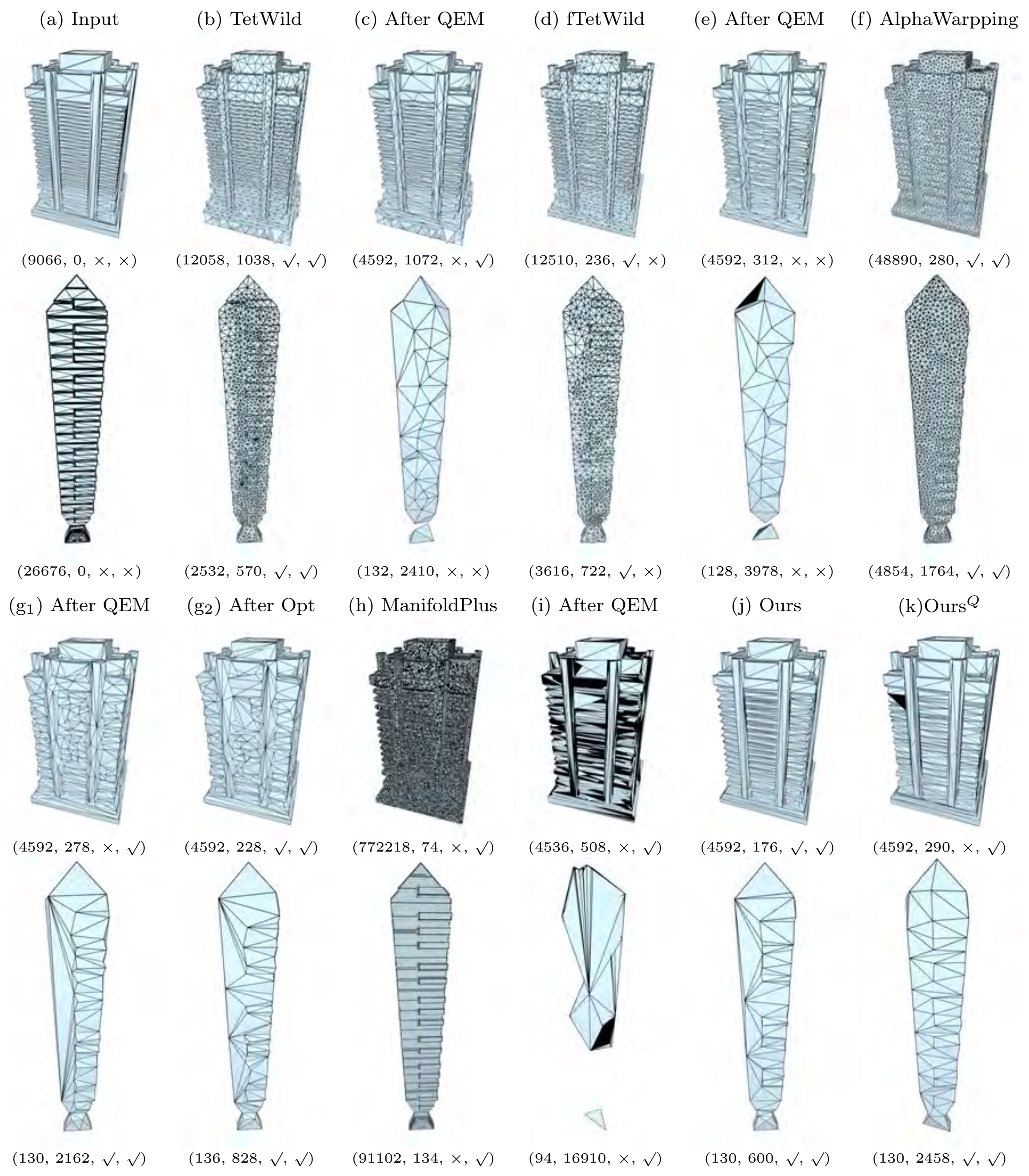


Figure 4.21: Comparison with variants of the pipeline of first mesh repairing and then mesh simplification. For (g_2), we apply the proposed mesh optimization on the AlphaWrapping (AW) output; for (k), we combine our mesh extraction with QEM simplification. All the back faces are rendered black. (•, •, •, •) indicates (face number, light field distance, self-intersection-free flag, manifoldness flag).

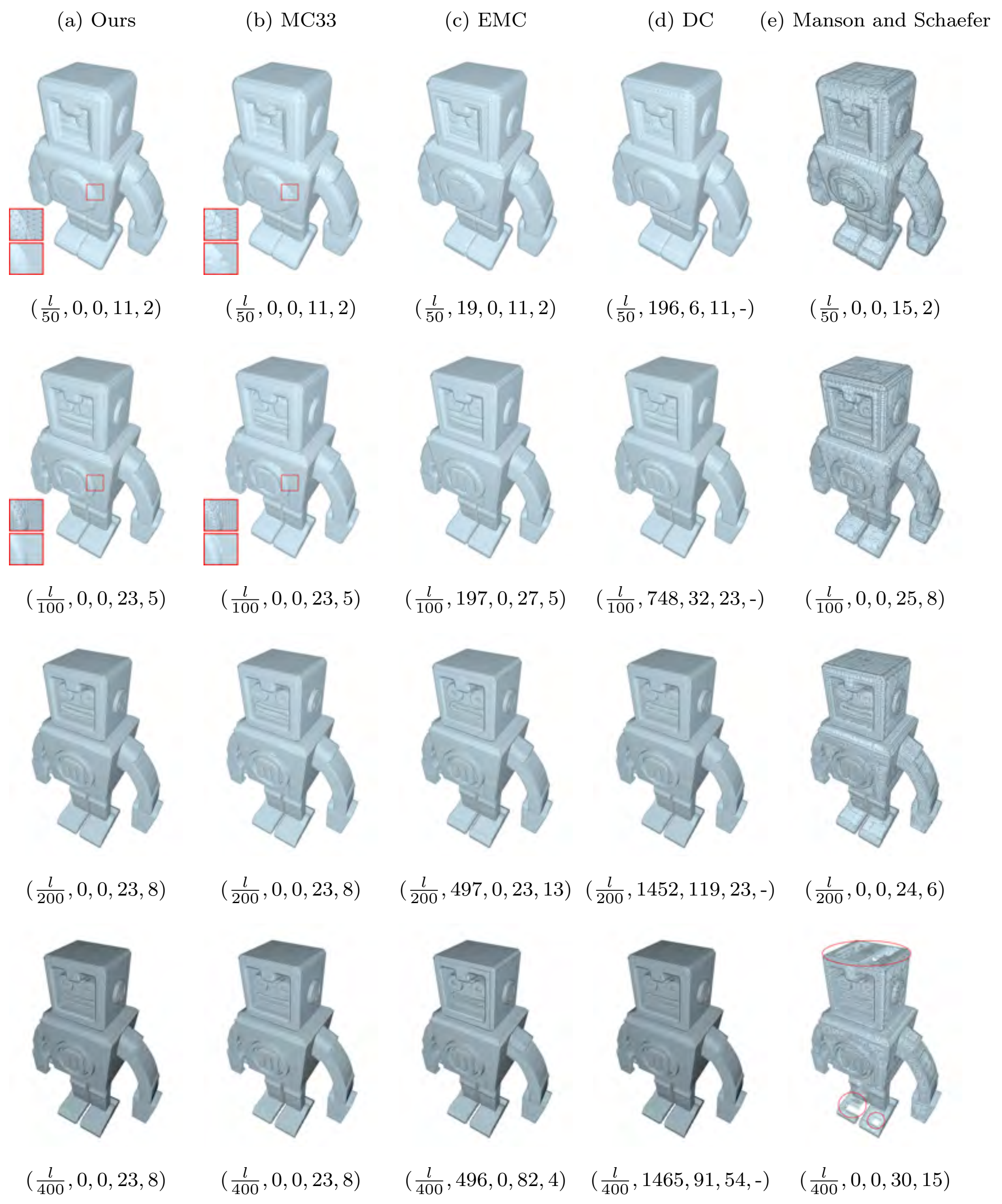

(a) Ours
(b) MC33
(c) EMC
(d) DC
(e) Manson and Schaefer
$(\frac{l}{50}, 0, 0, 11, 2)$
$(\frac{l}{50}, 0, 0, 11, 2)$
$(\frac{l}{50}, 19, 0, 11, 2)$
$(\frac{l}{50}, 196, 6, 11, \text{-})$
$(\frac{l}{50}, 0, 0, 15, 2)$
$(\frac{l}{100}, 0, 0, 23, 5)$
$(\frac{l}{100}, 0, 0, 23, 5)$
$(\frac{l}{100}, 197, 0, 27, 5)$
$(\frac{l}{100}, 748, 32, 23, \text{-})$
$(\frac{l}{100}, 0, 0, 25, 8)$
$(\frac{l}{200}, 0, 0, 23, 8)$
$(\frac{l}{200}, 0, 0, 23, 8)$
$(\frac{l}{200}, 497, 0, 23, 13)$
$(\frac{l}{200}, 1452, 119, 23, \text{-})$
$(\frac{l}{200}, 0, 0, 24, 6)$
$(\frac{l}{400}, 0, 0, 23, 8)$
$(\frac{l}{400}, 0, 0, 23, 8)$
$(\frac{l}{400}, 496, 0, 82, 4)$
$(\frac{l}{400}, 1465, 91, 54, \text{-})$
$(\frac{l}{400}, 0, 0, 30, 15)$

Figure 4.22: The comparison of different iso-surface extraction method. $(\bullet, \bullet, \bullet, \bullet, \bullet)$ are (iso-value, #self-intersected faces, #non-manifold edges, #comps, #genus), where l is the diagonal length of the bounding box of the input mesh. "-" means the genus is not well defined given the mesh is not manifold

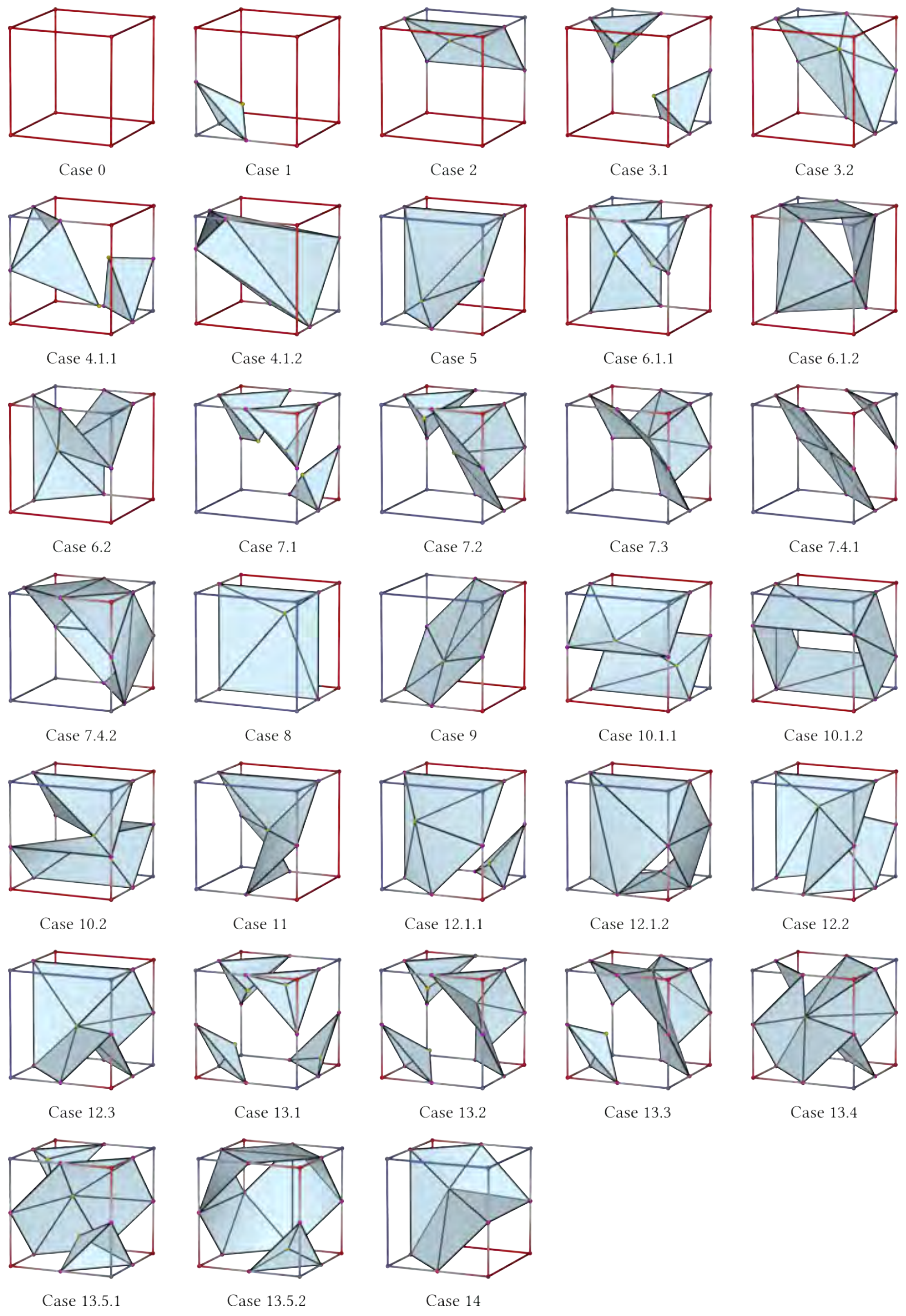


Figure 4.23: We show our lookup table. The cube vertices are colored by their signs, with red for positive and blue for negative. The surface-cube intersections on edges are the pink points, while the yellow points are the inserted feature points. Notice that case 12.2 and 12.3 are symmetric, as well as case 11 and 14.

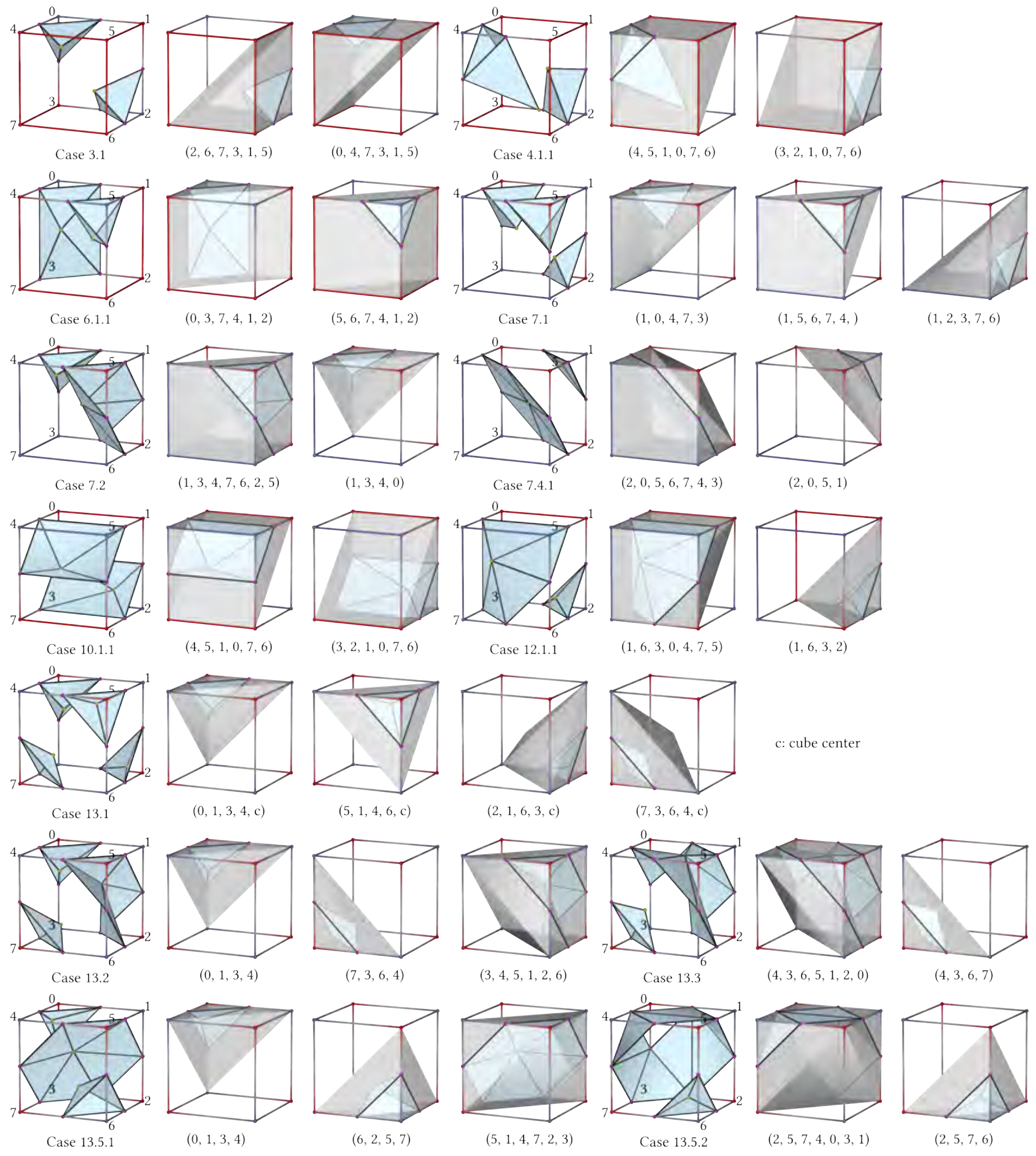


Figure 4.24: Cube division policies for MC33 cases with more than one components. The numbers inside the parentheses are the cube vertices which form the constraint polyhedra (rendered in gray) for the corresponding components (rendered in blue)

Chapter 5: Conclusion and Future Directions

Simulating and manipulating wrinkles plays a pivotal role in computer graphics, fashion design, and artistic endeavors. Yet, the prevalent tools in this domain encounter a significant trade-off between detail and computational efficiency. Low-resolution meshes fall short in capturing intricate details, whereas high-resolution counterparts, despite their accuracy, compromise on efficiency. This dilemma is further exacerbated by the complexities of collision and friction dynamics. The models and algorithms delineated in this thesis offer innovative solutions to these challenges, thereby advancing the field of wrinkle simulation.

Chapter 2 introduced the Tension Field + Wrinkles (*TFW*) model, a groundbreaking method for depicting detailed wrinkles on coarse base meshes. This model leverages spectral representation to capture high-frequency wrinkles, a novel approach that builds upon the established elastic model (*StVK*). Through rigorous experimentation, we validated the *TFW* model's accuracy and effectiveness, particularly in the static simulation of high-frequency wrinkles.

Furthering this innovation, Chapter 3 presented the Complex Wrinkle Field (*CWF*), coupled with state-of-the-art interpolation and upsampling algorithms. This development facilitates the rapid generation of high-frequency wrinkle animations on a single CPU and supports the design of wrinkle patterns with specific directions and amplitudes, underscoring its industrial potential.

In pursuit of a suitable coarse base mesh, Chapter 4 detailed an algorithm capable of transforming any high-resolution mesh into a coarse, manifold, and intersection-free version, independent of its original topology and geometry. Demonstrated on a subset of the Thingi10K models, our method not only achieved a 100% success rate but also surpassed the performance of commercial (e.g., Simplygon), open-source software, and other academic algorithms. This technique proves invaluable not only for mesh generation for *CWF* and *TFW* models but also holds extensive applicability in game development, particularly for Level of Detail (*LOD*) modeling.

The contributions of this thesis signify a novel approach to representing and understanding high-frequency wrinkles, thereby enabling a reduction in the degrees of freedom. Additionally, our remeshing and repair algorithm facilitates the handling of real-world data. Nevertheless, these advancements merely scratch the surface of what's possible in real-time simulation of real-world data. Two primary concerns warrant further investigation: the robustness of simulating poorly meshed thin shells and the efficiency of dynamic simulations.

Addressing robustness, traditional remeshing techniques, while effective, often introduce extra collisions and inadvertently alter the geometry. A promising solution lies in the adoption of intrinsic triangulation, specifically intrinsic Delaunay retriangulation. This approach preserves the underlying geometry while enhancing mesh quality, thereby offering a potential solution to classical remeshing challenges. Exploring simulation models that operate on this representation could revolutionize the handling of geometric data.

Efficiency remains another critical area for exploration. While the proposed *StVK* and *CWF* models adeptly address keyframe interpolation for animation, the dynamic aspects of *CWF* require further development. Pursuing research in GPU acceleration could open new avenues for efficient simulation. Additionally, exploring generative models for simulating wrinkles might offer novel insights and methodologies for future research. For example, rather than solely generating mesh models, an intriguing approach would be to incorporate physical properties throughout the process. This could involve creating models with varied stiffness or incorporating equilibrium as one of the training objectives to ensure the generated models are physically stable.

In summary, this thesis lays the groundwork for future studies in wrinkle simulation, addressing both the technical and conceptual challenges within the field.

The journey towards real-time, real-world data simulation is fraught with challenges, yet the potential rewards for computer graphics, fashion design, and beyond are immense.

Appendix A: Fine Wrinkling on Coarsely Meshed Thin Shells Appendices

A.1 Kirchhoff-Love Shells

$$\boldsymbol{s}(u,v,t) = \boldsymbol{r}(u,v) + t\hat{\boldsymbol{n}}(u,v)$$

where $\hat{\boldsymbol{n}} = \frac{\boldsymbol{r}_u \times \boldsymbol{r}_v}{\|\boldsymbol{r}_u \times \boldsymbol{r}_v\|}$ is the unit normal of the midsurface. This map induces a (volumetric) metric $\boldsymbol{g}$ on the slab $U \times [-\frac{h}{2}, \frac{h}{2}]$:

$$\boldsymbol{g} = \begin{bmatrix} \boldsymbol{I} - 2t\boldsymbol{II} + O(t^2) & 0 \\ 0 & 1 \end{bmatrix}, \tag{A.1}$$

where

$$\boldsymbol{I} = \mathrm{d}\boldsymbol{r}^T \mathrm{d}\boldsymbol{r}, \;\; \boldsymbol{II} = -\mathrm{d}\boldsymbol{r}^T \mathrm{d}\hat{\boldsymbol{n}}$$

are the first and second fundamental forms of the midsurface.

If the residual strains in the shell are linear in the thickness direction, the shell's rest state can be recorded in terms of a "rest metric" with similar expression van Rees et al. (2017); Chen et al. (2018a):

$$\overline{\boldsymbol{g}} = \begin{bmatrix} \boldsymbol{I}_u - 2t\boldsymbol{II}_u & 0 \\ 0 & 1 \end{bmatrix}. \tag{A.2}$$

Notice that if we take the rest state as the parameter domain, which is almost always the case for sewn garments, then $\boldsymbol{I}_u = \boldsymbol{id}$ and $\boldsymbol{II}_u = 0$.

For a St. Venant-Kirchhoff material, and assuming in-plane strain is $O(h)$, the elastic energy in Equation (2.1) can be derived from the above setup, by integrating through the midsurface direction, and dropping energy terms of order higher than cubic Weischedel (2012).

As mentioned in the main text, the matrix norm in the Koiter energy describes the elastic constitutive law,

$$\|M\|^2_{\mathrm{SV}} = \frac{\alpha}{2}\,\mathrm{tr}^2(M) + \beta\,\mathrm{tr}(M^2),$$

where the Lamé parameters are related to the Young's modulus Y and Poisson's ratio ν by

$$\alpha = \frac{Y\nu}{1-\nu^2}, \beta = \frac{Y}{2(1+\nu)}. \tag{A.3}$$

We discretize the first fundamental form as piecewise constant, derived directly from the definition $\boldsymbol{I} = \mathrm{d}\boldsymbol{r}^T\mathrm{d}\boldsymbol{r}$ and the fact that $\boldsymbol{r}$ is linear within each triangle face $\mathbf{f}_{ijk}$:

$$\boldsymbol{I} = \begin{bmatrix} \|\mathbf{v}_j - \mathbf{v}_i\|^2 & (\mathbf{v}_j - \mathbf{v}_i)\cdot(\mathbf{v}_k - \mathbf{v}_i) \\ (\mathbf{v}_k - \mathbf{v}_i)\cdot(\mathbf{v}_j - \mathbf{v}_i) & \|\mathbf{v}_k - \mathbf{v}_i\|^2 \end{bmatrix},$$

Here the first fundamental form is expressed in the triangle's own barycentric coordinates.

For the second fundamental form, we follow the discretization of Grinspun et al. (2003) and others Weischedel (2012); Chen et al. (2018a) based on jumps in the mid-edge normal:

$$\boldsymbol{II}_b = 2\begin{bmatrix} (\mathbf{v}_j - \mathbf{v}_i)\cdot(\hat{\mathbf{n}}_j - \hat{\mathbf{n}}_i) & (\mathbf{v}_k - \mathbf{v}_i)\cdot(\hat{\mathbf{n}}_j - \hat{\mathbf{n}}_i) \\ (\mathbf{v}_k - \mathbf{v}_i)\cdot(\hat{\mathbf{n}}_j - \hat{\mathbf{n}}_i) & (\mathbf{v}_k - \mathbf{v}_i)\cdot(\hat{\mathbf{n}}_k - \hat{\mathbf{n}}_i) \end{bmatrix},$$

where $\hat{\mathbf{n}}_i$ is the mid-edge normal on the edge $\mathbf{e}_{jk}$ opposite to $\mathbf{v}_i$ on face $\mathbf{f}_{ijk}$. We take this mid-edge normal to be

- the unit face normal if $\mathbf{e}_{jk}$ is a boundary edge;
- the average of the face normals on the two adjacent faces of $\mathbf{e}_{jk}$, otherwise.

A.2 Tension Field Theory

Given a parameter domain $U \subset \mathbb{R}^2$ with material metric $\boldsymbol{I}_u$ as described in Appendix A.1, the stretching energy density W_s in Equation (2.2) can be written as:

$$\begin{aligned} W_s(\lambda_1, \lambda_2) &= \|\boldsymbol{I}_u^{-1}\boldsymbol{I} - \boldsymbol{id}\|^2_{\mathrm{SV}} \\ &= \frac{\alpha}{2}(\lambda_1 + \lambda_2)^2 + \beta(\lambda_1^2 + \lambda_2^2) \end{aligned}. \tag{A.4}$$

Here $\lambda_{1,2}$ are the eigenvalues of the Green strain $\boldsymbol{I}_u^{-1}\boldsymbol{I} - \boldsymbol{id}$. Without loss of generality, we assume $\lambda_1 \geq \lambda_2$.

In tension field theory, this stretching energy density is modified to be identical to W_s in regions of pure tension, but so that the material exerts no force resisting compressive stress. In terms of the principal strains, this behavior is captured by the formula

$$\tilde{W}_s(\lambda_1, \lambda_2) = \begin{cases} 0, & \lambda_1 < 0, \lambda_2 < 0 \\ W_s\left(\lambda_1, \hat{\lambda}_2(\lambda_1)\right), & \lambda_1 \geq 0, \lambda_2 < \hat{\lambda}_2(\lambda_1) \\ W_s(\lambda_1, \lambda_2), & \lambda_1 \geq 0, \lambda_2 \geq \hat{\lambda}_2(\lambda_1), \end{cases}$$

where

$$\hat{\lambda}_2(\lambda_1) = \arg\min_{\lambda_2} W_s(\lambda_1, \lambda_2).$$

See Montes et al. (2020) for the full derivation of these expressions.

A.3 In-plane Wrinkle Correction Calculation

In this section, we use the notation and definitions of Section 2.3.4 for $\boldsymbol{w}, \boldsymbol{w}^\perp$ and $\mathrm{d}\phi^\perp$, and derive the expressions for $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ cited in Chapter 2.

First, notice that we can zero out the term (2.15) by simply setting

$$\boldsymbol{v}_2 = \frac{a^2}{8} \boldsymbol{I}_b^{-1} \mathrm{d}\phi^T.$$

The $\boldsymbol{v}_1$ term is more involved. Writing $X = -2a\boldsymbol{II}_b$ and $\boldsymbol{y} = \boldsymbol{I}_b \boldsymbol{v}_1$, and using the notation $[M]_T = M + M^T$, minimizing the term (2.14) amounts to solving:

$$\min_{\boldsymbol{v}_1 \in \mathbb{R}^2} \left\| \boldsymbol{I}_u^{-1} \left(X + \left[\boldsymbol{y} \mathrm{d}\phi^T \right]_T \right) \right\|_{\mathrm{SV}}^2 \tag{A.5}$$

One can check that, for any symmetric matrix M,

$$\begin{aligned} \|\boldsymbol{I}_u^{-1} M\|_{\mathrm{SV}}^2 = & \frac{\alpha}{2} \left(\|\hat{\boldsymbol{w}}\|_M^2 + \left\|\hat{\boldsymbol{w}}^\perp\right\|_M^2 \right) \\ & + \beta \left(\|\hat{\boldsymbol{w}}\|_M^2 + 2 \left(\hat{\boldsymbol{w}}^T M \hat{\boldsymbol{w}}^\perp \right)^2 + \left\|\hat{\boldsymbol{w}}^\perp\right\|_M^2 \right) \end{aligned} \tag{A.6}$$

using the fact

$$\mathrm{tr}(\boldsymbol{I}_u^{-1} M) = \hat{\boldsymbol{w}}^T M \hat{\boldsymbol{w}} + \left(\hat{\boldsymbol{w}}^\perp\right)^T M \hat{\boldsymbol{w}}^\perp$$

where $\hat{\boldsymbol{w}}$ and $\hat{\boldsymbol{w}}^\perp$ are the normalized $\boldsymbol{w}$ and $\boldsymbol{w}^\perp$ under $\boldsymbol{I}_u$ norm. By setting (recall: $\boldsymbol{w} = \boldsymbol{I}_u^{-1} d\phi^T$)

$$\tilde{\boldsymbol{y}} = \|\boldsymbol{w}\|_{\boldsymbol{I}_u} \boldsymbol{y}, \quad M = X + [\tilde{\boldsymbol{v}} \hat{\boldsymbol{w}}^T \boldsymbol{I}_u]_T$$

and representing $\tilde{\boldsymbol{y}}$ in the basis $\{\boldsymbol{I}_u \hat{\boldsymbol{w}}, \boldsymbol{I}_u \hat{\boldsymbol{w}}^\perp\}$,

$$\tilde{\boldsymbol{y}} = \tilde{x}_1 \boldsymbol{I}_u \hat{\boldsymbol{w}} + \tilde{x}_2 \boldsymbol{I}_u \hat{\boldsymbol{w}}^\perp,$$

Equation (A.5) can be converted into a quadratic problem

$$\min_{\tilde{x}_1, \tilde{x}_2} \frac{\alpha}{2}(c_1 + 2\tilde{x}_1 + c_2)^2 + \beta\left((c_1 + 2\tilde{x}_1)^2 + c_2^2 + 2(c_3 + \tilde{x}_2)^2\right)$$

where $c_1 = \hat{\boldsymbol{w}}^T X \hat{\boldsymbol{w}}$, $c_2 = \left(\hat{\boldsymbol{w}}^\perp\right)^T X \hat{\boldsymbol{w}}^\perp$ and $c_3 = \hat{\boldsymbol{w}}^T X \hat{\boldsymbol{w}}^\perp$. The optimal solution is given by

$$\begin{aligned}
\tilde{x}_1^* &= -\frac{1}{2}\left(c_1 + \frac{\alpha}{\alpha + 2\beta} c_2\right) = -\frac{1}{2}\frac{\boldsymbol{w}^T X \boldsymbol{w}}{\boldsymbol{w}^T \boldsymbol{I}_u \boldsymbol{w}} - \frac{\alpha}{2\alpha + 4\beta} \frac{\left(\boldsymbol{w}^\perp\right)^T X \boldsymbol{w}^\perp}{\left(\boldsymbol{w}^\perp\right)^T \boldsymbol{I}_u \boldsymbol{w}^\perp} \\
\tilde{x}_2^* &= -c_3 = -\frac{\boldsymbol{w}^T X \boldsymbol{w}^\perp}{\|\boldsymbol{w}\|_{\boldsymbol{I}_u} \|\boldsymbol{w}^\perp\|_{\boldsymbol{I}_u}} \\
\boldsymbol{y}^* &= -\left(\frac{\alpha}{2\alpha + 4\beta} \frac{\operatorname{tr}(\boldsymbol{I}_u^{-1} X)}{\|\boldsymbol{w}\|_{\boldsymbol{I}_u}^2} + \frac{\beta}{\alpha + 2\beta} \frac{\boldsymbol{w}^T X \boldsymbol{w}}{\|\boldsymbol{w}\|_{\boldsymbol{I}_u}^4}\right) \mathrm{d}\phi^T \\
&\quad - \frac{\boldsymbol{w}^T X \boldsymbol{w}^\perp}{\|\boldsymbol{w}\|_{\boldsymbol{I}_u}^2 \|\boldsymbol{w}^\perp\|_{\boldsymbol{I}_u}^2} \left(\mathrm{d}\phi^\perp\right)^T
\end{aligned}$$

and the optimal value attained is

$$\frac{\beta(\alpha + \beta)}{(\alpha/2 + \beta)} c_2^2 = \frac{\beta(\alpha + \beta)}{(\alpha/2 + \beta)} \frac{\left[\left(\boldsymbol{w}^\perp\right)^T X \boldsymbol{w}^\perp\right]^2}{\|\boldsymbol{w}^\perp\|_{\boldsymbol{I}_u^4}}.$$

Unwinding the changes of variables leads to the formula for $\boldsymbol{v}_1$ in Section 2.3.4.

A.4 Derivation of Stretching Term in *TFW* Model

In Equation (2.19), we have

$$\boldsymbol{r} \approx \boldsymbol{r}_b + \mathrm{d}\boldsymbol{r_b} \boldsymbol{I}_b^{-1}\left(a \sin\phi\, \boldsymbol{v} + \frac{a^2}{8} \sin 2\phi\, \mathrm{d}\phi^T\right) + a\cos\phi \hat{\boldsymbol{n}}_b.$$

Applying the assumptions about fast and slow variables stated in Section 2.3.3, we can compute $\mathrm{d}\boldsymbol{r}$ in the following way:

$$
\begin{aligned}
\mathrm{d}\boldsymbol{r} =&\mathrm{d}\boldsymbol{r_b}\Bigg(\boldsymbol{id} - a\cos\phi \boldsymbol{I}_b^{-1}\boldsymbol{II}_b + \sin\phi\left(\boldsymbol{I}_b^{-1}\boldsymbol{v}\right)\mathrm{d}a + a\cos\phi\left(\boldsymbol{I}_b^{-1}\boldsymbol{v}\right)\mathrm{d}\phi \\
&+ \frac{a}{4}\sin 2\phi\left(\boldsymbol{I}_b^{-1}\mathrm{d}\phi^T\right)\mathrm{d}a + \frac{a^2}{4}\cos 2\phi\left(\boldsymbol{I}_b^{-1}\mathrm{d}\phi^T\right)\mathrm{d}\phi\Bigg) \\
&+ \hat{\boldsymbol{n}}_b\Bigg(\cos\phi\mathrm{d}a - a\sin\phi\mathrm{d}\phi + a\sin\phi\left(\boldsymbol{I}_b^{-1}\boldsymbol{v}\right)^T\boldsymbol{II}_b \\
&+ \frac{a^2}{8}\sin 2\phi\left(\mathrm{d}\phi\boldsymbol{I}_b^{-1}\right)\boldsymbol{II}_b\Bigg) \\
=&\mathrm{d}\boldsymbol{r_b}(\boldsymbol{id} + A) + \hat{\boldsymbol{n}}_b B,
\end{aligned}
$$

where the matrix A and vector B collect the various terms in the above expression. We then compute

$$
\begin{aligned}
(\boldsymbol{id} + A)^T\boldsymbol{I}_b(\boldsymbol{id} + A) =&\boldsymbol{I}_b + [\boldsymbol{I}_b A]_T + o(A) \\
\approx&\ \boldsymbol{I}_b - 2a\cos\phi\boldsymbol{II}_b + \sin\phi[\boldsymbol{v}\mathrm{d}a] + a\cos\phi[\boldsymbol{v}\mathrm{d}\phi]_T \\
&+ \frac{a}{4}\sin 2\phi[\mathrm{d}\phi^T\mathrm{d}a]_T + \frac{a^2}{2}\cos 2\phi\mathrm{d}\phi^T\mathrm{d}\phi \\
=&\ \boldsymbol{I}_b + \frac{a}{4}\sin 2\phi[\mathrm{d}\phi^T\mathrm{d}a]_T + \frac{a^2}{2}\cos 2\phi\mathrm{d}\phi^T\mathrm{d}\phi \\
&- 2a\cos\phi\boldsymbol{II}_b + a\cos\phi[\boldsymbol{v}\mathrm{d}\phi]_T + o(\|\mathrm{d}a\|)
\end{aligned}
$$

and

$$
\begin{aligned}
B^T B =&\cos^2\phi\mathrm{d}a^T\mathrm{d}a + a^2\sin^2\phi\mathrm{d}\phi^T\mathrm{d}\phi - a\sin\phi\cos\phi[\mathrm{d}\phi^T\mathrm{d}a] \\
&+ o\left(a^2\|\mathrm{d}\phi\|^2\right) + o\left(a\|\boldsymbol{v}\|\right) + o\left(a\|\mathrm{d}\phi^T\mathrm{d}a\|\right) \\
\approx&\cos^2\phi\mathrm{d}a^T\mathrm{d}a + a^2\sin^2\phi\mathrm{d}\phi^T\mathrm{d}\phi - a\sin\phi\cos\phi[\mathrm{d}\phi^T\mathrm{d}a].
\end{aligned}
$$

To avoid (even more) clutter, implicit in the above expressions is the use of one-form and vector norms $\boldsymbol{I}_u^{-1}$ and $\boldsymbol{I}_u$, respectively, and the matrix norm $\|\cdot\|_{\mathrm{SV}}$.

Then, after dropping the high order terms, we get

$$\begin{aligned}W_s =&\boldsymbol{I}_u^{-1}(\mathrm{d}\boldsymbol{r}\mathrm{d}\boldsymbol{r} - \boldsymbol{I}_u)\\ =&\boldsymbol{I}_u^{-1}(\boldsymbol{I}_b - \boldsymbol{I}_u + \frac{1}{2}\mathrm{d}a^T\mathrm{d}a + \frac{1}{2}a^2\mathrm{d}\phi^T\mathrm{d}\phi)\\ &+\cos\phi\left(\boldsymbol{I}_u^{-1}(-2a\boldsymbol{II}_b + a[\boldsymbol{v}\mathrm{d}\phi]_T)\right)\\ &+\sin 2\phi\left(\boldsymbol{I}_u^{-1}(-\frac{a}{4}[\mathrm{d}\phi^T\mathrm{d}a]_T)\right) + \cos 2\phi\left(\frac{1}{2}\boldsymbol{I}_u^{-1}\mathrm{d}a^T\mathrm{d}a\right).\end{aligned}$$

From here we recover the expression for the wrinkle field stretching energy in Section 2.3.5 by plugging in the definition of $\boldsymbol{v}$, and applying the coarse-graining operator.

A.5 Additional Performance Experiments and Data

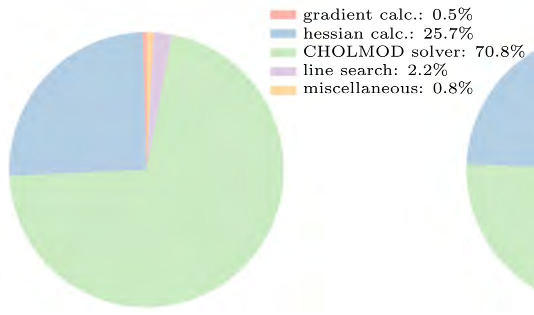
In this Appendix, we provide more detailed data and discussion related to the performance experiments reported in Section 2.6.5.

A.5.1 Per-Iteration Timing Breakdown

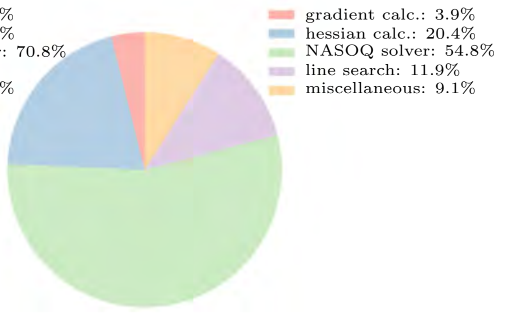
We instrumented the wall-clock time required by each component of one optimization step of *StVK* and *TFW*, and report these timings (averaged over experiments and iterations) in Figure A.1. For both methods, the base solver (QP for *TFW* and linear for *StVK*) takes the majority of time ($\approx 70\%$ for *StVK* and $\approx 55\%$ for *TFW*). We made a best effort to optimize both the *StVK* and *TFW* code, and that the QP solver time dominates in both methods in Figure A.1 confirms that there are no gross inefficiencies remaining in either implementation.

For *StVK*, we used parallel supernodal sparse Cholesky decomposition, provided by SuiteSparse Chen et al. (2008), as the solver, and the computational expense of the solve is due to the large size of the hessian matrix. For *TFW*, we use NASOQ Cheshmi et al. (2020) as our QP solver, and the bulk of the expense is due to the presence of the integrability equality constraints on ω and the inequality constraints on a. NASOQ is not optimized for our (very simple) box inequality constraints; in Table A.1, we list the average time per iteration spend by NASOQ, and compare to a baseline where we drop the constraints and use CHOLMOD to solve the *TFW* QP instead. The latter numbers are substantially faster than the former and give a sense of the performance ceiling for *TFW*, should NASOQ be replaced by a more performant QP solver.

Source code for our implementations of both *TFW* and *StVK* are available in this Github page.



(a) StVK



(b) TFW

Figure A.1: Breakdown of average time required by each component of *TFW* and *StVK* during one optimization iteration (averaged over all iterations of all collision-free examples in Chapter 2). See Tables A.1 and A.3 for additional timing breakdowns for *TFW* and *StVK*, respectively. "Miscellaneous" includes bookkeeping such as updating the state variables, printing information about the solver state to the console, checking for termination, etc.

A.5.2 Residual Plots for Each Example

For each of our examples, we provide plots in Figure A.2 of the gradient residual (for *StVK*) and of the stationarity residual, i.e gradient projected onto the constraint manifold using the current values of the Lagrange multipliers, for *TFW*. Both are plotted against wall clock time. We also show stills of each simulation at the chosen visually-stable time, as well as many iterations later, to illustrate that there is indeed no significant visual difference between the results at these two times.

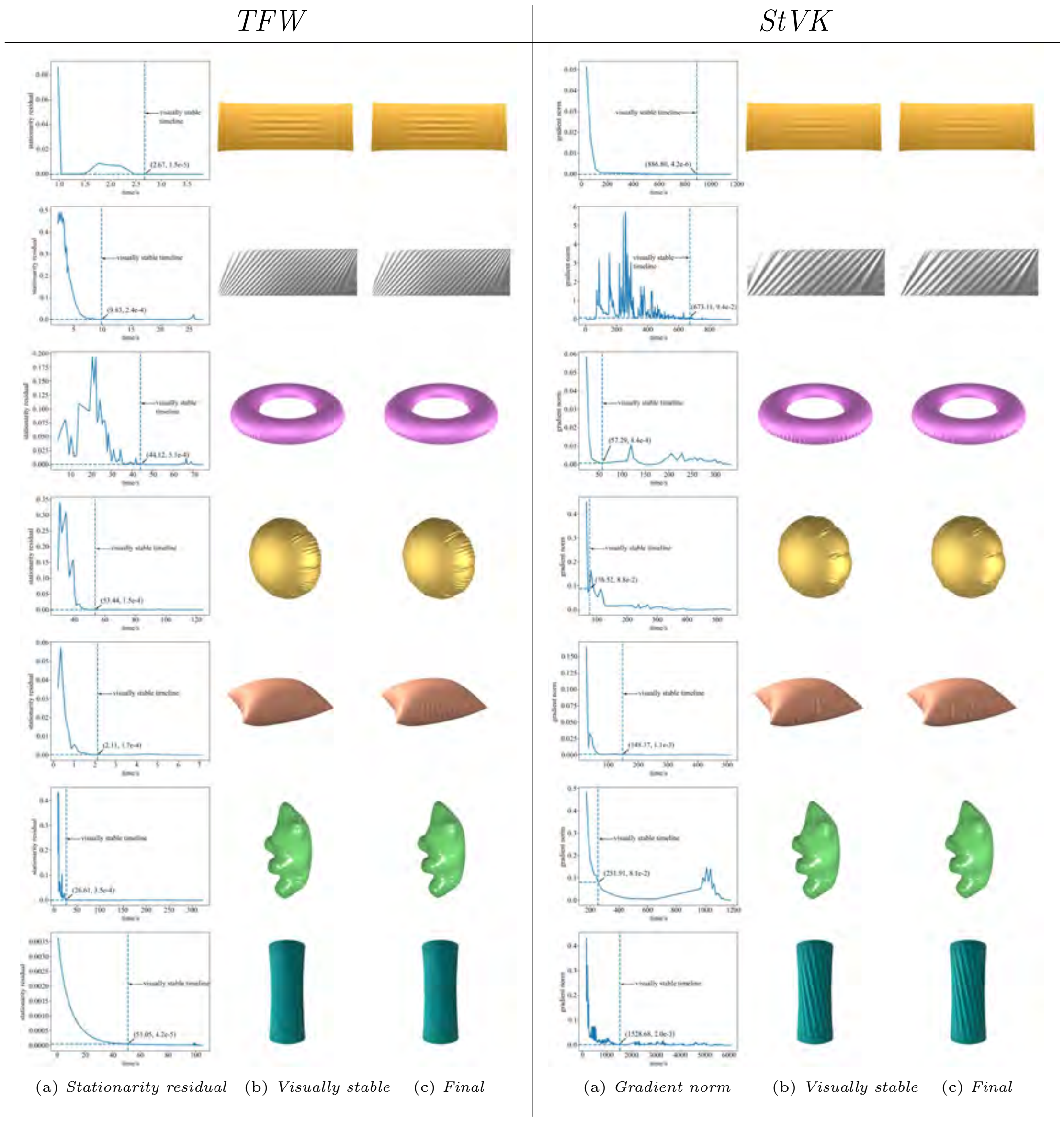


Figure A.2: Comparing residual and wall clock time for our examples: For *StVK*, it's the gradient norm; for *TFW*, it includes projection onto Equation A.9's constraints with $|\nabla L| = |\nabla E^{\text{wf}} + S^T y + C^T z|$, where y, z are Lagrange multipliers. Dashed lines mark when simulations appear visually stable upon inspection, using the infinity norm. The second and third columns display wrinkled surface snapshots at the visually stable point and post-experiment, respectively. Simulations end after 1000 iterations or when the residual norm is below 10^{-6}. Refer to Table 2.1 for detailed experiment data.

For *TFW*, we see that the point at which a simulation becomes visually stable (chosen by inspection of the simulation output) approximately matches the onset of a plateau in the residual plot. The situation for *StVK* is less clear. In future work, it would be practically useful to formalize these observations into a quantitative termination condition corresponding to being "visually stable."

A.5.3 Convergence Discussion

In Tables A.1 and A.2, we provide additional timing data about termination of the simulations shown in Figure 2.22: recall that termination occurs on each timeline where the background color changes from blue to green, and that we terminate when either the residual infinity norm is smaller than 10^{-6} or the optimization exceeds 1000 iterations. Note that the gradient of energy with respect to position has different units than the gradient with respect to amplitude a or to frequency ω; it is thus only meaningful to compare *TFW* results to each other and *StVK* results to each other, since each method is essentially using a different termination condition.

In Figure A.2, we observe that the gradient norm eventually converges quadratically to zero, as expected. On the other hand, for some examples the *TFW* stationarity residual appears to converge only linearly. We believe the reason for this behavior is the non-negativity constraint on a: near optimality, the optimization problem in Equation (A.9) becomes convex, but *only in the feasible cone* delineated by the equality constraints and active inequality constraints. In particular, near optimality the unconstrained Hessian can be indefinite, even though the second-order change in energy is positive in every feasible direction. Unfortunately, standard QP solvers, including the ones we currently use (NASOQ) generally only accepts positive quadratic forms, requiring us to project any indefinite Hessian to a nearby positive-definite matrix (see Appendix A.6 for details). The use of this modified Hessian during SQP prevents quadratic convergence; in future work, NASOQ could potentially be replaced by a QP solver that does not require modifying the Hessian in cases where it is indefinite despite the constrained problem being locally convex.

Models	#verts	*TFT*		*TFW*		Time/iter (s)		
		#iter	time (s)	#iter	time (s)	total	NASOQ	CHOLMOD
sphere drape	1.6k	—	—	46	15.06	0.33	0.18	0.013
symmetric dress	1.4k	—	—	72	29.74	0.41	0.28	0.011
asymmetric dress	3.4k	—	—	50	38.47	0.77	0.56	0.030
pants	4k	—	—	74	141.74	1.92	1.52	0.029
stretched sheet	522	10	0.45	23	3.93	0.17	0.10	0.0039
sheared rectangle	$1k$	21	2.27	147	27.52	0.19	0.073	0.0082
torus	$2k$	11	2.27	125	74.82	0.60	0.44	0.017
balloon	$5k$	33	16.04	91	125.33	1.38	1.08	0.039
teabag	936	8	0.78	46	7.32	0.16	0.057	0.0034
teddy*	$2k$	3	0.93	1001	322.25	0.32	0.19	0.0063
twisted cylinder	688	12	0.69	813	104.92	0.13	0.048	0.0056

Table A.1: Additional timing information for the *TFW* solver. The *TFT* columns list the number of iterations and wall clock time spent computing the base mesh (Section 2.4.1), and the *TFW* columns give the same information, for the solve for the wrinkle field variables (Sections 2.4.2 and 2.5.2). Note that the *TFW* times are for when we terminate the *TFW* simulation due to small stationarity residual, or maximum iterations, and do *not* correspond to termination at the visually-stable time (see Section A.5.3 for more discussion). Note that the time needed to compute the base mesh is negligible compared to the SQP solve for amplitude and frequency. The last three columns, from left to right, list: the average wall clock time required by one *TFW* iteration, the amount of that time spent specifically inside NASOQ (see also Figure A.1), and the amount of time CHOLMOD would require for the same solve if the constraints were ignored. This latter number serves as a ceiling for how efficient *TFW* might be, if the NASOQ solver were replaced by another, more efficient code.

A.5.4 Initialization of *StVK*

Like in all statics problems, our *StVK* optimization requires, and has performance sensitive to, an initial guess. In the main text, we propose using the *TFT* base mesh as the initial guess for *StVK*, both to maximize *StVK*'s performance, and for maintaining consistency of the experimental setup with *TFW*. We performed experiments to justify this choice, with results listed in Table A.3. Instead of the *TFT* base mesh, in these experiments we used a problem-specific "flat" state as the initial guess: for the stretched sheet, sheared rectangle, torus, and balloon problems, we simply take the 2D rest mesh as the flat state. For the teabag we export an unwrinkled,

Models	#verts	*TFT*		*StVK*		Time/iter (s)	
		#iter	time (s)	#iter	time (s)	total	CHOLMOD
stretched sheet	$260k$	17	640.39	22	1178.55	53.57	42.63
sheared rectangle	$20k$	49	101.77	307	936.55	3.05	2.16
torus	$40k$	13	84.47	46	337.91	3.25	2.02
balloon	$40k$	291	1907.74	67	547.68	8.00	5.63
teabag	$30k$	11	34.11	98	521.08	5.32	3.59
teddy	$98k$	10	214.48	59	1188.25	20.14	14.44
twisted cylinder	$960k$	33	308.03	424	6071.73	14.32	9.80

Table A.2: Additional timing information for the *StVK* solver. The *TFT* columns list the number of iterations and wall clock time spent computing the base mesh (Section 2.4.1), and the *StVK* columns give the same information, for the solve for the static shape using *TFT* as the initial guess. Note that the *StVK* times are for when we terminate the *StVK* simulation due to small gradient residual, or maximum iterations, and do *not* correspond to termination at the visually-stable time (see Section A.5.3 for more discussion). The last two columns list the average wall clock time required by one *StVK* iteration, and the amount of that time spent specifically inside CHOLMOD (see also Figure A.1).

pre-optimization initial guess from *Marvelous Designer*, and for the teddy, we start from the unwrinkled geometry provided by Skouras et al. (2014). For the twisted cylinder problem, we use an untwisted cylinder as the flat state. Unsurprisingly, the *TFT* solution is universally a better initial guess than these alternate flat states, since the *TFT* solution is expected to match the *StVK* optimum, up to missing fine wrinkles.

A.5.5 Video Comparisons

We provide videos of the equal-effort comparison experiments illustrated in Figure 2.22 (See this for details). We play back the optimization iterates for both *TFW* and *StVK*, where playback time is a multiple of wall clock time, chosen so that each clip plays at reasonable speed. For *TFW*, we visualize amplitude a as a scalar color field on the base mesh (white is zero amplitude), and likewise draw $\boldsymbol{I}_u^{-1}\omega^T$ as a vector field on $\boldsymbol{r}_b$ (left animation). We show the wrinkled surface $\boldsymbol{r}_w$ as well (middle animation; note that since we only recover phase ϕ up to an unknown global

Models	#verts	*StVK*		Time/iter (s)	
		#iter	time (s)	total	CHOLMOD
stretched sheet	$260k$	105	6564.25	62.52	47.47
sheared rectangle	$20k$	–	–	–	–
torus	$40k$	66	448.68	6.80	4.42
balloon	$40k$	204	1651.71	8.10	5.40
teabag	$30k$	229	1347.81	5.89	4.21
teddy	$98k$	140	2259.93	16.14	10.30
twisted cylinder	$960k$	596	10183.8	17.09	12.86

Table A.3: Timing for an alternative *StVK* setup where a problem-specific "flat" state is used as the initial guess rather than the *TFT* base mesh (see Section A.5.4). As in Table A.2, the *StVK* columns give the number of iterations, and wall clock time, when we terminate the simulation due to small gradient residual or maximum iterations. The last two columns list the average wall clock time required by one *StVK* iteration, and the amount of that time spent specifically inside CHOLMOD. The shared rectangle experiment failed completely (the simulation exploded after a few iterations, due to the excessive strain in the initial guess). Notice that the *TFT* initial guess leads to faster static solvers in all cases.

phase shift, which is not necessarily temporally coherent between solver iterations, the wrinkles on the surface can "drift" incoherently between iterations; see Section 2.7 for more discussion of the phase ambiguity). For *StVK*, we show the predicted vertex positions at each iteration (right animation).

We use the same background color as in Figure 2.22 to indicate the status of *TFW* and *StVK* at the each frame of the animations. We also show, at the bottom of the video, a wall-clock time axis for each animation, on which we mark the transition times where each simulation becomes visually stable or terminates. In most cases, termination of the *TFW* algorithm occurs well after *TFW* has reached a visually stable state, and well before *StVK* does so. We speed up the second portion of the animation (where *TFW* has terminated and *StVK* is still running) to keep the movie length reasonable—precise playback speed information is provided above the time axis. Each clip ends when the *StVK* simulation terminates, and we pause each animation for five seconds at the end.

Some notable behavior that we observe in the videos: for the *TFW* stretched sheet experiment, amplitude first vanishes globally over the sheet, and then wrinkles emerge at the center of the sheet, as predicted by theory. Notice also that in some examples (such as the Mylar balloon and torus), the vector field $\boldsymbol{I}_u^{-1}\omega^T$ has large magnitude near singularities. This behavior is expected (and corresponds to high-frequency, low amplitude wrinkling at points where wrinkles converge to a singular points).

A.6 Additional Implementation Details

In this appendix, we flesh out some of the steps described at a high level in Section 2.5 of Chapter 2.

A.6.1 Triangulation

We Delaunay-triangulate the parameter domain U to create a simulation mesh. In many of our examples, the parameter domain is given as a set of disconnected patches, which are to be sewn together into a garment or balloon along shared boundaries; we do so to generate a single connected simulation mesh K.

In Section 2.6.3, we analyze the effect of meshing on the *TFW* results. We observe that although our method is fairly robust to mesh resolution and tessellation, using a highly symmetric mesh, or one whose edge directions have consistent bias, can result in artifacts, both during simulation of *TFT* and during Loop subdivision. We therefore recommend always using an irregular but coarse Delaunay mesh.

A.6.2 Initialization

In this step, we choose initial guesses for a and ω, based on the strain of the base mesh embedding V_b. We assume each triangle $\mathbf{f}$ in F has constant strain (see Appendix A.1 for details on how we discretize strain and related quantities), and compute the direction $\boldsymbol{w}$ and magnitude $\epsilon_{\boldsymbol{w}}$ of the most negative principal strain. We initialize amplitude to $k\sqrt{2\epsilon_{\boldsymbol{w}}}$, where k is an arbitrary constant (in our experiments, this constant is set to $\min\left[0.01, 0.1 \cdot \text{bbox}(V_b)\right]$, where $\text{bbox}(V_b)$ is the diameter of the bounding box around the base mesh), and compute a target frequency $\omega_{\mathbf{e}}$ for each edge $\mathbf{e}$ of $\mathbf{f}$ using Equation (2.8). (If the triangle has no negative principal strain, we use $a = 0$ and $\omega_{\mathbf{e}} = 0$ instead.)

Note that this procedure does not generally yield an integrable one-form ω (in fact, the two triangles neighboring $\mathbf{e}$ usually will not even agree on $\omega_{\mathbf{e}}$). We project to the closest curl-free one-form ω in the least-squares sense using the Helmholtz decomposition.

A.6.3 Amplitude and Phase Optimization Details

Recall from Section 2.5 that we discretize the parameter domain U by a coarse mesh $K = (V_u, F, E)$, and discretize a as a piecewise-linear function on K and ϕ as a one-form on the edges E. We can then write the objective function in Equation (2.30) as a sum of contributions from triangles not in W:

$$E^{\text{wf}} = \frac{1}{2} \sum_{\mathbf{f} \in F, \mathbf{f} \notin W} \int_{\mathbf{f}} \left(\frac{h}{4} W_s^{\text{wf}} + \frac{h^3}{12} W_b^{\text{wf}} \right) \sqrt{\det \boldsymbol{I}_u}\, dA, \tag{A.7}$$

where W_s^{wf} and W_b^{wf} collect the terms in Equations (2.20) and (2.26). $\mathrm{d}a$ is constant over each triangle, as are the fundamental forms $\boldsymbol{I}_u, \boldsymbol{II}_u$. The other terms we need in order to implement Equation (A.7) are $\kappa_\perp$ and $d\phi$.

To estimate curvature of the wrinkle crests, we first compute a vector $\boldsymbol{w}$ per triangle, as described in Section 2.4.2, based on that triangle's (constant) base mesh strain tensor. We then robustly estimate principal curvatures and curvature directions via quadratic fitting Panozzo et al. (2010), which allows us to compute $\kappa_\perp$. Note that $\kappa_\perp$ is a constant: it does not vary over the course of optimizing E^{wf}.

Integrability induces one linear equality constraint for each triangle not in W, enforcing that the circulation $\mathrm{d}\omega$ of ω around the face is zero:

$$\omega_{ij} + \omega_{jk} + \omega_{ki} = 0 \tag{A.8}$$

for each face $\mathbf{f}_{ijk} \notin W$ with edges $\{\mathbf{e}_{ij}, \mathbf{e}_{jk}, \mathbf{e}_{ki}\}$.

On faces not in W, integrability allows us to recover a constant $\mathrm{d}\phi$ on that face from the values of ω on the face's edges, from the defining equations

$$\mathrm{d}\phi\,(\boldsymbol{v}_j - \boldsymbol{v}_i) = \omega_{ij}, \quad \mathrm{d}\phi\,(\boldsymbol{v}_k - \boldsymbol{v}_j) = \omega_{jk}, \quad \mathrm{d}\phi\,(\boldsymbol{v}_i - \boldsymbol{v}_k) = \omega_{ki}.$$

The integrability constraint on ω is exactly the condition that enforces that this overconstrained system of equations has a solution (in practice we compute $\mathrm{d}\phi$ using only the first two equations, and discarding the third as redundant).

Forming a global vector $\boldsymbol{x} \in \mathbb{R}^{|V_b|+|E|}$ by concatenating the unknowns a and ω, applying the above discretization and three-point quadrature Zhang et al. (2009) to compute the integrals in Equation (A.7) yields a degree-eight polynomial function $E(\boldsymbol{x})$ discretizing E^{wf}. We minimize this polynomial subject to the simple non-negativity constraints on amplitudes and the curl constraint, $C\boldsymbol{x} = 0$, on non-W faces,

$$\min_{\boldsymbol{x}} E(\boldsymbol{x}) \quad \text{s.t.} \quad S\boldsymbol{x} \geq 0,\ C\boldsymbol{x} = 0 \tag{A.9}$$

where S is the selector matrix extracting amplitudes from x. We solve this system via sequential quadratic programming. At each iteration, we use the NASOQ QP solver Cheshmi et al. (2020) to compute a descent directions $\delta\boldsymbol{x}$, by solving the sparse, linearly constrained quadratic minimization problem

$$\min_{\delta\boldsymbol{x}} \frac{1}{2}\delta\boldsymbol{x}^T H \delta\boldsymbol{x} \quad \text{s.t.} \quad S\left[\boldsymbol{x}_k + \delta\boldsymbol{x}\right] \geq 0,\ C\delta\boldsymbol{x} = 0$$

where $\boldsymbol{x}_k$ is the current iterate of $\boldsymbol{x}$ and H is a convexification of the energy Hessian $HE(x_k)$ (see below). We perform a line search Moré and Thuente (1994) in the $\delta\boldsymbol{x}$ direction to ensure each SQP iteration decreases the energy and does not violate the inequality constraints. We terminate this optimization process when one of following termination criteria are satisfied: (1) change in energy is smaller than 10^{-10}, (2) the stationarity residual of (A.9) is smaller than 10^{-6}, (3) the update to $\boldsymbol{x}$ is smaller than 10^{-10}, (4) reach the maximum iteration steps (1000 by default). We observe that, as expected, maximum (unit-length) step sizes are accepted near optimality and so always generate a feasible solution satisfying the constraints.

Hessian Projection The quadratic form H must be positive-definite in order for the above SQP scheme to succeed, since otherwise the search direction $\delta\boldsymbol{x}$ cannot be guaranteed to be a descent direction. We therefore select H using one of two methods that ensure it is positive-definite:

- computing the Hessian of each triangle's contribution to the sum in Equation (A.7), projecting that local Hessian to the closest positive-definite matrix (using SVD), and then summing those projected local Hessians to yield H;
- setting $H = HE(x_k) + \epsilon\boldsymbol{id}$, where ϵ is a constant larger than the most-negative eigenvalue in $HE(x_k)$, found via binary search.

We leave further research into methods for projecting the energy Hessian, or for combining the existing approaches into a high-performance metastrategy, to future work; for the results shown in Chapter 2 we use the first method at the beginning of the optimization, and switch to the second once the energy decrease per step becomes smaller than 10^{-8}. See Section A.5.3 for convergence plots of the SQP and additional discussion.

A.6.4 Phase Field Extraction

To visualize the wrinkled surface, we need to convert ω back into a phase field ϕ. Although the curl constraints ensure that ω is *locally* integrable, there is no guarantee that a ϕ globally exists with $\mathrm{d}\phi = \omega$. In the sphere drape example, for instance, it is possible that the optimal ω encodes a fractional number of wrinkles around the cloth circumference.

We borrow from the parameterization literature Bommes et al. (2009) the idea of *rounding* ω to the nearest ϕ: we take K, remove the wrinkle-free faces F, and cut the result into a topological disk. We use Gurobi LLC (2020) to solve

$$\min_{\phi} \|\mathrm{d}\phi - \omega\|^2$$

subject to the constraint that the jump at each cut edge is an integer multiple of 2π.

The resulting ϕ is defined on a triangle soup made from F; i.e. two neighboring faces on K that were cut along their common edge might disagree on the value of ϕ at their shared vertices. But since this disagreement is always a multiple of 2π, the cuts are invisible during visualization.

A.6.5 Upsampling and Visualization

The output of the above *TFW* pipeline is the very coarse base mesh, and the wrinkle field (a, ϕ) defined on its vertices. To visualize the final wrinkled shell, the mesh and wrinkle fields must be upsampled (note that directly displacing the vertices of K according to the wrinkle field is not useful, as a single triangle often hosts multiple wavelengths of wrinkles). We explicitly materialize an upsampled mesh by applying Loop subdivision on K (although, in principle, the visualization could alternately be done on the fly with a tessellation shader) and also applying the subdivision stencil to a and ϕ. Some care is needed when subdividing ϕ to correctly account for: (1) the integer period jumps that can occur in ϕ across neighboring triangles, and (2) triangles W where ϕ is missing.

We then displace the vertices of the upsampled base mesh using Equation (2.19). Whereas the $\boldsymbol{v}_1$ term in this equation is crucial to the correct physical modeling of the elastic energy landscape, we find that this term has only a very slight effect on the visual appearance of the wrinkled surface during the upsampling. We thus drop the the $\boldsymbol{v}_1$ term from just this final visualization (but not from prior computation of the *TFW* model) when upsampling and visualizing all examples shown in Chapter 2. The $\boldsymbol{v}_1$ term is nontrivial to estimate on the upsampled base mesh, and subject to noise in regions where $\mathrm{d}\phi$ is small. By contrast, the $\boldsymbol{v}_2$ term is included in this visualization step as it is critical throughout. If we were to omit the $\boldsymbol{v}_2$ in-plane term we would obtain unnatural-looking undulations (see Figure 2.19, right, for an example) rather than the natural-looking, "bulging" wrinkles with clear overhangs between wrinkles seen in Figure 2.19, left.

Appendix B: Complex Wrinkle Field Evolution Appendices

B.1 Geodesic Discussion

In this section, we derive the geodesic formula given in the main text.

B.1.1 Distance Energy Derivation

We start with the full derivation of the distance-measure energy term. To study the wrinkle evolution controlled by CWF, we assume that the underlying base surface and the corresponding rest shape do not change over time. Therefore

$$\dot{\epsilon} = -\frac{\mathrm{d}(\Re(\boldsymbol{z}))}{\mathrm{d}t}F - \Re(\boldsymbol{z})\frac{\mathrm{d}F}{\mathrm{d}t}, \tag{B.1}$$

$$F := \boldsymbol{I}^{-1}(\mathrm{d}\arg\boldsymbol{z})^T(\mathrm{d}\arg\boldsymbol{z}). \tag{B.2}$$

Letting $\omega = \mathrm{d}\arg\boldsymbol{z}$ and $\boldsymbol{z} = a\tilde{\boldsymbol{z}} = a(\cos\theta + i\sin\theta)$, we have

$$\begin{aligned}\|\dot{\epsilon}\|^2_{\mathrm{SV}} = &\Big((\dot{a})^2\cos^2\theta + (a\dot{\theta})^2\sin^2\theta - a\dot{a}\dot{\theta}\sin 2\theta\Big)\,\|F\|^2_{\mathrm{SV}} \\ &+ a^2\cos^2\theta\|\dot{F}\|^2_{\mathrm{SV}} + 2a\cos\theta(\dot{a}\cos\theta - a\dot{\theta}\sin\theta)(F:\dot{F}),\end{aligned} \tag{B.3}$$

where

$$\|M\|^2_{\mathrm{SV}} = \frac{\alpha}{2}\mathrm{Tr}^2(M) + \beta\mathrm{Tr}(M^2), \tag{B.4}$$

with α and β the Lamé parameters, and

$$A : B := \frac{\alpha}{2}\mathrm{Tr}(A)\mathrm{Tr}(B) + \beta\mathrm{Tr}(AB) = B : A. \tag{B.5}$$

Following the coarse-graining idea of Aharoni et al. (2017), the trigonometric terms can be eliminated by averaging with the conjugate correspondence:

$$\begin{aligned}\|\dot{\bar{\epsilon}}\|^2_{\mathrm{SV}} = &\Big((\dot{a})^2\sin^2\theta + (a\dot{\theta})^2\cos^2\theta + a\dot{a}\dot{\theta}\sin 2\theta\Big)\,\|F\|^2_{\mathrm{SV}} \\ &+ a^2\sin^2\theta\|\dot{F}\|^2_{\mathrm{SV}} + 2a\sin\theta(\dot{a}\sin\theta + a\dot{\theta}\cos\theta)(F:\dot{F}).\end{aligned} \tag{B.6}$$

Therefore,

$$\begin{aligned}\int_{\mathcal{M}} \|\dot{\epsilon}\|^2_{\text{SV}}\, \mathrm{d}A &\approx \int_{\mathcal{M}} \frac{1}{2}\left(\|\dot{\epsilon}\|^2_{\text{SV}} + \|\dot{\bar{\epsilon}}\|^2_{\text{SV}}\right)\, \mathrm{d}A \\ &= \int_{\mathcal{M}} \frac{(\dot{a})^2 + a^2\left|\dot{\tilde{\boldsymbol{z}}}\right|^2}{2}\|F\|^2_{\text{SV}} + \frac{a^2}{2}\|\dot{F}\|^2_{\text{SV}} + a\dot{a}(F : \dot{F})\, \mathrm{d}A \\ &= \frac{1}{2}\int_{\mathcal{M}} \left|\dot{\tilde{\boldsymbol{z}}}\right|^2 \|F_{a,\omega}\|^2_{\text{SV}} + \|\dot{F}_{a,\omega}\|^2_{\text{SV}}\, \mathrm{d}A,\end{aligned} \tag{B.7}$$

where we use

$$\left|\dot{\tilde{\boldsymbol{z}}}\right|^2 = \left|(\exp i\theta)\,\dot{\theta}\right| = |\dot{\theta}| \tag{B.8}$$

$$F_{a,\omega} = a\boldsymbol{I}^{-1}\omega^T\omega. \tag{B.9}$$

Thus the distance on paths $\gamma(t)$, which connects two endpoints $\boldsymbol{z}^0$ and $\boldsymbol{z}^1$ is given by:

$$d(\gamma) = \frac{1}{2}\int_0^1 \int_{\mathcal{M}} \left|\dot{\tilde{\boldsymbol{z}}}\right|^2 \|F_{a,\omega}\|^2_{\text{SV}} + \|\dot{F}_{a,\omega}\|^2_{\text{SV}}\, \mathrm{d}A\ \mathrm{d}t, \tag{B.10}$$

with $\boldsymbol{z} = a\tilde{\boldsymbol{z}}$ and $\mathrm{d}\arg\boldsymbol{z} = \omega$, or equivalently $(\mathrm{d} - i\omega)\tilde{\boldsymbol{z}} = 0$..

B.1.2 Geodesics Computation

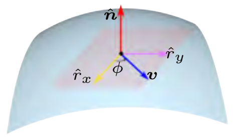
The geodesic γ^* between two endpoints $\boldsymbol{z}^0$ and $\boldsymbol{z}^1$ can be found by solving the following variational problem:

$$\begin{aligned}\min_{\boldsymbol{z}} \frac{1}{2}\int_0^1 \int_{\mathcal{M}} &\left|\dot{\tilde{\boldsymbol{z}}}\right|^2 \|F_{a,\omega}\|^2_{\text{SV}} + \|\dot{F}_{a,\omega}\|^2_{\text{SV}}\, \mathrm{d}A\ \mathrm{d}t \\ \text{s.t.}\quad &(\mathrm{d} - i\omega)\tilde{\boldsymbol{z}} = 0,\ \boldsymbol{z} = a\tilde{\boldsymbol{z}},\ |\tilde{\boldsymbol{z}}| = 1 \\ &\boldsymbol{z}(0) = \boldsymbol{z}^0,\ \boldsymbol{z}(1) = \boldsymbol{z}^1.\end{aligned} \tag{B.11}$$

To solve Equation (B.11), we write $u = \sqrt{a}\omega$, and $f = \|u\|^2_{\boldsymbol{I}^{-1}}$, so that we have

$$\begin{aligned}\|F_{a,\omega}\|^2_{\text{SV}} &= (\frac{\alpha}{2} + \beta)f^2 \\ \|\dot{F}_{a,\omega}\|^2_{\text{SV}} &= 2(\alpha + \beta)\left(u\boldsymbol{I}^{-1}\dot{u}^T\right)^2 + 2\beta f\|\dot{u}\|^2_{\boldsymbol{I}^{-1}}.\end{aligned} \tag{B.12}$$

Consider the tangent vector $\boldsymbol{v}$ corresponding to the one-form u, where $\boldsymbol{v} = \mathrm{d}r\boldsymbol{I}^{-1}u^T$.This can be expressed as the rotation of the **unit** basis tangent vector $\hat{r}_x$ w.r.t **unit** base surface normal $\hat{\boldsymbol{n}}$, with rotation angle $\phi(t)$, and rescaling by its norm:



$$\boldsymbol{v} = \|\boldsymbol{v}\| R(\hat{\boldsymbol{n}}, \phi(t))\hat{r}_x = \sqrt{f} R(\hat{\boldsymbol{n}}, \phi(t))\hat{r}_x, \tag{B.13}$$

where $\|\boldsymbol{v}\|^2 = \boldsymbol{v}^T\boldsymbol{v} = u\boldsymbol{I}^{-1}u^T = \|u\|^2_{\boldsymbol{I}^{-1}} = f$ and $R(\hat{\boldsymbol{n}}, \phi(t))$ is the rotation matrix with axis $\hat{\boldsymbol{n}}$ and angle ϕ. This expression gives us

$$\begin{aligned} u\boldsymbol{I}^{-1}\dot{u}^T &= \frac{1}{2}\dot{f} \\ \|\dot{u}\|^2_{\boldsymbol{I}^{-1}} &= \frac{1}{4f}\dot{f}^2 + f\dot{\phi}^2 \hat{r}_x^T \left(\frac{\partial R}{\partial \phi}\right)^T \frac{\partial R}{\partial \phi}\hat{r}_x. \end{aligned} \tag{B.14}$$

Given that $\|\hat{\boldsymbol{n}}\| = \|\hat{r}_x\| = 1$, $\hat{\boldsymbol{n}} \cdot \hat{r}_x = 0$, and $\hat{\boldsymbol{n}}$ is the rotation axis, it is easy to check that

$$\hat{r}_x^T \left(\frac{\partial R}{\partial \phi}\right)^T \frac{\partial R}{\partial \phi}\hat{r}_x = 1. \tag{B.15}$$

Therefore we have

$$\|\dot{F}_{a,\omega}\|^2_{\mathrm{SV}} = (\frac{\alpha}{2} + \beta)\dot{f}^2 + 2\beta f^2\dot{\phi}^2. \tag{B.16}$$

The optimization (B.11) becomes:

$$\begin{aligned} \underset{f,\phi,\tilde{\boldsymbol{z}},a}{\arg\min} \frac{1}{2}\int_0^1 \int_{\mathcal{M}} &(\frac{\alpha}{2} + \beta)(f^2 \left|\dot{\tilde{\boldsymbol{z}}}\right|^2 + \dot{f}^2) + 2\beta f^2\dot{\phi}^2 \,\mathrm{d}A\mathrm{d}t \\ \text{s.t.} \quad & \omega(t) = \boldsymbol{u}(t)/\sqrt{a(t)} = \sqrt{f(t)/a(t)}\hat{r}_x^T \left[R(\hat{n}, \phi(t))\right]^T \mathrm{d}r \\ & \phi(0) = \phi^0,\ \phi(1) = \phi^1,\ \tilde{\boldsymbol{z}}(0) = \tilde{\boldsymbol{z}}^0,\ \tilde{\boldsymbol{z}}(1) = \tilde{\boldsymbol{z}}^1 \\ & f(0) = f^0,\ f(1) = f^1,\ a(0) = a^0,\ a(1) = a^1 \\ & [\mathrm{d} - i\omega(t)]\,\tilde{\boldsymbol{z}}(t) = 0,\ |\tilde{\boldsymbol{z}}(t)| = 1. \end{aligned} \tag{B.17}$$

Similarly to what we discussed in the main text, we replace the last two constraints by $\tilde{\boldsymbol{z}} \in \mathrm{Opt}_\omega$, leading to the following well-defined optimization problem:

$$\begin{aligned} &\underset{f,\phi,\tilde{\boldsymbol{z}},a}{\arg\min} \frac{1}{2}\int_0^1 \int_{\mathcal{M}} (\frac{\alpha}{2}+\beta)(f^2 \left|\dot{\tilde{\boldsymbol{z}}}\right|^2 + \dot{f}^2) + 2\beta f^2 \dot{\phi}^2 \; \mathrm{d}A \; \mathrm{d}t \\ &\quad \text{s.t.} \quad \tilde{\boldsymbol{z}} \in \mathrm{Opt}_\omega, \\ &\mathrm{Opt}_\omega := \underset{\tilde{\boldsymbol{z}}}{\arg\min} \int_{\mathcal{M}} \left[\left\|(\mathrm{d} - i\omega(t))\,\tilde{\boldsymbol{z}}\right\|^2 + \left(|\tilde{\boldsymbol{z}}|^2 - 1\right)^2 \right] \; \mathrm{d}A, \end{aligned} \tag{B.18}$$

with the same boundary conditions.

We solve it by a penalty method as:

$$\begin{aligned} &\underset{f,\phi,\tilde{\boldsymbol{z}},a}{\arg\min} \frac{1}{2}\int_0^1 \int_{\mathcal{M}} (\frac{\alpha}{2}+\beta)(f^2 \left|\dot{\tilde{\boldsymbol{z}}}\right|^2 + \dot{f}^2) + 2\beta f^2 \dot{\phi}^2 \; \mathrm{d}A \; \mathrm{d}t \\ &\qquad + c_1 \int_0^1 \int_{\mathcal{M}} \left\|(\mathrm{d} - i\omega)\,\tilde{\boldsymbol{z}}\right\|^2 + \left(|\tilde{\boldsymbol{z}}|^2 - 1\right)^2 \; \mathrm{d}A \; \mathrm{d}t, \end{aligned} \tag{B.19}$$

with boundary conditions:

$$\phi(0) = \phi^0,\ \phi(1) = \phi^1,\ \theta(0) = \theta^0,\ \theta(1) = \theta^1 \tag{B.20}$$

$$f(0) = f^0,\ f(1) = f^1,\ a(0) = a^0,\ a(1) = a^1, \tag{B.21}$$

$$\tag{B.22}$$

and

$$\omega = \sqrt{f(t)/a(t)}\,\hat{r}_x^T \left[R(\hat{n}, \phi(t))\right]^T \mathrm{d}r. \tag{B.23}$$

We further approximate this problem by replacing f^2 by its average over time:

$$\begin{aligned} &\underset{f,\phi,\tilde{\boldsymbol{z}},a}{\arg\min} \frac{1}{2} \int_{\mathcal{M}} \left(\frac{(f^0)^2 + (f^1)^2}{2} \int_0^1 \left(\frac{\alpha}{2} + \beta\right) \left|\dot{\tilde{\boldsymbol{z}}}\right|^2 + 2\beta\dot{\phi}^2 \mathrm{d}t \right) \mathrm{d}A \\ &\qquad + \left(\frac{\alpha}{4} + \frac{\beta}{2}\right) \int_{\mathcal{M}} \int_0^1 \dot{f}^2 \; \mathrm{d}t \; \mathrm{d}A \\ &\qquad + c_1 \int_{\mathcal{M}} \int_0^1 \left\|(\mathrm{d} - i\omega)\,\tilde{\boldsymbol{z}}\right\|^2 + \left(|\tilde{\boldsymbol{z}}|^2 - 1\right)^2 \; \mathrm{d}t \; \mathrm{d}A. \end{aligned} \tag{B.24}$$

This indicates that

$$f(t) = (1-t)f^0 + tf^1.$$

The optimal ϕ should satisfy

$$\begin{aligned}\beta\ddot{\phi} &= -c_1\overline{(\mathrm{d}-i\omega)\tilde{\boldsymbol{z}}}\left(i\tilde{\boldsymbol{z}}\frac{\mathrm{d}\omega}{\mathrm{d}\phi}\right)\\ &= -c_1\overline{(\mathrm{d}-i\omega)\tilde{\boldsymbol{z}}}\left(i\tilde{\boldsymbol{z}}\omega^{\mathrm{perp}}\right),\end{aligned} \tag{B.25}$$

where

$$\omega^{\mathrm{perp}}(t) = \sqrt{f(t)/a(t)}\hat{r}_x^T\left[R\left(\hat{n},\phi(t)+\frac{\pi}{2}\right)\right]^T\mathrm{d}r. \tag{B.26}$$

Away from the singularities, $\tilde{\boldsymbol{z}}$ and ω are almost compatible, that is $(\mathrm{d}-i\omega)\tilde{\boldsymbol{z}}\approx 0$. Therefore we have $\ddot{\phi}=0$, which indicates

$$\phi(t) = (1-t)\phi^0 + t\phi^1. \tag{B.27}$$

Near the singularities, $\tilde{\boldsymbol{z}}$ is close to zero. In this case, the wrinkles displacement $(= a\Re(\tilde{\boldsymbol{z}})\hat{\boldsymbol{n}}\approx 0)$ is negligible. The value of ϕ does not matter. Therefore, we can extend Equation (B.27) for the whole surface. We further assume that amplitude varies slowly temporally, so that it can be approximated using the linear relation,

$$a(t) = (1-t)a^0 + ta^1. \tag{B.28}$$

We can get the final remaining $\tilde{\boldsymbol{z}}$ by solving the optimization problem:

$$\begin{aligned}\arg\min_{\tilde{\boldsymbol{z}}} &\int_{\mathcal{M}}\left(g_{\mathrm{bd}}\int_0^1\left|\dot{\tilde{\boldsymbol{z}}}\right|^2\mathrm{d}t\right)\mathrm{d}A\\ &+ c\int_{\mathcal{M}}\int_0^1\|(\mathrm{d}-i\omega(t))\,\tilde{\boldsymbol{z}}\|^2 + \left(|\tilde{\boldsymbol{z}}|^2-1\right)^2\ \mathrm{d}t\mathrm{d}A,\end{aligned} \tag{B.29}$$

where

$$c = c_1/\left(\frac{\alpha}{4}+\frac{\beta}{2}\right),\ g_{\mathrm{bd}} = \frac{(f^0)^2+(f^1)^2}{2} \tag{B.30}$$

To sum up, in the smooth case, the approximated geodesic can be computed as follows:

$$a(t) = (1-t)a^0 + ta^1 \tag{B.31}$$

$$f(t) = (1-t)f^0 + tf^1 = (1-t)a^0\|\omega^0\|^2_{\boldsymbol{I}^{-1}} + ta^1\|\omega^1\|^2_{\boldsymbol{I}^{-1}} \tag{B.32}$$

$$\phi(t) = (1-t)\phi^0 + t\phi^1 \tag{B.33}$$

$$\omega(t) = \sqrt{f(t)/a(t)}\hat{r}_x^T\left[R(\hat{n},\phi(t))\right]^T\mathrm{d}r, \tag{B.34}$$

and $\tilde{\boldsymbol{z}}$ is the optimal solution for Equation (B.29) with constant penalty coefficient c. For the regions with singularities, where $a^0|_{\boldsymbol{P}} = 0$, or $a^1|_{\boldsymbol{P}} = 0$, $\phi(t)$ is not well defined. In that case, we simply linearly interpolate $\omega(t)$ as $(1-t)\omega^0 + t\omega^1$.

B.1.3 Discrete Geodesics

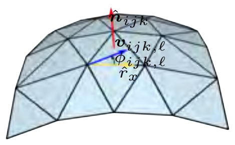
In the previous section, we provide the formula to compute the smooth geodesic between two complex fields $\boldsymbol{z}^0$ and $\boldsymbol{z}^1$. In this section, we provide the corresponding discretization w.r.t. two *CWF*s: $(\omega^0, a^0, \tilde{\boldsymbol{z}}^0)$ and $(\omega^1, a^1, \tilde{\boldsymbol{z}}^1)$, where $\omega^{0,1}$ are the edge-discretized one-forms, while $a^{0,1}$ and $\tilde{\boldsymbol{z}}^{0,1}$ are defined per-vertex.

We start with some notation. Let N_j be the set of all triangle faces incident to vertex $\mathbf{v}_j$, and N_{jk} the faces incident to edge $\mathbf{e}_{jk}$. Notice that Equation (B.31) can be directly discretized for per-vertex amplitude as

$$a_j^t = (1-t)a_j^0 + ta_j^1. \tag{B.35}$$

What remains is to discuss the discretization of Equation (B.34) for ω, and the corresponding optimization (B.29) for $\tilde{\boldsymbol{z}}$.

Discrete f We first discuss the discretization of f. From Equation (B.13), we know that in the smooth setting, f represents the squared magnitude of the tangential vector $\boldsymbol{v} = \sqrt{a}\mathrm{d}r\boldsymbol{I}^{-1}\omega$. We discretize it as a scalar per face-vertex; that is, for each face $\mathbf{f}_{jk\ell}$ and each vertex $\mathbf{v}_m, m \in \{j, k, \ell\}$, we assign an $f_{jk\ell,m}$. More specifically, for each vertex $\mathbf{v}_m, m \in \{j, k, \ell\}$, we discretize $\boldsymbol{v}$ as follows:



$$\boldsymbol{v}_{jk\ell,m} = \sqrt{a_m}\boldsymbol{u}_{jk\ell,m} \tag{B.36}$$

where $\boldsymbol{u}_{jk\ell,m}$ can be computed according to Equation (B.51) in Section B.3. Then we set

$$f_{jk\ell,m} = \|\boldsymbol{v}_{jk\ell,m}\|^2 = a_m \|\boldsymbol{u}_{jk\ell,m}\|^2. \tag{B.37}$$

Since a^0_m, $a^1_,$, $\boldsymbol{u}^0_{jk\ell,m}$, and $\boldsymbol{u}^1_{jk\ell,m}$ are given by the boundary conditions, we conclude that

$$f^t_{jk\ell,m} = (1-t)a^0_m \left\|\boldsymbol{u}^0_{jk\ell,m}\right\|^2 + ta^1_m \left\|\boldsymbol{u}^1_{jk\ell,m}\right\|^2, \tag{B.38}$$

and that for each vertex $\mathbf{v}_j$,

$$\begin{aligned}(g_{\text{bd}})_j &= \left(\frac{(f^0)^2 + (f^1)^2}{2}\right)_j && \text{(B.39)}\\ &= \frac{1}{2|\mathrm{N}_j|} \sum_{F_{jk\ell}\in \mathrm{N}_j} \left(a^0_j \left\|\boldsymbol{u}^0_{jk\ell,j}\right\|^2\right)^2 + \left(a^1_j \left\|\boldsymbol{u}^1_{jk\ell,j}\right\|^2\right)^2. && \text{(B.40)}\end{aligned}$$

Discrete ω formula. Our next step is to discretize Equation (B.34). Consider an edge $\mathbf{e}_{jk}$. Consider one of its incident faces $F_{jk\ell}$ and one of its endpoints $\mathbf{v}_m$, $m \in \{j,k\}$. We can get the corresponding a^t_m and $f^t_{jk\ell,m}$ according to Equations (B.35) and (B.38) respectively. We choose

$$\begin{aligned} r_x &= e_{jk} := \mathbf{v}_k - \mathbf{v}_j && \text{(B.41)}\\ \hat{r}_x &= r_x/\|r_x\|, && \text{(B.42)}\end{aligned}$$

and $\phi_{jk\ell,m}$ is the corresponding angle between $\hat{r}_x$ and $\boldsymbol{v}_{jk\ell,m}$, we have

$$\phi^t_{jk\ell,m} = (1-t)\phi^0_{jk\ell,m} + t\phi^1_{jk\ell,m}. \tag{B.43}$$

The one-form contribution from vertex $\mathbf{v}_m$ on face $F_{jk\ell}$, is then computed as

$$\begin{aligned}\omega^t_{jk\ell,m} &= \sqrt{f^t_{jk\ell,m}/a^t_m}\,\hat{e}^T_{jk} \left[R(\hat{\boldsymbol{n}}_{jk\ell}, \phi^t_{jk\ell,m})\right]^T e_{jk} && \text{(B.44)}\\ &= \|e_{jk}\| \sqrt{f^t_{jk\ell,m}/a^t_m} \cos\left(\phi^t_{jk\ell,m}\right). && \text{(B.45)}\end{aligned}$$

Therefore

$$\omega^t_{jk} = \frac{\|\mathbf{v}_k - \mathbf{v}_j\|}{2|\mathrm{N}_{jk}|} \sum_{\mathbf{f}_{jk\ell}\in \mathrm{N}_{jk}} \sum_{m\in\{j,k\}} \sqrt{f^t_{jk\ell,m}/a^t_m} \cos\left(\phi^t_{jk\ell,m}\right). \tag{B.46}$$

Discrete $\tilde{\boldsymbol{z}}$ formula. The final step is to discretize Equation (B.29). In the main text, we have already provided the discretization of $|(\mathrm{d}-i\omega)\tilde{\boldsymbol{z}}|$ and $(|\tilde{\boldsymbol{z}}|^2-1)^2$. Additionally, Equation (B.40) offers the discretization of g_{bd}. The only remaining part of Equation (B.29) is $\dot{\tilde{\boldsymbol{z}}}$, which can be directly discretized using standard finite difference:

$$\dot{\tilde{\boldsymbol{z}}}_j^t = \frac{\tilde{\boldsymbol{z}}_j^{t+\delta t} - \tilde{\boldsymbol{z}}_j^t}{\delta t}. \tag{B.47}$$

Optimizing Equation (B.29) with respect to $\left\{\tilde{\boldsymbol{z}}_j^{n\cdot\delta t} = x_j^{n\cdot\delta t} + iy^{n\cdot\delta t}j\right\}_j^n$ yields the final optimal $\tilde{\boldsymbol{z}}$.

B.2 Loop Subdivision Rules for 1-forms

We modified the Loop scheme provided in Wang et al. (2006); de Goes et al. (2016a), such that the corresponding mask for the one-form is still consistent with the boundary fixed Loop scheme. We listed the changes to the boundary rules in Figure B.1. For the interior rules, we refer the reader to the supplementary material provided by de Goes et al. (2016a).

B.3 Contangent Vector Computation

In this section, we provide the formula of acting the cotangent vector ω at a point $\boldsymbol{P}$ (in the tangent space) on a tangent direction d. Setting $\boldsymbol{P} = \boldsymbol{P}_0$ and $d = \boldsymbol{P}_1^* - \boldsymbol{P}_0$ will give us the corresponding formual mentioned in the main text.

To begin with, consider a face $\mathbf{f}$ with vertices $\mathbf{v}_0, \mathbf{v}_1$ and $\mathbf{v}_2$, and edge 1-forms ω_{01}, ω_{12}, and ω_{20}. The barycentric coordinates of $\boldsymbol{P}$ is (α_0, α_1, α_2):

$$\boldsymbol{P} = \alpha_0\mathbf{v}_0 + \alpha_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2 \tag{B.48}$$

For any tangent direction d,

$$\omega(\boldsymbol{P})(d) = \boldsymbol{v}_{\tan}(\boldsymbol{P}) \cdot d \approx \sum_{i=0}^{2} \alpha_i(\boldsymbol{v}_i \cdot d) \tag{B.49}$$

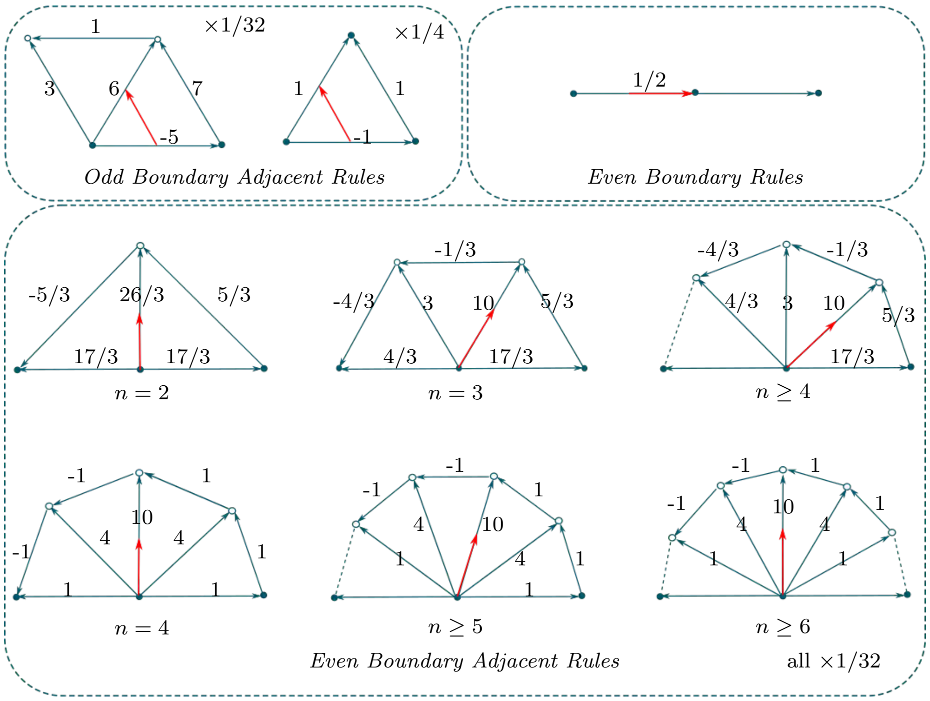


Figure B.1: Loop subdivision rules for boundary 1-forms

where $\boldsymbol{v}_{\text{tan}}$ is the corresponding tangent gradient of ω. Here we approximate $\boldsymbol{v}_{\text{tan}}$ by linearly blending from the corresponding values at triangle corners (denoted as $\boldsymbol{v}_i$).

The tangential gradient $\boldsymbol{v}_0$ of vertex $\mathbf{v}_0$ should satisfy

$$\begin{aligned} \boldsymbol{v}_0 \cdot (\mathbf{v}_1 - \mathbf{v}_0) &= \omega_{01} \\ \boldsymbol{v}_0 \cdot (\mathbf{v}_2 - \mathbf{v}_0) &= -\omega_{20} \end{aligned} \tag{B.50}$$

Expressing $\boldsymbol{v}_0$ in the tangential plane basis gives us:

$$\boldsymbol{v}_0 = x(\mathbf{v}_1 - \mathbf{v}_0) + y(\mathbf{v}_2 - \mathbf{v}_0), \tag{B.51}$$

where

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \|\mathbf{v}_1 - \mathbf{v}_0\|^2 & (\mathbf{v}_1 - \mathbf{v}_0) \cdot (\mathbf{v}_2 - \mathbf{v}_0) \\ (\mathbf{v}_1 - \mathbf{v}_0) \cdot (\mathbf{v}_2 - \mathbf{v}_0) & \|\mathbf{v}_2 - \mathbf{v}_0\|^2 \end{bmatrix}^{-1} \begin{bmatrix} \omega_{01} \\ -\omega_{20} \end{bmatrix} \tag{B.52}$$

$\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ can be computed in a similar manner.

Models	Figures	|V|	|F|	|V′|	|F′|	#iter	T_{CHOL} (s)	T_{tol} (s)
torus global rotation	B.10	400	800	400K	800K	243	123.30	127.13
cylinder local rotation	B.2	639	1222	246K	491K	129	30.61	31.92
bunny global rotation	3.10	502	1000	512K	1.0M	381	254.53	263.09
bunny local rotation	3.7	502	1000	512K	1.0M	127	89.36	92.31
fertility global enlargement	3.5	494	1000	511K	1.0M	298	178.91	185.74
fertility local enlargement	3.6	494	1000	511K	1.0M	94	63.52	65.69
pantasma local enlargement	B.9	850	1696	868K	1.7M	109	172.01	176.05
spot composite editing	3.19	502	1000	512K	1.0M	154	82.19	85.65
spot user designed editing	B.5	502	1000	512K	1.0M	212	123.21	127.92
user designed face	B.4	1026	1999	1.0M	2.0M	151	334.46	341.79
user designed hand	B.3	1012	2000	1.0M	2.0M	94	173.12	177.69
user designed dress	B.6	1333	2586	1.5M	3.0M	144	324.59	333.65
pants	B.8	1677	3304	1.6M	3.3M	262	710.14	728.63
dress	B.7	1525	2950	1.5M	3.0M	96	303.08	310.65
Average	-	820	1617	802K	1.6M	188	225.17	231.62

Table B.1: The time and convergence information for all the examples showing in this chapter. |V| and |F| are the number of vertices and faces of the base coarse mesh, and |V′| and |F′| are the corresponding values of the upsampled wrinkled mesh, where 4 or 5 levels of upsampling are used. The major part of time (∼90%) is costed by the CHOLMOD linear solve within the Newton's method. Note that the efficiency can be improved by fine-tuning the number of frames we use. See Appendix B.7.1 for more discussion.

B.4 Timing

We report the timing of our interpolation approach in Table B.1. We ran our experiments on a desktop with a 8-core Intel Core i9-9900K CPU, clocked at 3.6 GHz and 64 GB of memory. We use 50 frames in total and linear interpolants inbetween to get a final set of 200 frames used for animation generation. To get optimized results, we let the Newton solver run until the gradient L2-norm is smaller than 10^{-6}.

B.5 Additional Editing Examples

We used our wrinkle editing tools to create several additional examples of smooth interpolation of wrinkles on complex surfaces. Please see the supplemental video for animations of these examples. Figure B.2 shows the result of locally rotating a small user-specified patch of the circular wave on the cylinder by ninety degrees.

Figure B.3 shows interpolation between two wrinkle patterns on a hand, where the second keyframe was created from the first by rotating the left red patch 45 degrees and halving the wrinkle frequency and doubling the amplitude in the right red patch. In Figure B.4, we generate the target frame by rotating the wrinkles on the cheek (the left red patch) by 90 degrees, increasing the wrinkle frequency by 1.5 times in the middle red patch beneath the mouth, and doubling the frequency on the bottom red patch. Figure B.5 shows an interpolation of two user-designed waves on Spot, where different red patches have different rotation angles. Figure B.6 shows the possibility to design and interpolate wrinkles on cloth. All of these examples show complicated wrinkle shapes with many singularities (the blue dots in the amplitude figures). We successfully get *local* smooth wrinkle evolution on all of these user-designed examples.

Editing Simulation Results Our tools can be used to edit wrinkles originally computed via physical simulation. We show two examples: Chen et al. (2021b)'s dress and pants. For the dress (Figure B.7), we set the second keyframe by uniformly enlarging the wrinkle frequency and at the same time halving the wrinkle amplitude to mimic a change in cloth thickness. For the pants (Figure B.8), we change the frequency on the two legs differently, by enlarging the frequency in the middle of the left leg and shrinking the frequency in the middle of the right leg. Again we changed the amplitude to keep $\|a \cdot \omega\|$ constant. We find that our approach mimic wrinkle motion reasonably well.

B.6 Extra Comparison Examples

Figures B.9 and B.10 show extra comparisons between our approach and alternative keyframe interpolation methods.

B.7 Ablation Study

In this section we study different choices for the number of guide frames N' (with $N = 200$ fixed), and solver penalty c.

B.7.1 Number of Guide Frames

In this section, we show the results of different choices of number of guide frames N', where we fix the penalty coefficient to $10^3 g_{\text{ave}}$. In Figures B.11, B.12, B.13, we show the results of wrinkles, amplitude and phase patterns respectively for different number of frames, where $N' = 1$ implies that we directly linearly interpolate keyframes. We find that as we increase the number of frames, the final interpolated results turn to converge. In particular, the differences between $N' = 25, 50, 100$ and 200 are subtle, especially for the last three. (You can see subtle differences between $N' = 25$ and the others at $t = 0.5$ and $t = 0.875$.)

B.7.2 Penalty Coefficient

As we discussed in the main text, when actually solving for the geodesic between two *CWF*s, we use a penalty $c = 10^3 g_{\text{ave}}$ to control the balance between temporal coherence and compatibility, where the later affects the semantic meaning of the final upsampled wrinkled surface. In Figures B.14, B.15, we show different choices of c with 50 frames. If c is too small, for example $c = 10 g_{\text{ave}}$, the kinetic term dominates the optimization, and the end result has bad compatibility. The corresponding final wrinkled surface has lots of unexpected singularities, with undesired aliased visual artifacts. On the other hand, if c is too large, for example $c = 10^5 g_{\text{ave}}$, the compatibility term and unit term dominate the optimization, and temporal incoherence will not be penalized. This leads to spatially appealing, but temporally incoherent, wrinkle sequences. For the values in between, the smaller the c, the more "local" patches of rotating regions we will have. At the same time, E_{compat} is a quadratic convex term, while E_{unit} is a quartic, non-convex term. Increasing c will increase the impact of these two terms. We found that it will slow down the optimization, then speed it up (see the first column in Figure B.14). How to find the best choice for c for each example to achieve the best trade-off between animation quality (spatial and temporal) and solving speed needs further exploration, and we treat this as future work.

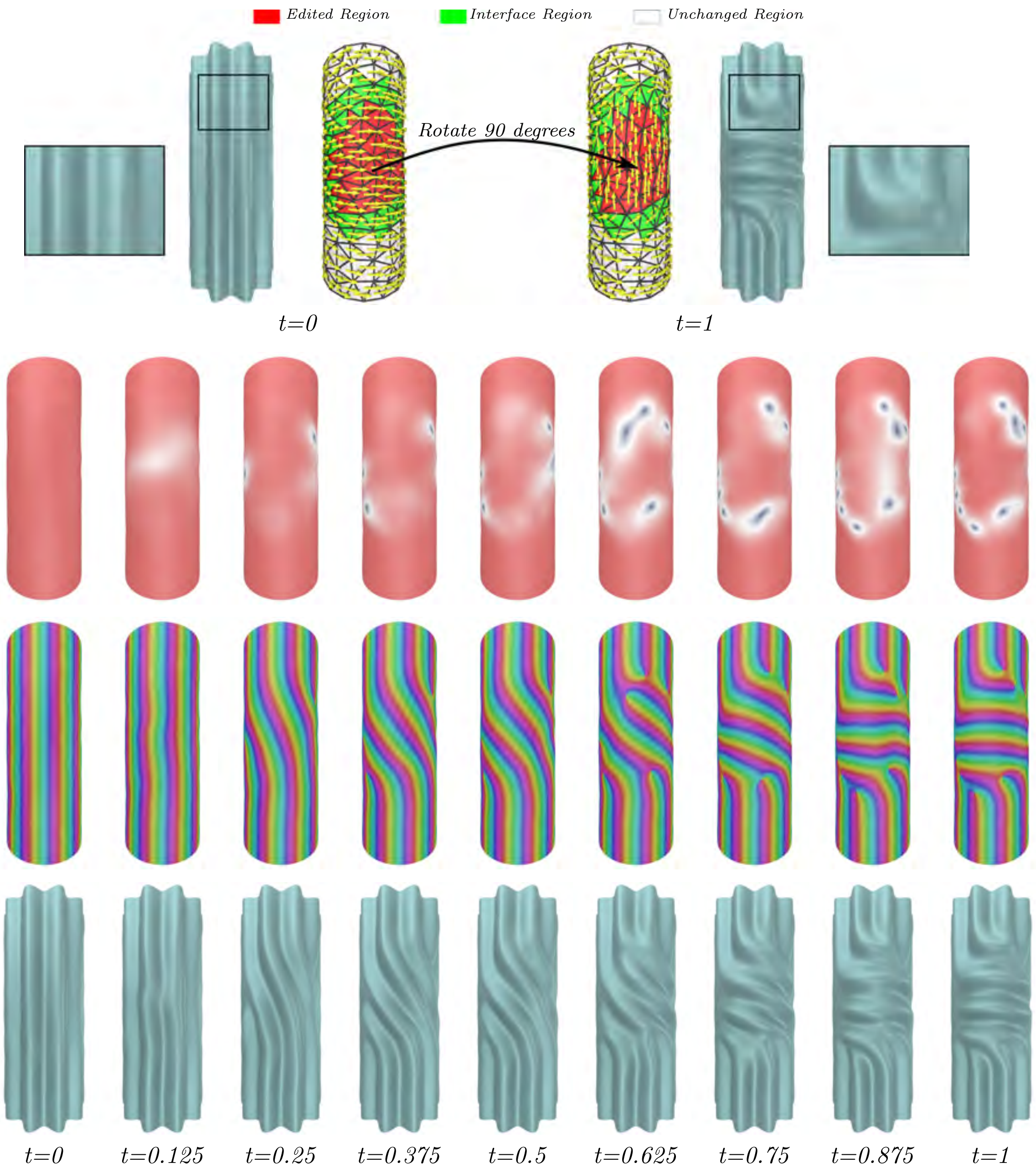


Figure B.2: Local rotation of an axial wrinkle pattern on a cylinder. The two keyframes are given in the left/right side; the wrinkle frequency has been rotated by ninety degrees in the red region of the second keyframe. The corresponding wrinkle animation is given in the **Interpolation Results** video in the supplementary materials near 00:20-00:37.

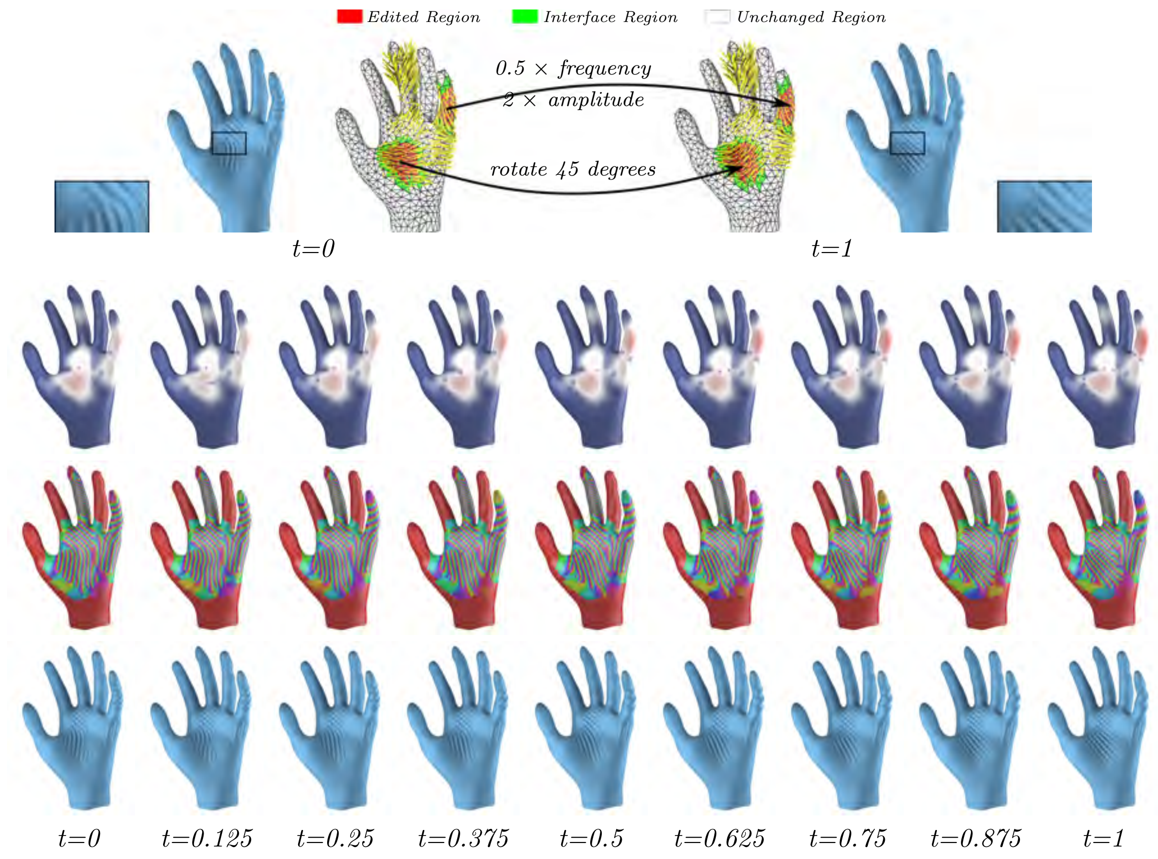


Figure B.3: *CWF* interpolation of two wrinkle patterns on the hand. Please check the **Interpolation Results** video in the supplementary materials for more details (timestamp 03:13-03:29).

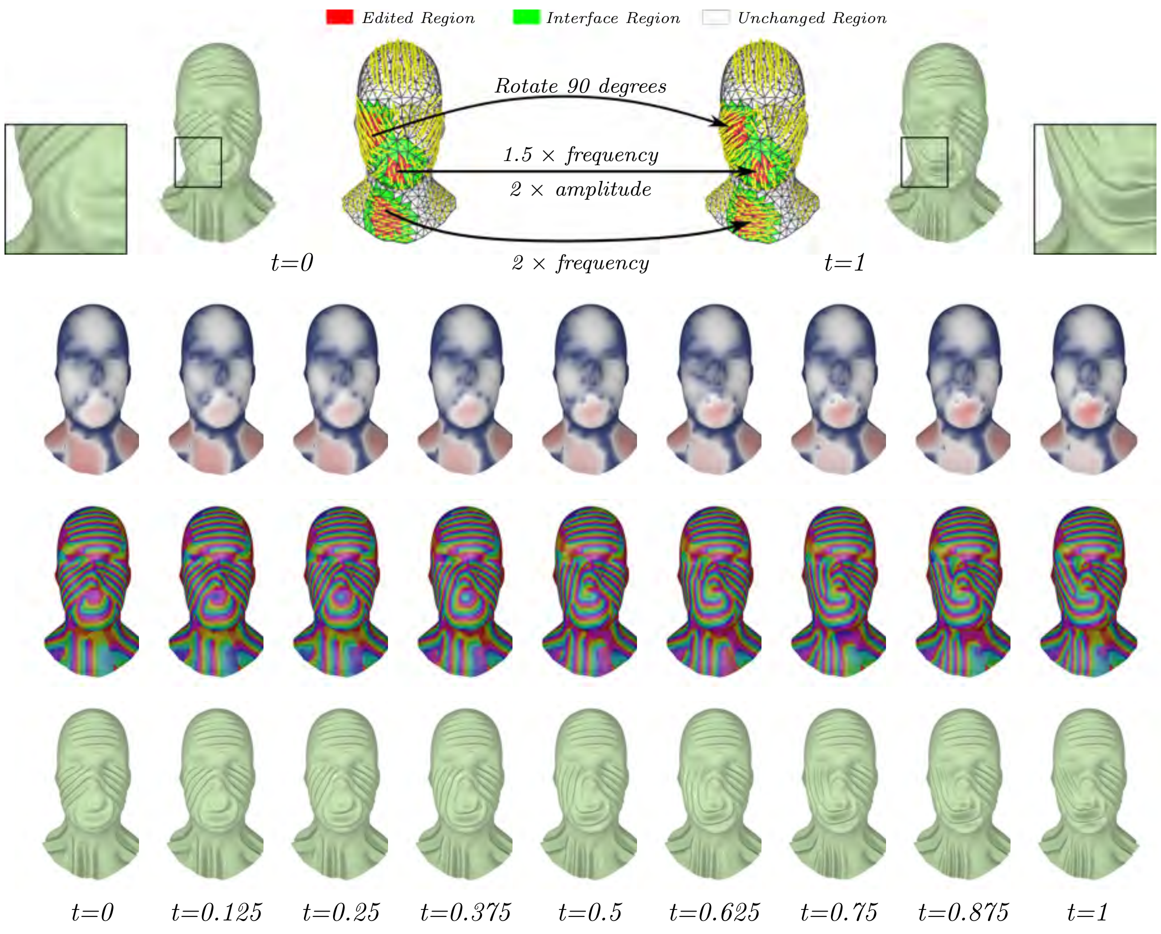


Figure B.4: *CWF* interpolation of two wrinkle patterns on the face. Please check the **Interpolation Results** video in the supplementary materials for more details (timestamp 02:56–03:12).

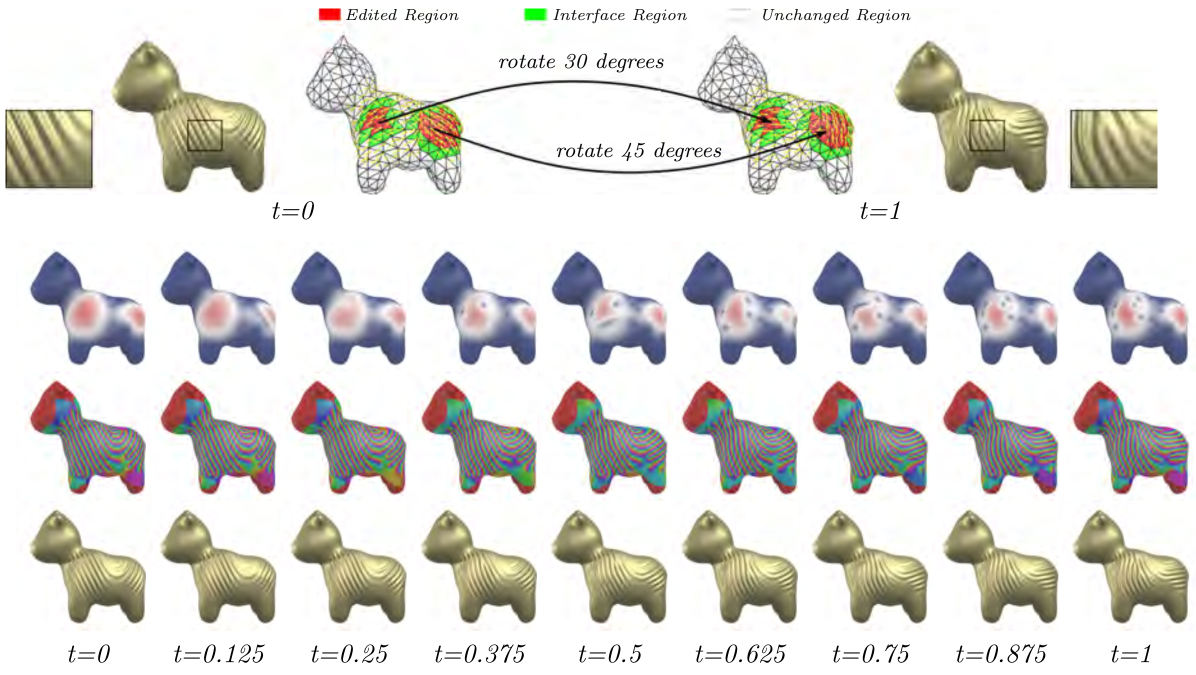


Figure B.5: *CWF* interpolation of two wrinkle patterns on Spot the cow. Please refer to the **Interpolation Results** video in the supplementary materials for the corresponding wrinkle animation (timestamp 02:38-02:55).

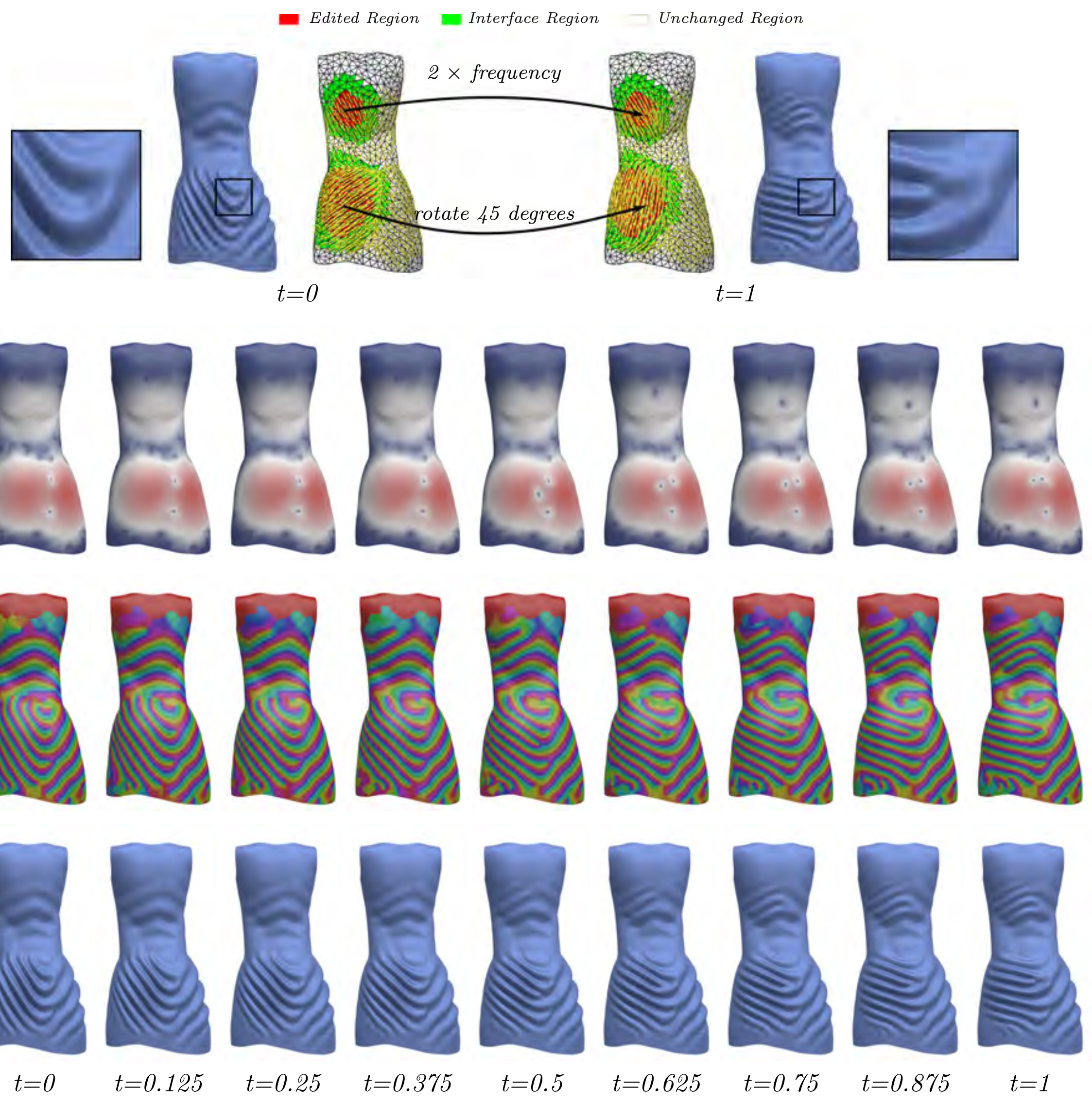


Figure B.6: *CWF* interpolation of two wrinkle patterns on the dress. One can easily see how the wrinkles and the singularities move in the **Interpolation Results** video of the supplementary materials (timestamp 03:30-03:46).

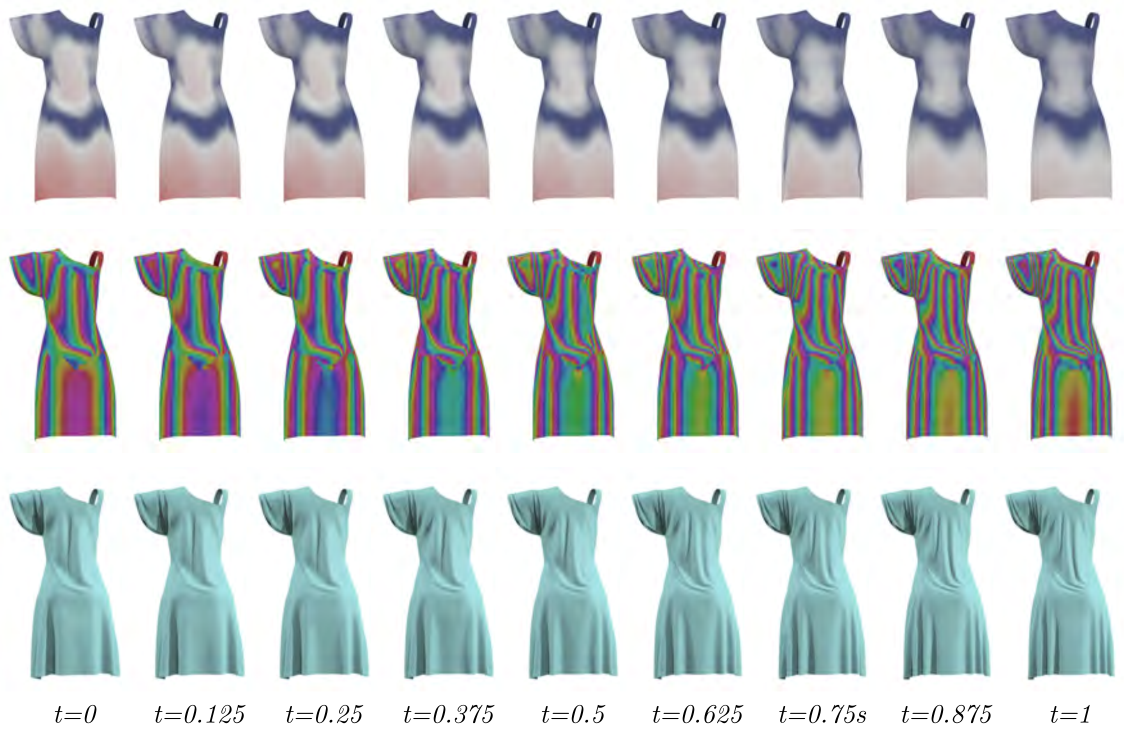


Figure B.7: We uniformly double the wrinkle frequency and halve the amplitude on the dress examples from Chen et al. (2021b), emulating replacing the cloth with a thinner material. Please check the **Interpolation Results** video in the supplementary materials at 03:47-04:03 for the corresponding wrinkle animation.

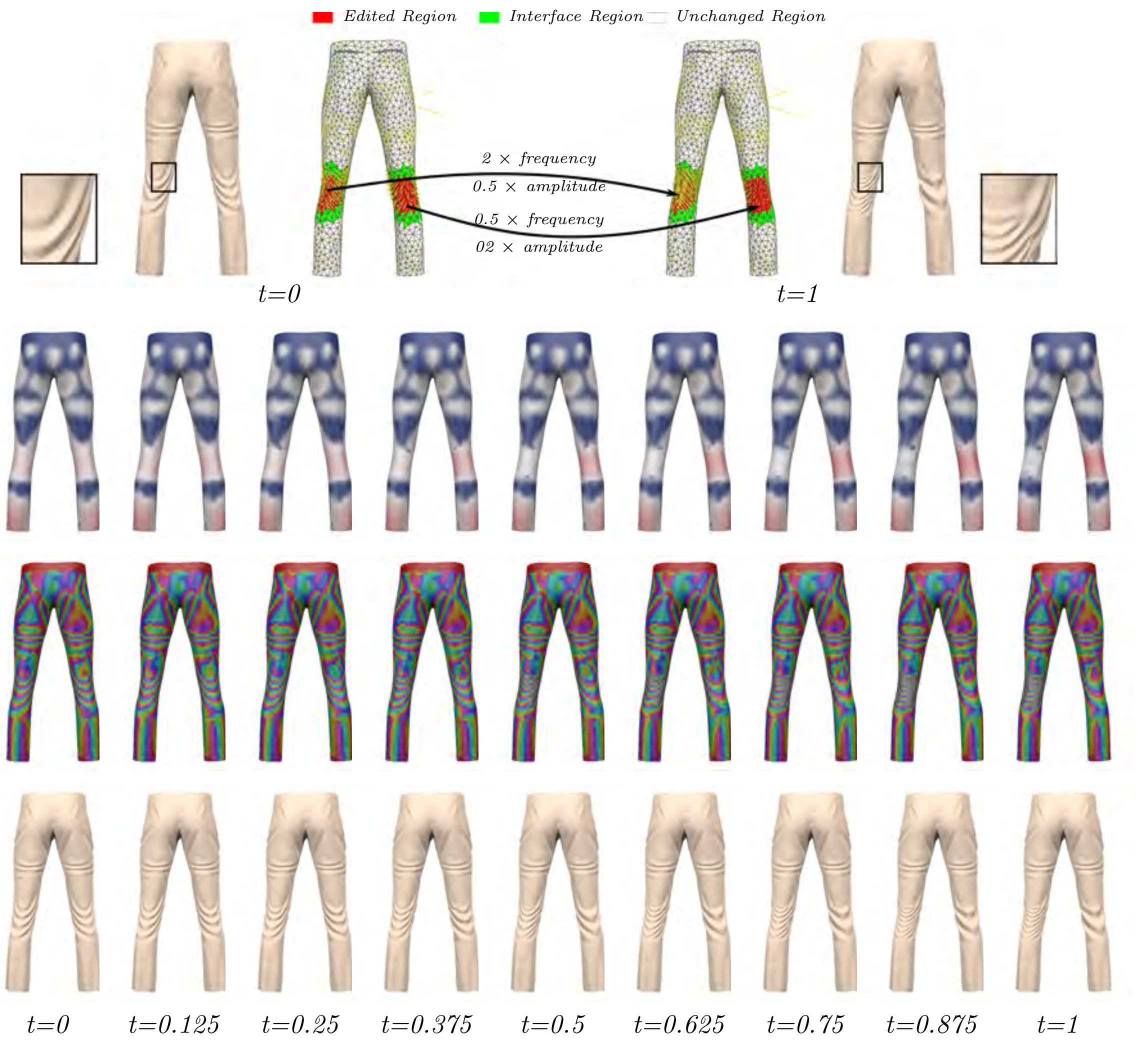


Figure B.8: *CWF* interpolation of two wrinkle patterns on the pants, where the initial frames are generated using the code provided by Chen et al. (2021b). Please check the **Interpolation Results** video in supplementary materials at 04:04-04:21 for the corresponding wrinkle animation.

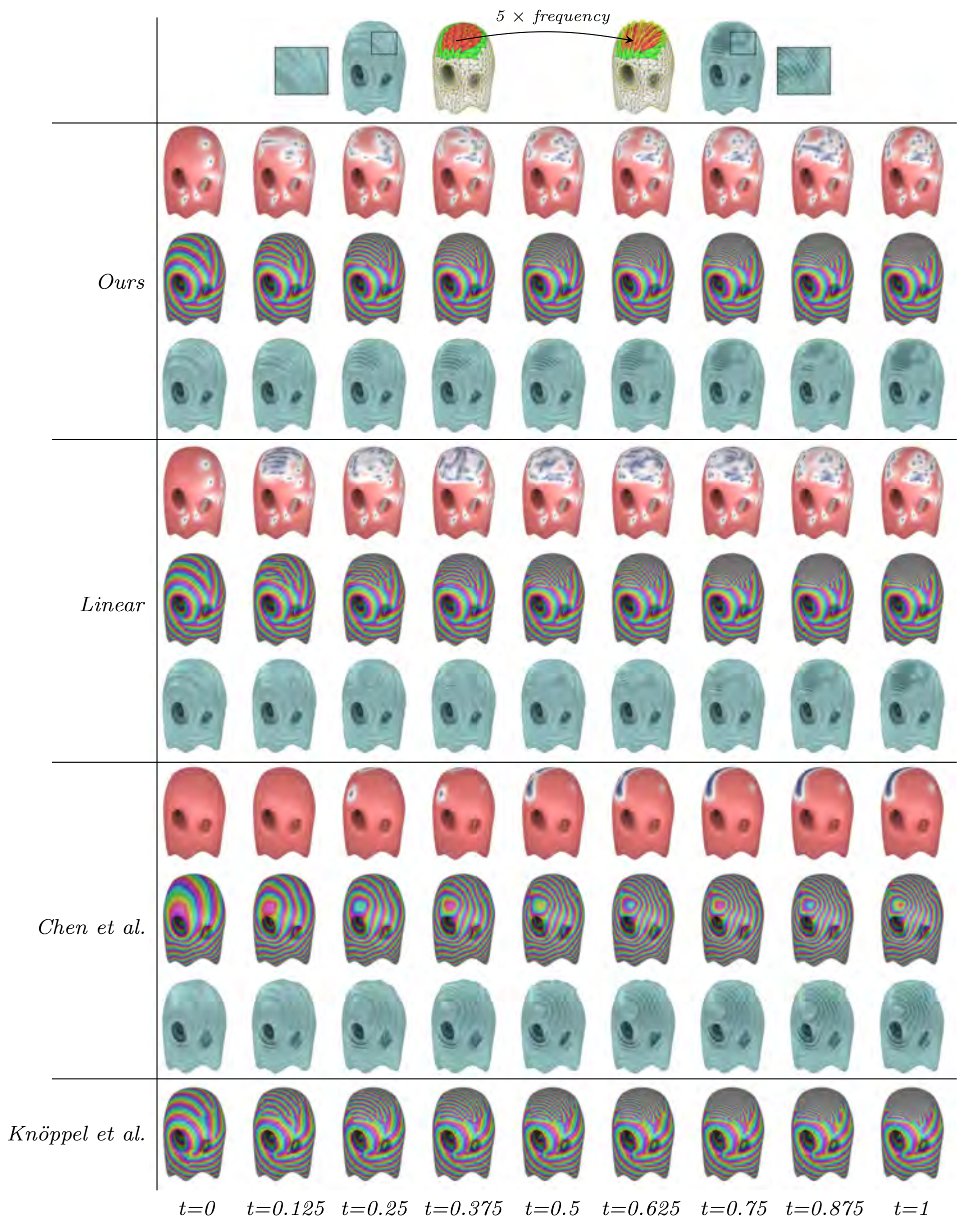


Figure B.9: The results of enlarging a local patch of the phantasma model (see the first row for details). You can see clear amplitude artifacts when using direct linear interpolation, and strange phase patterns when using the method proposed by Chen et al. (2021b). Although the method of Knöppel et al. (2015) produces beautiful static phase patterns, it suffers from temporal incoherence. Similar temporal incoherence can be seen in Chen et al. (2021b). These coherence issues can be better visualize in the **main** supplementary video (at 04:05-04:27) and in the **Comparisons** supplementary video (at 01:57-02:58).

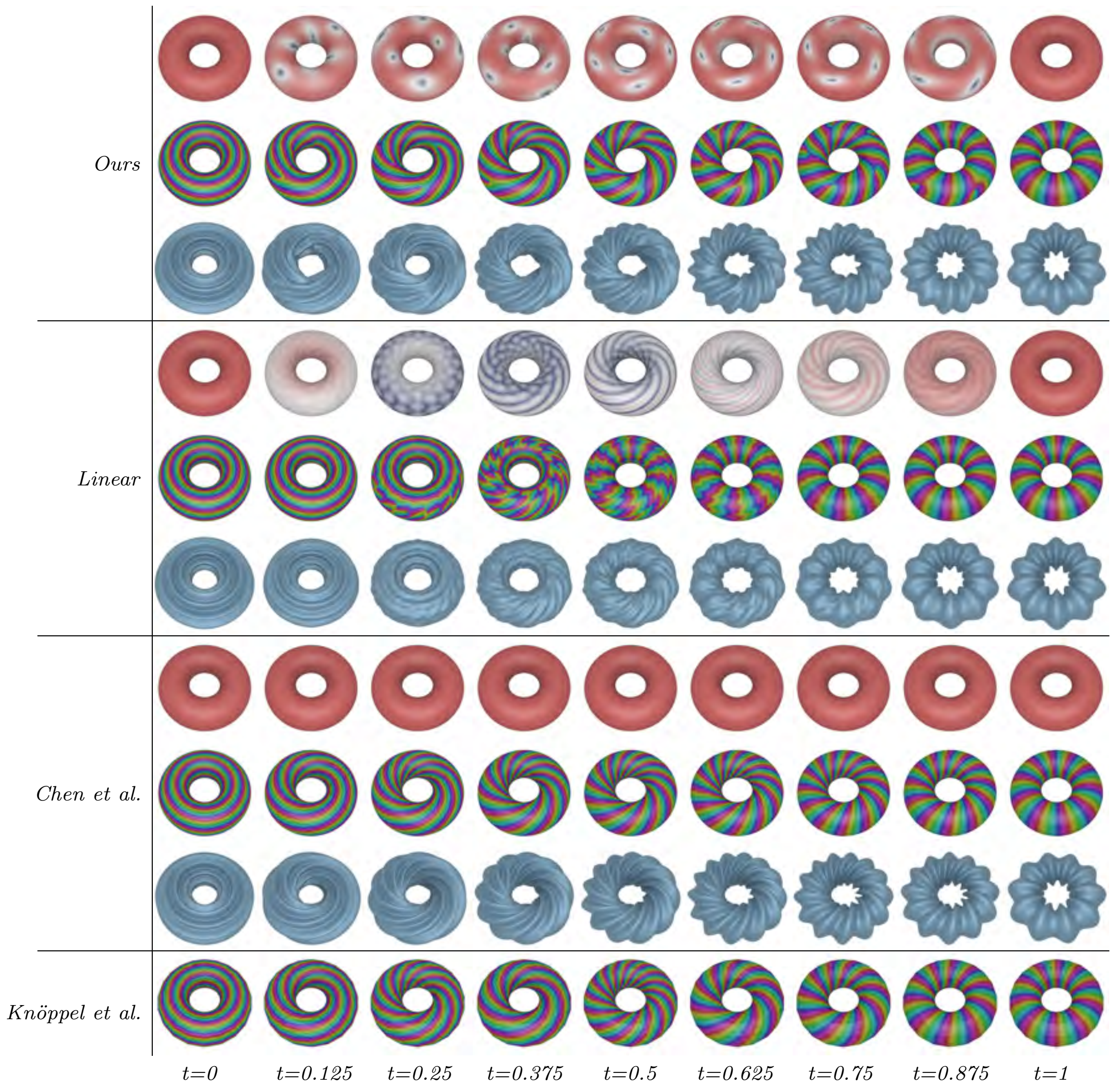


Figure B.10: The results of rotating wrinkles on a torus by ninety degrees. You can see clear amplitude artifacts when using direct linear interpolation. Although in this example Chen et al. (2021b) and Knöppel et al. (2015) produce beautiful static phase patterns with no singularities, the corresponding results suffer from temporal incoherence. These coherence issues can be seen in the **main** supplementary video at 02:50-03:44 and in the **Comparisons** supplementary video at 00:05-00:59.

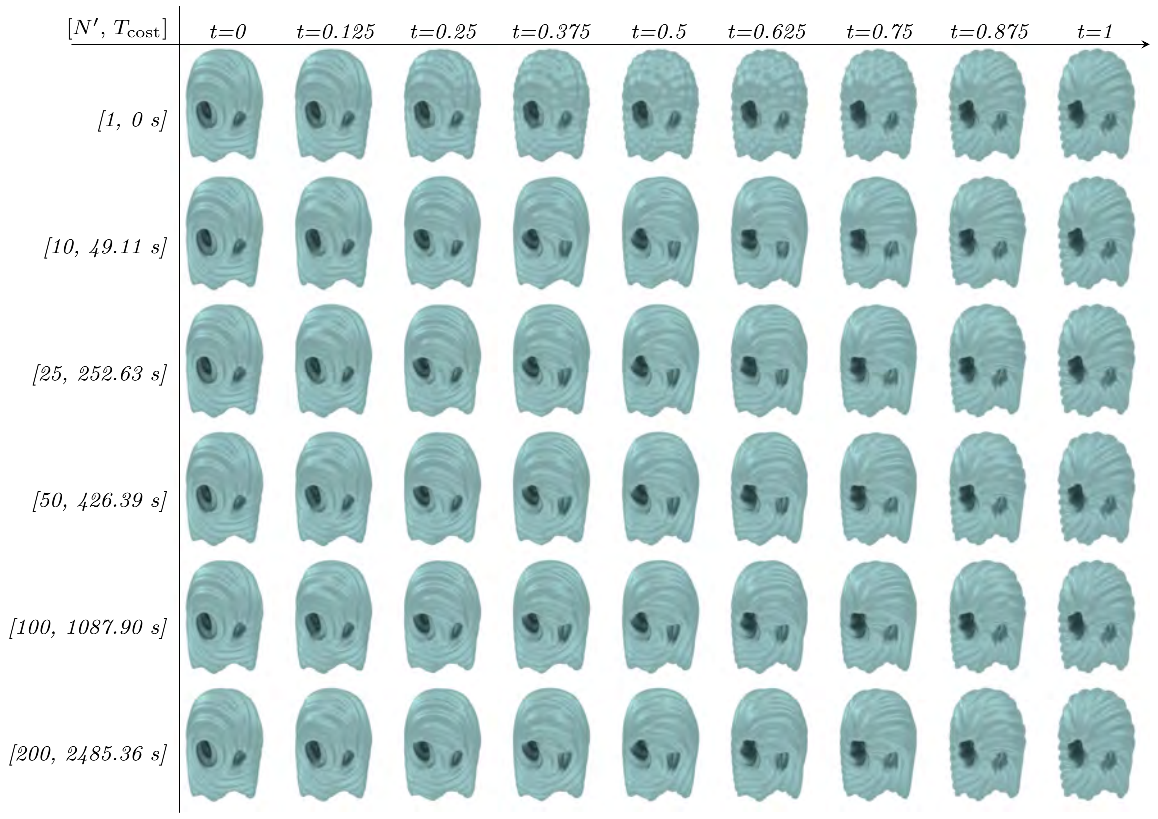


Figure B.11: Interpolation of rotating wrinkle patterns on the phantasma model by ninety degrees, where $[\bullet, \bullet]$ are the number of guide frames, and the computation time (in seconds), respectively. Notice that the differences between $N' = 25, 50, 100$ and 200 are subtle. You can refer to the **Ablation Experiments** video in the supplementary materials (00:14-00:31) for details.

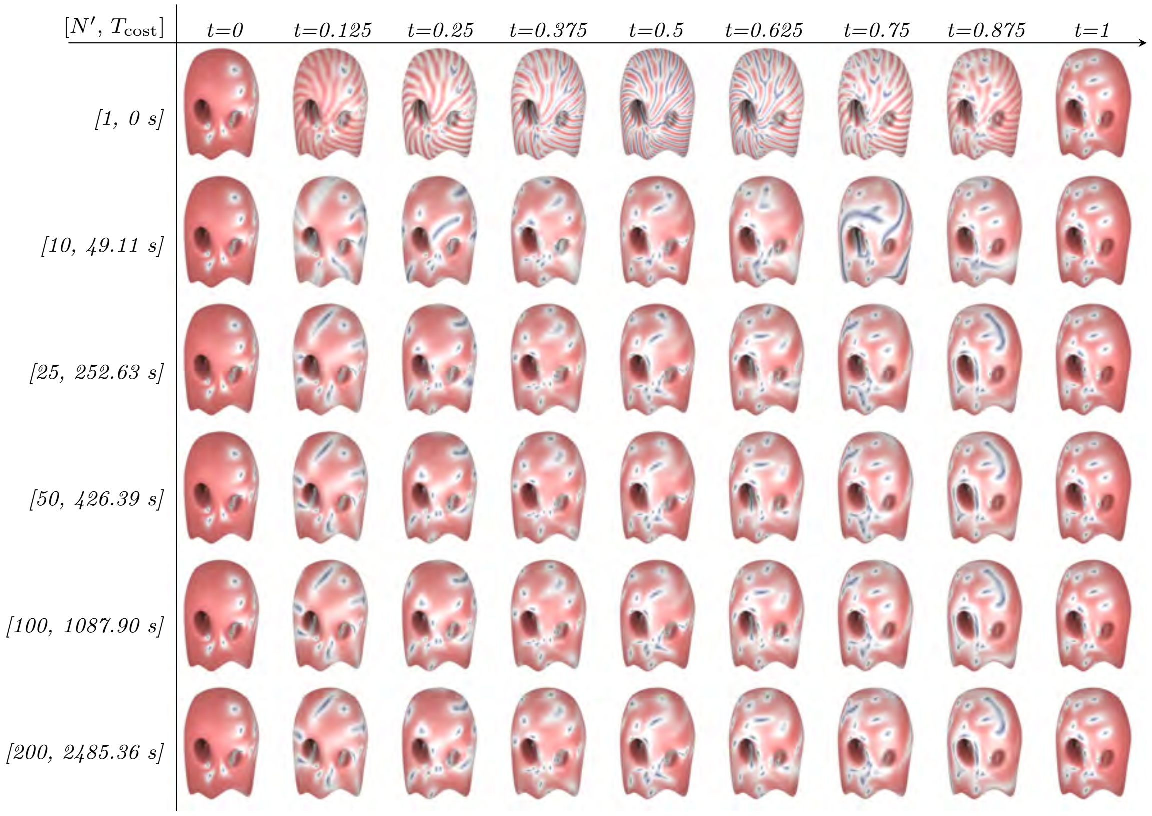


Figure B.12: The corresponding amplitude patterns of the example in Figure B.11. Again, please refer to the **Ablation Experiments** video in the supplementary materials (00:14-00:31) for details.

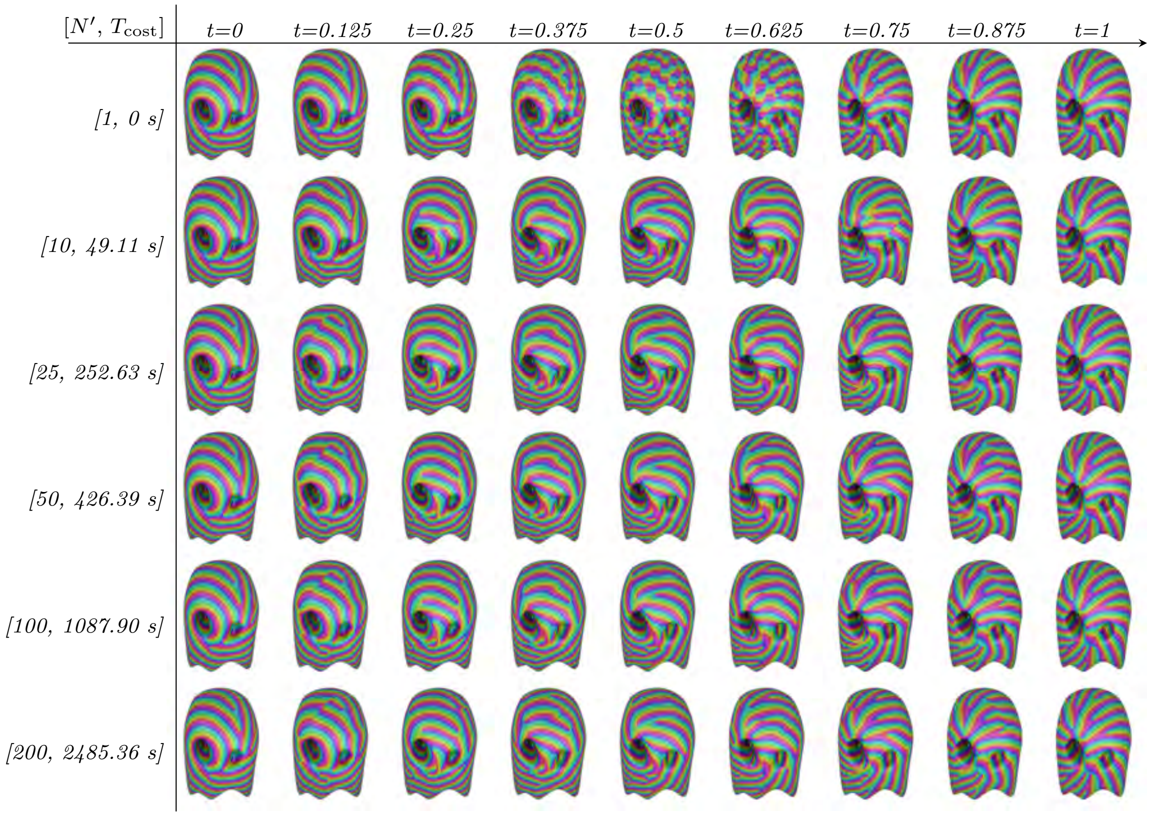


Figure B.13: The corresponding phase patterns of the example in Figure B.11. Again, please refer to the **Ablation Experiments** video in the supplementary materials (00:14-00:31) for details.

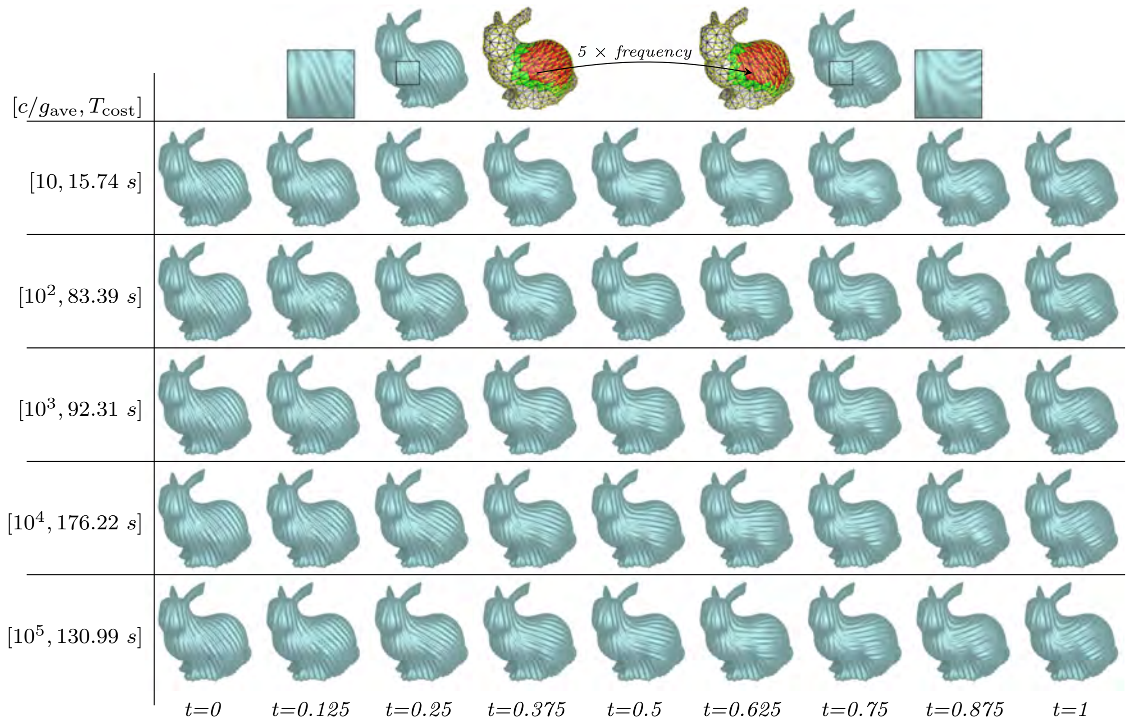


Figure B.14: *CWF* interpolation of two wrinkle patterns on the Stanford bunny model, where the target wrinkles are from locally rotating the corresponding patch of the first keyframe by ninety degrees. T_{cost} is the time cost of the interpolation solve. Please refer to the **Ablation Experiments** video in the supplementary materials at timestamp 00:44-01:07 for details.

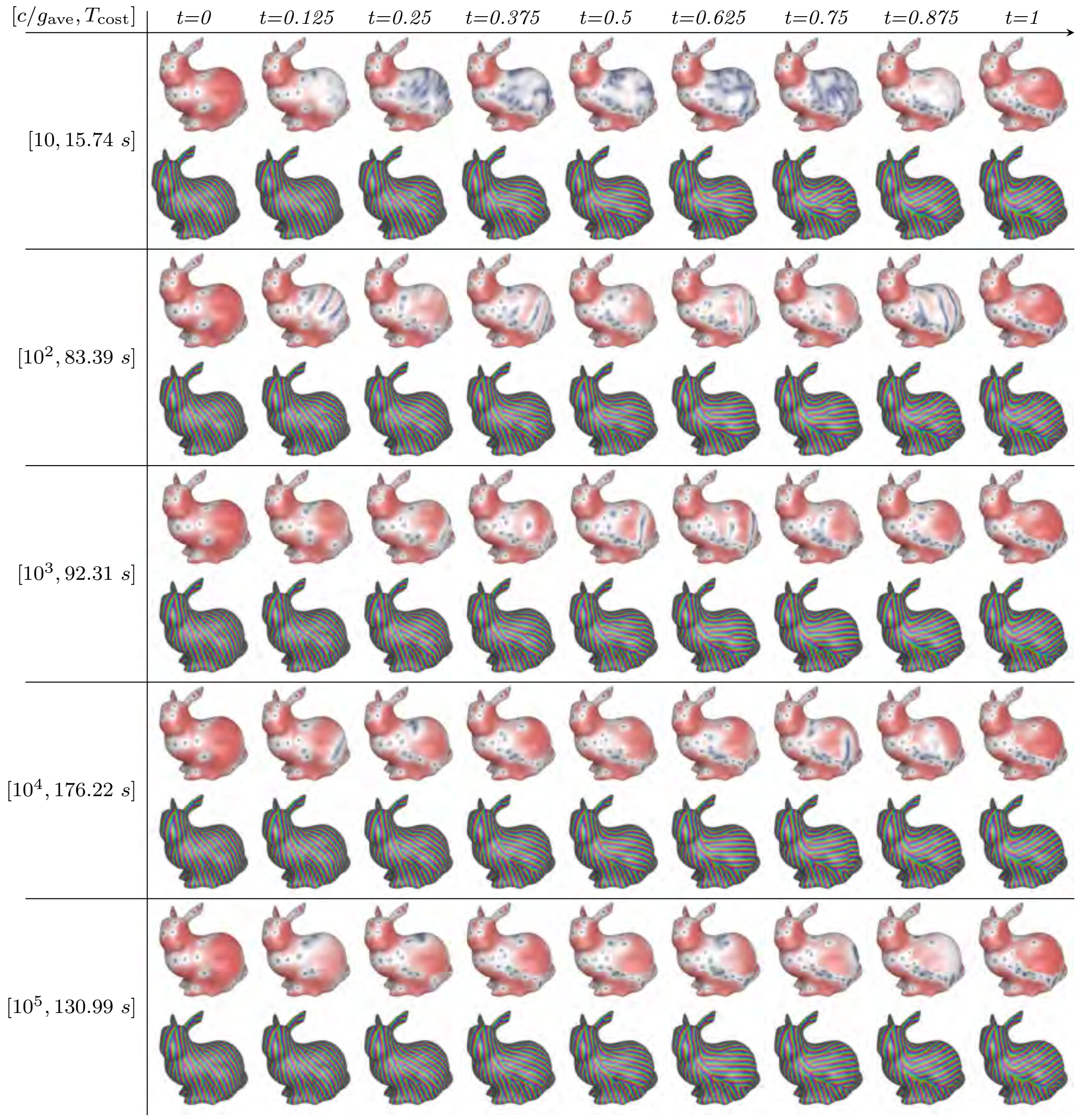


Figure B.15: The corresponding amplitude (odd rows) and phase (even rows) patterns of the example in Figure B.14. Please refer to the **Ablation Experiments** video in the supplementary materials at timestamp 00:44-01:07 for details.

Works Cited

Donya Labs AB. Simplygon 9, 2022. URL `https://www.simplygon.com/Home/Index#section-solutions`.

Hillel Aharoni, Desislava Todorova, Octavio Albarran, Lucas Goehring, Randall Kamien, and Eleni Katifori. The smectic order of wrinkles. *Nature Communications*, 8:15809, 07 2017. doi: 10.1038/ncomms15809.

George Biddell Airy. Tides and waves. 1842.

Marco Attene. A lightweight approach to repairing digitized polygon meshes. *The Visual Computer*, 26:1393–1406, 2010.

Marco Attene, Daniela Giorgi, Massimo Ferri, and Bianca Falcidieno. On converting sets of tetrahedra to combinatorial and pl manifolds. *Computer Aided Geometric Design*, 26(8):850–864, 2009. ISSN 0167-8396.

Frank Baginski. On the design and analysis of inflated membranes: Natural and pumpkin shaped balloons. *SIAM Journal on Applied Mathematics*, 65(3): 838–857, 2005. doi: 10.1137/S0036139903438478. URL `https://doi.org/10.1137/S0036139903438478`.

David Baraff and Andrew Witkin. Large steps in cloth simulation. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 43–54, New York, NY, USA, 1998. ACM. ISBN 0-89791-999-8. doi: 10.1145/280814.280821.

Gavin Barill, Neil G. Dickson, Ryan Schmidt, David I. W. Levin, and Alec Jacobson. Fast winding numbers for soups and clouds. *ACM Trans. Graph.*, 37(4), jul 2018. ISSN 0730-0301.

Jean-Philippe Bauchet and Florent Lafarge. Kinetic shape reconstruction. *ACM Trans. Graph.*, 39(5), jun 2020. ISSN 0730-0301.

Miklós Bergou, Max Wardetzky, David Harmon, Denis Zorin, and Eitan Grinspun. A quadratic bending model for inextensible surfaces. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, SGP '06, page 227–230, Goslar, DEU, 2006. Eurographics Association. ISBN 3905673363. doi: 10.1145/1281957.1281987.

Miklós Bergou, Saurabh Mathur, Max Wardetzky, and Eitan Grinspun. Tracks: Toward directable thin shells. *ACM Trans. Graph.*, 26(3):50–es, jul 2007a. ISSN 0730-0301. doi: 10.1145/1276377.1276439.

Miklós Bergou, Saurabh Mathur, Max Wardetzky, and Eitan Grinspun. Tracks: Toward directable thin shells. *ACM Trans. Graph.*, 26(3):50–es, July 2007b. ISSN 0730-0301. doi: 10.1145/1276377.1276439. URL `https://doi.org/10.1145/1276377.1276439`.

Narasimha Boddeti, Yunlong Tang, Kurt Maute, David W Rosen, and Martin L Dunn. Optimal design and manufacture of variable stiffness laminated continuous fiber reinforced composites. *Scientific reports*, 10(1):1–15, 2020.

David Bommes, Henrik Zimmer, and Leif Kobbelt. Mixed-integer quadrangulation. *ACM Trans. Graph.*, 28(3), jul 2009. ISSN 0730-0301. doi: 10.1145/1531326.1531383. URL `https://doi.org/10.1145/1531326.1531383`.

Eddy Boxerman and Uri Ascher. Decomposing cloth. page 153–161, 2004. doi: 10.1145/1028523.1028543. URL `https://doi.org/10.1145/1028523.1028543`.

Hervé Brönnimann, Andreas Fabri, Geert-Jan Giezeman, Susan Hert, Michael Hoffmann, Lutz Kettner, Sylvain Pion, and Stefan Schirra. 2D and 3D linear geometry kernel. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.5.1 edition, 2022.

Stéphane Calderon and Tamy Boubekeur. Bounding proxies for shape approximation. *ACM Trans. Graph.*, 36(4), jul 2017. ISSN 0730-0301.

Juan J. Casafranca, Gabriel Cirio, Alejandro Rodríguez, Eder Miguel, and Miguel A. Otaduy. Mixing yarns and triangles in cloth simulation. *Computer Graphics Forum*, 39(2):101–110, 2020. doi: 10.1111/cgf.13915. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13915`.

Enrique A. Cerda. Mechanics of scars. *Journal of Biomechanics*, 38(8):1598–1603, 2005. ISSN 0021-9290. doi: https://doi.org/10.1016/j.jbiomech.2004.07.026. URL `https://www.sciencedirect.com/science/article/pii/S0021929004003793`.

Enrique A. Cerda and Lakshminarayanan Mahadevan. Geometry and physics of wrinkling. *Physical review letters*, 90:074302, 03 2003. doi: 10.1103/PhysRevLett.90.074302.

Dominique Chapelle and K.J. Bathe. *The Finite Element Analysis of Shells - Fundamentals - Second Edition*. Computational Fluid and Solid Mechanics. Springer, 2011. doi: 10.1007/978-3-642-16408-8. URL `https://hal.inria.fr/hal-00654533`.

Anne-Laure Chauve, Patrick Labatut, and Jean-Philippe Pons. Robust piecewise-planar 3d reconstruction and completion from large-scale unstructured point data. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1261–1268, 2010.

Ding-Yun Chen, Xiao-Pei Tian, Yu-Te Shen, and Ming Ouhyoung. On visual similarity based 3d model retrieval. *Computer Graphics Forum*, 22(3):223–232, 2003.

Guoning Chen, Vivek Kwatra, Li-Yi Wei, Charles D. Hansen, and Eugene Zhang. Design of 2d time-varying vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 18(10):1717–1730, 2012. doi: 10.1109/TVCG.2011.290.

Hsiao-Yu Chen, Arnav Sastry, Wim M. van Rees, and Etienne Vouga. Physical simulation of environmentally induced thin shell deformation. *ACM Trans. Graph.*, 37(4):146:1–146:13, July 2018a. ISSN 0730-0301. doi: 10.1145/3197517.3201395. URL `http://doi.acm.org/10.1145/3197517.3201395`.

Lan Chen, Juntao Ye, Liguo Jiang, Chengcheng Ma, Zhanglin Cheng, and Xiaopeng Zhang. Synthesizing cloth wrinkles by cnn-based geometry image superresolution. *Computer Animation and Virtual Worlds*, 29:e1810, 05 2018b. doi: 10.1002/cav.1810.

Lan Chen, Juntao Ye, and Xiaopeng Zhang. Multi-feature super-resolution network for cloth wrinkle synthesis. *Journal of Computer Science and Technology*, 36:478–493, 06 2021a. doi: 10.1007/s11390-021-1331-y.

Yanqing Chen, Timothy A. Davis, William W. Hager, and Sivasankaran Rajamanickam. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Trans. Math. Softw.*, 35(3), October 2008. ISSN 0098-3500. doi: 10.1145/1391989.1391995. URL `https://doi.org/10.1145/1391989.1391995`.

Zhen Chen, Hsiao-Yu Chen, Danny M. Kaufman, Mélina Skouras, and Etienne Vouga. Fine wrinkling on coarsely meshed thin shells. *ACM Trans. Graph.*, 40(5), aug 2021b. ISSN 0730-0301. doi: 10.1145/3462758.

Zhen Chen, Danny Kaufman, Mélina Skouras, and Etienne Vouga. Complex wrinkle field evolution. *ACM Trans. Graph.*, 42(4), jul 2023a. ISSN 0730-0301. doi: 10.1145/3592397. URL `https://doi.org/10.1145/3592397`.

Zhen Chen, Zherong Pan, Kui Wu, Etienne Vouga, and Xifeng Gao. Robust low-poly meshing for general 3d models. *ACM Trans. Graph.*, 42(4), jul 2023b. ISSN 0730-0301. doi: 10.1145/3592396. URL `https://doi.org/10.1145/3592396`.

Zhiqin Chen and Hao Zhang. Neural marching cubes. *ACM Trans. Graph.*, 40(6), dec 2021. ISSN 0730-0301.

Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. Bsp-net: Generating compact meshes via binary space partitioning. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 42–51, Los Alamitos, CA, USA, jun 2020. IEEE Computer Society.

Zhiqin Chen, Andrea Tagliasacchi, Thomas Funkhouser, and Hao Zhang. Neural dual contouring. *ACM Trans. Graph.*, 41(4), jul 2022a. ISSN 0730-0301.

Zhiqin Chen, Kangxue Yin, and Sanja Fidler. Auv-net: Learning aligned uv maps for texture transfer and synthesis. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1455–1464, Los Alamitos, CA, USA, jun 2022b. IEEE Computer Society.

Evgeni V Chernyaev. Marching cubes 33: construction of topologically correct isosurfaces. *Tech. Rep. CERN CN/95-17*, 1995.

Kazem Cheshmi, Danny M. Kaufman, Shoaib Kamil, and Maryam Mehri Dehnavi. Nasoq: Numerically accurate sparsity-oriented qp solver. *ACM Trans. Graph.*, 39(4), July 2020. ISSN 0730-0301. doi: 10.1145/3386569.3392486. URL `https://doi.org/10.1145/3386569.3392486`.

Kwang-Jin Choi and Hyeong-Seok Ko. Stable but responsive cloth. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, page 604–611, New York, NY, USA, 2002. Association for Computing Machinery. ISBN 1581135211. doi: 10.1145/566570.566624. URL `https://doi.org/10.1145/566570.566624`.

Philippe G. Ciarlet. *Theory of Shells, Volume 3 (Mathematical Elasticity).* North Holland, 2000.

P. Cignoni, C. Rocchini, and R. Scopigno. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2):167–174, 1998.

Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. MeshLab: an Open-Source Mesh Processing Tool. In Vittorio Scarano, Rosario De Chiara, and Ugo Erra, editors, *Eurographics Italian Chapter Conference*. The Eurographics Association, 2008. ISBN 978-3-905673-68-5.

Marvelous Designer CLO Virtual Fashion Inc. Marvelous designer, best realistic cloth making program for 3d artists, 2020. URL `https://www.marvelousdesigner.com`.

David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. Variational shape approximation. *ACM Trans. Graph.*, 23(3):905–914, aug 2004. ISSN 0730-0301.

Keenan Crane, Mathieu Desbrun, and Peter Schröder. Trivial connections on discrete surfaces. *Computer Graphics Forum*, 29(5):1525–1533, 2010.

Bram Custers and Amir Vaxman. Subdivision directional fields. *ACM Trans. Graph.*, 39(2), feb 2020. ISSN 0730-0301. doi: 10.1145/3375659.

Lis Custodio, Tiago Etiene, Sinesio Pesco, and Claudio Silva. Practical considerations on marching cubes 33 topological correctness. *Computers and Graphics*, 37(7):840–850, 2013. ISSN 0097-8493.

Lawrence D. Cutler, Reid Gershbein, Xiaohuan Corina Wang, Cassidy Curtis, Erwan Maigret, Luca Prasso, and Peter Farson. An art-directed wrinkle system for cg character clothing. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '05, page 117–125, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595931988. doi: 10.1145/1073368.1073384.

B. R. de Araújo, Daniel S. Lopes, Pauline Jepp, Joaquim A. Jorge, and Brian Wyvill. A survey on implicit surface polygonization. *ACM Comput. Surv.*, 47 (4), may 2015. ISSN 0360-0300.

Fernando de Goes, Mathieu Desbrun, Mark Meyer, and Tony DeRose. Subdivision exterior calculus for geometry processing. *ACM Trans. Graph.*, 35(4), jul 2016a. ISSN 0730-0301. doi: 10.1145/2897824.2925880.

Fernando de Goes, Mathieu Desbrun, and Yiying Tong. Vector field processing on triangle meshes. In *ACM SIGGRAPH 2016 Courses*, SIGGRAPH '16, New York, NY, USA, 2016b. Association for Computing Machinery. ISBN 9781450342896. doi: 10.1145/2897826.2927303. URL `https://doi.org/10.1145/2897826.2927303`.

Olga Diamanti, Amir Vaxman, Daniele Panozzo, and Olga Sorkine-Hornung. Designing n-polyvector fields with complex polynomials. *Comput. Graph. Forum*, 33(5):1–11, aug 2014. ISSN 0167-7055.

Olga Diamanti, Amir Vaxman, Daniele Panozzo, and Olga Sorkine-Hornung. Integrable polyvector fields. *ACM Trans. Graph.*, 34(4), jul 2015. ISSN 0730-0301. doi: 10.1145/2766906. URL `https://doi.org/10.1145/2766906`.

Lorenzo Diazzi and Marco Attene. Convex polyhedral meshing for robust solid modeling. *ACM Trans. Graph.*, 40(6), dec 2021. ISSN 0730-0301.

Akio Doi and Akio Koide. An efficient method of triangulating equi-valued surfaces by using tetrahedral cells. *IEICE Transactions on Information and Systems*, 1991.

Matin J. Dürst. Letters: additional reference to marching cubes. *Computer Graphics*, 1988.

Zuenko Evgeny and Matthias Harders. Wrinkles, folds, creases, buckles: Small-scale surface deformations as periodic functions on 3d meshes. *IEEE Transactions on Visualization and Computer Graphics*, PP:1–1, 05 2019. doi: 10.1109/TVCG.2019.2914676.

D. Ezuz, B. Heeren, O. Azencot, M. Rumpf, and M. Ben-Chen. Elastic correspondence between triangle meshes. *Computer Graphics Forum*, 38(2):121–134, 2019. doi: https://doi.org/10.1111/cgf.13624. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13624`.

Hao Fang and Florent Lafarge. Connect-and-slice: An hybrid approach for reconstructing 3d objects. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13487–13495, 2020.

Hao Fang, Florent Lafarge, and Mathieu Desbrun. Planar shape detection at structural scales. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2965–2973, 2018.

Gerald Farin. Triangular bernstein-bézier patches. *Computer Aided Geometric Design*, 3(2):83–127, 1986. ISSN 0167-8396. doi: https://doi.org/10.1016/0167-8396(86)90016-6. URL `https://www.sciencedirect.com/science/article/pii/0167839686900166`.

Matthew Fisher, Peter Schröder, Mathieu Desbrun, and Hugues Hoppe. Design of tangent vector fields. 26(3), jul 2007. ISSN 0730-0301.

Xifeng Gao, Hanxiao Shen, and Daniele Panozzo. Feature preserving octree-based hexahedral meshing. *Computer Graphics Forum*, 38:135–149, 08 2019.

Xifeng Gao, Kui Wu, and Zherong Pan. Low-poly mesh generation for building models. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Proceedings*, SIGGRAPH22 Conference Proceeding, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393379.

Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, page 209–216, USA, 1997. ACM Press/Addison-Wesley Publishing Co. ISBN 0897918967.

Russell Gillette, Craig Peters, Nicholas Vining, Essex Edwards, and Alla Sheffer. Real-time dynamic wrinkling of coarse animated cloth. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '15, page 17–26, New York, NY, USA, 2015a. Association for Computing Machinery. ISBN 9781450334969. doi: 10.1145/2786784.2786789.

Russell Gillette, Craig Peters, Nicholas Vining, Essex Edwards, and Alla Sheffer. Real-time dynamic wrinkling of coarse animated cloth. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '15, page 17–26, New York, NY, USA, 2015b. Association for Computing Machinery. ISBN 9781450334969. doi: 10.1145/2786784.2786789. URL `https://doi.org/10.1145/2786784.2786789`.

Eitan Grinspun, Petr Krysl, and Peter Schröder. Charms: A simple framework for adaptive simulation. *ACM Trans. Graph.*, 21(3):281–290, July 2002. ISSN 0730-0301. doi: 10.1145/566654.566578. URL `https://doi.org/10.1145/566654.566578`.

Eitan Grinspun, Anil N. Hirani, Mathieu Desbrun, and Peter Schröder. Discrete Shells. In D. Breen and M. Lin, editors, *Symposium on Computer Animation*. The Eurographics Association, 2003. ISBN 1-58113-659-5. doi: 10.2312/SCA03/062-067.

Peng Guan, Loretta Reiss, David Hirshberg, Alexander Weiss, and Michael Black. Drape: Dressing any person. *ACM Transactions on Graphics - TOG*, 31, 07 2012. doi: 10.1145/2185520.2185531.

Fabian Hahn, Bernhard Thomaszewski, Stelian Coros, Robert W. Sumner, Forrester Cole, Mark Meyer, Tony DeRose, and Markus Gross. Subspace clothing simulation using adaptive bases. *ACM Trans. Graph.*, 33(4):105:1–105:9, July 2014. ISSN 0730-0301. doi: 10.1145/2601097.2601160. URL `http://doi.acm.org/10.1145/2601097.2601160`.

Dong-Hoon Han, Chang-Jin Lee, Sangbin Lee, and Hyeong-Seok Ko. Tight Normal Cone Merging for Efficient Collision Detection of Thin Deformable Objects. In Holger Theisel and Michael Wimmer, editors, *Eurographics 2021 - Short Papers*. The Eurographics Association, 2021. ISBN 978-3-03868-133-5.

Jon Hasselgren, Jacob Munkberg, Jaakko Lehtinen, Miika Aittala, and Samuli Laine. Appearance-Driven Automatic 3D Model Simplification. In Adrien Bousseau and Morgan McGuire, editors, *Eurographics Symposium on Rendering - DL-only Track*, Prague, Czech Republic, 2021. The Eurographics Association. ISBN 978-3-03868-157-1.

Timothy J. Healey, Qingdu Li, and Ron-Bin Cheng. Wrinkling behavior of highly stretched rectangular elastic films via parametric global bifurcation. *Journal of Nonlinear Science*, 23:777–805, 2013. URL `https://doi.org/10.1007/s00332-013-9168-3`.

B. Heeren, M. Rumpf, M. Wardetzky, and B. Wirth. Time-discrete geodesics in the space of shells. *Computer Graphics Forum*, 31(5):1755–1764, 2012. doi: https://doi.org/10.1111/j.1467-8659.2012.03180.x. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2012.03180.x`.

B. Heeren, M. Rumpf, P. Schröder, M. Wardetzky, and B. Wirth. Exploring the geometry of the space of shells. *Computer Graphics Forum*, 33(5):247–256, 2014. doi: https://doi.org/10.1111/cgf.12450. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12450`.

Hugues Hoppe. Progressive meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, page 99–108, New York, NY, USA, 1996. Association for Computing Machinery. ISBN 0897917464.

Hugues Hoppe. New quadric metric for simplifiying meshes with appearance attributes. In *Proceedings of the Conference on Visualization '99: Celebrating Ten Years*, VIS '99, page 59–66, Washington, DC, USA, 1999. IEEE Computer Society Press. ISBN 078035897X.

Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, and Daniele Panozzo. Tetrahedral meshing in the wild. *ACM Trans. Graph.*, 37(4):60:1–60:14, July 2018. ISSN 0730-0301.

Yixin Hu, Teseo Schneider, Bolun Wang, Denis Zorin, and Daniele Panozzo. Fast tetrahedral meshing in the wild. *ACM Trans. Graph.*, 39(4), July 2020. ISSN 0730-0301.

Jin Huang, Tengfei Jiang, Zeyun Shi, Yiying Tong, Hujun Bao, and Mathieu Desbrun. $\ell 1$-based construction of polycube maps from complex shapes. *ACM Trans. Graph.*, 33(3), jun 2014. ISSN 0730-0301.

Jingwei Huang, Yichao Zhou, and Leonidas Guibas. Manifoldplus: A robust and scalable watertight manifold surface generation method for triangle soups. 2020.

Alec Jacobson, Daniele Panozzo, et al. libigl: A simple C++ geometry processing library, 2018. https://libigl.github.io/.

Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. Instant field-aligned meshes. *ACM Trans. Graph.*, 34(6), oct 2015. ISSN 0730-0301. doi: 10.1145/2816795.2818078.

Stefan Jeschke and Chris Wojtan. Water wave animation via wavefront parameter interpolation. *ACM Trans. Graph.*, 34(3), may 2015. ISSN 0730-0301. doi: 10.1145/2714572. URL `https://doi.org/10.1145/2714572`.

Stefan Jeschke and Chris Wojtan. Water wave packets. *ACM Trans. Graph.*, 36(4), July 2017. ISSN 0730-0301. doi: 10.1145/3072959.3073678. URL `https://doi.org/10.1145/3072959.3073678`.

Stefan Jeschke, Tomáš Skřivan, Matthias Müller-Fischer, Nuttapong Chentanez, Miles Macklin, and Chris Wojtan. Water surface wavelets. *ACM Trans. Graph.*, 37(4), jul 2018. ISSN 0730-0301. doi: 10.1145/3197517.3201336. URL `https://doi.org/10.1145/3197517.3201336`.

Ning Jin, Wenlong Lu, Zhenglin Geng, and Ronald P. Fedkiw. Inequality cloth. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '17, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450350914. doi: 10.1145/3099564.3099568. URL `https://doi.org/10.1145/3099564.3099568`.

Tao Ju and Tushar Udeshi. Intersection-free contouring on an octree grid. *Pacific Graphics Poster*, 01 2006.

Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual contouring of hermite data. *ACM Trans. Graph.*, 21(3):339–346, jul 2002. ISSN 0730-0301.

Felix Kälberer, Matthias Nieser, and Konrad Polthier. Quadcover - surface parameterization using branched coverings. *Comput. Graph. Forum*, 26:375–384, 09 2007. doi: 10.1111/j.1467-8659.2007.01060.x.

Jonathan M. Kaldor, Doug L. James, and Steve Marschner. Simulating knitted cloth at the yarn level. In *ACM SIGGRAPH 2008 Papers*, SIGGRAPH '08, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781450301121. doi: 10.1145/1399504.1360664. URL `https://doi.org/10.1145/1399504.1360664`.

Tero Karras. Maximizing parallelism in the construction of bvhs, octrees, and k-d trees. In *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics*, EGGH-HPG'12, page 33–37, Goslar, DEU, 2012. Eurographics Association. ISBN 9783905674415.

Ladislav Kavan, Dan Gerszewski, Adam Bargteil, and Peter-Pike Sloan. Physics-inspired upsampling for cloth simulation in games. *ACM Trans. Graph.*, 30: 93, 07 2011. doi: 10.1145/2010324.1964988.

Tom Kelly, John Femiani, Peter Wonka, and Niloy J. Mitra. Bigsur: Large-scale structured urban reconstruction. *ACM Trans. Graph.*, 36(6), nov 2017. ISSN 0730-0301.

Dawar Khan, Alexander Plopski, Yuichiro Fujimoto, Masayuki Kanbara, Gul Jabeen, Yongjie Jessica Zhang, Xiaopeng Zhang, and Hirokazu Kato. Surface remeshing: A systematic literature review of methods and research directions. *IEEE Transactions on Visualization and Computer Graphics*, 28(3):1680–1713, 2022.

Doyub Kim, Woojong Koh, Rahul Narain, Kayvon Fatahalian, Adrien Treuille, and James F. O'Brien. Near-exhaustive precomputation of secondary cloth effects. *ACM Trans. Graph.*, 32(4), July 2013. ISSN 0730-0301. doi: 10.1145/2461912.2462020. URL `https://doi.org/10.1145/2461912.2462020`.

Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. Globally optimal direction fields. *ACM Trans. Graph.*, 32(4), jul 2013. ISSN 0730-0301. doi: 10.1145/2461912.2462005.

Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. Stripe patterns on surfaces. *ACM Trans. Graph.*, 34(4), July 2015. ISSN 0730-0301. doi: 10.1145/2767000. URL `https://doi.org/10.1145/2767000`.

Leif P. Kobbelt, Mario Botsch, Ulrich Schwanecke, and Hans-Peter Seidel. Feature sensitive surface extraction from volume data. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, page 57–66, New York, NY, USA, 2001. Association for Computing Machinery. ISBN 158113374X.

Robert V. Kohn. Energy-driven pattern formation. *Proceedings of the International Congress of Mathematicians*, 2006. doi: 10.4171/022-1/15. URL `https://www.math.nyu.edu/~kohn/papers/kohn-icm.pdf`.

Zorah Lähner, Daniel Cremers, and Tony Tung. Deepwrinkles: Accurate and realistic clothing modeling. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 698–715, Cham, 2018a. Springer International Publishing. ISBN 978-3-030-01225-0. URL `https://doi.org/10.1007/978-3-030-01225-0_41`.

Zorah Lähner, Daniel Cremers, and Tony Tung. Deepwrinkles: Accurate and realistic clothing modeling. In *ECCV*, 2018b.

Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. Modular primitives for high-performance differentiable rendering. *ACM Trans. Graph.*, 39(6), nov 2020. ISSN 0730-0301.

Thibault Lescoat, Hsueh-Ti Derek Liu, Jean-Marc Thiery, Alec Jacobson, Tamy Boubekeur, and Maks Ovsjanikov. Spectral mesh simplification. *Computer Graphics Forum*, 39(2):315–324, 2020.

Thomas Lewiner, Hélio Lopes, Antônio Wilson Vieira, and Geovan Tavares. Efficient implementation of marching cubes' cases with topological guarantees. *Journal of Graphics Tools*, 8(2):1–15, 2003.

Jie Li, Gilles Daviet, Rahul Narain, Florence Bertails-Descoubes, Matthew Overby, George Brown, and Laurence Boissieux. An Implicit Frictional Contact Solver for Adaptive Cloth Simulation. *ACM Transactions on Graphics*, 37(4):1–15, August 2018. doi: 10.1145/3197517.3201308. URL `https://hal.inria.fr/hal-01834705`.

Minchen Li, Danny M. Kaufman, and Chenfanfu Jiang. Codimensional incremental potential contact. *ACM Trans. Graph.*, 40(4), jul 2021. ISSN 0730-0301.

Minglei Li and Liangliang Nan. Feature-preserving 3d mesh simplification for urban buildings. *ISPRS Journal of Photogrammetry and Remote Sensing*, 173: 135–150, 2021. ISSN 0924-2716.

Qingdu Li and Timothy J. Healey. Stability boundaries for wrinkling in highly stretched elastic sheets. *Journal of the Mechanics and Physics of Solids*, 97:260 – 274, 2016. ISSN 0022-5096. doi: https://doi.org/10.1016/j.jmps.2015.12.001. URL `http://www.sciencedirect.com/science/article/pii/S0022509615303185`. SI:Pierre Suquet Symposium.

Wan-chiu Li, Bruno Vallet, Nicolas Ray, and Bruno Levy. Representing higher-order singularities in vector fields on piecewise linear surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1315–1322, 2006.

Yiyi Liao, Simon Donné, and Andreas Geiger. Deep marching cubes: Learning explicit surface representations. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2916–2925, 2018.

Nils Lichtenberg, Noeska Smit, Christian Hansen, and Kai Lawonn. Real-time field aligned stripe patterns. *Computers and Graphics*, 74:137–149, aug 2018.

Peter Lindstrom and Greg Turk. Fast and memory efficient polygonal simplification. In *Proceedings of the Conference on Visualization '98*, VIS '98, page 279–286, Washington, DC, USA, 1998. IEEE Computer Society Press. ISBN 1581131062.

Peter Lindstrom and Greg Turk. Image-driven simplification. *ACM Trans. Graph.*, 19(3):204–241, jul 2000. ISSN 0730-0301.

Ruotian Ling, Jin Huang, Bert Jüttler, Feng Sun, Hujun Bao, and Wenping Wang. Spectral quadrangulation with feature curve alignment and element size control. *ACM Trans. Graph.*, 34(1), dec 2015. ISSN 0730-0301. doi: 10.1145/2653476. URL `https://doi.org/10.1145/2653476`.

Songrun Liu, Zachary Ferguson, Alec Jacobson, and Yotam Gingold. Seamless: Seam erasure and seam-aware decoupling of shape from mesh resolution. *ACM Trans. Graph.*, 36(6), nov 2017. ISSN 0730-0301.

Gurobi Optimization LLC. Gurobi optimizer reference manual, 2020. URL `http://www.gurobi.com`.

Charles T. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah, 1987.

Adriano Lopes and Ken Brodlie. Improving the robustness and accuracy of the marching cubes algorithm for isosurfacing. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):16–29, 2003.

William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, page 163–169, New York, NY, USA, 1987. Association for Computing Machinery. ISBN 0897912276.

Fujun Luan, Shuang Zhao, Kavita Bala, and Zhao Dong. Unified shape and svbrdf recovery using differentiable monte carlo rendering. *Computer Graphics Forum*, 40(4):101–113, 2021.

Eric Harold Mansfield. *The Bending and Stretching of Plates*. Cambridge University Press, 2 edition, 1989. doi: 10.1017/CBO9780511525193.

Josiah Manson and Scott Schaefer. Isosurfaces over simplicial partitions of multiresolution grids. *Computer Graphics Forum*, 29(2):377–385, 2010.

Jan Möbius Mario Botsch. Isoex, 2015. URL `https://www.graphics.rwth-aachen.de/IsoEx/index.html`.

Sergey V. Matveyev. Approximation of isosurface in the marching cube: Ambiguity problem. In *Proceedings of the Conference on Visualization '94*, VIS '94, page 288–292, Washington, DC, USA, 1994. IEEE Computer Society Press. ISBN 0780325214.

Ravish Mehra, Qingnan Zhou, Jeremy Long, Alla Sheffer, Amy Gooch, and Niloy J. Mitra. Abstraction of man-made shapes. *ACM Trans. Graph.*, 28(5): 1–10, dec 2009. ISSN 0730-0301.

Juan Montes, Bernhard Thomaszewski, Sudhir Mudur, and Tiberiu Popa. Computational design of skintight clothing. *ACM Trans. Graph.*, 39(4), July 2020. ISSN 0730-0301. doi: 10.1145/3386569.3392477. URL `https://doi.org/10.1145/3386569.3392477`.

Jorge J. Moré and David J. Thuente. Line search algorithms with guaranteed sufficient decrease. *ACM Trans. Math. Softw.*, 20(3):286–307, September 1994. ISSN 0098-3500. doi: 10.1145/192115.192132. URL `https://doi.org/10.1145/192115.192132`.

Matthias Müller and Nuttapong Chentanez. Wrinkle meshes. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, page 85–92, Goslar, DEU, 2010. Eurographics Association. doi: 10.2312/SCA/SCA10/085-091.

Jacob Munkberg, Wenzheng Chen, Jon Hasselgren, Alex Evans, Tianchang Shen, Thomas Muller, Jun Gao, and Sanja Fidler. Extracting triangular 3d models, materials, and lighting from images. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8270–8280, Los Alamitos, CA, USA, jun 2022. IEEE Computer Society.

Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *Journal of Visual Communication and Image Representation*, 18(2):109 – 118, 2007. ISSN 1047-3203. doi: https://doi.org/10.1016/j.jvcir.2007.01.005. URL `http://www.sciencedirect.com/science/article/pii/S1047320307000065`.

Liangliang Nan and Peter Wonka. Polyfit: Polygonal surface reconstruction from point clouds. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2372–2380, 2017.

Rahul Narain, Armin Samii, and James F. O'Brien. Adaptive anisotropic remeshing for cloth simulation. *ACM Trans. Graph.*, 31(6), nov 2012a. ISSN 0730-0301. doi: 10.1145/2366145.2366171. URL `https://doi.org/10.1145/2366145.2366171`.

Rahul Narain, Armin Samii, and James F. O'Brien. Adaptive anisotropic remeshing for cloth simulation. *ACM Trans. Graph.*, 31(6), November 2012b. ISSN 0730-0301. doi: 10.1145/2366145.2366171. URL `https://doi.org/10.1145/2366145.2366171`.

Rahul Narain, Tobias Pfaff, and James F. O'Brien. Folding and crumpling adaptive sheets. *ACM Trans. Graph.*, 32(4), July 2013. ISSN 0730-0301. doi: 10.1145/2461912.2462010. URL `https://doi.org/10.1145/2461912.2462010`.

Baptiste Nicolet, Alec Jacobson, and Wenzel Jakob. Large steps in inverse rendering of geometry. *ACM Trans. Graph.*, 40(6), dec 2021. ISSN 0730-0301.

Gregory M. Nielson. On marching cubes. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):283–297, 2003.

Gregory M. Nielson. Dual marching cubes. In *Proceedings of the Conference on Visualization '04*, VIS '04, page 489–496, USA, 2004. IEEE Computer Society. ISBN 0780387880.

Gregory M. Nielson and Bernd Hamann. The asymptotic decider: resolving the ambiguity in marching cubes. In *Proceeding Visualization '91*, pages 83–91, 1991.

Yuta Noma, Nobuyuki Umetani, and Yoshihiro Kawahara. Fast editing of singularities in field-aligned stripe patterns. In *SIGGRAPH Asia 2022 Conference Papers*, SA '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450394703.

Igor Pak and Jean-Marc Schlenker. Profiles of inflated surfaces. *Journal of Nonlinear Mathematical Physics*, 17(02):145–157, 2010. doi: 10.1142/S140292511000057X.

Daniele Panozzo, E. Puppo, and Luigi Rocca. Efficient multi-scale curvature and crease estimation. *2nd International Workshop on Computer Graphics, Computer Vision and Mathematics, GraVisMa 2010 - Workshop Proceedings*, pages 9–16, 01 2010.

Daniele Panozzo, Enrico Puppo, Marco Tarini, and Olga Sorkine-Hornung. Frame fields: Anisotropic and non-orthogonal cross fields. *ACM Trans. Graph.*, 33(4), jul 2014. ISSN 0730-0301. doi: 10.1145/2601097.2601179. URL `https://doi.org/10.1145/2601097.2601179`.

Joseph D. Paulsen, Evan Hohlfeld, Hunter King, Jiangshui Huang, Zhanlong Qiu, Thomas P. Russell, Narayanan Menon, Dominic Vella, and Benny Davidovitch. Curvature-induced stiffness and the spatial variation of wavelength in wrinkled sheets. *Proceedings of the National Academy of Sciences*, 113(5): 1144–1149, 2016. ISSN 0027-8424. doi: 10.1073/pnas.1521520113. URL `https://www.pnas.org/content/113/5/1144`.

William H. Paulsen. What is the shape of a mylar balloon? *The American Mathematical Monthly*, 101(10):953–958, 1994. ISSN 00029890, 19300972. doi: 10.2307/2975161.

Allen C. Pipkin. The Relaxed Energy Density for Isotropic Elastic Membranes. *IMA Journal of Applied Mathematics*, 36(1):85–99, 01 1986. ISSN 0272-4960. doi: 10.1093/imamat/36.1.85. URL `https://doi.org/10.1093/imamat/36.1.85`.

Allen C. Pipkin. Relaxed energy densities for large deformations of membranes. *IMA Journal of Applied Mathematics*, 52(3):297–308, 09 1994. ISSN 0272-4960.

doi: 10.1093/imamat/52.3.297. URL https://doi.org/10.1093/imamat/52.3.297.

Cédric Portaneri, Mael Rouxel-Labbé, Michael Hemmer, David Cohen-Steiner, and Pierre Alliez. Alpha Wrapping with an Offset. *ACM Transactions on Graphics*, 41(4):1–22, June 2022. URL https://hal.inria.fr/hal-03688637.

Nicolas Ray, Wan Chiu Li, Bruno Lévy, Alla Sheffer, and Pierre Alliez. Periodic global parameterization. *ACM Trans. Graph.*, 25(4):1460–1485, oct 2006. ISSN 0730-0301. doi: 10.1145/1183287.1183297.

Nicolas Ray, Bruno Vallet, Wan Chiu Li, and Bruno Lévy. N-symmetry direction field design. 27(2), may 2008. ISSN 0730-0301.

Nicolas Ray, Bruno Vallet, Laurent Alonso, and Bruno Levy. Geometry-aware direction field processing. 29(1), dec 2009. ISSN 0730-0301.

Olivier Rémillard and Paul G. Kry. Embedded thin shells for wrinkle simulation. *ACM Trans. Graph.*, 32(4), jul 2013a. ISSN 0730-0301. doi: 10.1145/2461912.2462018.

Olivier Rémillard and Paul G. Kry. Embedded thin shells for wrinkle simulation. *ACM Trans. Graph.*, 32, July 2013b. doi: http://doi.acm.org/10.1145/2461912.2462018. URL http://doi.acm.org/10.1145/2461912.2462018.

Damien Rohmer, Tiberiu Popa, Marie-Paule Cani, Stefanie Hahmann, and Alla Sheffer. Animation wrinkling: Augmenting coarse cloth simulations with realistic-looking wrinkles. *ACM Transaction on Graphics*, 29, 12 2010. doi: 10.1145/1866158.1866183.

Martin Rumpf and Benedikt Wirth. Variational time discretization of geodesic calculus. *IMA Journal of Numerical Analysis*, 35(3):1011–1046, 05 2014. ISSN 0272-4979. doi: 10.1093/imanum/dru027. URL https://doi.org/10.1093/imanum/dru027.

Leonardo Sacht, Etienne Vouga, and Alec Jacobson. Nested cages. *ACM Trans. Graph.*, 34(6), nov 2015. ISSN 0730-0301.

D. Salinas, F. Lafarge, and P. Alliez. Structure-aware mesh decimation. *Computer Graphics Forum*, 34(6):211–227, 2015.

Igor Santesteban, Miguel Otaduy, and Dan Casas. Learning-based animation of clothing for virtual try-on. *Computer Graphics Forum*, 38:355–366, 05 2019a. doi: 10.1111/cgf.13643.

Igor Santesteban, Miguel A. Otaduy, and Dan Casas. Learning-Based Animation of Clothing for Virtual Try-On. *Computer Graphics Forum*, 38(2): 355–366, 2019b. doi: 10.1111/cgf.13643.

Josua Sassen, Klaus Hildebrandt, and Martin Rumpf. Nonlinear deformation synthesis via sparse principal geodesic analysis. *Computer Graphics Forum*, 39(5):119–132, 2020. doi: https://doi.org/10.1111/cgf.14073. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14073`.

Syuhei Sato, Yoshinori Dobashi, and Tomoyuki Nishita. Editing fluid animation using flow interpolation. *ACM Trans. Graph.*, 37(5), sep 2018. ISSN 0730-0301. doi: 10.1145/3213771. URL `https://doi.org/10.1145/3213771`.

Scott Schaefer and Joe Warren. Dual marching cubes: Primal contouring of dual grids. pages 70– 76, 11 2004. ISBN 0-7695-2234-3.

Scott Schaefer, Tao Ju, and Joe Warren. Manifold dual contouring. *IEEE Transactions on Visualization and Computer Graphics*, 13(03):610–619, may 2007. ISSN 1941-0506.

Martin Seiler, Jonas Spillmann, and Matthias Harders. Enriching Coarse Interactive Elastic Objects with High-Resolution Data-Driven Deformations. In Jehee Lee and Paul Kry, editors, *Eurographics/ ACM SIGGRAPH Symposium* *on Computer Animation*. The Eurographics Association, 2012. ISBN 978-3-905674-37-8. doi: 10.2312/SCA/SCA12/009-017.

Andrew Selle, Jonathan Su, Geoffrey Irving, and Ronald Fedkiw. Robust high-resolution cloth using parallelism, history-based collisions, and accurate friction. *IEEE Transactions on Visualization and Computer Graphics*, 15:339–350, 2009. doi: 10.1109/TVCG.2008.79.

Nicholas Sharp, Yousuf Soliman, and Keenan Crane. The vector heat method. *ACM Trans. Graph.*, 38(3), jun 2019. ISSN 0730-0301. doi: 10.1145/3243651.

Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Jonathan Richard Shewchuk. Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In Ming C. Lin and Dinesh Manocha, editors, *Applied Computational Geometry Towards Geometric Engineering*, pages 203–222, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. ISBN 978-3-540-70680-9. doi: 10.1007/BFb0014497.

Melina Skouras, Bernhard Thomaszewski, Peter Kaufmann, Akash Garg, Bernd Bickel, Eitan Grinspun, and Markus Gross. Designing inflatable structures. *ACM Transactions on Graphics*, 33:1–10, 07 2014. doi: 10.1145/2601097.2601166.

Justin Solomon and Amir Vaxman. Optimal transport-based polar interpolation of directional fields. *ACM Trans. Graph.*, 38(4), jul 2019. ISSN 0730-0301. doi: 10.1145/3306346.3323005.

Georg Sperl, Rahul Narain, and Chris Wojtan. Homogenized yarn-level cloth. *ACM Trans. Graph.*, 39(4), July 2020. ISSN 0730-0301. doi: 10.1145/3386569.3392412. URL https://doi.org/10.1145/3386569.3392412.

David Steigmann. Tension-field theory. *Royal Society of London Proceedings Series A*, 429:141–173, 05 1990. doi: 10.1098/rspa.1990.0055.

Rasmus Tamstorf, Toby Jones, and Stephen F. McCormick. Smoothed aggregation multigrid for cloth simulation. *ACM Trans. Graph.*, 34(6), October 2015. ISSN 0730-0301. doi: 10.1145/2816795.2818081. URL https://doi.org/10.1145/2816795.2818081.

Greg Turk. Generating textures on arbitrary surfaces using reaction-diffusion. *SIGGRAPH Comput. Graph.*, 25(4):289–298, jul 1991. ISSN 0097-8930. doi: 10.1145/127719.122749.

Wim M. van Rees, Etienne Vouga, and L. Mahadevan. Growth patterns for shape-shifting elastic bilayers. *Proceedings of the National Academy of Sciences*, 114(44):11597–11602, 2017. ISSN 0027-8424. doi: 10.1073/pnas.1709025114. URL https://www.pnas.org/content/114/44/11597.

Hugues Vandeparre, Miguel Piñeirua, Fabian Brau, Benoit Roman, José Bico, Cyprien Gay, Wenzhong Bao, Chun Ning Lau, Pedro M. Reis, and Pascal Damman. Wrinkling hierarchy in constrained thin sheets from suspended graphene to curtains. *Physical Review Letters*, 106(22), Jun 2011. ISSN 1079-7114. doi: 10.1103/physrevlett.106.224301. URL http://dx.doi.org/10.1103/PhysRevLett.106.224301.

Amir Vaxman, Marcel Campen, Olga Diamanti, Daniele Panozzo, David Bommes, Klaus Hildebrandt, and Mirela Ben-Chen. Directional Field Synthesis, Design, and Processing. *Computer Graphics Forum*, 2016. ISSN 1467-8659. doi: 10.1111/cgf.12864.

David Vega, Javier Abache, and David Coll. A fast and memory saving Marching Cubes 33 implementation with the correct interior test. *Journal of Computer Graphics Techniques (JCGT)*, 8(3):1–18, August 2019. ISSN 2331-7418.

Roman Vetter, Norbert Stoop, Falk K Wittel, and Hans J Herrmann. Simulating thin sheets: Buckling, wrinkling, folding and growth. *Journal of Physics: Conference Series*, 487:012012, mar 2014. doi: 10.1088/1742-6596/487/1/012012. URL `https://doi.org/10.1088/1742-6596/487/1/012012`.

Ryan Viertel and Braxton Osting. An approach to quad meshing based on harmonic cross-valued maps and the ginzburg–landau theory. *SIAM Journal on Scientific Computing*, 41(1):A452–A479, 2019.

P. Volino and N.M. Thalmann. The spherigon: a simple polygon patch for smoothing quickly your polygonal meshes. In *Proceedings Computer Animation '98 (Cat. No.98EX169)*, pages 72–78, 1998. doi: 10.1109/CA.1998.681910.

Pascal Volino and Nadia Magnenat Thalmann. Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. *Computer Graphics Forum*, 1994. ISSN 1467-8659.

Christoph Von-Tycowicz, Christian Schulz, Hans-Peter Seidel, and Klaus Hildebrandt. Real-time nonlinear shape interpolation. *ACM Trans. Graph.*, 34(3), may 2015. ISSN 0730-0301. doi: 10.1145/2729972. URL `https://doi.org/10.1145/2729972`.

Huamin Wang. Gpu-based simulation of cloth wrinkles at submillimeter levels. *ACM Trans. Graph.*, 40(4), jul 2021. ISSN 0730-0301. doi: 10.1145/3450626.3459787.

Huamin Wang, Florian Hecht, Ravi Ramamoorthi, and James F. O'Brien. Example-based wrinkle synthesis for clothing animation. *ACM Trans. Graph.*, 29(4), July 2010. ISSN 0730-0301. doi: 10.1145/1778765.1778844. URL https://doi.org/10.1145/1778765.1778844.

Ke Wang, Weiwei, Yiying Tong, Mathieu Desbrun, and Peter Schröder. Edge subdivision schemes and the construction of smooth vector fields. *ACM Trans. Graph.*, 25(3):1041–1048, jul 2006. ISSN 0730-0301. doi: 10.1145/1141911.1141991.

Ting Wang, Fu Chenbo, Fan Xu, Yongzhong Huo, and Michel Potier-Ferry. On the wrinkling and restabilization of highly stretched sheets. *International Journal of Engineering Science*, 136:1–16, 12 2018. doi: 10.1016/j.ijengsci.2018.12.002.

Tongtong Wang, Zhihua Liu, Min Tang, Ruofeng Tong, and Dinesh Manocha. Efficient and reliable self-collision culling using unprojected normal cones. *Computer Graphics Forum*, 36(8):487–498, 2017.

Yu Wang, Alec Jacobson, Jernej Barbič, and Ladislav Kavan. Linear subspace design for real-time shape deformation. *ACM Trans. Graph.*, 34(4), July 2015. ISSN 0730-0301. doi: 10.1145/2766952. URL https://doi.org/10.1145/2766952.

Xinyue Wei, Minghua Liu, Zhan Ling, and Hao Su. Approximate convex decomposition for 3d meshes with collision-aware concavity and tree search. *ACM Trans. Graph.*, 41(4), jul 2022. ISSN 0730-0301.

Clarisse Weischedel. *A discrete geometric view on shear-deformable shell models*. PhD dissertation, Georg-August-Universität Göttingen, 2012.

Andrew Witkin and Michael Kass. Reaction-diffusion textures. *SIGGRAPH Comput. Graph.*, 25(4):299–308, jul 1991. ISSN 0097-8930. doi: 10.1145/127719.122750.

Wesley Wong and Sergio Pellegrino. Wrinkled membranes part i: Experiments. *Journal of Mechanics of Materials and Structures - J MECH MATER STRUCT*, 1:3–25, 05 2006. doi: 10.2140/jomms.2006.1.3.

Kui Wu, Xu He, Zherong Pan, and Xifeng Gao. Occluder generation for buildings in digital games. *Computer Graphics Forum*, 41(7):205–214, 2022.

Geoff Wyvill, Craig McPheeters, and Brian Wyvill. Data structure for soft objects. *The Visual Computer - VC*, 2:227–234, 08 1986.

Kaizhi Yang and Xuejin Chen. Unsupervised learning for cuboid shape abstraction via joint segmentation from point clouds. *ACM Trans. Graph.*, 40 (4), jul 2021. ISSN 0730-0301.

Mulin Yu and Florent Lafarge. Finding good configurations of planar primitives in unorganized point clouds. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6357–6366, Los Alamitos, CA, USA, jun 2022. IEEE Computer Society.

J. Zavala-Hidalgo, M.A. Bourassa, S.L. Morey, J.J. O'Brien, and P. Yu. A new temporal interpolation method for high-frequency vector wind fields. In *Oceans 2003.*, volume 2, pages 1050–1053 Vol.2, 2003. doi: 10.1109/OCEANS.2003.178485.

Eugene Zhang, Konstantin Mischaikow, and Greg Turk. Vector field design on surfaces. 25(4):1294–1326, oct 2006. ISSN 0730-0301.

Linbo Zhang, Tao Cui, and Hui Liu. A set of symmetric quadrature rules on triangles and tetrahedra. *Journal of Computational Mathematics*, 27:89–96, 01 2009.

Meng Zhang, Tuanfeng Wang, Duygu Ceylan, and Niloy J Mitra. Deep detail enhancement for any garment. In *Computer Graphics Forum*, volume 40, pages 399–411. Wiley Online Library, 2021. doi: 10.1111/cgf.142642.

Muyang Zhang, Jin Huang, Xinguo Liu, and Hujun Bao. A wave-based anisotropic quadrangulation method. *ACM Trans. Graph.*, 29(4), jul 2010. ISSN 0730-0301. doi: 10.1145/1778765.1778855. URL `https://doi.org/10.1145/1778765.1778855`.

Qingnan Zhou and Alec Jacobson. Thingi10k: A dataset of 10,000 3d-printing models. 2016.

Vita

Zhen Chen was born in Hunan, China. He obtained his Bachelor of Science degree in Information and Computing Science from the School of Mathematics, University of Science and Technology of China. Subsequently, he commenced his graduate studies at the University of Texas at Austin in August 2018, under the mentorship of Dr. Etienne Vouga. Throughout his graduate career, Zhen dedicated himself to research in the fields of physical simulations and geometry processing, with a particular emphasis on cloth wrinkle simulation and surface mesh repair. He harbors a profound interest in exploring mathematical concepts related to geometry and engaging in practical research that contributes to product development.

Address: zhenjaychen@gmail.com

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.