Discrete differential geometry driven methods for architectural geometry

Yao, Sidan

2021

Yao, S. (2021). Discrete differential geometry driven methods for architectural geometry.
Doctoral thesis, Nanyang Technological University, Singapore.
<https://hdl.handle.net/10356/155036>

<https://hdl.handle.net/10356/155036>

<https://doi.org/10.32657/10356/155036>

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0
International License (CC BY-NC 4.0).

Downloaded on 26 Apr 2023 18:40:56 SGT

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

**DISCRETE DIFFERENTIAL GEOMETRY DRIVEN
METHODS FOR ARCHITECTURAL GEOMETRY**

**YAO SIDAN
SCHOOL OF COMPUTER SCIENCE AND
ENGINEERING
2021**

DISCRETE DIFFERENTIAL GEOMETRY DRIVEN METHODS FOR ARCHITECTURAL GEOMETRY

YAO SIDAN

School of Computer Science and Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Doctor of Philosophy

2021

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

08 November 2021

.....

Date

NTU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU

.....

Yao Sidan

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

08 November 2021

.....

Date

NTU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU

He Ying

Authorship Attribution Statement

This thesis contains material from 2 papers published in the following peer-reviewed journal(s) / from papers accepted at conferences in which I am listed as an author.

Please amend the typical statements below to suit your circumstances if (B) is selected.

Chapter 4 is published as Sidan Yao, Xiao Ling, Fiona Nueesch, Gerhard Schrotter, Simon Schubiger, Zheng Fang, Long Ma, and Zhen Tian. Maintaining Semantic Information across Generic 3D Model Editing Operations. Remote Sensing 12, no. 2 (2020): 335.

The contributions of the co-authors are as follows:

- Prof. Simon Schubiger provided the initial project direction.
- Prof. Simon Schubiger, Dr. Xiao Ling and Dr. Gerhard Schrotter contributed to discuss the construction of the technical framework.
- Ms. Fiona Nueesch offered user interface support as a user experience expert in the experiments.
- I designed the algorithm pipeline with assistance from Prof. Simon Schubiger and Dr. Xiao Ling.
- I wrote the C++ and JAVA codes to implement the framework.
- I prepared the manuscript drafts. Dr. Long Ma and Dr. Zheng Fang provided revision and proof reading. Mr. Zhen Tian provided partial revision.

Chapter 5 is published as Long Ma*, Sidan Yao*, Jianmin Zheng, Yang Liu, Yuanfeng Zhou, Shi-Qing Xin, Ying He. Constructing Self-Supporting

Surfaces with Planar Quadrilateral Elements. Computational Visual Media, 2021.

The contributions of the co-authors are as follows:

- Dr Ma provided the initial project direction and edited the manuscript drafts.
- I prepared the manuscript drafts. The manuscript was revised by A/Prof He and A/Prof Zheng.
- I implemented the algorithm. Dr Ma assisted in the implementation.
- I co-designed the experiments with Dr Ma and A/Prof He and performed all the experimental work at the school of Computer Science and Engineering. I also analyzed the data.
- Dr Liu assisted in the experiments design. I did the coding for experiments.
- A/Prof Zhou and A/Prof Xin assisted in amendment of paperwork.

08 November 2021

.....

Date

NTU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU

.....

Yao Sidan

Abstract

With the rapid growth of free form architectures, the demand for architectural geometry technologies increases dramatically in recent years. Architectural geometry contains knowledge highly relevant to computer graphics and geometry especially discrete differential geometry. In addition, architectural geometry provides new context for some well-established concepts and brings new challenges and new objectives. In this dissertation, we tackle four challenges in architectural geometry. These four proposed solutions cover various processes including architecture maintenance, architecture construction, architecture texture mapping and architecture decoration.

In Chapter 4, we present a data model synchronization method that preserves semantic information across editing operations relying only on geometry, UV mappings, and materials. This enables easy integration of existing and future 3D editing techniques with rich data models. The method links the original data model to the edited geometry using point set registration, recovering the existing information based on spatial and UV search methods, and automatically labels the newly created geometry. The implementation synchronized changes in the 3D geometry with a CityGML data model.

In Chapter 5, we present a simple yet effective method for constructing 3D self-supporting surfaces with planar quadrilateral (PQ) elements. Starting with a self-supporting surface in triangle mesh, we first compute the princi-

pal curvatures and directions of each triangular face using a new approach, yielding more accurate results than existing methods. Then, we smooth the principal direction field to reduce the number of singularities and partition all faces into two groups in terms of principal curvature difference. For each face with small curvature difference, we compute a stretch matrix that turns the principal directions into a pair of conjugate directions. Finally, applying a mixed-integer programming solver to the mixed principal and conjugate direction field, we obtain a planar quadrilateral mesh. Experimental results show that our method is computationally efficient and can yield high-quality PQ meshes that well approximate the geometry of the input surfaces and maintain their self-supporting properties.

In Chapter 6, we present a simple and robust algorithm to compute quad layout. We first propose an interpolation strategy to find tracing directions for singularities: given a singularity with index k (which is a multiple of $\frac{1}{4}$), there are exactly $4 - 4k$ searching directions produced. We then trace integral curves with a rounding strategy to encourage them to go through mesh vertices, which can effectively reduce the number of short segments. Finally, we partition the triangle mesh along the integral curves to extract the quadrilateral patches. Our method does not require any numerical solver and is easy to implement and computationally efficient. Computational results show that our results have consistently fewer quad patches than those of the existing methods. For models with rich geometric details, our method can save up to 50% patches in the quad layout.

In Chapter 7, we present research in contrast-enhanced high-relief modeling. We present a simple and effective method to generate contrast-enhanced high-reliefs. Our key idea is a depth compression function with only two variables for 3D models. In this function, normals and mean curvatures are

utilized to enhance the contrast of the resulting high-reliefs. To calculate the two variables in depth compression function, we construct an optimization framework with two objectives, volume minimization and contrast maximization. The variables are obtained when the two terms reach a balance point. Our method narrows down the solution space to only two variables and is easy to implement and produces good-quality high-reliefs. We show results on a range of real 3D models including real-world and synthetic models.

In summary, we present four algorithms in architectural geometry. These solutions cover many applications including semantic information update, free form surface construction, surface parameterization and architecture surface decoration. We demonstrate the effectiveness of the proposed methods through extensive evaluation and comparison with the state-of-the-art methods.

Acknowledgements

Above all, I would offer my genuine gratitude to my respectable supervisor Prof. He Ying for his continuous and patient support of my Ph.D. study. It's my lifelong pleasure to be his student. Prof. He taught me the method to complete research and to present the results as clear as possible. It has been an unforgettable time working with Prof. He, for his precise research altitude, clear instructions, personal fascination and constant encouragement that I have learned. I would also like to thank Dr. Ma Long for his guidance in differential geometry. Thanks to Prof. Simon Schubiger for his advice on architectural geometry and software development and thanks to Dr. Ling Xiao for his advice in research. Specially, thanks to Prof. Ying Xiang for recommending me to Prof. He and encouraging me all the time.

Thanks to my labmates, Dr. Fang Zheng, Prof. Hou Fei, Prof. Sun Qian, Dr. Du Jie, Dr. Fu Qian and Prof. Lu Xuequan for guiding me. Meanwhile, thanks to my colleagues at Virtual Singapore, Ms. Fiona Nueesch, Prof. Zeng Wei, Mr. Zhang Chi, Prof. Stefan Müller Arisona, Prof. Armin Gruen, Dr. Gerhard Schrotter, Prof. Qin Rongjun, Dr. Xiao Changlin and Mr. Li Jiaqiang for their patient collaboration.

Thanks to my friends Ms. Zhou Xing, Ms. Li Shiyong and Mr. Tian Zhen for their accompany. I sincerely appreciate my parents for their support and care. Specially, thanks to my son Tiantian, for his unconditional love.

Publications

Long Ma*, **Sidan Yao***, Jianmin Zheng, Yang Liu, Yuanfeng Zhou, Shi-Qing Xin, Ying He. Constructing Self-Supporting Surfaces with Planar Quadrilateral Elements. Computational Visual Media, in press, 2021. (* joint first author)

Long Ma*, **Sidan Yao***, Ying He, Jianmin Zheng, Junhui Hou, Yuanfeng Zhou. A Simple and Robust Method for Computing Field-Coherent Quad Layout on Triangle Meshes. Submitted to Computer-Aided Design, 2021. (* joint first author)

Sidan Yao, Xiao Ling, Fiona Nueesch, Gerhard Schrotter, Simon Schubiger, Zheng Fang, Long Ma, and Zhen Tian. "Maintaining Semantic Information across Generic 3D Model Editing Operations." Remote Sensing 12, no. 2 (2020): 335. (Impact factor: 4.509).

Contents

Abstract	i
Acknowledgements	iv
Publications	v
List of Figures	xxi
List of Tables	xxii
List of Algorithms	xxiii
1 Introduction	1
1.1 Background	1
1.2 Research Objectives	6
1.3 Technical Contributions	9
1.4 Dissertation Organization	14
2 Related Work	16
2.1 Architectural Geometry	16
2.1.1 Semantic Information	16
2.1.2 Editing Techniques for Architectural Models	17
2.1.3 Information Synchronization	18

2.1.4	Self-Supporting Surfaces	19
2.1.5	Relief Modeling	20
2.2	Discrete Differential Geometry	20
2.2.1	Model Registration	20
2.2.2	Cross Fields	21
2.2.3	Principal Directions and Curvatures	22
2.2.4	Quadrilateral Layouts	23
2.2.5	Conjugate Direction Fields	23
3	Preliminaries	25
3.1	Discrete Normals and Mean Curvatures	25
3.2	Self-supporting Surfaces with Isotropic Stresses	27
3.3	Conformal Parametrization	28
3.4	Smooth Cross Fields	29
4	Maintaining Semantic Information across Architectural Model Editing	30
4.1	Problem Statement	30
4.2	Method Overview	31
4.2.1	Face Matching	32
4.2.2	Model Registration	34
4.2.3	Semantic Information Transfer	39
4.2.4	Labeling	41
4.3	Discussions	44
4.3.1	Face Deletion for Shelters	45
4.3.2	Template Application For Openings	47
4.3.3	Commercial Tools	47
4.4	Summary	52

5	Constructing Self-Supporting Surfaces with Planar Quadri-	
	lateral Elements	55
5.1	Problem Statement	55
5.2	Algorithm	57
5.2.1	Overview	57
5.2.2	Computing Principal Curvatures and Directions	57
5.2.3	Smoothing Principal Directions	60
5.2.4	Fixed Chart and Free Chart Distinguish	62
5.2.5	Stretch Matrix Construction	62
5.2.6	Parameterization & Generating PQ Meshes	67
5.2.7	Airy Stress Function	68
5.3	Finite Element Method Analysis for PQ Meshes	69
5.3.1	Accuracy Measures	70
5.3.2	Force Balance on Bending Thin Shells	71
5.3.3	Thin Shell Problem on Parametric Surface	74
5.3.4	Balance between Stress Tensor and Gravity	75
5.4	Experimental Results	76
5.4.1	Principal Curvatures and Directions	76
5.4.2	Evaluation	77
5.4.3	Comparison	78
5.4.4	Discussion	82
5.5	Summary	83
6	Field-Coherent Quad Layout Computation for Architectural	
	Parameterization	86
6.1	Problem Statement	86
6.2	Data Structures	88
6.2.1	Discrete Cross Fields	88

6.2.2	Discrete Integral Curves	89
6.3	Algorithm	91
6.3.1	Searching Ports	91
6.3.2	Rounding Strategy	93
6.3.3	Pairing and Smoothing	94
6.3.4	T-junctions	96
6.3.5	Facet Separation	96
6.4	Results and Comparison	98
6.4.1	Evaluation	98
6.4.2	Comparison	101
6.4.3	Limitation	103
6.5	Application	103
6.6	Summary	105
7	Contrast-Enhanced High-Relief Modeling for Architectural Decoration	108
7.1	Problem Statement	108
7.2	Algorithm	109
7.2.1	Method Overview	109
7.2.2	Depth Compression Function	110
7.2.3	Optimization Framework	111
7.2.4	Transplant to Curved Background Surfaces	114
7.3	Results	115
7.4	Summary	118
8	Conclusion and Future Work	119
8.1	Conclusion	119
8.2	Future Work	121

List of Figures

1.1	(a) The frame field of Botijo model. (b) The frame field of Elk model. (c) Four pairwise orthogonal unit length vectors in the tangent plane. Frame fields are visualized in line integral convolution (LIC), which is widely used to visualize vector field.	2
1.2	Traditional high-reliefs on traditional Chinese ancient architectures. Brick relief is a technique of sculpture in which subjects are left in relief on a brick-built surface or wall. Brick relief are widely used in Huizhou architecture.	6
1.3	(a) French 20th-century bas-relief. (b) Relatively modern high-relief in Bishopsgate, London [6]. Relief is a form of sculpture used for thousands of years. Bas-relief has been studied extensively in the graphics community. High-relief modeling is quite challenging since it deals with non-height fields. The full range of relief includes shallow-relief, bas-relief, mid-relief and high-relief, this dissertation focuses on bas-relief and high-relief.	6
1.4	Given a self-supporting surface in triangular mesh, our method converts it into a planar quadrilateral (PQ) mesh that approximates the input self-supporting surface faithfully both in geometrical performance and physical performance.	12
1.5	Quad layout and texture mapping on a complex human body.	13

1.6	Given a smooth cross field, our method computes a quad layout that 1) well aligns with the field directions; and 2) does not contain any new vertex other than T-junctions. Therefore, the computed layout preserves both the geometrical and topological features of the input cross field. For better visualization, we color the positive (resp. negative) singularities of the cross field in red (resp. blue).	13
1.7	(a) Front and side view of input model. (b) Front and side view of generated contrast-enhanced high-relief. Given a real 3D model, our method calculate the normals and mean curvatures of vertices, calculate the variables for depth compression function and produce a contrast-enhanced high-relief.	14
2.1	Self-supporting surfaces generated by Ma et al.'s algorithm [74].	20
3.1	Tangent vector and normal vector in curved surface.	26
3.2	Reducing area distortion in conformal parameterization. (a) The parameterization has no singularity but very large area distortion. One can see the sizes of the quad elements vary significantly. Also, its iso-parameter lines are not orthogonal to the boundary. (b) We improve the parameterization quality by adding 4 singularities (in red). Note the area distortion is reduced and the iso-parameter lines are aligned with the boundary.	28
4.1	Flow chart of automatic 3D model synchronization.	32

4.2	UV space of a building model. In UV space, each vertex v_i of the geometry has a corresponding texture coordinate t_i . The UV space simplifies matching faces, since texture coordinates remain unchanged or are re-sampled during deformation, push-pull, and template applications.	33
4.3	A building model with symmetries. Using a uniform sampling strategy will prevent ICP from recovering a rotation by 180° . .	35
4.4	Edges and diagonals of a face are sampled with a random number generator which is initialized by the texture coordinates of the vertices of the edge or diagonal.	36
4.5	Example of a generated source point set (red) and generated target point set (yellow) for the iterative closest point (ICP) algorithm.	37
4.6	A complete building model to derive the maximum number of iterations.	38
4.7	The source point set and target point set for the building model in Figure 4.6	38
4.8	Experimental results under different numbers of iterations. . .	39
4.9	Registration result. The white point set is the transformed source point set. It overlaps with the point set generated from the edited geometry M_e	39
4.10	A building model and its octree.	40
4.11	Deformation operations on vertex, edge, and face. Deformation will change f_{normal} , $f_{centroid}$ and f_{length} . A UV space search was performed for these faces.	41
4.12	Relevant parts of the CityGML data model for LoD3 modelling.	41

4.13	Opening generated by push-pull. This editing operation will not split the boundary surface but generate a hole on the boundary surface.	42
4.14	Openings generated by template application. This editing operation will split the boundary surface and generate several new faces co-planar with the boundary surface.	42
4.15	(a) f_ϕ : the angle between the face normal f_{normal} and the up vector of the coordinate system. (b)Openings detected by topology relationships.	43
4.16	Adding roof infrastructure for building models by push-pull. .	44
4.17	Face split operation.	45
4.18	user interface of VSE.	46
4.19	Deleting faces for shelters in the editor.	46
4.20	FZKViewer: Data model update after deletion. Roofs surfaces in red, wall surfaces in grey.	47
4.21	Façade template application.	48
4.22	FZKViewer: Data model update for template application. Roofs surfaces in red, wall surfaces in grey, windows in turquoise. .	48
4.23	A building model and its semantic information for comparisons among VSE, CityEditor, and CityGML Importer.	52

5.1	(a) Input, (b) Principal directions, (c) Smoothed directions (d) Free and fixed faces, (e) Mixed field, (f) PQ mesh. Algorithmic pipeline. The input is a 3D self-supporting surface represented by a triangle mesh. First, we compute principal curvatures and principal directions for each triangular face. Then, we group the faces with similar principal curvatures, which are called free faces (teal). We compute a smooth cross field on the free faces and generate a pair of conjugate directions by stretching the principal directions. Next, we parameterize the triangle mesh using the mixed principal direction field and conjugate direction field. Finally, we extract the planar quadrilateral mesh from the parameterization.	58
5.2	The difference between the observation point movement strategy used in Rusinkiewicz’s method [96] and our method. (a) In [96], the derivatives are $\delta \mathbf{r}_1 = \overrightarrow{CB}$, $\delta \mathbf{r}_2 = \overrightarrow{AC}$ and $\delta \mathbf{r}_3 = \overrightarrow{BA}$, which are based on mesh vertices. (b) In our method, we define $\delta \mathbf{r}_i = \overrightarrow{OO_i}$, where O and O_i are the centers of the circumscribed circles of $\triangle ABC$ and its neighbors.	58
5.3	Smoothing principal directions. (a) There are many singularities in the principal direction field. (b) The singularities are reduced significantly after smoothing. The red (resp. blue) spheres are singularities with index $\frac{1}{4}$ (resp. $\frac{-1}{4}$).	61
5.4	Distinguishing fixed and free faces. (a) We compute the difference λ between principal curvatures κ_1 and κ_2 on each triangular face, $\lambda = \frac{\min\{ \kappa_1 , \kappa_2 \}}{\max\{ \kappa_1 , \kappa_2 \}}$. (b) Faces with $\lambda \geq \varepsilon$ are marked as “fixed” (brown) and the rest are “free” (teal) faces.	63

5.5	Consider a circle (blue) with two perpendicular diameters. Stretching the circle with matrix $\mathbf{S}$ yields an ellipse and the two perpendicular diameters become conjugate.	66
5.6	Computing conjugate directions on free faces. (a) The smoothed principal directions. (b) The conjugate directions. .	66
5.7	Comparisons with Rusinkiewicz’s method [96] in terms of accuracy. We generate sequences of isotropic (first two rows) and anisotropic meshes (last two rows) with increasing number of faces. We compute the angle differences $\delta\theta$ between the resulting principal directions and the ground-truths and visualize the errors using heat color map, where the warm (resp. cold) colors are large (resp. small) angles. For each model, we show our results in the top and theirs in the bottom. τ is the anisotropy measure and $ F $ is the face count.	78
5.8	(a) Convergence plots of our method (orange) and Rusinkiewicz’s method [96] (gray). Both methods converge with the increasing number of resolutions. However, our method is more accurate and robust than theirs, especially on anisotropic meshes. (b) and (c) The smoothness measure is the average of the discrete geodesic curvatures of the parameter lines.	79
5.9	Non-height fields. Row 1: the triangle meshes generated by Ma et al.’s method [74]; Rows 2 and 3: the PQ meshes produced by our method and Liu et al.’s method [69], respectively.	82
5.10	The distribution of ϵ and θ_S produced by Liu et al.’s method [69] and our method.	82

5.11	Results and comparisons. For each model, the three rows show the triangle mesh produced by Ma et al.'s method [74] and the PQ meshes produced by our method and Liu et al.'s method [69], respectively. We visualize the quality of the results using 3 quantitative measures, i.e., normal displacement θ , stress tensor error σ_s and differential stress σ_D . The lower the measures, the higher the quality of the resulting PQ meshes. Images are rendered in high resolution, allowing close-up examination.	85
6.1	The reference direction and the first unit vector in a random triangle.	89
6.2	A random highlighted half-edge e , going from vertex a to vertex b , half-edges are directed, so order matters. The corresponding half-edge in the opposite direction is denoted as opposite e . e is part of face f (blue shaded area).	90
6.3	Technical challenges in tracing separatrices. In a continuous setting, each singularity with index $\frac{1}{4}$ (resp. $-\frac{1}{4}$) is the source of 3 (resp. 5) separatrices. However, due to discretization error, redundant and missing directions often occur. (a) One of the 4 pink directions is redundant, and should be discarded. (b) Only 4 directions are detected and the dashed direction is missing. (c)-(d) We propose an interpolation strategy so that for a singularity with index k there are exactly $4 - 4k$ ports.	92

6.4	Chart interpolation. (a) Given an edge $v_i v_j$, we define the edge chart θ_{ij} by averaging the two charts on the adjacent faces. See Equation (6.3). (b) The chart for a polar angle α is determined by linear interpolating the charts of the adjacent edges. See Equation(6.4).	92
6.5	A tracing line enters $\triangle v_0 v_1 v_2$ at P and leaves it at Q	93
6.6	Eliminating short segments. (a) A path near a vertex, yielding very short segments and small facets. (b) Making the path going through vertex fixes the problem.	94
6.7	Path linking. (a) Directly linking two paths often produces a non-smooth path. (b) Adjusting the linking points improves smoothness.	95
6.8	There are two non-parallel separatrices in face f . The tracing of the current separatrix Γ stops if the intersection point T is inside f , and continues otherwise.	95
6.9	Different topological types of facets split by tracing lines. In (a), (b) and (c) nodes on a same path are matched; in (d) and (e), nodes on edges are marked. Even though n_i are coincide with some v_j , extracting facets from marked or matched vertices is also robust.	98

6.10	Algorithmic pipeline. (a) The input cross field has two singularities of indices $\pm\frac{1}{4}$, respectively. (b) For a singularity with index k , we generate $4 - 4k$ ports (pink vectors). (c) We trace a separatrix for each port with a rounding strategy that encourages the separatrix to go through vertices. Each separatrix terminates at another singularity, a T-junction or a boundary. (d) Finally, we generate the layout by extracting the quadrilateral patches.	100
6.11	Close-up views of the Bunny model. Besides T-junctions, Pietroni et al.'s method [84] produces a large number valence-4 vertices (highlighted by red circles) in the quad layout. Our method, in contrast, does not require the valence-4 vertices. . .	103
6.12	The distribution of ε and δ . produced by Pietroni et al.'s method [84] and our method.	103
6.13	(a) Conformal parameterization of reconstructed mesh. (b) A quadrilateral patch denoted by $ABCD$	104
6.14	Results on scanned data. The models are taken from the 8i Voxelized Surface Light Field (8iVSLF) Dataset. For each model, we show the quad layout, 2D tessellation and normal map. From left to right, top to bottom: <i>Red and Black</i> (Frames 1677 and 1491), <i>Loot</i> , <i>Long Dress</i> (Frames 1063 and 1270), and <i>Soldier</i> . We slightly modified some shapes by removing thin, tube-shaped parts, which cause difficulty in computing cross fields.	106

6.15	Comparison with Pietroni et al.'s method [84]. Both methods are able to produce field-coherent quad layouts. However, our results have consistently fewer patches than theirs. In particular, for models with rich geometric details, our method can save up to 54% quad patches. Moreover, our method does not add any new vertex other than T-junctions in the computed layout, whereas Pietroni et al.'s method adds a large number of non-T-junction vertices to maintain the quad topology. The 2-tuple below each figure is the number of quad patches N_p and the number of non-T-junction vertices added in the quad layout N_n . Random colors are used to visualize the layout.	107
7.1	(a) A real 3D model before scaling. (b) A real 3D model compressed with scale ratio $s = 0.3$. (c) Enhance contrast for compressed model with $d = 0.3$.	110
7.2	Setting the parameter ω_1 to the longest edge of bounding box' reciprocal and parameter ω_2 to 1.	114
7.3	Transplant the Dragon relief to a curved background surface.	115
7.4	(a) The result generated by bas-relief modeling techniques. (b) Transplant a high-relief to a curved background surface.	115
7.5	Distribution of z -coordinate before and after depth compression. Distributions indicate that our method maintains the geometry details of the input real 3D models.	116
7.6	Contrast-enhanced high-reliefs generated by our method. The first and second columns are input 3D models and the third and forth columns are high-reliefs produced by our method.	116
7.7	Contrast-enhanced high-reliefs generated from scanned models by our method.	118

List of Tables

4.1	Test cases for verifying our model synchronization method. . .	53
4.2	Comparisons among VSE, CityEditor, and CityGML Importer.	54
5.1	Notations for constructing self-supporting surfaces with planar quadrilateral elements	56
5.2	Evaluation of the physical measures θ , σ_D and σ_S , and planarity measure γ . For each model, we show the MIQ method [20]'s result (first row) and our result (second row). For each quality measure, the lower the value, the higher the quality. .	80
5.3	Evaluation of the physical measures θ , σ_D and σ_S , geometric measures ϕ and γ and running time t_{CDF} and t_P . For each model, we show Ma et al. [74]'s result (first row), our result (second row) and Liu et al. [68]'s result (third row). For each quality measure, the lower the value, the higher the quality. $\#\square$ is the number of quadrilateral elements. t_{CDF} is the time cost of calculating CDF and t_P is the time cost of extracting PQ mesh.	84

6.1	Statistics. $ V $: the number of mesh vertices; N_s : the number of singularities in the cross field; N_p : the number of quadrilateral patches; N_t : the number of T-junctions; N_n : the number of <i>new</i> vertices other than T-junctions added to the quad layout; $\bar{\varepsilon}$: the average quad regularity metric; $\bar{\delta}$: the average field coherence metric; and T : processing times in seconds. For each model, we show both our results (first row) and the results of Pietroni et al.'s method [84] (second row).	102
7.1	Variables s and d generated from different ω_1 . Experiments show that parameter ω_1 should be set to the longest edge of bounding box' reciprocal and ω_2 should be set to 1.	113
7.2	Variables s and d in depth compression function generated by solving an optimization problem.	117

List of Algorithms

1	Constructing 3D self-supporting surfaces with planar quadri- lateral elements	59
2	Computing principal curvatures and directions on triangle meshes	61
3	Computing separatrices	97
4	Extracting quad layout	99

Chapter 1

Introduction

1.1 Background

Deleting straight lines, sharp corners and acute angles, freeform architectures emphasize futuristic with sweeping lines, curves and irregular shapes. Thanks to modern computer-aided design (CAD) tools which boost the rapid development of freeform architectures. Many commercial products such as 3DMax[5], AutoCAD[6] and Maya [9] adapt various 3D modeling and processing techniques thanks to the progressively research results in architectural geometry. Around 2005, the geometry processing community realized that freeform architecture involves many problems highly relevant with computer graphics and geometry, especially with discrete differential geometry. This new application context also brings new objects and challenges and this area of study called architectural geometry has become a promising field.

This dissertation focus on four stage of architectural geometry, they are architecture maintenance, architecture construction, architecture texture mapping and architecture decoration. The most widely used polygonal mesh representation is used in this dissertation, the so-called face-vertex meshes,

consisting of a simple list of vertices and a set of faces that point to their vertices [48] in all proposed solutions. In addition to face-vertex meshes, direction field is also used in this dissertation. Direction field is defined on each face of a mesh by a pair of – possibly non-orthogonal – tangent vectors, and their opposite vectors. They are piece-wise constant on the faces of a triangle mesh (See Figure 1.1(c)). Figure 1.1(a) and (b) show frame field on triangle meshes, Figure 1.1 (c) shows four pairwise orthogonal unit length vectors in the tangent plane. Comparing to cross fields, the tangent vectors in direction fields are possibly non-orthogonal.

Figure 1.1: (a) The frame field of Botijo model. (b) The frame field of Elk model. (c) Four pairwise orthogonal unit length vectors in the tangent plane. Frame fields are visualized in line integral convolution (LIC), which is widely used to visualize vector field.

Many of today’s data models for 3D applications, such as CityGML or IFC, encode rich semantic information in addition to the traditional geometry and materials representation. Semantic information enables a wide range of application scenarios beyond visualization. Specifically, semantic and hierarchical information of building models is important for building information analysis [47], urban analytics [87], deployment of facilities [86], predic-

tion [111], and governance [51]. Semantic information provides the identification of key components in a generic 3D model and enables users to differentiate walls, roofs, ground, and other surfaces in a 3D building model. Hierarchical information carries the relationships between surfaces in 3D models; for instance, in 3D building models, it tells how a window is related to a wall surface. Nowadays, a variety of techniques are available for editing generic 3D models. However, the maintenance of semantic and hierarchical information through the editing process is only supported by specific applications that are closely tied to the underlying data model. We identified five main classes of popular geometry editing techniques: transformation, deletion, deformation, push-pull [66, 107], and parametric (procedural) templates [82, 78, 39]. Most editing tools preserve only geometric information, UV mappings, and materials. In order to provide a flexible integration of existing as well as future editing tools, an automatic matching and synchronization method for the modified geometry and the data model is needed. Many of today’s 3D applications provide SDKs and APIs where 3D geometry, UVs, and materials are exposed, and thus enable an easy integration of the presented method.

Besides architecture maintenance, architecture construction also brings new challenging problems for architectural geometry. Many of today’s architectures are designed in free form geometry. One major challenge in modelling of free form geometry in architectural scale is providing appropriate support structures. Timoshenko and Gere [106] introduce the principles and theory of structural stability to evaluate the mechanical resistance and stability of buildings. Self-supporting surfaces, as one of the most ancient and elegant techniques for building curved shapes, can be found in many architectural heritages, because of advantageous structural properties and efficient use of material [74]. A 3D surface is self-supporting if the internal stresses

are along the tangent directions so that they balance the surface’s gravity and there is no shear stress and bending moment inside the surface [45, 108, 55]. Designing self-supporting surfaces is an over-constrained problem since there are 3 constraints in the equilibrium equation whereas the surface has only 2 degrees of freedom, which are the isotropic stress function and the z-coordinate. Constructing self-supporting surfaces has received considerable attention in architectural geometry [85] since 2000s, leading to several computational tools [33, 68, 112, 16, 74]. Most of the existing self-supporting surfaces generation methods are particularly designed for triangle meshes, which are the dominant shape representation scheme in digital geometry processing thanks to its flexibility and ease of construction. However, triangle meshes require more number of edges and are heavier than quad meshes in architectural scale [85]. In addition, quad meshes provide more aesthetic value. Planar quadrilateral (PQ) meshes are highly desired in reducing gravity and modeling architectural glass structures [44, 70, 121]. We notice that quadrilateral meshes are highly desired in architectural geometry since most smooth surfaces have two dominating local directions, associated with principal curvature directions, and aligning quad elements to given directions is a natural way to capture shape features. Using triangle meshes, however, necessitates an arbitrary choice of a third edge direction. Therefore, constructing self-supporting surfaces with planar quadrilateral elements is an important problem with practical values.

Besides self-supporting surface construction problem, quad layout is also important for architecture texture mapping. Quadrilateral layouts separate the triangle mesh into quad patches, which provide an effective structure for parametrization and shape abstraction. Quad layouts are widely used in surface parameterization [21, 25], quadrangulation [35], quad-dominant

remeshing [4], quad-shaped domains decomposition [57], and subdivision surfaces [22]. As a semi-regular structure, the more patches close to a topological rectangle, the higher the quality of quad layout. This quality has significant influence on downstream applications. In spite of several existing methods, automatic methods are still far from providing high quality results on shapes with complicated geometry details.

Excepting computing quad layouts, high-relief modeling is also a challenging problem in architecture decoration. Relief is an art technique between sculptural and painting. Relief provides an impression that the emerged elements raised above the background. One of the biggest advantages of relief is that relief is less delicate and more securely fixed than a sculpture, for instance, a standing figure made from stone has weak point at the ankles. Reliefs are widely used in traditional Chinese ancient architectures, examples are shown in Figure 1.2. Types of relief are distinguished based on the degree of projection of the sculpted form from the field. The full range includes high relief, mid-relief, bas-relief, and shallow-relief. The distinction between high-relief and bas-relief is the most obvious and most significant, and these two are the only words used in this dissertation. Figure 1.3 shows a famous bas-relief and a modern high-relief. Unlike bas-relief, high-relief has materials undercut from background. Although bas-relief modeling is enforced thanks to height field, high-relief modeling is quite challenging since for each (x, y) , there can exist multiple z values. Existing methods enforce bas-relief modeling by utilizing techniques developed for image processing, high-relief modeling is still a challenging problem.

Figure 1.2: Traditional high-reliefs on traditional Chinese ancient architectures. Brick relief is a technique of sculpture in which subjects are left in relief on a brick-built surface or wall. Brick relief are widely used in Huizhou architecture.

Figure 1.3: (a) French 20th-century bas-relief. (b) Relatively modern high-relief in Bishopsgate, London [6]. Relief is a form of sculpture used for thousands of years. Bas-relief has been studied extensively in the graphics community. High-relief modeling is quite challenging since it deals with non-height fields. The full range of relief includes shallow-relief, bas-relief, mid-relief and high-relief, this dissertation focuses on bas-relief and high-relief.

1.2 Research Objectives

In this dissertation, we are interested in challenges in architecture maintenance, architecture construction, architecture texture mapping and archi-

texture decoration. According to the aforementioned background, there are various research objectives for each stage which are introduced in the following content.

Firstly, in architecture maintenance, there are three major challenges to maintaining semantic information across the editing process: the first one is to decide to what extent to depend on the specific edit operation. On the one hand, recovering information based on an edit operation is an intuitive way to maintain semantic information. On the other hand, this would mandate the same number of data model update strategies as edit operations, severely limiting its flexibility. Therefore, an automatic matching method independent of the specific edit operation is required. The second challenge is how to deal with both rigid and non-rigid transformations of polygons as well as how to distinguish existing faces from newly added faces. New faces need to be inserted and classified according to the data model. The third challenge is the performance of the automatic matching process, since it very likely runs after each interactive edit operation, and thus may impact the user experience by its running time.

Secondly, in architecture construction, modeling self-supporting surfaces with *planar quadrilateral* (PQ) meshes are highly desired in glass structures [44, 70, 121]. Designing self-supporting surfaces is an over-constrained problem since there are 3 constraints in the equilibrium equation whereas the surface has only 2 degrees of freedom, which are the isotropic stress function and the z-coordinate. A commonly used technique is thrust network analysis (TNA) [112, 16], which simplifies the 3D equilibrium problem into a 2D problem by projecting the surface to 2D space($\mathbb{R}^2$). Though TNA works for models with complex boundary conditions, the resulting surfaces are an only *approximate* solution, which contains parts with large stress error that may

break the self-supporting structure. Recently, Ma et al. [74] proposed a new idea to formulate the 3D equilibrium problem as a potential energy minimization problem in 4D, which can be solved efficiently using a variational method. Using a centroidal Voronoi tessellation (CVT)-like technique, their method produces high-quality triangle meshes which are more accurate in terms of mechanics measures than the results of TNA. Reparameterizing the results of [68, 74] usually fall short to this task, since there is no guarantee that the generated PQ meshes have planar quadrilateral elements. The main requirement in this reconstruction task is maintaining the original geometry and hereby keeping the equilibrium condition to satisfy the physical requirements. Since PQ meshes are the discrete counterpart of conjugate curve networks [17, 100], computing conjugate directions plays a critical role in PQ mesh construction [70, 85]. The existing methods [70, 69, 121, 36] apply global optimization to compute conjugate directions on polygonal meshes. Though they can yield smooth PQ meshes, these methods are computationally expensive, and are not easy to find physically-valid solutions due to high non-linearity in the optimization.

Besides constructing self-supporting surface in planar quad mesh, computing quad layout also faces difficulties. A commonly used approach for generating quad layouts on triangle meshes is to compute a smooth cross field and then trace the field lines. However, due to discretization error, it often yields redundant and/or missing curves and adds new singularities, resulting a quad layout with different topology as the cross field. To increase the robustness of curve tracing, Pietroni et al. [84] formulated an integer linear programming and solved it using the SCIP solver [2]. Pietroni et al.’s method is fully automatic and can compute field-aligned quad layout. However, their method requires new vertices, which are neither the singularities of

the input cross field nor T-junctions, to ensure that each patch is a quadrilateral. Moreover, their method is computationally expensive due to the nature of integer programming.

Lastly, for architecture decoration, unlike bas-relief, high-relief allows the materials undercut from background. Existing approaches developed for image processing cannot be applied to high-relief modeling. The core problem is high-relief cannot be represented as height fields which have a single z depth for each (x, y) position. To produce 3D printable high-reliefs, scaling is in need to reduce the volume of the real 3D model. However, scaling can also reduce the contrast of the input model. An effective method to enhance the contrast of the real 3D model is important in high-relief modeling.

1.3 Technical Contributions

This dissertation presents four solutions for challenges in architectural geometry, including maintaining semantic information across generic 3D model editing operations, constructing self-supporting surfaces with planar quadrilateral elements, computing field-coherent quad layout on triangle meshes and contrast-enhanced high-reliefs modeling. These four solutions covers various stages in architecture geometry including architecture maintenance, architecture construction, architecture texture mapping and architecture decoration.

In architecture maintenance process, maintaining semantic information across generic 3D model editing operations allows various editing operations to update the architecture while preserving the semantic information. The presented automatic matching method is based on a closed loop of interleaving interactive edit operations and automatic data model synchronizations.

Starting from a source data model with geometry, UVs and materials, the method preserves semantic information after geometry changes through a process of registration, recovering, and labeling. The method is independent of the specific editing operation, and can thus be applied to a wide range of interactive 3D tools or 3D batch processing jobs. The main contributions of the method are:

- Independence of edit operation: The method assumes no knowledge about the edit operation. It is able to recover information only based on the source data model and the modified 3D geometry.
- Integration flexibility: Since the method is only based on 3D geometry, UVs and materials, which are usually exposed through SDKs and APIs, integration in existing 3D applications is simple.
- Information preservation: The method can preserve all of the semantic information of the original model, classify newly added polygons, and label them according to the data model.

The method was implemented and tested as part of a LoD3 building editor for the Virtual Singapore project using CityGML as the underlying data model. It features standard 3D transformation tools, push-pull tools, and procedural façade templates through the integration of 3rd party libraries. These libraries disregard the data model and operate on the 3D geometry only. The CityGML data model is updated automatically after each edit operation. Thirty editing test cases were designed to verify that the data model was consistent, and the semantic information was maintained after each edit operation.

In architecture construction process, constructing self-supporting in PQ mesh is a challenging question. To overcome the aforementioned challenges

in existing methods, we propose a simple yet effective method for constructing self-supporting surfaces with planar quadrilateral elements. Our idea is to parameterize an existing triangle-mesh-based self-supporting surface and then extract a PQ mesh from the parameterization. Our method takes a triangular discretization of a self-supporting surface as input, and the principal curvatures and directions for each triangular face are calculated using a new discrete differential geometry approach, yielding more accurate results than existing methods. Then, the principal direction field is smoothed to reduce the number of singularities. Next, all faces are partitioned into two groups in terms of principal curvature differences. We compute a stretch matrix for faces with similar principal curvatures; it turns the principal directions into a pair of conjugate directions. For faces with large curvature difference, we keep the smoothed principal directions. Finally, a mixed-integer programming solver is applied to the mixed principal directions and conjugate directions to compute a planar quadrilateral mesh. Since our method does not require any non-linear optimization for computing conjugate direction fields (CDFs), it is easy to implement and computationally efficient. Experimental results on various self-supporting surfaces (including both height and non-height fields) demonstrate the effectiveness and computational efficiency of our method when generating high-quality self-supporting surface with planar quadrilateral elements. Figure 1.4 illustrates an example generated by our method.

In architecture texture mapping stage, quad layout is important for the following parameterization steps. The challenges in the existing quad layout computation methods are tackled with a new and practical method that can compute field-coherent quad layout in an efficient and robust manner. We

Figure 1.4: Given a self-supporting surface in triangular mesh, our method converts it into a planar quadrilateral (PQ) mesh that approximates the input self-supporting surface faithfully both in geometrical performance and physical performance.

propose an interpolation strategy to determine the ports¹ for each singularity. Specifically, given a singularity with index k (which is a multiple of $\frac{1}{4}$), there are exactly $4 - 4k$ ports produced. We also propose a rounding strategy that encourages the paths going through mesh vertices to reduce short segments and small faces. After obtaining the separatrices, we adopt a simple and effective approach to extract the quadrilateral patches that can deal with various types of topological cases in a unified way. Our method does not require any numerical solver, hereby is easy to implement and highly efficient. It takes only a few seconds for triangle meshes with 50K faces, whereas the existing optimization method often takes minutes. Computational results show that our results have consistently fewer quad patches than the state-of-the-art methods. For models with rich geometric details, our method can save up to 50% patches in the quad layout. In contrast to the existing method, our method does not add new vertices other than T-junctions in the computed

¹The tracing paths start from singularities. The direction of a tracing path is called *port*.

layout, hereby can preserve both the geometrical and topological features of the input cross field. Figure 1.6 illustrates a quad layout on Dragon model.

Figure 1.5: Quad layout and texture mapping on a complex human body.

Figure 1.6: Given a smooth cross field, our method computes a quad layout that 1) well aligns with the field directions; and 2) does not contain any new vertex other than T-junctions. Therefore, the computed layout preserves both the geometrical and topological features of the input cross field. For better visualization, we color the positive (resp. negative) singularities of the cross field in red (resp. blue).

In architecture decoration stage, high-relief is widely used for thousands of years in monumental sculpture and architecture. We present a simple and

effective solution to generate contrast-enhanced high-reliefs to solve the challenges in the existing methods. Our key idea is a depth compression function with only two variables for 3D models. In this function, normals and mean curvatures are utilized to enhance the contrast of the resulting high-reliefs. To calculate the two variables in depth compression function, we construct an optimization framework with two objectives, volume minimization and contrast maximization. The variables are obtained when the two terms reach a balance point. Our method narrows down the solution space to only two variables and is easy to implement and produces good-quality high-reliefs. We show results on a range of real 3D models include real-world and synthetic models. Figure 1.6 illustrates a high-relief generated by compressing a real 3D model.

Figure 1.7: (a) Front and side view of input model. (b) Front and side view of generated contrast-enhanced high-relief. Given a real 3D model, our method calculate the normals and mean curvatures of vertices, calculate the variables for depth compression function and produce a contrast-enhanced high-relief.

1.4 Dissertation Organization

The dissertation is organized into following structure: Chapter 2 introduces the preliminary knowledge of self-supporting surfaces, mean curvatures and

normals and smooth cross fields. The related literatures and work are covered in Chapter 3 including 3D model matching methods, state-of-the-art 3D editing techniques, state of the art self-supporting surface construction techniques, discrete differential geometry, generation of quad layout. Chapter 4 presents a data model synchronization method that preserves semantic information across editing operation relying only on geometry, UV mappings, and materials. Chapter 5 describes the method for constructing self-supporting surfaces with planar quadrilateral elements. Chapter 6 presents a simple and robust method for computing field-coherent quad layout on triangle meshes. Chapter 7 introduces a contrast-enhanced high-relief modeling method. Finally, Chapter 8 concludes the dissertation and discusses the future directions.

Chapter 2

Related Work

Several research areas are relevant for understanding the dissertation’s context. They are grouped in architectural geometry and discrete differential geometry. In architectural geometry, include: semantic information, editing techniques, information synchronization, self-supporting surfaces and relief modeling. In discrete differential geometry include: model registration, cross fields, principal directions and curvatures, quadrilateral layouts and conjugate direction fields.

2.1 Architectural Geometry

2.1.1 Semantic Information

A semantically-enriched 3D scene is crucial for non-visualization oriented applications. Alegre [3] proposed a probabilistic approach to the semantic interpretation of building façades. Verdie [110] reconstructed a 3D urban scene by classifying and abstracting the original meshes. With the assistance of Markov Random Field, Verdie’s method is able to distinguish ground, trees, façades, and roof surfaces. Zhu [129] proposed a point cloud classifi-

cation method based on multi-level semantic relationships, including point-homogeneity, supervoxel-adjacency, and class-knowledge constraints. Furthermore, Wu [119] presented an algorithm to produce hierarchical labeling of an RGB-D scene for robotics. Rook [95] automatically labeled faces using a decision tree but without support for LoD3 labels. All these methods address the broader problem of creating semantic information directly from 3D faces or point cloud data, and thus do not take into account prior information available in a rich source data model.

2.1.2 Editing Techniques for Architectural Models

Nowadays, a variety of techniques are available for editing generic 3D models. Push-pull (press-pull or sculpting) is a popular, intuitive editing technique which unifies previously distinct tools such as edge move, face move, face split, and extrude into a single tool. Having a single tool minimizes tool changes and thus improves the efficiency of the editing process. It has been adopted by many commercial products, such as AutoCAD [6], SketchUp [107], and Maya [9]. In particular, PushPull++[66] used in CityEngine [38], and ArcGIS Pro [37] employs methods for adaptive face insertion, adjacent face updates, edge collapse handling, and an intuitive user interface that automatically proposes useful drag directions. PushPull++ has been verified to be able to reduce the complexity of common modeling tasks by up to an order of magnitude comparing to existing tools, and is therefore well suited for editing building models and is integrated as part of the Virtual Singapore Editor (VSE). Esri’s procedural runtime [39] is used to generate parametrized geometry from façade templates. The procedural runtime is a library implementing the computer graphics architecture (CGA) shape grammar introduced in [82, 78], which defines a set of operations for

transforming shapes in space, with a special focus on patterns found in architecture.

2.1.3 Information Synchronization

While the method is independent of the data model, the VSE is based on CityGML [12], introduced by Kolbe [56], covering the geometrical, topological, and semantic aspects of 3D city models. CityGML defines the classes and relations for relevant topographic objects in cities and regional models with respect to their geometrical, topological, semantical, and appearance properties. There is wide industry support for importing CityGML, (e.g. Bentley Map [10], BS Contact Geo [14], CityServer3D [49], CodeSynthesis XSD [103], FME from Safe Software [99], and many more), but editing of the data model is supported by only a few options (e.g. [1, 109]). The method regards edit operations as black boxes, but looking at current tools, it helps to distinguish five main types of editing techniques: transformation, deletion, deformation, push-pull, and template applications. The following section gives an overview of the method and presents each step. Then, the key tasks of registration, recovery, information transfer between source and edited model, and automatic labeling are discussed. Internally, 3D geometry is represented as face-vertex meshes [42] which incorporate a list of vertices and a set of faces referencing their vertices. Both the source geometry and edited geometry are handled as meshes (M_s and M_e). Mesh M_s is represented by vertices $v \in V$ which are points in three-dimensional space. A vertex v_i may optionally be annotated by a two-dimensional texture coordinate $t_i \in T$. For each pair of connected vertices in f , an edge $e = (v_i, v_j)$ is defined. Planar faces $f \in F$ are defined by a list of counter clockwise oriented vertices $(v_1, v_2, \dots, v_n)$. Opening (holes) are represented as faces referencing

their boundary (parent) faces.

2.1.4 Self-Supporting Surfaces

self-supporting surfaces are often designed by TNA, which is a popular computational tool developed by Block and his colleagues [16, 15]. TNA simplifies the problem by approximates the stress field through uniaxial singular stresses and factors the over-constrained 3D equilibrium problem into horizontal and vertical sub-problems. TNA is applied in Vouga et al.’s research [113] for approximating freeform shapes by self-supporting ones. Since their method solves an over-constrained linear system, it cannot guarantee convergence when vertices have high valence and/or large distortion is needed to make the reference mesh self-supporting [74]. Miki et al. [77] generalized the Airy stress function in the planar elasticity problem to the 3D thin shell problem and developed a method for computing self-supporting spline surfaces with high-order smoothness. However, the mean curvature of the thin shell will cause the Airy stress function to be inconsistent with the true mechanical equilibrium. Therefore, this method can only compute approximate solutions. Both Liu et al. [68] and de Goes et al. [33] projected 3D surfaces onto 2D domain to simplify the problem. de Goes’s method allows the user to specify the stress distribution, which has greater flexibility. Liu’s method uses regular triangulation to ensure the equilibrium in the x - and y -dimensions.

Ma et al. [74] proved that the hyper-generatrix of a 4D minimal hypersurface of revolution is equivalent to a 3D self-supporting surface. As a result, constructing a 3D self-supporting surface is parallel to volume minimization, which can be further converted into a surface reconstruction problem with mean curvature constraint [76, 71]. Ma et al. showed that their method

Figure 2.1: Self-supporting surfaces generated by Ma et al.’s algorithm [74].

computed more accurate results of the over-constrained equilibrium problem than the TNA-based approaches [113, 68, 33]. The input of this dissertation is 3D self-supporting surfaces generated by Ma et al.’s method [74].

2.1.5 Relief Modeling

Cignoni et al. [30] originally generated reliefs by compressing the 3D objects nonlinearly. Consequently, following research works were presented to strengthen the visual details in the relief. Direct methods [101, 124, 123] first extract details from the input 3D models, then add the details to the compressed surface to improve the visual details of the relief. However, one important defect of these methods is weak control of depth which yields large depth discontinuities at silhouettes.

2.2 Discrete Differential Geometry

2.2.1 Model Registration

Model registration between the source geometry and the edited geometry allows recovering the transformation applied by an edit operation. Various methods were developed over the past to fully or partially match geometries and extract transformations between matching parts. Sundar [102]

proposed a skeleton based method for comparing and searching 3D models. The method expresses the model by a skeleton graph and uses graph searching techniques to match 3D models. DeepShape [120] extracts high-level shape features to match geometric deformations of shapes. A shape signature is computed with the method introduced in [102], which makes it possible to measure similarity of different shapes. This method represents the signature of an object as a shape distribution sampled from a shape function measuring global geometric properties of the object. Iterative closest point (ICP) is a widely used registration method introduced by Besl and McKay [11] and Zhang [128]. This efficient point set registration method meets well, the requirements in terms of interactive edit performance and simple geometry representation. ICP was proved to be adequate for the editing operations used in the test cases (see Table 4.1). If necessary, ICP variants such as [97] and [41] can be integrated. Other alternatives include: Coherent point drift(CPD), a probabilistic method introduced by Myronenko [80] for both rigid and non-rigid point set registration. Ma proposed his vector field consensus algorithm in [72]. Ge [43] presented the global-local topology preservation (GLTP) algorithm for non-rigid point set registration.

2.2.2 Cross Fields

Cross fields were introduced by Hertzmann and Zorin [48] for non-photorealistic rendering by means of cross-hatching. Palacios et al. [81] and Ray et al. [89] generalized cross fields to N -rotational symmetric directional field (N -RoSy). If singularities are given (i.e., their number, placement and indices are fixed), smooth cross fields can be computed efficiently by solving a sparse linear system [32]. Bommes et al. [20] formulated a mixed integer programming to compute smooth cross field, which is aligned with sharp

features and user-specified directional constraints. The smoothness energy is defined by measuring the angle difference of the crosses of adjacent triangles. However, since integer programming is NP-hard, it is difficult to compute a globally optimal solution. Cross fields can also be produced by prescribing a sparse set of directional constraints and letting the topology emerge from a smoothing process [88, 54]. Cross fields are widely used in non-photorealistic rendering [48, 81], texture synthesis [59], parameterization [52, 90] and quadrangulation [20, 58, 83]. In Frame field, one pair of vectors are allowed to be non-orthogonal. [65] using eigenfunctions from the Laplace-Beltrami operator for surface parameterization. The generated parametric lines are insensitive to local detailed features on the surface with low-mode eigenfunctions. [64] simplify low-contributing atoms and generate quality T-meshes for biomolecular surfaces. A cross field-based parametrization method is developed by extending the error-bounded atom elimination method.

2.2.3 Principal Directions and Curvatures

Discrete constructions of surfaces require estimations of differential geometric conceptions on a triangular mesh. The estimation of mean curvature is usually based on the Laplace operators [76, 73] and has been used to construct constant mean curvature surfaces [23] and self-supporting surfaces [73, 74]. The estimation of Gauss curvature is based on the angular defect [76]. The principal curvatures can be solved by the estimations. However, in order to construct conjugate directions, the estimation of principal directions becomes the core problem. On a smooth surface, the principal directions are the eigenvector of the surface curvature tensor at one point [4]. On a triangular mesh, the estimation of the surface curvature tensor are usually based

on the 1-ring neighbor vertices[96] or edges[31] around one vertex. These estimations are used to construct PQ meshes[70, 69]. We propose a more accurate method by updating the observation point movement strategy and avoid errors from normal of vertices. Comparisons are presented in Section 5.2.2.

2.2.4 Quadrilateral Layouts

Quadrilateral layouts are often used in conformal parameterization and quadrangulations [35, 20, 83, 21, 25]. Boier-Martin et al. [18] introduced the idea of producing a coarse quad layout for triangular meshes. The main technical challenge is to robustly trace curves over a piecewise linear, irregularly sampled surface, which often requires a careful and sophisticated implementation. With T-junctions, Myles et al.’s method [79] enlarges the solution space, allowing for coarser layouts with milder distortion. There are roughly three types of methods for computing quad layouts, including greedy strategies [19, 105], user-guided methods [50, 26, 104, 75] and optimization-based methods [91, 122, 84]. Quad mesh generation with guaranteed quality is a related and challenging research problem. [63] presents an algorithm to match the generated interior and exterior meshes with conformal boundary, preserving the guaranteed angle bounds. In addition, we introduce another automatic and robust approach to preserve sharp features. [62] and [61] study similar problems while [62] is not fully automatic and [61] is unable to preserve sharp features.

2.2.5 Conjugate Direction Fields

Conjugate Direction Fields are the core of constructing PQ meshes, As mentioned in Chapter 1, PQ mesh appears in discrete geometry as counterparts

of conjugate curve networks. Liu et al. [70] first proposed a general method to design and compute PQ meshes by producing a conjugate direction field (CDF) based on using nonlinear optimization. The nonlinear constraint in this method is face planarity and the objective function is the fairness of the main mesh polylines. Based on measuring the smoothness of the representation vector field, Zadavec et al.'s method [121] is capable of producing a smooth CDF. However, only singularities with indices of $\pm k/2 (k \in \mathbb{Z}^2)$ can be modeled and it fails in handling $\pm k/4 (k \in \mathbb{Z}^2)$ singularities, such as a surface with convex corners (e.g., a round cube). Liu et al.'s method [69] is based on the study of conjugate direction fields (CDF) which allow the presence of $\pm k/4 (k \in \mathbb{Z}^2)$ singularities. Our method is more computational efficient than these existing methods since we adapt stretch matrix to produce conjugate directions. Quad meshes have better performance by providing nice tensor-product structure for IGA with many superior applications in engineering than the traditional finite element method. For instance, shell analysis [28, 27], vesicle and capsule [29].

Chapter 3

Preliminaries

This chapter presents formal mathematical definitions appear in this dissertation. First, we introduce discrete normals, mean curvatures and principal curvatures of vertex, it is helpful in understanding the construction of the mixed direction fields. Then, self-supporting surfaces with isotropic stresses are introduced with the formal definition, it is an introduction to our input models. Next, conformal parameterization is introduced, it is meaningful for understanding the downstream application of quad layout. Finally, smooth cross field are discussed.

3.1 Discrete Normals and Mean Curvatures

Computing differential properties, such as normals and curvatures, plays a critical role in geometry processing. On a regular surface, we compute normal $\mathbf{n}$ (See Figure 3.1) as

$$\mathbf{n} = \frac{\mathbf{r}_u \times \mathbf{r}_v}{\|\mathbf{r}_u \times \mathbf{r}_v\|},$$

Figure 3.1: Tangent vector and normal vector in curved surface.

and mean curvature H as

$$H = \frac{EN - 2FR + GL}{2(EG - F^2)},$$

where $L = \mathbf{r}_{uu} \cdot \mathbf{n}$, $R = \mathbf{r}_{uv} \cdot \mathbf{n}$ and $N = \mathbf{r}_{vv} \cdot \mathbf{n}$ are the coefficients of the second fundamental form.

Discrete differential geometry [76] is used to compute discrete normals and mean curvatures on triangle meshes. Given a manifold triangle mesh M , denote by V its total volume and A the surface area. Let ∇_i be the gradient operator with respect to the coordinates of vertex $\mathbf{v}_i$. The vertex normal $\mathbf{n}_i$ and mean curvature H_i can be computed by measuring local area and volume variations

$$\mathbf{n}_i = \frac{\nabla_i V}{\|\nabla_i V\|}, \quad (3.1)$$

and

$$H_i = \frac{1}{2} \frac{\nabla_i A \cdot \nabla_i V}{\|\nabla_i V\|^2}. \quad (3.2)$$

The above discrete measures are used in constructing constant mean curvature surfaces [73] and self-supporting surfaces [74].

3.2 Self-supporting Surfaces with Isotropic Stresses

Ma et al. [74] showed that computing a 3D self-supporting surface is equivalent to constructing a 4D minimal hypersurfaces of revolution. They proved a 3D surface that minimizes the following energy functional is a self-supporting structure with isotropic stresses

$$E_{sss} = \int (z_0 - z) dA, \quad (3.3)$$

where z_0 is the zero potential level, which is higher than the peak of the surface. Using variational method, they derived a sufficient condition of self-supporting structures

$$2(z_0 - z)H = n_z, \quad (3.4)$$

where n_z is the z -coordinate of normal.

Ma et al. [74] developed an iterative method for constructing self-supporting surfaces with triangle elements. Their method allows the user to specify the target volume. With each iteration, it updates the zero potential level z_0 as

$$z_0 = \frac{\sum \mathbf{n}_{zi} + 2z_i H_i}{2 \sum H_i}, \quad (3.5)$$

and then computes the normal and tangential displacements to update the vertex positions to satisfy (3.5). We refer the user to [74] for details. It is worth noting that Ma et al.'s method produces triangle meshes only.

Figure 3.2: Reducing area distortion in conformal parameterization. (a) The parameterization has no singularity but very large area distortion. One can see the sizes of the quad elements vary significantly. Also, its iso-parameter lines are not orthogonal to the boundary. (b) We improve the parameterization quality by adding 4 singularities (in red). Note the area distortion is reduced and the iso-parameter lines are aligned with the boundary.

3.3 Conformal Parametrization

Let $\mathbf{r} = \mathbf{r}(u, v)$ be a regular surface in $\mathbb{R}^3$. Its intrinsic geometry is characterized by the first fundamental form $E = \mathbf{r}_u \cdot \mathbf{r}_u$, $F = \mathbf{r}_u \cdot \mathbf{r}_v$, and $G = \mathbf{r}_v \cdot \mathbf{r}_v$. If the parameterization is conformal, the gradients of the parameters are orthogonal and have the same magnitude $\nabla u \perp \nabla v$ and $\|\nabla u\| = \|\nabla v\|$. As a result, $E = G$ and $F = 0$. Conformal parameterization preserves angles, but area distortion is inevitable unless the surface is developable. A typical method for reducing area distortion is to add singularities. See Figure 3.2 for an example.

The difference of an arbitrary parameterization and a conformal parametrization can be measured by the conformal energy, defined as follows:

$$E_{conf} = \int \left(\frac{\lambda_1}{\lambda_2} + \frac{\lambda_2}{\lambda_1} \right) dA, \quad (3.6)$$

where λ_1 and λ_2 are the eigen values of the matrix composed of the first fundamental form $[E, F; F, G]$, and dA is the area element. For a conformal parametrization, we have $E = G$ and $F = 0$. As a result, the eigen values

are equal $\lambda_1 = \lambda_2 = E$, and the conformal energy is twice the area of the surface.

3.4 Smooth Cross Fields

Cross field is composed of a pair of perpendicular field on a surface. On a triangular mesh, the cross field are commonly expressed by putting a cross chart on each facet. Every cross chart is composed of 4 vectors. The chart rotation $\Delta\theta$ between a pair of adjacent facets are defined as the least angle among all angles between the vectors from the two charts. Under this definition, the singularity index $I(v_i)$ of each vertex v_i can be defined:

$$I(v_i) = \left(\sum_{vi \in e_{ij}} \Delta\theta|_{e_{ij}} + \delta(v_i) \right) / \frac{\pi}{2} \quad (3.7)$$

where the chart rotation $\Delta\theta_{e_{ij}}$ is counted as the anti-clockwise around v_i , $\delta(v_i)$ is the angle defect:

$$\delta(v_i) = 2\pi - \sum_{v_i \in F_{ijk}} \angle v_k v_i v_j$$

For most vertices v_i , $I(v_i) = 0$. But usually, there are several vertices S_i with $I(S_i) \neq 0$ and $I(S_i) \in \mathbb{Z}$. Such vertices are called singularities. The charts around a singularity is not coherent. So a field line touching the singularity is ended here.

Chapter 4

Maintaining Semantic Information across Architectural Model Editing

4.1 Problem Statement

Many of today's data models for 3D applications, such as City Geography Markup Language (CityGML) or Industry Foundation Classes (IFC) encode rich semantic information in addition to the traditional geometry and materials representation. However, 3D editing techniques fall short of maintaining the semantic information across edit operations if they are not tailored to a specific data model. While semantic information is often lost during edit operations, geometry, UV mappings, and materials are usually maintained. This chapter presents a data model synchronization method that preserves semantic information across editing operation relying only on geometry, UV mappings, and materials. This enables easy integration of existing and future 3D editing techniques with rich data models. The method links the original

data model to the edited geometry using point set registration, recovering the existing information based on spatial and UV search methods, and automatically labels the newly created geometry. An implementation of a Level of Detail 3 (LoD3) building editor for the Virtual Singapore project, based on interactive push-pull and procedural generation of façades, verified the method with 30 common editing tasks. The implementation synchronized changes in the 3D geometry with a CityGML data model and was applied to more than 100 test buildings.¹

4.2 Method Overview

Figure 4.1 gives an overview of the model synchronization method. Applying an edit operation to a source geometry (M_s) results in an edited geometry (M_e) deprived of semantic information because we assume that these are not preserved by the edit operation.

The model synchronization method can be divided into three steps:

- **Model registration:** M_s is the source point set for the ICP algorithm, whereas M_e is its target point set. The output of ICP is a transformation matrix which is applied to the source M_s to yield a transformed point set M_t . An octree storing additional information (face normal, face centroid, length of edges) is constructed from the transformed model M_t .
- **Semantic information transfer:** An octree search with faces from M_e yields matching rigid transformed faces in M_t and their original semantic labels in M_s . Deformed faces are handled in a similar way by an additional UV space search.

¹The work in this chapter has been accepted by Remote Sensing.

Figure 4.1: Flow chart of automatic 3D model synchronization.

- **Labeling:** Each face in the edited geometry M_e is incorporated into a topological tree to detect openings and their boundary surfaces. Boundary surfaces are labeled as roof, wall, or ground faces based on their normal direction. For openings, the octree and UV space are used again to find the position in the data model, finally resulting in an updated data model that preserves as much semantic information as possible and which is updated with new faces labeled according to their topology.

4.2.1 Face Matching

Since M_e contains only geometric information after an edit operation, faces $f_e \in M_e$ have to be matched with corresponding faces $f_s \in M_s$ to find related semantic information. The following feature are used for face matching:

- Face normal f_{normal} .
- Face centroid $f_{centroid}$.
- Length of edges f_{length} .
- Set of texture coordinates f_T .

If the distance between two vertices v_i and v_j is below a given threshold ϵ_v , they are considered the same. Likewise, if the UV distance of two texture coordinates t_i and t_j is below a given threshold ϵ_t , their UVs are considered the same (See Figure 4.2).

Figure 4.2: UV space of a building model. In UV space, each vertex v_i of the geometry has a corresponding texture coordinate t_i . The UV space simplifies matching faces, since texture coordinates remain unchanged or are re-sampled during deformation, push-pull, and template applications.

Two faces $f_t \in M_t$ and $f_e \in M_e$ are matched if:

- f_t and f_e have same f_{normal} , $f_{centroid}$, and f_{length} .
- f_t and f_e share same set of texture coordinates.

For matching faces, semantic information from $f_s \in M_s$ can be transferred to $f_e \in M_e$ via $f_t \in M_t$.

4.2.2 Model Registration

Several editing operations include rigid transformation and scaling, which can be represented by a transformation matrix. The ICP algorithm is used by the method to calculate the transformation matrix between the source M_s and the edited geometry M_e . ICP can be applied to the source geometry as a whole to compute the global transformation of the geometry or partially to recover local transformations.

ICP Algorithm

The ICP algorithm is a simple and efficient way to find the best match between two point sets by solving an optimization problem. The inputs of the ICP algorithm are two point sets, the source point set X (see Equation (4.1)), and target point set P (see Equation (4.2)); the point sets may have different sizes, and the distance between two points is defined by an euclidean metric.

$$X = \{x_1, \dots, x_{N_x}\} \quad (4.1)$$

$$P = \{p_1, \dots, p_{N_p}\}. \quad (4.2)$$

The ICP algorithm finds the transformation T in an iterative approach, applying transformation T to X and computing the fitness error $E(T)$, until $E(T)$ is less than a given threshold.

$$E(T) = \frac{1}{N_p} \sum_{i=1}^{N_p} \|x_i - T(p_i)\|^2. \quad (4.3)$$

Point Set Generation

In order to increase the quality of the ICP result for a given edit operation, more points than just the vertices of the source geometry are usually required. Generating points by sampling uniformly the edges is easy to implement but will result in the same points for symmetries, and thus prevent ICP from recovering, e.g., rotation or mirroring along these symmetries (see Figure 4.3). Random sampling is a widely used strategy [13, 94], but may add an error to the ICP result because of the noise introduced by the random sampling.

Figure 4.3: A building model with symmetries. Using a uniform sampling strategy will prevent ICP from recovering a rotation by 180° .

Our method uses random sampling along edges and diagonals but with a consistent random sequence based on a random seed computed from the texture coordinate t_i of vertex v_i . As illustrated in Figure 4.4, additional points for the ICP algorithm are generated along diagonals by connecting opposite vertices. If the source geometry does not have texture coordinates, they are generated beforehand. Initializing the random number generator from the texture coordinates results in a unique sequence of sampling points for every edge and diagonal in M_s , and hence in M_e , which then can be unambiguously registered by the ICP algorithm.

The edge and diagonal sampling is done as follows:

For a segment S_{ij} with vertices (v_i, v_j) and texture coordinates (t_i, t_j) :

- Calculate two random seeds $seed_i$ and $seed_j$ based on the texture coordinates of t_i and t_j .
- Generate n random numbers $r_{i_1}, r_{i_2}, \dots, r_{i_n}$ based on $seed_i$, and n random numbers $r_{j_1}, r_{j_2}, \dots, r_{j_n}$ based on $seed_j$. By applying these random numbers, we obtain n sampling points s_{ij_k} and n sampling points s_{ji_k} (see Equation (4.4))
- Add the points obtained by the second step to the source point set and target point set of ICP algorithm

$$\begin{aligned} s_{ij_k} &= v_i + r_{i_k}(v_j - v_i) \\ s_{ji_k} &= v_j + r_{j_k}(v_i - v_j). \end{aligned} \tag{4.4}$$

This enables the ICP algorithm to register the source geometry to the edited geometry, since there exist unique corresponding point sets for each segment in M_s and M_e (See Figure 4.5).

Figure 4.4: Edges and diagonals of a face are sampled with a random number generator which is initialized by the texture coordinates of the vertices of the edge or diagonal.

Parameter Analysis

Two parameters in ICP need to be determined for our method. First, the fitness error between M_s and M_e ; 0.01 m is sufficient for buildings reconstructed

Figure 4.5: Example of a generated source point set (red) and generated target point set (yellow) for the iterative closest point (ICP) algorithm.

from airborne and ground-based point clouds. Second, the maximum number of iterations.

Considering that VSE supports five main types of editing operations, six experiments were designed to derive the value of maximum number of iterations:

1. Translate 1000 m;
2. Translate 10 m;
3. Deletion on a face;
4. Deformation on a face;
5. Push-pull on a face;
6. Template application on a face.

These experiments were executed on a complete building model with 495 faces (see Figure 4.6). Figure 4.7 shows source point set and target point set.

Figure 4.8 shows that deletion, deformation, push-pull, and template applications terminate ICP algorithm in the very beginning. For all experiments, fitness error falls below the threshold after no more than 90 iterations. In these experiments, maximum number of iterations can be set to

Figure 4.6: A complete building model to derive the maximum number of iterations.

an arbitrary value over 90. It was set to 300 to deal with potential complicated situations.

The registration result shown in Figure 4.9 indicates that 300 ICP alignment iterations and a fitness error of 0.01 m are sufficient to calculate the transformation matrix T . The red point set was generated from M_s , the yellow point set was generated from M_e , and the white point set was the transformed source point set by applying T to source point set.

Figure 4.7: The source point set and target point set for the building model in Figure 4.6

Figure 4.8: Experimental results under different numbers of iterations.

Figure 4.9: Registration result. The white point set is the transformed source point set. It overlaps with the point set generated from the edited geometry M_e .

4.2.3 Semantic Information Transfer

To match semantic information from the source model to the edited geometry, a search for matching faces had to be performed. The implementation uses an octree to speed up the search process.

An octree is a hierarchical data structure allowing efficient search operations in three-dimensional space. To match the semantic information of the

original model, we constructed an octree based on the transformed source geometry $M_t = T \times M_s$. The root node of the octree is the bounding box of the source model. The bounding box is recursively divided into eight nodes until each box contains only one node as shown in Figure 4.10. Each node stores the face features from Section 4.2.1.

Figure 4.10: A building model and its octree.

Whether a face f_e in the edited geometry matches a face in the source data model is determined by searching for the corresponding face features (see Section 4.2.1) in the octree. Matching faces allow the transfer of semantic information from the source model M_s to the edited geometry M_e .

As illustrated in Figure 4.11, the deformation of a vertex, edge, or face changes f_{normal} , $f_{centroid}$, and f_{length} . Such a deformed face will not be found by an octree search. However, deforming edit operations only affect the geometry but preserve texture coordinates. Therefore, these faces can still be matched by an UV space search, a method also used for texture transfer (see for example [34]), and hence can be used to transfer semantic information.

After the octree and UV search, unmatched faces in the source data model M_s are deleted, and unmatched faces in the edited geometry M_e are considered new faces. For these, semantic information has to be generated through labeling, which will be discussed in the next section.

Figure 4.11: Deformation operations on vertex, edge, and face. Deformation will change f_{normal} , $f_{centroid}$ and f_{length} . A UV space search was performed for these faces.

4.2.4 Labeling

Semantic information transfer based on faces can be applied as described to any data model. However, labeling new faces is domain specific and the approach described here is based on CityGML, which is the underlying data model of the implementation. According to the CityGML building model [46] the relevant classes for LoD3 building model are divided into three levels (see Figure 4.12), which are building, boundary surface, and opening. Specifically, the boundary surfaces are roof surface, wall surface, and ground surface, whereas the openings include windows and doors.

Figure 4.12: Relevant parts of the CityGML data model for LoD3 modelling.

Typical edit operations creating new boundary surfaces or openings are push-pull (see Figure 4.13) and template applications (see Figure 4.14). New boundary surfaces are classified according to the following heuristic where

f_ϕ is the angle between the face normal f_{normal} and the up vector of the coordinate system, as shown in Figure 4.15(a). This basically assumes that walls are within $+/-15$ degree from the vertical (90 degree):

$$\begin{cases} f_\phi < 75^\circ, & \text{roof surface} \\ 75^\circ \leq f_\phi < 105^\circ, & \text{wall surface} \\ f_\phi \geq 105^\circ, & \text{ground surface} \end{cases} \quad (4.5)$$

Figure 4.13: Opening generated by push-pull. This editing operation will not split the boundary surface but generate a hole on the boundary surface.

Figure 4.14: Openings generated by template application. This editing operation will split the boundary surface and generate several new faces co-planar with the boundary surface.

Starting from a boundary surface (labeled either by semantic information transfer or by the heuristic), topological relationships are constructed by the iterative method depicted in Figure 4.15(b).

Figure 4.15: (a) f_ϕ : the angle between the face normal f_{normal} and the up vector of the coordinate system. (b) Openings detected by topology relationships.

This iterative approach constructs a topological tree based on the edited geometry. First, all new faces which are co-planar with labeled faces are collected and added to the topological tree as the first level. Second, it is determined whether the rest of the newly added faces *share* an edge or have an edge *on the face* of the leaf node in the topological tree. If yes, we add these faces to corresponding leaf nodes. This process is repeated until there are no new faces left. Finally, leaf nodes of all wall surfaces and with tree depth larger than two are considered as openings. Openings are classified as follows with h denoting the height and w denoting the width of the bounding

box of the opening in meters. The constraints for doors are taken from [24].

$$\begin{cases} 2.0 < h < 2.5 \wedge 0.8 < w < 1.2, & \text{door} \\ \text{else,} & \text{window} \end{cases}.$$

In addition, roof infrastructures generated by push-pull operations (see Figure 4.16) are not openings, since in the topological tree they are children of roof surfaces.

Figure 4.16: Adding roof infrastructure for building models by push-pull.

4.3 Discussions

The method was implemented as part of a standalone JavaFX based LoD3 building editor for the Virtual Singapore project. The Esri procedural runtime provides a façade template application and the PCL (Point Cloud Library [pcl2019pcl]) and ICP implementations, along with other utilities for point set management. Besides camera control, CityGML import/export, and point cloud visualization, selection, and texture assignment, Figure 4.18

shows the user interface of VSE, the following editing operations are supported by the editor:

- Model move, scale, rotate (see Figure 4.9).
- Vertex, edge, and face move (see Figure 4.11).
- Face and edge push-pull (see Figure 4.13).
- Template application (see Figure 4.14).
- Face split (see Figure 4.17).
- Vertex, edge, and face deletion (see Figure 4.19).

Figure 4.17: Face split operation.

We used FZKViewer [53] as 3rd party software to verify that the CityGML attributes were maintained when editing the building model. To illustrate typical edit operations and how the method updates the data model, two use cases are presented in the following sections: face deletion for shelters and template application for openings.

4.3.1 Face Deletion for Shelters

A typical example for superfluous faces and incorrect semantic information are faces which are present because of occlusions at ground level. The passageway under the shelter, shown in Figure 4.19, was constructed as a volume

Figure 4.18: user interface of VSE.

instead of just the roof section of the passageway. The editor allows the user to remove superfluous faces through deletion and the data model is updates accordingly (see Figure 4.20).

Figure 4.19: Deleting faces for shelters in the editor.

Figure 4.20: FZKViewer: Data model update after deletion. Roofs surfaces in red, wall surfaces in grey.

4.3.2 Template Application For Openings

Procedural modelling is used for adding regular structures; e.g., extending façade patterns into occluded areas or correcting measurement errors in the source model. Figure 4.21 shows hundreds of added openings in a high-rise building. Through the labeling process of the method, openings are correctly detected and classified as windows in the data model (see Figure 4.22).

These two use cases illustrate the automatic data model update process, which is essential for an effective editing workflow. While the increase in efficiency, compared to manual labeling, largely depends on the editing tasks, data model consistency has shown itself to be one of the greatest benefits of the method.

4.3.3 Commercial Tools

Several commercial tools are available which support editing of CityGML models. The following sections discuss a selection of these tools in terms of their advantages and limitations regarding editing of LoD3 CityGML build-

Figure 4.21: Façade template application.

Figure 4.22: FZKViewer: Data model update for template application. Roofs surfaces in red, wall surfaces in grey, windows in turquoise.

ing models.

CityEditor

Trimble SketchUp [107] is a complete and extensible 3D modelling package, widely used in 3D modelling and architectural design. CityEditor [1] is an extension for SketchUp, which supports import/export and editing of CityGML files. As an extension, CityEditor leverages the full capabilities of

SketchUp, going far beyond the editing operations that the VSE supports. However, it lacks specific LoD3 tools, such as façade template application. Furthermore, CityEditor discards all semantic information when importing a CityGML file, and each surface is labeled "unclassified." Users need to assign semantic information for each surface manually. CityEditor also supports limited automatic surface classification depending on the tilt of the surface. The surface classification can then be exported to a CityGML file.

CityGML Importer

FME Desktop from Safe Software [99] is data conversion software which supports more than 450 formats and applications to help integrate and transform data. CityGML Importer [98] is also based on Safe Software's data transformation technology and focuses on loading and converting CityGML data; e.g., into IMX files which can be edited in InfraWorks [8] and Map 3D [7]. While loading a CityGML file, CityGML importer maintains the semantic information of each surface by storing them in separate layers. However, the hierarchical structure of the CityGML file is lost. For instance, building parts and boundary surfaces are at different hierarchical levels in the data model, but CityGML Importer discards the relationship and puts the layers on the same level. The VSE maintains the semantic information as well as the hierarchical structure of the CityGML data model.

RhinoCity

Rhino3D [93] is a popular architectural design software which has extensive support for curves and free-form surfaces. The polygon-based editing operations of the VSE are not well suited for curved surfaces and further development is required to provide a good user experience. Through the

Grasshopper extension, parametric 3D modeling can be added to Rhino3D which offers similar capabilities as the CGA based template application of the VSE. RhinoCity [92] is another extension for Rhino3D and focuses on generation of building models and supports import/export of CityGML files. RhinoCity leverage the full editing tools from Rhino and offers a far richer set of editing operations than the VSE. RhinoCity also supports automatic generation of CityGML data during the 3D model creation phase but lacks the maintenance of semantic information during further editing.

AutoCAD Map 3D

AutoCAD [6] is a commercial computer-aided design (CAD) and drafting software which has several domain-specific enhancements. For instance, AutoCAD Map 3D [7] is one of AutoCAD’s vertical applications. AutoCAD Map 3D incorporates geographic information system and CAD data with an industry-specific toolset for GIS and 3D mapping. AutoCAD Map 3D supports importing and exporting objects into CityGML format. However, CityGML import is limited to level of detail 2 (LoD2), whereas the VSE is designed explicitly for LoD3 editing. Furthermore, much of the semantic information is lost when exporting to CityGML format from Map 3D.

Comparisons among VSE, CityEditor, and CityGML Importer

We compared our method with CityEditor and CityGML Importer to edit the building model shown in Figure 4.23. Fourteen test cases in Table 4.1 were chosen while other test cases were ignored, since they were not applicable in Skethup or 3ds max. For instance, test case 29 and 30 were façade template applications provided by Esri procedural runtime which were not feasible in Sketchup and 3ds max. One metric was employed: the mini-

mum amount of mouse and keyboard interactions required for updating the semantic information of the edited geometry.

To record the metric, one user performed the 14 test cases per software. In GUI of VSE, the user can click the refresh button after an edit operation to update semantic information of the edited geometry. In CityEditor, the user need to assign semantic label manually in the context menu. CityGML Importer translates CityGML files into IMX files and stores each polygon into separate layers. The name of the layer is the semantic label of the polygon. The user can edit the name of the layer to update the semantic information. The results are shown in Table 4.2.

There are two observations from Table 4.2. First, our method is helpful for an efficient workflow with multiple newly added faces. For transformation, deletion, and deformation, no extra mouse clicks are needed to update the semantic information in Sketchup and 3ds max. For push-pull, which generates newly added faces, users need several mouse clicks to manually assign semantic labels for the newly added faces. Thanks to automatic labeling introduced in Section 4.2.4, manual labor is saved in updating the edited geometry. Second observation is our method has advantages in importing and exporting CityGML data models. As mentioned in Section 4.3.3, CityEditor lost all semantic information in loading the CityGML data model. The comparison excludes the number of mouse clicks in initializing the semantic label of the model in VSE, CityEditor, and CityGML Importer. Extra tools are needed to export the shape in 3ds max into CityGML format, while VSE supports the direct export of CityGML files.

Figure 4.23: A building model and its semantic information for comparisons among VSE, CityEditor, and CityGML Importer.

4.4 Summary

This chapter introduced a data model synchronization method which preserves semantic information across editing operation. It is independent of the edit operation and depends only on geometry, UV mappings, and materials. This enables easy integration of existing and future 3D editing tools with rich data models in a broad range of 3D applications. The method was implemented in a LoD3 building editor for the Virtual Singapore project, including interactive push-pull and procedural generation of façade provided by 3rd party libraries. The quality of the method was verified with 30 common editing tasks and compared with four commercial tools. The VSE internally uses a polygon-based representation for the geometry which is not well suited for freeform surfaces. Further research is required for recovering surface parametrization and designing an easy to use toolset for editing curved surfaces. Currently, the focus is on interactive editing operations only. It would be interesting to see how accuracy our labeling process will be in more practice cases and how the method performs in batch processing of 3D geometry, e.g., for change detection and labeling, an important topic for maintaining and updating 3D city models.

Table 4.1: Test cases for verifying our model synchronization method.

Transformation	Translate the entire model	01
	Rotate the entire model	02
Deletion	Delete single vertex	03
	Delete multiple vertices	04
	Delete edge between two non-planar faces	05
	Delete edge between two co-planar faces	06
	Delete face with holes	07
	Delete face without holes	08
	Delete face with child faces	09
	Delete face without child faces	10
Deformation	Move single vertices	11
	Move multiple vertices	12
	Move single edge	13
	Move multiple edges	14
	Move single face	15
	Move multiple faces	16
	Move arbitrary selection of faces / edges / vertices	17
Push-pull	Push-pull of existing face follow the face normal	18
	Push-pull on existing face with an angle	19
	Push-pull on existing face in corner vertically	20
	Push-pull on existing face in corner with an angle	21
	Push-pull on existing edge	22
	Push-pull after creation of a face on an existing face	23
	Push-pull after creation of a edge on an existing face	24
	Push-pull after creation of multiple face on an existing face	25
	Push-pull after creation of multiple face on multiple existing face	26
	Push-pull after creation of a face on an existing face with holes	27
	Push-pull after creation of a circle on an existing face	28
Template	Apply procedural templates with one level of opening extrusion	29
	Apply procedural templates with two levels of opening extrusion	30

Table 4.2: Comparisons among VSE, CityEditor, and CityGML Importer.

Initialization	0	90	1
01	1	0	0
02	1	0	0
05	1	0	0
08	1	0	0
10	1	0	0
13	1	0	0
14	1	0	0
15	1	0	0
16	1	0	0
18	1	4	4
20	1	4	4
23	1	10	10
25	1	20	20
26	1	20	20

Chapter 5

Constructing Self-Supporting Surfaces with Planar Quadrilateral Elements

5.1 Problem Statement

This chapter presents a simple yet effective method for constructing 3D self-supporting surfaces with planar quadrilateral (PQ) elements. Starting with a triangular discretization of a self-supporting surface, we first compute the principal curvatures and directions of each triangular face using a new discrete differential geometry approach, yielding more accurate results than existing methods. Then, we smooth the principal direction field to reduce the number of singularities. Next, we partition all faces into two groups in terms of principal curvature difference. For each face with small curvature difference, we compute a stretch matrix that turns the principal directions into a pair of conjugate directions. For the rest of triangular faces, we simply keep the smoothed principal directions. Finally, we apply the mixed-integer pro-

gramming solver to the mixed principal directions and conjugate directions and compute a planar quadrilateral mesh. Experimental results demonstrate that our method is computationally efficient and can yield high-quality PQ meshes. Since our method does not require any non-linear optimization for computing conjugate direction fields (CDFs), it is easy to implement and highly efficient. Computational results on various self-supporting surfaces (including both height and non-height fields) demonstrate the effectiveness and computational efficiency of our method in generating high-quality self-supporting surface with planar quadrilateral elements.¹

To ease reading, we list the major notations in Table 5.1.

Table 5.1: Notations for constructing self-supporting surfaces with planar quadrilateral elements

Item	Description
$\Omega \subset \mathbb{R}^2$	2D domain
$u, v \in \Omega$	parameters
$\mathbf{r}(u, v) \subset \mathbb{R}^3$	3D surface
$M = (V, E, F)$	Input 3D triangular mesh
$\mathbf{e}_1, \mathbf{e}_2$	Local coordinate axes for each face
$\mathbf{n}$	Unit normal
$\mathbf{C}$	Curvature tensor
κ_1, κ_2	Principal curvatures
$\mathbf{d}_1, \mathbf{d}_2$	Principal directions
$\mathbf{S}$	Stretch matrix
$\mathbf{u}, \mathbf{v}$	Conjugate tangent directions

¹The work in this chapter has been accepted by Computational visual Media.

5.2 Algorithm

5.2.1 Overview

Our method takes a self-supporting surface represented by triangle mesh as input. We first compute the principal curvatures and directions for each triangular face using a new method for computing curvature tensor on triangle meshes. Then we group the triangular faces whose principal curvature difference is small. We call these faces “free faces” and the rest faces “fixed faces”. Since the principal directions on fixed charts are already conjugated, our goal is to construct a pair of conjugate directions for each free chart. Towards this goal, we compute a stretching matrix that turns the principal directions of free charts into conjugate directions. The principal directions of the fixed charts and the conjugate directions of the free charts induce a conjugate direction field on the input mesh. Applying the mixed integer quadrangulation method [20] to the CDF, we parameterize the triangle mesh and extract the planar quadrilateral elements. We document the algorithmic details in Section 3.2 to 3.6. We show the algorithmic pipeline of our method in Figure 5.1 and the pseudocode in Algorithm 1.

5.2.2 Computing Principal Curvatures and Directions

Curvature is the amount by which a curve deviates from being a straight line, or a surface deviates from being a plane. For a smooth 3D surface $\mathbf{r}(u, v)$, the curvature tensor $\mathbf{C}$ satisfies $\mathbf{C}\mathbf{r}_u = -\mathbf{n}_u$ and $\mathbf{C}\mathbf{r}_v = -\mathbf{n}_v$, where $\mathbf{r}_u, \mathbf{r}_v$ are the tangent vectors of S and $\mathbf{n}_u, \mathbf{n}_v$ the partial derivatives of unit normal $\mathbf{n}$.

Since we deal with 3D self-supporting surfaces represented by triangle meshes, we need to figure out the discretization of $\mathbf{C}$. The key idea is to con-

Figure 5.1: (a) Input, (b) Principal directions, (c) Smoothed directions (d) Free and fixed faces, (e) Mixed field, (f) PQ mesh. Algorithmic pipeline. The input is a 3D self-supporting surface represented by a triangle mesh. First, we compute principal curvatures and principal directions for each triangular face. Then, we group the faces with similar principal curvatures, which are called free faces (teal). We compute a smooth cross field on the free faces and generate a pair of conjugate directions by stretching the principal directions. Next, we parameterize the triangle mesh using the mixed principal direction field and conjugate direction field. Finally, we extract the planar quadrilateral mesh from the parameterization.

sider $\mathbf{r}_u, \mathbf{r}_v$ as the movement of observation points on the surface. Consider a triangular face $\triangle ABC$ and its neighbors $\triangle CBD$, $\triangle ACF$ and $\triangle BAE$. Denote by $\mathbf{n}, \mathbf{n}_i, i = 1, 2, 3$, the normals of $\triangle ABC$, $\triangle CBD$, $\triangle ACF$ and $\triangle AEB$, and $\delta \mathbf{n}_i$ the derivatives of normals.

Figure 5.2: The difference between the observation point movement strategy used in Rusinkiewicz's method [96] and our method. (a) In [96], the derivatives are $\delta \mathbf{r}_1 = \overrightarrow{CB}$, $\delta \mathbf{r}_2 = \overrightarrow{AC}$ and $\delta \mathbf{r}_3 = \overrightarrow{BA}$, which are based on mesh vertices. (b) In our method, we define $\delta \mathbf{r}_i = \overrightarrow{OO_i}$, where O and O_i are the centers of the circumscribed circles of $\triangle ABC$ and its neighbors.

Observe that per-face normal is the cross product of two edge vectors, and per-vertex normal is computed as the weighted average of per-face nor-

Algorithm 1: Constructing 3D self-supporting surfaces with planar quadrilateral elements

```

1 Require: A self-supporting surface in triangular mesh  $M = (V, E, F)$ 
   and the threshold  $\varepsilon$  of principal curvature difference
2 Result: A planar quad mesh representing the surface
3 Compute the principal curvatures and directions using Algorithm 2
4 Smooth the principal directions to reduce singularities
5 for each triangular face  $f \in F$  do
6   if  $\kappa_1 \kappa_2 < 0$  then
7     | Mark  $f$  “fixed”
8   end
9   else
10    | Compute  $\lambda = \frac{\min\{|\kappa_1|, |\kappa_2|\}}{\max\{|\kappa_1|, |\kappa_2|\}}$ 
11    | if  $\lambda < \varepsilon$  then
12      | | Mark  $f$  “free”
13      | | Compute stretching matrix  $\mathbf{S}$  using Equation (5.9)
14      | | Stretch the principal directions with  $\mathbf{S}$ 
15    | end
16    | else
17      | | Mark  $f$  “fixed”
18    | end
19  end
20 end
21 Apply the MIQ solver to the mixed cross and conjugate direction fields
22 Extract quad mesh from parameterization

```

mals. Therefore, per-face normals are usually more accurate than per-vertex normals on triangle meshes. Our movement strategy is to use the centers O and O_i , $i = 1, 2, 3$, of the circumscribed circles of $\triangle ABC$ and its neighbors $\triangle BCD$, $\triangle ACF$ and $\triangle BAE$. We compute the derivatives as

$$\delta \mathbf{r}_i = \overrightarrow{OO_i}, \quad (5.1)$$

and

$$\delta \mathbf{n}_i = \mathbf{n} - \mathbf{n}_i, \quad (5.2)$$

for $i = 1, 2, 3$. Since the curvature tensor $\mathbf{C}$ satisfies $\mathbf{C}\delta\mathbf{r}_i = -\delta\mathbf{n}_i$, $i = 1, 2, 3$, we form an over-constrained linear system

$$\begin{bmatrix} \delta\mathbf{r}_1 \cdot \mathbf{e}_1 & \delta\mathbf{r}_1 \cdot \mathbf{e}_2 & 0 \\ 0 & \delta\mathbf{r}_1 \cdot \mathbf{e}_1 & \delta\mathbf{r}_1 \cdot \mathbf{e}_2 \\ \delta\mathbf{r}_2 \cdot \mathbf{e}_1 & \delta\mathbf{r}_2 \cdot \mathbf{e}_2 & 0 \\ 0 & \delta\mathbf{r}_2 \cdot \mathbf{e}_1 & \delta\mathbf{r}_2 \cdot \mathbf{e}_2 \\ \delta\mathbf{r}_3 \cdot \mathbf{e}_1 & \delta\mathbf{r}_3 \cdot \mathbf{e}_2 & 0 \\ 0 & \delta\mathbf{r}_3 \cdot \mathbf{e}_1 & \delta\mathbf{r}_3 \cdot \mathbf{e}_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \delta\mathbf{n}_1 \cdot \mathbf{e}_1 \\ \delta\mathbf{n}_1 \cdot \mathbf{e}_2 \\ \delta\mathbf{n}_2 \cdot \mathbf{e}_1 \\ \delta\mathbf{n}_2 \cdot \mathbf{e}_2 \\ \delta\mathbf{n}_3 \cdot \mathbf{e}_1 \\ \delta\mathbf{n}_3 \cdot \mathbf{e}_2 \end{bmatrix} \quad (5.3)$$

where $\mathbf{e}_1$ and $\mathbf{e}_2$ are the axes of the local coordinate system on $\triangle ABC$. Then we compute x_i , $i = 1, 2, 3$, by solving a linear least square. After that, we can form the curvature tensor $\mathbf{C}$

$$\mathbf{C} = \begin{bmatrix} x_1 & x_2 \\ x_2 & x_3 \end{bmatrix}$$

since $\mathbf{C}$ is symmetric. The eigenvalues κ_i and eigenvectors $\mathbf{d}_i$, $i = 1, 2$, of $\mathbf{C}$ are the principal curvatures and directions of face $\triangle ABC$. Algorithm 2 shows the pseudocode of computing principal curvatures and directions on triangle meshes.

5.2.3 Smoothing Principal Directions

The computed principal directions are usually not smooth due to discretization. As a result, the principal direction fields contain many undesired singularities, which compromises the parameterization quality. To reduce singularities, we apply [ma2021] to locally adjust singularities. Specifically, the method moves singularities based on their indices: the singularities of the

Algorithm 2: Computing principal curvatures and directions on triangle meshes

```

1 Data: A self-supporting surface in triangular mesh  $M = (V, E, F)$ 
2 Result: Principal curvatures and directions on each triangular face
    $f \in F$ 
3 for each face  $f \in F$  do
4   | Compute the circumscribed circle center of  $f$ 
5   | Compute the per-face normal using cross product
6   | Construct a local coordinate system of  $f$ 
7 end
8 for each face  $f \in F$  do
9   | for  $i = 1$  to 3 do
10  | | Compute  $\delta \mathbf{r}_i$  and  $\delta \mathbf{n}_i$  ▷ Equations (5.1) and (5.2)
11  | end
12  | Solve Equation 5.3 using linear least square
13  | Form matrix  $\mathbf{C}$  Compute the eigenvalues and eigenvectors of  $\mathbf{C}$ 
14 end

```

Figure 5.3: Smoothing principal directions. (a) There are many singularities in the principal direction field. (b) The singularities are reduced significantly after smoothing. The red (resp. blue) spheres are singularities with index $\frac{1}{4}$ (resp. $-\frac{1}{4}$).

same index propel and opposite index attract. When a pair of opposite-index singularities meet, they cancel out each other, reducing the number of singularities by two. Once all singularities stop moving, we obtain a smoothed principal direction field with fewer singularities than the original one. Figure 5.3 shows the principal curvature direction field before and after optimiza-

tion. We observe that the number of singularities can be significantly reduced using this local smoothing method.

5.2.4 Fixed Chart and Free Chart Distinguish

In self-supporting surfaces, there exists some parts similar to sphere (See Figure 5.4(a)). It is difficult to calculate principal directions for these parts. The resulting direction field we produced in Section 5.2.2 are not accurate for these sphere-like parts. However, based on Liu et al.’s research [70, 69], not only principal direction fields are used for PQ mesh, but also conjugate direction fields can produce PQ mesh. Thus we need to manipulate these cross vectors into conjugate directions.

Regions with large principal curvature difference can be ignored, since we can directly use the computed principal directions, which are also conjugate. We need to pay more attention to the regions with small principal curvature difference (i.e., sphere-like regions), on which principal directions do not capture the local geometry features. In our implementation, we set the curvature difference threshold $\varepsilon = \sqrt{3}$. A face is classified as free face when its principal curvatures satisfy $\frac{\min\{|\kappa_1|, |\kappa_2|\}}{\max\{|\kappa_1|, |\kappa_2|\}} < \varepsilon$. See Figure 5.4 for two examples on classification of fixed and free faces.

5.2.5 Stretch Matrix Construction

To generate conjugate directions for free faces, we compute a matrix $\mathbf{S}$, which stretches the principal directions.

Let $\mathbf{u}$ and $\mathbf{v}$ be the tangent vectors at point $\mathbf{p}$ on the surface. If $\mathbf{p}$, $\mathbf{p} + s\mathbf{u}$, $\mathbf{p} + t\mathbf{v}$ and $\mathbf{p} + s\mathbf{u} + t\mathbf{v}$ form a planar quadrilateral for some scalars s and t ,

Figure 5.4: Distinguishing fixed and free faces. (a) We compute the difference λ between principal curvatures κ_1 and κ_2 on each triangular face, $\lambda = \frac{\min\{|\kappa_1|, |\kappa_2|\}}{\max\{|\kappa_1|, |\kappa_2|\}}$. (b) Faces with $\lambda \geq \varepsilon$ are marked as “fixed” (brown) and the rest are “free” (teal) faces.

they satisfy the following constraint

$$(s\mathbf{u} + t\mathbf{v})^T \mathbf{C}(s\mathbf{u} + t\mathbf{v}) = s^2 \mathbf{u}^T \mathbf{C} \mathbf{u} + t^2 \mathbf{v}^T \mathbf{C} \mathbf{v},$$

where $\mathbf{C}$ is the curvature tensor at $\mathbf{p}$. This implies the tangent directions $\mathbf{u}$ and $\mathbf{v}$ satisfy

$$\mathbf{u}^T \mathbf{C} \mathbf{v} = 0,$$

which are conjugate directions. If the edges of each quadrilateral face follow the conjugate directions, its vertices are co-planar. Therefore, the conjugate directions are often used in constructing planar quadrilateral meshes [70, 121, 69].

The surface’s curvature tensor $\mathbf{C}$ is

$$\mathbf{C} = L\mathbf{r}_u \otimes \mathbf{r}_u + M(\mathbf{r}_u \otimes \mathbf{r}_v + \mathbf{r}_v \otimes \mathbf{r}_u) + N\mathbf{r}_v \otimes \mathbf{r}_v,$$

where L , M , and N are the coefficients of the second fundamental form, and $\otimes$ is tensor product. It is well known that the principal directions $\mathbf{d}_1$ and $\mathbf{d}_2$ are perpendicular to each other and they satisfy $\mathbf{d}_1^T \mathbf{C} \mathbf{d}_2 = 0$. Also, the principal curvatures κ_1 and κ_2 are the eigenvalues of $\mathbf{C}$, i.e., $\mathbf{C} \mathbf{d}_i = \kappa_i \mathbf{d}_i$, $i = 1, 2$.

Consider a unit vector $\mathbf{v}$, $\mathbf{v} \otimes \mathbf{v}$ is a second order tensor, for an arbitrary vector $\mathbf{r}$, we have $(\mathbf{v} \otimes \mathbf{v}) \cdot \mathbf{r} = (\mathbf{v} \cdot \mathbf{r}) \mathbf{v}$, if $\mathbf{r}$ and $\mathbf{v}$ share the same direction, the result is equal to $\mathbf{r}$, if $\mathbf{r}$ and $\mathbf{v}$ are perpendicular to each other, the result is $\mathbf{0}$. Therefore, this operation $\mathbf{v} \otimes \mathbf{v}$ extracts $\mathbf{r}$'s component along $\mathbf{v}$'s direction. Now consider $\mathbf{I} - \mathbf{v} \otimes \mathbf{v}$, where $\mathbf{I}$ is the identity matrix, apply to $\mathbf{r}$ and yields $\mathbf{r} - (\mathbf{v} \cdot \mathbf{r}) \mathbf{v}$, it maintains $\mathbf{r}$'s component perpendicular to $\mathbf{v}$. Stretch s times of $\mathbf{r}$ along the direction of $\mathbf{v}$ is equal to stretch $\mathbf{r}$'s component along $\mathbf{v}$'s direction and maintain $\mathbf{r}$'s component perpendicular to $\mathbf{v}$'s direction. The combination is $(\mathbf{I} - \mathbf{v} \otimes \mathbf{v}) \cdot \mathbf{r} + s(\mathbf{v} \otimes \mathbf{v}) \cdot \mathbf{r} = (\mathbf{I} + (s - 1)\mathbf{v} \otimes \mathbf{v}) \cdot \mathbf{r}$. Therefore, the stretch matrix is $\mathbf{I} + (s - 1)\mathbf{v} \otimes \mathbf{v}$. Now we stretch the cross vector on free faces along the principal directions $\mathbf{d}_i$ by $\frac{1}{\sqrt{\kappa_i}}$ times. Stretch matrix $\mathbf{S}_i$ for each direction $\mathbf{d}_i$ is defined as

$$\mathbf{S}_i = \mathbf{I} + \left(\frac{1}{\sqrt{\kappa_i}} - 1 \right) \frac{1}{\mathbf{d}_i \cdot \mathbf{d}_i} \mathbf{d}_i \otimes \mathbf{d}_i, \quad i = 1, 2, \quad (5.4)$$

Since $\mathbf{d}_1$ and $\mathbf{d}_2$ are perpendicular to each other, therefore, multiplying $\mathbf{S}_1$ and $\mathbf{S}_2$ will not influence each direction.

Multiplying $\mathbf{S}_1$ and $\mathbf{S}_2$ yields the stretch matrix $\mathbf{S}$

$$\mathbf{S} = \mathbf{S}_1 \mathbf{S}_2 = \mathbf{I} + \sum_{i=1}^2 \left(\frac{1}{\sqrt{\kappa_i}} - 1 \right) \frac{1}{\mathbf{d}_i \cdot \mathbf{d}_i} \mathbf{d}_i \otimes \mathbf{d}_i, \quad (5.5)$$

since $(\mathbf{d}_1 \otimes \mathbf{d}_1)\mathbf{C}(\mathbf{d}_2 \otimes \mathbf{d}_2) = 0$. Also notice that

$$\frac{1}{\mathbf{d}_1 \cdot \mathbf{d}_1} \mathbf{d}_1 \otimes \mathbf{d}_1 + \frac{1}{\mathbf{d}_2 \cdot \mathbf{d}_2} \mathbf{d}_2 \otimes \mathbf{d}_2 + \mathbf{n} \otimes \mathbf{n} = \mathbf{I},$$

where $\mathbf{n}$ is unit normal. We can re-write $\mathbf{S}$ as

$$\mathbf{S} = \sum_{i=1}^2 \frac{1}{\sqrt{\kappa_i}} \frac{1}{\mathbf{d}_i \cdot \mathbf{d}_i} \mathbf{d}_i \otimes \mathbf{d}_i + \mathbf{n} \otimes \mathbf{n}. \quad (5.6)$$

Matrix $\mathbf{S}$ is symmetric. Now we show that the matrix $\mathbf{S}$ turns two perpendicular tangent directions into a pair of conjugate directions. Since the principal curvatures κ_i and principal directions $\mathbf{d}_i$ satisfy the following equations

$$\begin{cases} (\mathbf{d}_1 \otimes \mathbf{d}_1)\mathbf{C}(\mathbf{d}_1 \otimes \mathbf{d}_1) &= \kappa_1(\mathbf{d}_1 \cdot \mathbf{d}_1)\mathbf{d}_1 \otimes \mathbf{d}_1, \\ (\mathbf{d}_2 \otimes \mathbf{d}_2)\mathbf{C}(\mathbf{d}_2 \otimes \mathbf{d}_2) &= \kappa_2(\mathbf{d}_2 \cdot \mathbf{d}_2)\mathbf{d}_2 \otimes \mathbf{d}_2, \\ (\mathbf{d}_1 \otimes \mathbf{d}_1)\mathbf{C}(\mathbf{d}_2 \otimes \mathbf{d}_2) &= 0, \\ (\mathbf{d}_2 \otimes \mathbf{d}_2)\mathbf{C}(\mathbf{d}_1 \otimes \mathbf{d}_1) &= 0, \end{cases}$$

we obtain

$$\mathbf{S}\mathbf{C}\mathbf{S} = \sum_{i=1}^2 \frac{1}{\mathbf{d}_i \cdot \mathbf{d}_i} \mathbf{d}_i \otimes \mathbf{d}_i = \mathbf{I} - \mathbf{n} \otimes \mathbf{n}.$$

Given two arbitrary perpendicular tangent vectors $\mathbf{u}$ and $\mathbf{v}$, we can easily verify that the stretched vectors $\mathbf{S}\mathbf{u}$ and $\mathbf{S}\mathbf{v}$ are conjugate due to

$$(\mathbf{S}\mathbf{u})^T \mathbf{C}(\mathbf{S}\mathbf{v}) = \mathbf{u}^T (\mathbf{S}^T \mathbf{C} \mathbf{S}) \mathbf{v} = \mathbf{u}^T \mathbf{v} - \mathbf{u}^T (\mathbf{n} \otimes \mathbf{n}) \mathbf{v} = 0.$$

See Figure 5.5 for an illustration. Therefore, the stretch matrix $\mathbf{S}$ turns perpendicular tangent directions into conjugate directions. In our implementation, we stretch the cross vectors on free faces to generate conjugate

directions.

Figure 5.5: Consider a circle (blue) with two perpendicular diameters. Stretching the circle with matrix $\mathbf{S}$ yields an ellipse and the two perpendicular diameters become conjugate.

Figure 5.6: Computing conjugate directions on free faces. (a) The smoothed principal directions. (b) The conjugate directions.

As a local operation, computing stretch matrix $\mathbf{S}$ is computationally efficient. Figure 5.6 shows examples of conjugate directions generated from stretching principal directions.

5.2.6 Parameterization & Generating PQ Meshes

Let $(\mathbf{u}, \mathbf{v}, -\mathbf{u}, -\mathbf{v})$ be the directional field, consisting of principal directions on fixed faces and conjugate directions on free faces, which is used as a guide to parameterize the triangle mesh and extract PQ elements.

We need to cut the input mesh into a topological disk so that the singularities of the directional field is on the disk boundary. To do that, we take the singularities as seeds and apply Dijkstra’s algorithm to compute a Voronoi diagram on the input mesh. The dual graph of the Voronoi diagram is a triangulation with singularities as vertices. We then segment the triangle mesh into patches along the edges of Delaunay triangulation. Then, starting with an arbitrary patch, we iteratively glue the patches together by traversing them in a breadth-first-search order. Since each patch is used only once, the resulting shape is a topological disk with singularities on its boundary.

To compute the parameterization, we adapt the global parametrization used in Liu et al.’s work [69], which improves the global parametrization method in [20] and [35]. Denote by (s, t) the parameters assigned to each vertex of the input triangle mesh. We minimize the following energy function

$$E_p = \sum_{f_i \in F} A_i \left[\left(\frac{\nabla s_i}{\|\nabla s_i\|} \cdot \mathbf{u}_i^\perp \right)^2 + \left(\frac{\nabla t_i}{\|\nabla t_i\|} \cdot \mathbf{v}_i^\perp \right)^2 \right], \quad (5.7)$$

where A_i is the area of face f_i and $\perp$ denotes 90 degrees rotation about the normal. To make the parameterization seamless, for each edge on the cut graph, we require the projections of parameter gradients on the two adjacent faces match, leading to integer translations on the parameters.

Equation (5.7) is a nonlinear energy function with integer variables introduced at the topology cut and singularities [20]. Similar to Liu et al.’s method [69], we solve the mixed integer programming using a simple round-off heuristic.

tic. We perform three operations in each iteration: First, we relax the integer constraints and minimize the function using the L-BFGS method [67]; second, pick on integer variable, round it off to its nearest integer and then fix it; repeat the first two operations until all integer variables are fixed.

5.2.7 Airy Stress Function

In this section, comparisons between our method and Vouga et al.’s work [112] is introduced. Vouga et al.’s method [112] also constructs self-supporting surface and generates PQ mesh of the self-supporting surface.

On plains, since $\mathbf{n}$ is constant, this integral is 0. However, in 2D manifold, if the surface has distortion, Airy stress function can’t guarantee the balance of stress between each region.

If Airy stress function is applied to construct self-supporting surface, the surface $\mathbf{r}(u, v)$ and its Airy stress function $\varphi(u, v)$ satisfy

$$-\oint_{\partial\Omega} \nabla\varphi \times d\mathbf{n} + \iint_{\Omega} dA\mathbf{c} = 0$$

$\mathbf{c}$ is constant of scale and direction of gravity. The number of constraints(3 constraints for 3 directions) is more than number of variables(2), thus can’t guarantee the existence of solution.

As a result, in construction of 3D self-supporting surface, both Airy stress function [112] and TNA [16] simplify the 3 degrees of freedom to 1 variable. Thus, lead to too many constraints, can’t guarantee existence of exact solution.

Our method is better than Vouga et al.’s method from two aspects. First, [112] didn’t take sphere-like part into account, they compute principal directions, however, for sphere-like parts, principal directions are not available.

Second, Vouga et al.’s method apply a global optimization method to satisfy planarity. The planarity energy $\sum_f (2\pi - \theta_f)^2$ is taken, the sum runs over faces and θ_f is the sum of the interior angles of face f . This energy is adding to the objective functions of the position update. As a result, their method is computational expensive.

5.3 Finite Element Method Analysis for PQ Meshes

Ma et al.’s research [74] proposed three discrete measures to evaluate the stability of the resulting triangular self-supporting surfaces, i.e., normal displacement θ , stress tensor error σ_S , and differential stress σ_D . Specifically, differential stress is the difference between the greatest and the least compressive stress experienced by the architecture, which characterizes homogeneity of stress. Stress tensor error measures the difference between the actual stress and the assembly stress. A large stress tensor error means stress distribution is far from the ideal case, leading to stress concentration, which weakens the building. Accumulated stress tensor error leads to vertex displacement, which in turns changes the orientation of faces. A large normal displacement results in shear stress and bending moment that may break the surface. A surface is self-supporting only if all three measures are small.

To evaluate the physical properties of the self-supporting surface, one could simply apply the triangle mesh-based FEM [74]. However, it requires a re-triangulation of the quad mesh, which may introduce unnecessary discretization error. In this section, we present the finite element method that can work directly for PQ meshes.

On a parametrized surface, Finite Element Analysis (FEA) method can

achieve a higher accuracy because parametric splines can be used to approximate the real solution with a higher degree (e.g. 3 with cubic B spline). On one side, splines can be used to represent a smooth surface, on the other side, the solution of mechanical problems are also represented by splines. The main work of FEA is transforming the PDE into a linear equation whose unknowns are composed of the coefficients of the splines. The displacement is based on the calculation of the derivatives of the splines shown as follows.

5.3.1 Accuracy Measures

The normal displacement of mesh M is

$$\theta(M) = \sqrt{\sum_{f \in F} \theta^2(f) A(f) / \sum_{f \in F} A(f)}.$$

Consider a quadrilateral face $f = (\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3) \in F$, denote by $\mathbf{n}(f)$ the outward normal of f and $\mathbf{x}_i$, $i = 0, 1, 2, 3$, the vertex displacement. The normal displacement of face f is defined as

$$\begin{aligned} \theta(f) &= \langle \mathbf{n}(f), ((\overrightarrow{\mathbf{x}_1 \mathbf{x}_0} + \overrightarrow{\mathbf{p}_1 \mathbf{p}_0}) \times (\overrightarrow{\mathbf{x}_2 \mathbf{x}_1} + \overrightarrow{\mathbf{p}_2 \mathbf{p}_1})) + ((\overrightarrow{\mathbf{x}_3 \mathbf{x}_2} + \overrightarrow{\mathbf{p}_3 \mathbf{p}_2}) \times (\overrightarrow{\mathbf{x}_2 \mathbf{x}_1} + \overrightarrow{\mathbf{p}_2 \mathbf{p}_1})) \rangle, \\ &= \frac{\left\| \sum_{i=0}^3 (\mathbf{x}_{[i+2]} - \mathbf{x}_{[i+1]}) \times \mathbf{x}_i \times \mathbf{n}(f) \right\|}{2A(f)} + O(\|\mathbf{x}\|^2). \end{aligned}$$

To ensure stress follows the ideal distribution, an isotropic assembly stress $\bar{\sigma}$ for each face is specified. The stress $\bar{\sigma}$ for $\square \mathbf{p}_0 \mathbf{p}_1 \mathbf{p}_2 \mathbf{p}_3$ is determined by the distance between the face's mass center and the zero potential surface

$$\bar{\sigma}(\square \mathbf{p}_0 \mathbf{p}_1 \mathbf{p}_2 \mathbf{p}_3) = \rho g \left(z_0 - \frac{z_{\mathbf{p}_0} + z_{\mathbf{p}_1} + z_{\mathbf{p}_2} + z_{\mathbf{p}_3}}{4} \right),$$

where g is the acceleration of gravity and ρ is the density.

The differential stress σ_D characterizes the difference between the two principal stresses

$$\sigma_D(f) = \frac{\sigma_1(f) - \sigma_2(f)}{\sigma_1(f) + \sigma_2(f)}.$$

The stress tensor error characterizes how a material tends to expand in directions perpendicular to the direction of compression. For a discrete face $f \in F$, the stress tensor error $\sigma_S(f)$ is

$$\sigma_S(f) = \frac{\sigma_1(f) + \sigma_2(f) - 2\bar{\sigma}}{2\bar{\sigma}},$$

where $\bar{\sigma} = \frac{\sum_{i=0}^3 \sigma(\mathbf{x}_i)}{4}$.

Displacement $\mathbf{x}$ is important in calculating normal displacement θ , stress tensor error σ_S , and differential stress σ_D . We present how to derive $\mathbf{x}$ for the finite element method to work directly for PQ meshes.

5.3.2 Force Balance on Bending Thin Shells

The derivative on curved surface is

$$\frac{\partial \mathbf{r}^i}{\partial u^j} = - \sum_{k=1}^2 \Gamma_{jk}^i \mathbf{r}^k + \sum_{k=1}^2 g^{ik} L_{jk} \mathbf{n}$$

The stress tensor inside the shell is

$$\mathbf{T} = \sum_{ij=1}^2 t_{ij} \mathbf{r}^i \otimes \mathbf{r}^j = \sum_{ij=1}^2 t^{ij} \mathbf{r}_i \otimes \mathbf{r}_j$$

Then the tension to region Ω (parameter field D) is

$$\begin{aligned}\mathbf{f}_\Omega &= \oint_{\partial\Omega} \mathbf{T} \cdot (\mathrm{d}\mathbf{l} \times \mathbf{n}) \\ &= \oint_{\partial D} \sum_{ijk=1}^2 t^{ij} (\mathbf{r}_i \otimes \mathbf{r}_j) \cdot (\mathbf{r}_k \times \mathbf{n}) \mathrm{d}u^k\end{aligned}$$

Particularly

$$\mathbf{r}_j \cdot (\mathbf{r}_k \times \mathbf{n}) = (\mathbf{r}_j \times \mathbf{r}_k) \cdot \mathbf{n} = \begin{cases} \sqrt{g_{11}g_{22} - g_{12}g_{12}} & j=1, k=2 \\ -\sqrt{g_{11}g_{22} - g_{12}g_{12}} & j=2, k=1 \\ 0 & j=k \end{cases}$$

According to Green's Equation

$$\begin{aligned}\mathbf{f}_\Omega &= \oint_{\partial D} \sqrt{g_{11}g_{22} - g_{12}g_{12}} \sum_{i=1}^2 (t^{i1} \mathrm{d}u^2 - t^{i2} \mathrm{d}u^1) \mathbf{r}_i \\ &= \iint_D \sum_{ij=1}^2 \frac{\partial(t^{ij} \sqrt{g_{11}g_{22} - g_{12}g_{12}} \mathbf{r}_i)}{\partial u^j} \mathrm{d}u^1 \mathrm{d}u^2 \\ &= \iint_\Omega \sum_{ij=1}^2 \frac{1}{\sqrt{g_{11}g_{22} - g_{12}g_{12}}} \frac{\partial(t^{ij} \sqrt{g_{11}g_{22} - g_{12}g_{12}} \mathbf{r}_i)}{\partial u^j} \mathrm{d}A\end{aligned}$$

Therefore, this is the transformation from linear stress tensor to surface stress tensor, can be simplified to

$$\begin{aligned}& \sum_{ij=1}^2 \frac{1}{\sqrt{g_{11}g_{22} - g_{12}g_{12}}} \frac{\partial(t^{ij} \sqrt{g_{11}g_{22} - g_{12}g_{12}} \mathbf{r}_i)}{\partial u^j} \\ &= \sum_{ij=1}^2 \frac{\partial(t^{ij} \mathbf{r}_i)}{\partial u^j} + \sum_{ij=1}^2 \frac{1}{\sqrt{g_{11}g_{22} - g_{12}g_{12}}} t^{ij} \frac{\partial \sqrt{g_{11}g_{22} - g_{12}g_{12}}}{\partial u^j} \mathbf{r}_i\end{aligned}$$

Specially, the second term can be simplified to

$$\begin{aligned}
& \frac{1}{\sqrt{g_{11}g_{22} - g_{12}g_{12}}} \frac{\partial \sqrt{g_{11}g_{22} - g_{12}g_{12}}}{\partial u^j} = \frac{\partial \ln \sqrt{g_{11}g_{22} - g_{12}g_{12}}}{\partial u^j} \\
&= \frac{1}{2} \frac{\partial \ln(g_{11}g_{22} - g_{12}g_{12})}{\partial u^j} = \frac{1}{2} \frac{1}{g_{11}g_{22} - g_{12}g_{12}} \frac{\partial(g_{11}g_{22} - g_{12}g_{12})}{\partial u^j} \\
&= \frac{1}{2} \frac{1}{g_{11}g_{22} - g_{12}g_{12}} (2\Gamma_{1j1}g_{22} + 2\Gamma_{2j2}g_{11} - 2(\Gamma_{1j2} + \Gamma_{2j1})g_{12}) \\
&= \Gamma_{1j1}g^{11} + \Gamma_{2j2}g^{22} + \Gamma_{1j2}g^{21} + \Gamma_{2j1}g^{12} \\
&= \Gamma_{1j}^1 + \Gamma_{2j}^2 = \sum_{k=1}^2 \mathbf{r}_{jk} \cdot \mathbf{r}^k
\end{aligned}$$

This derive applies

$$\begin{pmatrix} g^{11} & g^{12} \\ g^{21} & g^{22} \end{pmatrix} = \frac{1}{g_{11}g_{22} - g_{12}g_{12}} \begin{pmatrix} g^{22} & -g^{12} \\ -g^{21} & g^{11} \end{pmatrix}$$

Thus, surface stress tensor can be simplified to

$$\begin{aligned}
& \sum_{i,j,k=1}^2 (\mathbf{r}^k \cdot \mathbf{r}_j) \frac{\partial(t^{ij}\mathbf{r}_i)}{\partial u^k} + \sum_{i,j,k=1}^2 t^{ij} \left(\mathbf{r}^k \cdot \frac{\mathbf{r}_j}{\partial u^k} \right) \mathbf{r}_i \\
&= \sum_{k=1}^2 \mathbf{r}^k \cdot \frac{\partial(t^{ij}\mathbf{r}_j \otimes \mathbf{r}_i)}{\partial u^k} = \sum_{k=1}^2 \mathbf{r}^k \cdot \frac{\partial \mathbf{T}}{\partial u^k} = \nabla \cdot \mathbf{T}
\end{aligned}$$

This is the complete transformation of internal stress. To reach the force balance, external force need to equal to area force.

For flat thin shell, stress tensor $\mathbf{T}$ and strain tensor $\mathbf{S}$ satisfy

$$\mathbf{S} = \frac{2(1+\mu)}{E} \mathbf{T} - \frac{2\mu}{E} \text{tr}(\mathbf{T}) \mathbf{I}$$

$\mathbf{I}$ is 2D identity matrix, its trace is 2, calculate trace on two sides

$$\text{tr}(\mathbf{S}) = \frac{2(1+\mu)}{E} \text{tr}(\mathbf{T}) - 2 \frac{2\mu}{E} \text{tr}(\mathbf{T})$$

Therefore

$$\text{tr}(\mathbf{T}) = \frac{E}{2(1-\mu)} \text{tr}(\mathbf{S})$$

Hooke's Law is

$$\mathbf{S} = \frac{2(1+\mu)}{E} \mathbf{T} - \frac{\mu}{1-\mu} \text{tr}(\mathbf{S}) \mathbf{I}$$

thus, stress tensor can be expressed by strain tensor

$$\mathbf{T} = \frac{E}{2(1+\mu)} \mathbf{S} + \frac{\mu E}{2(1+\mu)(1-\mu)} \text{tr}(\mathbf{S})$$

5.3.3 Thin Shell Problem on Parametric Surface

Assume the function of parametric surface is $\mathbf{r}(u^1, u^2)$, displacement of each vertex is $\mathbf{x}(u^1, u^2)$, derivative function of displacement is

$$\mathbf{M} = \nabla \otimes \mathbf{x} = \sum_{i=1}^2 \mathbf{r}^i \otimes \frac{\partial \mathbf{x}}{\partial u^i}$$

$\mathbf{r}^i$ is dual base vector, satisfies

$$\mathbf{r}^i = \sum_{j=1}^2 g^{ij} \mathbf{r}_j$$

Strain tensor can be calculated from displacement derivative function

$$\mathbf{S} = \frac{1}{2} (\nabla \otimes \mathbf{x} + (\nabla \otimes \mathbf{x})^T) = \frac{1}{2} \sum_{i=1}^2 \left(\mathbf{r}^i \otimes \frac{\partial \mathbf{x}}{\partial u^i} + \frac{\partial \mathbf{x}}{\partial u^i} \otimes \mathbf{r}^i \right)$$

The trace of strain tensor is

$$\text{tr}(\mathbf{S}) = \frac{1}{2} \sum_{i=1}^2 \left(\mathbf{r}^i \cdot \frac{\partial \mathbf{x}}{\partial u^i} + \frac{\partial \mathbf{x}}{\partial u^i} \cdot \mathbf{r}^i \right) = \sum_{i=1}^2 \mathbf{r}^i \cdot \frac{\partial \mathbf{x}}{\partial u^i}$$

According to Hooke's Law, stress is

$$\begin{aligned} \mathbf{T} &= \frac{E}{2(1+\mu)} \frac{1}{2} \sum_{i=1}^2 \left(\mathbf{r}^i \otimes \frac{\partial \mathbf{x}}{\partial u^i} + \frac{\partial \mathbf{x}}{\partial u^i} \otimes \mathbf{r}^i \right) \\ &+ \frac{\mu E}{2(1+\mu)(1-\mu)} \sum_{i=1}^2 \left(\mathbf{r}^i \cdot \frac{\partial \mathbf{x}}{\partial u^i} \right) (\mathbf{I} - \mathbf{n} \otimes \mathbf{n}) \end{aligned}$$

$\mathbf{n}$ is normal of the surface.

5.3.4 Balance between Stress Tensor and Gravity

For a small region on curved surface, the balance condition is

$$\nabla \cdot \mathbf{T} + \rho \mathbf{g} = 0$$

ρ is material area density, $\mathbf{g}$ is gravity Acceleration. Now calculate $\nabla \cdot \mathbf{T}$

$$\begin{aligned} \nabla \cdot \left(\sum_{i=1}^2 \mathbf{r}^i \otimes \frac{\partial \mathbf{x}}{\partial u^i} \right) &= \sum_{i,j=1}^2 \mathbf{r}_j \cdot \frac{\partial}{\partial u^j} \left(\mathbf{r}^i \otimes \frac{\partial \mathbf{x}}{\partial u^i} \right) \\ &= \sum_{i,j=1}^2 \left(\mathbf{r}_j \cdot \frac{\partial \mathbf{r}^i}{\partial u^j} \right) \frac{\partial \mathbf{x}}{\partial u^i} + \sum_{i,j=1}^2 (\mathbf{r}_j \cdot \mathbf{r}^i) \frac{\partial}{\partial u^j} \frac{\partial \mathbf{x}}{\partial u^i} \\ &= \sum_{i,j,k=1}^2 (\Gamma_{jk}^i \mathbf{r}^j \cdot \mathbf{r}^k) \frac{\partial \mathbf{x}}{\partial u^i} + \sum_{i,j=1}^2 (\mathbf{r}_j \cdot \mathbf{r}^i) \frac{\partial}{\partial u^j} \frac{\partial \mathbf{x}}{\partial u^i} \\ &= \sum_{i,j,k=1}^2 g^{jk} \Gamma_{jk}^i \frac{\partial \mathbf{x}}{\partial u^i} + \sum_{i=1}^2 \frac{\partial}{\partial u^i} \frac{\partial \mathbf{x}}{\partial u^i} \end{aligned}$$

On the parametric surface, the balance function of displacement $\mathbf{x}$ is

$$\begin{aligned}
& \frac{E}{2(1+\mu)} \left(- \sum_{i,j,k=1}^2 g^{jk} \Gamma_{jk}^i \frac{\partial \mathbf{x}}{\partial u^i} + \sum_{i=1}^2 g^{ij} \frac{\partial}{\partial u^j} \frac{\partial \mathbf{x}}{\partial u^i} \right) \\
& + \frac{E}{2(1+\mu)} \sum_{i,j=1}^2 \left(\mathbf{r}^j \cdot \left(\frac{\partial}{\partial u^j} \left(\frac{\partial \mathbf{x}}{\partial u^i} \right) - \sum_{k=1}^2 \Gamma_{ij}^k \frac{\partial \mathbf{x}}{\partial u^k} \right) \right) \mathbf{r}^i \\
& + \frac{E}{2(1+\mu)} \sum_{i,j,k=1}^2 \left(\mathbf{r}^j \cdot \frac{\partial \mathbf{x}}{\partial u^i} \right) g^{ik} L_{jk} \mathbf{n} \\
& + \frac{\mu E}{2(1+\mu)(1-\mu)} \sum_{i,j=1}^2 \left(\frac{\partial}{\partial u^j} \frac{\partial \mathbf{x}}{\partial u^i} \cdot \mathbf{r}^i - \sum_{k=1}^2 \Gamma_{jk}^i \frac{\partial \mathbf{x}}{\partial u^i} \cdot \mathbf{r}^k \right) \mathbf{r}^j \\
& + 2 \frac{\mu E}{2(1+\mu)(1-\mu)} H \mathbf{n} + \rho \mathbf{g} = 0
\end{aligned}$$

This equation has $3n$ variables and $3n$ constraints. Thus, there exist unique solution. Displacement $\mathbf{x}$ is utilized for calculating θ , σ_D and σ_S .

5.4 Experimental Results

We implemented our algorithm in C++ and evaluated it on a laptop with an Intel(R) Core(TM) i7 CPU 2.90 GHz and 8GB memory.

5.4.1 Principal Curvatures and Directions

To calculate the curvature tensor $\mathbf{C}$, our method uses the observation point movement strategy, based on the centers of circumscribed circles. Here we compare our method to Rusinkiewicz's method [96] in terms of accuracy and robustness on a torus model $((R + r \cos v) \cos u, (R + r \cos v) \sin u, r \sin v)^T$, setting $R = \sqrt{10}$ and $r = 1$. The principal curvatures are $\kappa_1 = \cos v / (R + r \cos v)$ and $\kappa_2 = 1/r$, while the principal directions are $\mathbf{d}_1 = (-\sin u, \cos u, 0)^T$ and $\mathbf{d}_2 = (-\sin v \cos u, -\sin v \sin u, \cos v)^T$. We gen-

erated two sets of triangle meshes with different degrees of anisotropy τ and resolutions. The anisotropy degree τ for a triangle f is defined as $\tau(f) = PH/2\sqrt{3}A$, where P , H , A are the half-perimeter, longest edge length and area of f , respectively. $\tau \geq 1$ for all triangles and equality holds for equilateral triangles. The numerical results show that both methods converge with increasing number of resolutions. However, our method is more accurate and robust than theirs, especially for anisotropic meshes. See Figure 5.7. Our method has better accuracy than Rusinkiewicz’s method [96] since we adopt per-face normals in computing the curvature tensor $\mathbf{C}$, while Rusinkiewicz’s method [96] uses per-vertex normals, which are more sensitive to triangulation quality than per-face normals and often result in large accumulation errors on anisotropic meshes.

5.4.2 Evaluation

We evaluate our method and the other approaches in terms of both physical and geometrical properties. Similar to [74], we adopt three physical measures, i.e., normal displacement θ , stress tensor error σ_S , and differential stress σ_D , to evaluate the stability of self-supporting surfaces. We also measure the smoothness of parameter lines and the planarity of the resulting PQ meshes.

We define the smoothness measure by averaging the discrete geodesic curvatures of the parameter lines (see Figure 5.8(b))

$$\phi = \frac{1}{2n} \sum_{i=1}^n (|\pi - \alpha_i| + |\pi - \beta_i|), \quad (5.8)$$

where n is the number of vertices of the quad mesh. Given a PQ mesh with m patches, we define the planarity measure γ by averaging the volume-to-base

Figure 5.7: Comparisons with Rusinkiewicz’s method [96] in terms of accuracy. We generate sequences of isotropic (first two rows) and anisotropic meshes (last two rows) with increasing number of faces. We compute the angle differences $\delta\theta$ between the resulting principal directions and the ground-truths and visualize the errors using heat color map, where the warm (resp. cold) colors are large (resp. small) angles. For each model, we show our results in the top and theirs in the bottom. τ is the anisotropy measure and $|F|$ is the face count.

ratio

$$\gamma = \frac{1}{m} \sum_{i=1}^m \frac{V_i}{\sqrt{A_i^3}}, \quad (5.9)$$

where V_i is the volume of the tetrahedron formed by the four vertices of the i -th quad patch and A_i is the area of the patch. Obviously $\gamma = 0$ for a completely planar quad mesh. Therefore, the lower the value γ , the better the quality of PQ mesh in terms of planarity.

5.4.3 Comparison

We compare our method with the mixed integer quadrangulation method [20] using the aforementioned physical measure and planarity measure. As shown

Figure 5.8: (a) Convergence plots of our method (orange) and Rusinkiewicz’s method [96] (gray). Both methods converge with the increasing number of resolutions. However, our method is more accurate and robust than theirs, especially on anisotropic meshes. (b) and (c) The smoothness measure is the average of the discrete geodesic curvatures of the parameter lines.

in Table 5.2, the MIQ method produces quad mesh with high self-supporting performance than our method. However, our method outperforms MIQ in terms of planarity. This comparison indicates that conjugate direction field is important for constructing self-supporting surfaces with planar quadrilateral elements. Without conjugate direction field, the resulting mesh is far from PQ mesh which is difficult for constructing glass structures.

We compare our method with Ma et al.’s method [74] and Liu et al.’s method [69] using the aforementioned physical and geometrical measures. As shown in Table 5.3, Ma et al.’s method produces the most accurate self-supporting surfaces in terms of the three physical measures, but their results are triangular meshes. Both Liu et al.’s method and ours produce planar

Table 5.2: Evaluation of the physical measures θ , σ_D and σ_S , and planarity measure γ . For each model, we show the MIQ method [20]’s result (first row) and our result (second row). For each quality measure, the lower the value, the higher the quality.

Model	$\theta(\times 10^{-3})$	σ_D	σ_S	$\gamma(\times 10^{-5})$
Eich (Figure 9, (row 1, R))	1.51	0.79%	0.88%	9.64
	4.04	0.91%	1.27%	2.11
Tea (Figure 9, (row 3, L))	2.09	0.46%	0.73%	12.06
	3.38	0.81%	0.79%	2.56
Oct (Figure 9, (row 4, L))	2.41	0.33%	0.53%	2.92
	3.11	0.38%	1.08%	0.45

quadrilateral meshes. To make a fair comparison with their method, we generate PQ meshes with a similar number of quads. We observe that Liu et al.’s results are smoother than ours, since their method minimizes a global smoothness energy. Also, the two methods yield results with comparable planarity. However, our method is more accurate in terms of physical measures. Furthermore, Liu et al.’s method solves a complex non-linear optimization problem, which typically takes 1 to 3 seconds, to compute CDF. In contrast, our method computes CDF using local computation, which takes less than 1 second for all test models. Therefore, our method is more computationally efficient than theirs.

Our method and Liu et al.’s method [69] adopt different strategies to deal with faces with small principal curvature difference. Liu et al.’s method [69] solves a global optimization problem to find two vectors $\mathbf{v}_p, \mathbf{w}_p$ satisfy the following form: $\kappa_{p,1}(\mathbf{v}_p \cdot \mathbf{e}_{p,1})(\mathbf{w}_p \cdot \mathbf{e}_{p,1}) + \kappa_{p,2}(\mathbf{v}_p \cdot \mathbf{e}_{p,2})(\mathbf{w}_p \cdot \mathbf{e}_{p,2}) = 0$ where $\kappa_{p,1}, \kappa_{p,2}$ are the principal curvatures at p , and $\mathbf{e}_{p,1}, \mathbf{e}_{p,2}$ are the corresponding principal directions. Since the principal directions are inaccurate in these sphere-like faces, the errors are accumulated during iterations. To improve

planarity, Liu et al.’s method [69] applies planarity optimization. However, this step causes vertices leave their original position, therefore, reduce the physical performance. On the contrast, our method generates conjugate directions by manipulating the principal directions using the stretch matrix within a single step. We calculate high quality conjugate directions and avoid planarity optimization. Therefore, our result approximates the input self-supporting surface faithfully both in geometrical performance and physical performance.

Figures 5.9 and 5.11 show the results of height and non-height fields, respectively. We visualize the measures using color maps, where warm (resp. cold) colors indicate large (resp. small) error/distortion. We observe that our results are comparable to Ma et al.’s method [74] that produces triangle meshes and better than Liu et al.’s method [69] that also produces PQ meshes.

We also compare with Vouga et al.’s method [112] in a qualitative manner. Their method allows the user to design self-supporting surfaces using a reference surface. To generate PQ meshes, they solve a complicated non-linear optimization that takes equilibrium condition, shape smoothness and planarity into consideration. Our method is based on local computation of conjugate direction field and then construct PQ meshes via mixed integer based parameterization. Our method aims not to change the geometry of the input shape (which is already self-supported), hence does not support reference surface. However, our method (including both CDF computation and global parameterization) is still more efficient than theirs.

Figure 5.9: Non-height fields. Row 1: the triangle meshes generated by Ma et al.'s method [74]; Rows 2 and 3: the PQ meshes produced by our method and Liu et al.'s method [69], respectively.

Figure 5.10: The distribution of ϵ and θ_S produced by Liu et al.'s method [69] and our method.

5.4.4 Discussion

Our method produces PQ meshes with higher physical quality than Liu et al.'s method [69], our parameter lines are not as smooth as theirs due to the local nature of our method. In the future, we will improve the smoothness of our results using a light-weight global optimization of the principal direction field. Specifically, this could be done by finding an optimal rotation to each cross while keeping the stretch matrix $\mathbf{S}$ fixed for all free faces. Since such an optimization is linear, it could be more computationally efficient than the non-linear optimization used in [69].

5.5 Summary

We developed a simple yet effective method for generating 3D self-supporting surfaces with planar quadrilateral elements. Our method takes a triangular discretization of a 3D self-supporting surface as input and generates PQ mesh via constructing a conjugate direction field. In contrast to the existing methods that compute conjugate direction fields using global optimization, our method is a light-weight approach since it computes a local stretch matrix for each triangular face with small principal curvature difference that turns principal directions into conjugate directions. Computational results show that the generated PQ meshes approximate the input shape very well, hereby satisfying the balance constraints. Comparisons with the existing methods demonstrate the effectiveness of our method.

There are a few interesting directions for future work. First, our method requires the input shape self-supporting. From the application point of view, it is highly desired to either construct PQ meshes directly from the user-specified boundary curves or take a reference surface as input and then optimize its geometry to fulfill the physical requirements of being self-supported. To make the method more flexible, we will also take the user-specified soft and/or hard directional constraints into consideration.

Table 5.3: Evaluation of the physical measures θ , σ_D and σ_S , geometric measures ϕ and γ and running time t_{CDF} and t_P . For each model, we show Ma et al. [74]’s result (first row), our result (second row) and Liu et al. [68]’s result (third row). For each quality measure, the lower the value, the higher the quality. $\#\square$ is the number of quadrilateral elements. t_{CDF} is the time cost of calculating CDF and t_P is the time cost of extracting PQ mesh.

Model	θ	σ_D	σ_S	$\#\square$	$\gamma(\times 10^{-5})$	ϕ	t_{CDF}	t_P
Coin	2.36‰	2.59%	5.34%	-	-	-	-	-
Fig. 4.9	2.28‰	1.29%	3.54%	1,662	1.05	0.210	0.18	1.07
(row 1, L)	7.19‰	3.46%	3.20%	1,666	0.95	0.207	2.72	1.07
Eich	0.51‰	0.37%	0.41%	-	-	-	-	-
Fig. 4.9	4.04‰	0.91%	1.27%	2,410	2.88	0.217	0.115	0.43
(row 1, R)	13.3‰	3.42%	4.21%	2,378	2.11	0.228	2.910	0.43
Snow	0.36‰	0.26%	0.29%	-	-	-	-	-
Fig. 4.9	1.31‰	0.58%	0.70%	2,377	0.67	0.309	0.146	1.82
(row 2, L)	13.9‰	0.63%	1.09%	2,458	0.72	0.302	1.250	1.82
Star	1.96‰	1.28%	2.83%	-	-	-	-	-
Fig. 4.9	8.46‰	2.27%	2.16%	1,850	1.92	0.349	0.049	0.37
(row 2, R)	11.5‰	1.67%	4.60%	1,832	1.79	0.331	0.510	0.37
Tea	0.46‰	0.29%	0.39%	-	-	-	-	-
Fig. 4.9	3.38‰	0.81%	0.79%	1,830	2.56	0.199	0.101	0.72
(row 3, L)	11.3‰	1.46%	3.79%	1,865	2.59	0.194	1.180	0.72
Hall	1.48‰	1.68%	1.89%	-	-	-	-	-
Fig. 4.9	10.5‰	1.83%	2.47%	1,951	4.07	0.275	0.178	2.11
(row 3, R)	33.2‰	4.15%	5.16%	1,953	4.13	0.260	2.720	2.11
Oct	0.48‰	0.16%	0.22%	-	-	-	-	-
Fig. 4.9	3.11‰	0.38%	1.08%	2,532	0.45	0.190	0.134	0.74
(row 4, L)	27.4‰	2.18%	5.86%	2,516	0.60	0.172	1.520	0.74
Bottle	0.67‰	0.51%	0.62%	-	-	-	-	-
Fig. 4.9	2.02‰	0.61%	0.82%	3,109	1.34	0.188	0.187	0.68
(row 4, R)	9.89‰	5.11%	7.55%	3,130	0.99	0.178	2.230	0.68
Ring	3.96‰	2.33%	3.50%	-	-	-	-	-
Fig. 4.10	8.49‰	0.99%	1.61%	2,027	2.37	0.265	0.184	1.02
	70.8‰	5.65%	2.13%	2,091	3.10	0.238	1.810	1.02

Figure 5.11: Results and comparisons. For each model, the three rows show the triangle mesh produced by Ma et al.'s method [74] and the PQ meshes produced by our method and Liu et al.'s method [69], respectively. We visualize the quality of the results using 3 quantitative measures, i.e., normal displacement θ , stress tensor error σ_s and differential stress σ_D . The lower the measures, the higher the quality of the resulting PQ meshes. Images are rendered in high resolution, allowing close-up examination.

Chapter 6

Field-Coherent Quad Layout Computation for Architectural Parameterization

6.1 Problem Statement

Quadrilateral layouts are a semi-regular structure defined on 3D surfaces and widely used in surface parameterization [21, 25], quadrangulation [35], quad-dominant remeshing [4], quad-shaped domains decomposition [57], and subdivision surfaces [22].

A commonly used approach for generating quad layouts on triangle meshes is to compute a smooth cross field and then trace the field lines. However, due to discretization error, it often yields redundant and/or missing curves and adds new singularities, resulting a quad layout with different topology as the cross field. To increase the robustness of curve tracing, Pietroni et al. [84] formulated an integer linear programming and solved it using the SCIP solver. Pietroni et al.'s method is fully automatic and can com-

pute field-aligned quad layout. However, their method requires new vertices, which are neither the singularities of the input cross field nor T-junctions, to ensure that each patch is a quadrilateral. Moreover, their method is computationally expensive due to the nature of integer programming.

To overcome the challenges in the existing methods, we develop a new and practical method that can compute field-coherent quad layout in an efficient and robust manner. We propose an interpolation strategy to determine the ports¹ for each singularity. Specifically, given a singularity with index k (which is a multiple of $\frac{1}{4}$), there are exactly $4 - 4k$ ports produced. We also propose a rounding strategy that encourages the paths going through mesh vertices to reduce short segments and small faces. After obtaining the separatrices, we adopt a simple and effective approach to extract the quadrilateral patches that can deal with various types of topological cases in a unified way.

Our method does not require any numerical solver, hereby is easy to implement and highly efficient. It takes only a few seconds for triangle meshes with 50K faces, whereas the existing optimization method often takes minutes. Computational results show that our results have consistently fewer quad patches than the state-of-the-art method. For models with rich geometric details, our method can save up to 50% patches in the quad layout. In contrast to the existing method, our method does not add new vertices other than T-junctions in the computed layout, hereby can preserve both the geometrical and topological features of the input cross field. In computing field-coherent quad layout, our method guarantees that a singularity with index k has exactly $4-4k$ separatrices. This property could be very useful for isogeometric analysis (IGA) basis function construction. Good quality quad

¹The tracing paths start from singularities. The direction of a tracing path is called *port*.

meshes can be used as the starting point of control meshes for subdivision, B-splines and T-splines. The basic functions [118, 60, 115, 116, 114, 117] defined around singularity nodes determine the convergence rate of the IGA method. Quadrilateral meshes have a lot of applications in finite element analysis. In particular, biomechanics applications in recent years have been advanced a lot. [126] addresses biomedical imaging, image processing, etc. It reviews image-mesh-simulation pipelines. [125] presents challenges and advances in image-based geometric modeling and mesh generation along with a comprehensive computational framework for analysis-suitable geometric modeling and mesh generation. [127] describes an algorithm to extract adaptive and quality quadrilateral/hexahedral meshes directly from volumetric data.

6.2 Data Structures

6.2.1 Discrete Cross Fields

Let $M = (V, E, F)$ be a manifold triangle mesh with the set of vertices $V = \{v_i\}_i$, edges $E = \{e_{ij}\}_{i,j}$ and faces $F = \{f_{ijk}\}_{i,j,k}$. A cross field is a rotational symmetry directional field that is invariant by $\frac{\pi}{2}$ rotations. In the discrete setting, each triangle is assigned a quadruple of unit vectors related by a rotation of $\frac{\pi}{2}$ radians. Pick one edge as the reference direction and use angle θ to determine the first unit vector.

Given two adjacent faces sharing a common edge e_{ij} , the chart rotation $\Delta\theta|_{e_{ij}}$ is defined as the least angle among all angles between the vectors of the two triangles.

The index of a cross field describes its behavior around a singularity.

Figure 6.1: The reference direction and the first unit vector in a random triangle.

Given a vertex v_i , we define its index as

$$I(v_i) = \frac{1}{2\pi} \left(\sum_{v_i \in e_{ij}} \Delta\theta|_{e_{ij}} + \delta(v_i) \right). \quad (6.1)$$

where $\delta(v_i)$ is the angle defect

$$\delta(v_i) = 2\pi - \sum_{v_i \in f_{ijk}} \angle v_k v_i v_j.$$

The index is zero for regular vertices and a multiple of $\frac{1}{4}$ for singularities.

A quad layout is determined by the separatrices of the singularities of the input cross field. Each separatrix terminates at either singularities, surface boundaries or a T-junction. On triangle meshes, we represent a separatrix as a sequence of nodes, which are either vertices or on mesh edges.

6.2.2 Discrete Integral Curves

A discrete integral curve consists of a sequence of nodes, which are either vertices or on mesh edges. In our tracing algorithm, we do not need to store all nodes. Instead, we only store the last node and the tracing direction to proceed the extending algorithm. Where the last knot n is expressed as the

half edge v_0v_1 through it and the divide proportion λ . n can be expressed as

$$n = \lambda v_1 + (1 - \lambda)v_0 \quad (6.2)$$

And from the half edge, we can get the facet and the chart to calculate the extending direction. We call the half edge as the extending gate of the path.

On one facet, the nodes on the edges contain all the information of this segment of the tracing line. On a mesh with cross field, there are pairs of vectors with cross directions on each facet, so if two perpendicular paths goes through a facet, 4 nodes are needed. They make a one-to-one mapping with the same directions of vectors in the cross chart. The index of the nodes on a facet will be used for the partition of the mesh.

Figure 6.2: A random highlighted half-edge e , going from vertex a to vertex b , half-edges are directed, so order matters. The corresponding half-edge in the opposite direction is denoted as $\text{opposite } e$. e is part of face f (blue shaded area).

Half-edges are among the edge-centric mesh representations. The reason for focusing on edges is quite simple: $\text{Face} \rightarrow \text{Edge}$, $\text{Face} \rightarrow \text{Vertex}$, $\text{Vertex} \rightarrow \text{Edge}$ and $\text{Vertex} \rightarrow \text{Face}$ are all one-to-many-type relationships in a general polygon mesh, but each edge links exactly 2 vertices and lies (in a closed, 2-manifold mesh) between exactly 2 faces – a fixed number of pointers (or indices), much nicer. Half-edges make this even more uniform by using

directed edges (in the direction of winding order). Each half-edge is incident to one face and one vertex, and a pair of half-edges makes up a full edge, hence the name. Figure 6.2 shows a random highlighted half-edge e .

6.3 Algorithm

Our algorithm consists of 3 steps. First, it computes the ports for each singularity. Given a singularity with index k , it produces exactly $4 - 4k$ ports. Then it traces the field directions with a rounding strategy that encourages the tracing paths going through mesh vertices. T-junctions are created when a new path hits an already-existed one. Each path terminates either at a T-junction or a singularity. Finally, it extracts the quadrilateral patches using a simple and effective approach that can deal with various types of topological cases in a unified way. In this section, we first present the data structures and then document the details of our algorithm. Figure 6.10 illustrates the pipeline of our algorithm on a 2D toy model.

6.3.1 Searching Ports

In the continuous setting, given a singularity with index k , there are exactly $4 - 4k$ separatrices emanating from it. However, in the discrete setting, missing and/or duplicating separatrices often occur. See Figure 6.3 (a)(b) for two typical examples. The reason of such mistake is the discontinuity of the charts on facets. In order to fix the issue, we propose an interpolation strategy. We put the cross charts on the edges around the singularity. The direction of the chart on each edge is the mean value of that on the 2 facets

Figure 6.3: Technical challenges in tracing separatrices. In a continuous setting, each singularity with index $\frac{1}{4}$ (resp. $-\frac{1}{4}$) is the source of 3 (resp. 5) separatrices. However, due to discretization error, redundant and missing directions often occur. (a) One of the 4 pink directions is redundant, and should be discarded. (b) Only 4 directions are detected and the dashed direction is missing. (c)-(d) We propose an interpolation strategy so that for a singularity with index k there are exactly $4 - 4k$ ports.

Figure 6.4: Chart interpolation. (a) Given an edge $v_i v_j$, we define the edge chart θ_{ij} by averaging the two charts on the adjacent faces. See Equation (6.3). (b) The chart for a polar angle α is determined by linear interpolating the charts of the adjacent edges. See Equation(6.4).

sharing the edge (Figure 6.4 (a)):

$$\theta_{ij} = (\theta_{ijk} + \theta_{ilj})/2 \quad (6.3)$$

To define the cross between adjacent two edges incident to a singularity, we use linear interpolation (see Figure 6.4 (b)) ,

$$\theta(\alpha) = \frac{(\alpha_1 - \alpha)\theta_0 + (\alpha - \alpha_0)\theta_1}{\alpha_1 - \alpha_0}, \quad (6.4)$$

where α_0 and α_1 are the polar angles of the edges. The function $\theta(\alpha)$ is a C^0 continuous monotonic function of α . Given a singularity with index k , the equation

$$\theta = \alpha + k2\pi, 4k \in \mathbb{Z} \quad (6.5)$$

has exactly $4 - 4k$ solutions (Figure 6.3 (c)(d)), which are the directions of the separatrices.

6.3.2 Rounding Strategy

Figure 6.5: A tracing line enters $\triangle v_0v_1v_2$ at P and leaves it at Q .

Consider a tracing line that enters triangle $\triangle v_0v_1v_2$ at P on oriented half-edge (v_1, v_2) with an angle γ . See Figure 6.5. Let $\lambda_0 = \frac{|v_1P|}{|v_1v_2|}$ be the length ratio and

$$\gamma_c = \angle v_0Pv_2 = \arctan \frac{v_0v_1 \sin \beta_1}{|v_0v_1| \cos \beta_1 - \lambda_0 |v_1v_2|}.$$

The tracing line leaves the triangle at edge (v_2, v_0) if $\gamma < \gamma_c$, and (v_0, v_1)

Figure 6.6: Eliminating short segments. (a) A path near a vertex, yielding very short segments and small facets. (b) Making the path going through vertex fixes the problem.

otherwise. It is easy to calculate the position of the leaving point Q using sine Law

$$\lambda_1 = \begin{cases} \frac{v_1 v_2 (1-\lambda) \sin \gamma}{v_0 v_2 \sin(\beta_2 + \gamma)}, & \text{if } \gamma < \gamma_c; \\ 1 - \frac{v_1 v_2 \lambda \sin \gamma}{v_0 v_1 \sin(\gamma - \beta_1)}, & \text{otherwise.} \end{cases} \quad (6.6)$$

To make tracing robust, we introduce an angle tolerance ϵ and round off λ_1 as follows:

$$\lambda_1 := \begin{cases} 0, & \gamma < \min\{\gamma_c, \epsilon\}; \\ 1, & \gamma_c - \epsilon < \gamma < \gamma_c + \epsilon; \\ 0, & \gamma_c < \gamma < \gamma_c + \epsilon; \\ 1, & \gamma_c < \gamma < \pi - \epsilon. \end{cases}$$

The angle tolerance ϵ , specified by the user, encourages the tracing line to pass through mesh vertices, so that the number of short segments can be reduced (see Figure 6.6).

6.3.3 Pairing and Smoothing

In our method, we assume each triangle contains at most two perpendicular paths. But experiments shows there are many coincidences that two paths with opposite extending direction reach one triangular facet. In this case, we

Figure 6.7: Path linking. (a) Directly linking two paths often produces a non-smooth path. (b) Adjusting the linking points improves smoothness.

join them together to compose one path linking two singularities (Figure 6.7). But such operation usually bring turning points that seriously break the smoothness of the path. So it is necessary to adjust the knots on edges.

It is not possible to ensure the paths strictly follow the field lines, the condition can be weakened as

$$\Delta \arg(\mathbf{t}) = \Delta \theta$$

This means the path keeps a constant angle from the field line. In fact, $\Delta \theta$ on a pair of adjacent facets are nearly 0, this means the tracing path is almost straight. In other words, although the operation is local, it monotonously shortens the total length of the path, hereby resulting in a fairer path.

Figure 6.8: There are two non-parallel separatrices in face f . The tracing of the current separatrix Γ stops if the intersection point T is inside f , and continues otherwise.

In our tracing algorithm, we choose the shortest tracing path to extend. So each path almost grows at a same speed. Each path will end when it reach a boundary of the surface or another path. The algorithm is shown in Algorithm 3.

6.3.4 T-junctions

Each tracing line starts at a singularity and terminates at either a singularity or a T-junction. In our method, we assume each triangle contains at most two perpendicular paths. we use vector cross product to compute their intersection. Denote by $\mathbf{u}_p = (p_x, p_y, 1)$ the homogeneous coordinates of point $p = (p_x, p_y)$. Let A, B, C and D be the endpoints of the two perpendicular lines in triangle $\triangle v_0 v_1 v_2$. We can easily compute the intersection of AB and CD as

$$\mathbf{u}_P = \frac{(\mathbf{u}_A \times \mathbf{u}_B) \times (\mathbf{u}_C \times \mathbf{u}_D)}{\|(\mathbf{u}_A \times \mathbf{u}_B) \times (\mathbf{u}_C \times \mathbf{u}_D)\|_1}, \quad (6.7)$$

where $\|\cdot\|_1$ is the sum of the coordinate components of a vector. If at least one of the Barycentric coordinates of P is negative, the intersection point P is outside the triangle and the path is not ended in this triangle (see Figure 6.8 (c)).

6.3.5 Facet Separation

After all tracing lines are determined, the facets split by tracing lines require to reconstruct. This seems to be a simple problem because there are only 5 cases to deal with(Figure 6.9). But our rounding strategy should take the coincidence between the intersections and vertices into account. The number of cases will increase terribly. So the reconstruction become awesome.

We adopt a strategy that can deal with each case of face split robustly.

Algorithm 3: Computing separatrices

input : A manifold triangular mesh $M = (V, E, F)$ and a smooth cross field θ defined on triangular faces, and angle error tolerance ϵ

output: Field-coherent separatrices

```
1                                     ▷ Initialize the path set  $\mathcal{P} \leftarrow \emptyset$ 
2 for each vertex  $v_i \in V$  do
3   Calculate index  $I(v_i)$                                      ▷ Eqn. (6.1)
4   if  $I(v_i) \neq 0$  then
5     for each incident edge  $v_i v_j$  do
6       Compute  $\theta_{ij}$                                          ▷ Eqn. (6.3)
7     for each facet  $\Delta v_i v_j v_k$  do
8       Search the solution  $\alpha$                                ▷ Eqn. (6.5)
9       if the direction  $\mathbf{t}$  with argument  $\alpha$  is in  $\Delta v_i v_j v_k$  then
10        Add a new open path containing  $p$  to  $\mathcal{P}$ 
11 while  $\mathcal{P} \neq \emptyset$  do
12   Find path  $\Gamma$  with the shortest length
13   Let  $f$  be the triangular face containing the open end of  $\gamma$ 
14   Compute the extending direction  $\Gamma$  in  $f$  and new node  $n$    ▷ Eqns. (6.5) and (6.6)
15   if  $n \in \partial M$  then
16      $\Gamma.\text{node} \leftarrow n$ 
17     Remove  $\Gamma$  from  $\mathcal{P}$ 
18   if  $f$  contains another open path  $\Gamma_1$  then
19     Compute the intersection  $T = \Gamma_1 \cap \Gamma$                ▷ Eqn. (6.7)
20     if  $T \in f$  then
21        $\Gamma.\text{node} \leftarrow T$ 
22       Remove  $\Gamma$  from  $\mathcal{P}$                                      ▷  $\Gamma$  terminates at  $T$ 
23   if  $\Gamma \in \mathcal{P}$  then
24     Round off  $\lambda$                                            ▷ Equation (6.6)
25     if  $\lambda == 0$  or  $\lambda == 1$  then
26       Record the vertex  $v$  that  $\Gamma$  goes through on  $f$ 
27       Search facets  $f_1$  around  $v$  for further extension of  $\Gamma$ 
28        $\Gamma.\text{len} \leftarrow \Gamma.\text{len} + |\Gamma.\text{node} - v|$ 
29        $\Gamma.\text{node} \leftarrow v$ 
```

Figure 6.9: Different topological types of facets split by tracing lines. In (a), (b) and (c) nodes on a same path are matched; in (d) and (e), nodes on edges are marked. Even though n_i are coincide with some v_j , extracting facets from marked or matched vertices is also robust.

The strategy is based on the circular array data structure. The vertices and intersections on the borders of the facets are arranged anticlockwise. If there is a duplication, the repeated point is put in the circular array only once. If there is a T-Shape junction (Figure 6.9 (d) and (e)) in the face, the points on the tracing lines are marked, each sub-facet is composed of the T junction and the points between 2 marked points. Otherwise, the points on the tracing lines are matched, the sub-facets composed of the points between each pair of matched points are constructed first, then the rest part is the last piece of the sub-facet (Figure 6.9 (a), (b) and (c)).

After the split mesh is constructed, the final step is to mark the edges on the split mesh and use flood-fill to mark the patches on each facet.

6.4 Results and Comparison

We implemented our algorithm in C++ and evaluated it on a laptop with an Intel(R) Core(TM) i7 CPU 2.90 GHz and 8 GB memory.

6.4.1 Evaluation

We evaluate the results using both geometric and topological measures.

Algorithm 4: Extracting quad layout

input : Triangle M with separatrices $\mathcal{P}$, whose nodes are stored on facets
output: Quad layout $\mathcal{Q}$

- 1 Create an empty mesh M_1
- 2 Copy all vertices of M to M_1
- 3 Append all separatrix nodes excluding vertices of M to M_1
- 4 **for** Every face $f \in M$ **do**
- 5 **if** *there is no node on the edge of f* **then**
- 6 Construct a facet of M_1 with vertices of f
- 7 **else**
- 8 Arrange all vertices and nodes on the edges of f as a ring R
- 9 **if** *there is a T -junction T in f* **then**
- 10 Mark all nodes of paths in R
- 11 **for** Each marked node $n_0 \in R$ **do**
- 12 Find the next marked node n_1 after n_0
- 13 Construct a facet of M_1 with vertices between n_0 and n_1 , including T , n_0 and n_1
- 14 **else**
- 15 Match all pairs of nodes on a same path
- 16 **for** each pair of matched nodes n_0 and n_1 **do**
- 17 **if** *there are no nodes on path between n_0 and n_1* **then**
- 18 Construct a facet of M_1 with vertices n_0 and n_1 , including n_0 and n_1
- 19 Remove the vertices between n_0 and n_1 excluding n_0 or n_1 from R
- 20 **if** $R \neq \emptyset$ **then**
- 21 Construct a facet of M_1 with all vertices remained in R
- 22 Mark all edges of M_1 on paths
- 23 Flood fill facets in M_1 under the separation of Marked edges
- 24 Extract facets of M_1 with same filled marks quad patch set $\mathcal{Q}$
- 25 Output $\mathcal{Q}$

Geometric measures. We define a regularity metric ε to measure how

Figure 6.10: Algorithmic pipeline. (a) The input cross field has two singularities of indices $\pm\frac{1}{4}$, respectively. (b) For a singularity with index k , we generate $4 - 4k$ ports (pink vectors). (c) We trace a separatrix for each port with a rounding strategy that encourages the separatrix to go through vertices. Each separatrix terminates at another singularity, a T-junction or a boundary. (d) Finally, we generate the layout by extracting the quadrilateral patches.

close a quad patch is to a rectangle,

$$\varepsilon = \max \left\{ \frac{|a - c|}{\max\{a, c\}}, \frac{|b - d|}{\max\{b, d\}} \right\}, \quad (6.8)$$

where a , b , c and d are the side lengths of the quad patch. The smaller the metric ε , the closer the quad patch to a rectangle.

To preserve the geometric features, the computed layout should follow the field directions of the input cross field. We measure the coherence by computing the angle difference between the motorcycle graph and the field directions. Specifically, given an separatrix γ , we measure the field coherence by computing the line integral

$$\delta = \frac{1}{\text{len}(\gamma)} \int_{\gamma} (1 - \mathbf{t} \cdot \mathbf{f}) \, dl, \quad (6.9)$$

where $\mathbf{t}$ is the unit tangent direction of γ , $\mathbf{f}$ the field line direction, and dl the line element. If $\delta = 0$, the separatrix matches the underlying cross field exactly. In general, the smaller the value δ , the higher the coherency. In Table 6.1, we report the average field coherence $\bar{\delta}$ and the average quad

regularity metric $\bar{\varepsilon}$.

Topological measures include the number of patches in the quad layout N_p , the number of T -junctions and the number of new vertices other than T -junctions N_n . The fewer the patches, the higher the quality of the quad layout. Our method does not add any new vertices other than T -junctions in the quad layout. Therefore, each separatrix terminates at singularities, boundary or a T -junction.

6.4.2 Comparison

We compare our method with Pietroni et al.’s method [84] in terms of geometric measures (quad regularity ε and field coherency δ) and topological measures (the number of quad patches N_q , the number of T -junctions N_t , the number of vertices other than T -junctions added in the layout N_n). It is worth noting that each separatrix in our results terminate at singularities, a boundary or a T -junction. Thus, our method does not add any new vertices other than T -junctions. In contrast, Pietroni et al.’s method often adds a large number of valence-4 vertices in the quad layout (see Figure 6.11). As a result, our method produces much fewer patches than Pietroni et al.’s method. For models with rich geometric details, such as Dragon, Lion and Redbox, our method can save up to 50% patches. See Table 6.1. Our method can also preserve geometric features well. Table 6.1 shows the mean values of the regularity and field coherency measures. For most models, our results have lower error metrics and higher quality than theirs. We also calculate the histogram of quad regularity ε and field coherency δ , they are shown in Figure 6.12.

Unlike [84], our method does not solve an integer programming nor require any numerical solver. Therefore, it is computationally efficient. It takes only

Table 6.1: Statistics. $|V|$: the number of mesh vertices; N_s : the number of singularities in the cross field; N_p : the number of quadrilateral patches; N_t : the number of T-junctions; N_n : the number of *new* vertices other than T-junctions added to the quad layout; $\bar{\varepsilon}$: the average quad regularity metric; $\bar{\delta}$: the average field coherence metric; and T : processing times in seconds. For each model, we show both our results (first row) and the results of Pietroni et al.’s method [84] (second row).

Model ($ V $, N_s)	N_p	N_t	N_n	$\bar{\varepsilon}$	$\bar{\delta}$	T
Aquadom (1K, 6)	21	11	0	0.539	0.265	0.51
	23	0	7	0.625	0.787	N.A.
Bunny (15K, 50)	116	136	0	0.513	0.356	0.95
	179	0	131	0.520	0.673	23.6
Casting (18K, 126)	286	288	0	0.459	0.409	1.13
	515	0	373	0.492	0.565	68.9
Dragon (12K, 343)	679	676	0	0.484	0.271	1.55
	1,024	90	608	0.536	0.707	57.6
Fertility (14K, 38)	93	98	0	0.485	0.307	0.98
	128	0	84	0.486	0.656	19.8
Gargoyle (10K, 141)	303	328	0	0.485	0.279	2.46
	528	44	370	0.526	0.698	16.3
Heptoroid (9K, 168)	469	518	0	0.431	0.305	1.11
	825	0	465	0.475	0.670	56.7
Lion (25K, 220)	543	650	0	0.485	0.352	2.30
	941	38	699	0.484	0.694	66.2
Lucy (25K, 173)	356	370	0	0.504	0.205	2.24
	671	62	474	0.539	0.821	90.9
Redbox (25K, 1170)	2069	1802	0	0.516	0.326	8.78
	4586	450	3141	0.493	0.642	188.1

a few seconds for all test models on a laptop. Since we did not find a reference implementation of Pietroni et al.’s method [84], we could not make a direct comparison on running times. The timings of their method in Table 6.1 are a rough estimation based on the timings reported in their paper [84] and the performance difference of the CPUs used in the tests.

Figure 6.11: Close-up views of the Bunny model. Besides T-junctions, Pietroni et al.’s method [84] produces a large number valence-4 vertices (highlighted by red circles) in the quad layout. Our method, in contrast, does not require the valence-4 vertices.

Figure 6.12: The distribution of ε and δ . produced by Pietroni et al.’s method [84] and our method.

6.4.3 Limitation

There is also a limitation of our work. On a path linking two singularities, the knots on vertices are fixed, so the path is not so smooth as the cross field. Make further smoothing of such paths is a valuable work in the future. If we want to get quad-mesh without T-junctions from these results, the paths are required to be separated by a set of consistent integers. Find the integers is also a further work based on this framework.

6.5 Application

Many applications apply quad layout as input, as upstream application, a high quality separation is important for these downstream applications. For

instance, convolutional neural network(CNN) requires 2D image as input. To extend CNN to 3D processing, 3D mesh can be separated to quad patches, then parameterization can be applied to map 3D surface to 2D field.

Figure 6.13: (a) Conformal parameterization of reconstructed mesh. (b) A quadrilateral patch denoted by $ABCD$.

We also test our method on scanned models [40]. Since the inputs are raw point clouds, we first apply Poisson surface reconstruction to construct triangle meshes. Then we apply the MIQ method [20] to compute smooth cross fields. Next we compute quad layouts using our algorithm. After that, we compute conformal parameterization to map each quadrilateral patch to a 2D rectangular domain. Denote by $ABCD$ a quadrilateral patch (see the inset). We solve two Laplacian equations with Dirichlet boundary conditions

$$\Delta u_0 = 0, u_0|_{AB} = 0, u_0|_{CD} = 1$$

$$\Delta v_0 = 0, v_0|_{DA} = 0, v_0|_{BC} = 1$$

Since our quad edges follow the directions of the cross field closely, the gradients of the parameters are orthogonal to each other $\nabla u_0 \perp \nabla v_0$. However they usually have different magnitudes. Let $\lambda = \frac{\|\nabla u\|}{\|\nabla v\|}$ be the average

length ratio and s the patch area. Then we compute the scaled parameters (u, v) as $u = u_0\sqrt{s\lambda}$, and $v = v_0\sqrt{s/\lambda}$, which serve as the parameters of the conformal parameterization. After mapping each quad patch to a 2D rectangle, we tessellate all rectangles into a large square using a simple greedy strategy to reduce the empty space in the square. The resulting tessellation is a multi-chart geometry image with low area and angle distortions. See Figure 6.14.

6.6 Summary

We proposed a simple method for computing field-coherent quad layout from a given cross field. Each edge of the layout terminates either at a singularity of the input cross field or a T-junction. Unlike the existing method, our method does not generate any new singularities. Moreover, the layout edge aligns with the field directions well. Therefore, the resulting layout preserves the geometrical and topological features of the input cross field. Evaluation shows that that our results have fewer quad patches than those of the existing method, and with comparable or higher quality in terms of field-coherence and patch regularity. Last but not the least, our method does not require any numerical solver, and is easy to implement and computationally efficient.

Our method can guarantee a singularity with index k has exactly $4 - 4k$ separatrices. However, our method does not solve the drifting issue: if the initial direction of a separatrix drifts a little bit, the separatrix, which is supposed to terminate at another singularity, may terminate at a T-junction instead. As a result, the quad layout contains skinny rectangles. A possible way to fix this issue is to develop a post-processing step to shrink the skinny rectangles. In our current implementation, we consider only singularities

Figure 6.14: Results on scanned data. The models are taken from the 8i Voxelized Surface Light Field (8iVSLF) Dataset. For each model, we show the quad layout, 2D tessellation and normal map. From left to right, top to bottom: *Red and Black* (Frames 1677 and 1491), *Loot*, *Long Dress* (Frames 1063 and 1270), and *Soldier*. We slightly modified some shapes by removing thin, tube-shaped parts, which cause difficulty in computing cross fields.

with indices $\pm\frac{1}{4}$. However, the proposed method is general in that it can also deal with other type of singularities. In the future, we will also extend it to

extract the layout for N -RoSy fields [89].

Figure 6.15: Comparison with Pietroni et al.'s method [84]. Both methods are able to produce field-coherent quad layouts. However, our results have consistently fewer patches than theirs. In particular, for models with rich geometric details, our method can save up to 54% quad patches. Moreover, our method does not add any new vertex other than T-junctions in the computed layout, whereas Pietroni et al.'s method adds a large number of non-T-junction vertices to maintain the quad topology. The 2-tuple below each figure is the number of quad patches N_p and the number of non-T-junction vertices added in the quad layout N_n . Random colors are used to visualize the layout.

Chapter 7

Contrast-Enhanced High-Relief Modeling for Architectural Decoration

7.1 Problem Statement

Relief is a form of sculpture used for thousands of years. Relief is an art technique between sculptural and painting where the emerged elements remain attached to a solid background of the same material. Relief is an art technique between sculptural and painting. Relief provides an impression that the emerged elements raised above the background. One of the biggest advantages of relief is that relief is less delicate and more securely fixed than a sculpture, for instance, a standing figure made from stone has weak point at the ankles. Type of relief are distinguished based on the degree of projection of the sculpted form from the field. The full range includes high relief, mid-relief, bas-relief, and shallow-relief. The distinction between high-relief and bas-relief is the most obvious and most significant, and these two are the

only words used in this dissertation. Unlike bas-relief, high-relief has materials undercut from background. Although bas-relief modeling is enforced thanks to height field, high-relief modeling is quite challenging since for each (x, y) , there can exist multiple z values. Existing methods enforce bas-relief modeling by utilizing techniques developed for image processing, high-relief modeling is still a challenging problem.

To generate 3D printable high-relief, volume minimization is in demand. However, the commonly used scaling process reduces the contrast of high-reliefs. Therefore, we propose a depth compression function to satisfy volume minimization and contrast maximization at the same time. There are only two variables in the depth compression function, we apply an optimization framework to calculate the value of these two variables. Our method is easy to implement and produces good-quality high-reliefs. We show results on a range of real 3D models include real-world and synthetic models. In particular, the concept good-quality pays more attention for 3D printing applications. Therefore, good-quality in this work means the volume is reduced more than 20% while the visual details are maintained in terms of normal preservation.

7.2 Algorithm

7.2.1 Method Overview

We present a simple and effective method to generate high-reliefs from real 3D models. To reduce the volume while maintain the contrast of the 3D models, we formulate a depth compression function with two variables and calculate the variables using an optimization framework. The optimization framework takes volume minimization and contrast maximization as two objectives. The

solution space is narrowed down to only two variables.

Our method can be divided into three main steps:

- **Depth Compression Function:** Constructing the depth compression function with two objectives, volume reduction and contrast enhancement.
- **Solving Optimal Variables:** Constructing an optimization framework to solve the two variables in the depth compression function.
- **Transplanting to Curved Background Surfaces:** Projecting the boundary to the curved background surface and solving the displacement of the high-relief.

7.2.2 Depth Compression Function

To produce 3D printable high-reliefs, volume reduction is in demand to save 3D printing materials. However, the commonly used scaling process reduce the contrast of the real 3D model.

Figure 7.1: (a) A real 3D model before scaling. (b) A real 3D model compressed with scale ratio $s = 0.3$. (c) Enhance contrast for compressed model with $d = 0.3$.

Figure 7.1(b) shows a real 3D model compressed with $ratio = 0.3$. The volume of the scaled model is reduced around 70%, however, the contrast is lost seriously at the same time. Therefore, we propose a depth compression function to enhance the contrast of the scaled model.

There are two main objectives in producing 3D printable high-reliefs. The first one is volume minimization to save 3D printing materials, the second one is contrast maximization to maintain geometry details.

Equation 7.1 is the depth compression function we used to reduce volume while enhance contrast for high-relief modeling. Figure 7.1(c) shows an example of contrast enhancement. There are two terms in the depth compression function. The first one is volume reduction with ratio s and the second term is contrast enhancement with variable d . This manipulation function narrows down the solution space due to the intrinsic property of the input model. Particularly, z_{i_0} is z-coordinate of the original model, $\mathbf{n}_{i_z}$ is z-coordinate of original normal $\mathbf{n}_i$ and H_i is original mean curvature at vertex v_i .

$$z_i(s, d) = z_{i_0} \cdot s + d \cdot \mathbf{n}_z \cdot H_i \quad (7.1)$$

Curvature is the amount by which a curve deviates from being a straight line, or a surface deviates from being a plane, and normal is a vector that is perpendicular to the vertex. We use these two value to determine the movement along z direction.

7.2.3 Optimization Framework

Apply Equation 7.1 to a real 3D model will generate a high-relief. To determine the value of s and d , we construct an optimization framework to

calculate the optimal value of s and d . There are two terms in the optimization framework. The first one is volume minimization and the second one is contrast maximization. Contrast maximization is equivalent to normal maintenance since depth compression reduces contrast.

$$E = \min(\omega_1(\frac{1}{n} \sum_{i=1}^n z_i^2) + \omega_2(\frac{1}{n} \sum_{i=1}^n (\mathbf{n}_i - \mathbf{n}'_i)^2)) \quad (7.2)$$

In Equation 7.2, $\mathbf{n}'_i$ is normal of vertices after updating and $\mathbf{n}_i$ is normal of vertices before updating. z_i after updating is calculated using Equation 7.1.

We use Ceres Solver to calculate the value of s and d . Ceres Solver is an open source C++ library for modeling and solving large, complicated optimization problems. It can be used to solve Non-linear Least Squares problems with bounds constraints and general unconstrained optimization problems. It is a mature, feature rich, and performant library that has been used in production at Google since 2010.

Notice that there are two parameters in the optimization framework, ω_1 and ω_2 . Before solving the value of s and d , we design experiments to determine the value of parameters. Based on the metrics of z^2 and $\mathbf{n} - \mathbf{n}'$, ω_2 is set to 1 and ω_1 is set to the longest edge of bounding box's reciprocal. To evaluate the effectiveness of this setting, we design a set of experiments to analysis the parameters in the optimization framework. The parameter ω_2 is set to 1 for all experiments, different value of ω_1 are tested and computed variables are shown in Table 7.1, the optimized volume E_V and normal difference $E_{\delta\mathbf{n}}$ are also shown in Table 7.1.

Experiments shows that setting ω_1 to the longest edge of bounding box' reciprocal and ω_2 to 1 is reasonable.

Table 7.1: Variables s and d generated from different ω_1 . Experiments show that parameter ω_1 should be set to the longest edge of bounding box' reciprocal and ω_2 should be set to 1.

ω_1	ω_2	s	d	$E_{\mathbf{n}}$	E_V
10	1	0.283939	4.3711	0.916506	21.383
1	1	0.396269	2.75144	0.404965	42.1093
0.3	1	0.398857	1.02561	0.12325	43.1386
0.2	1	0.399005	0.627327	0.0905417	43.3023
0.1	1	0.399164	0.236545	0.0701723	43.4743
0.08	1	0.399572	0.142112	0.0669127	43.5978
0.05	1	0.398278	-0.016237	0.0633912	43.3744
0.0266	1	0.397985	-0.106801	0.0621353	43.3447
0.02	1	0.394409	-0.141323	0.0625407	42.5826
0.01	1	0.391611	-0.192861	0.0627789	42.0003
0.005	1	0.342089	-0.22035	0.0733596	32.0666
0.001	1	0.38904	-0.0331845	0.0650583	41.3918
0.0001	1	0.679196	-0.147023	0.0174196	126.216

Figure 7.2: Setting the parameter ω_1 to the longest edge of bounding box' reciprocal and parameter ω_2 to 1.

Figure 7.2 shows the bounding box of a real 3D model. We calculate the longest edge of bounding box and set the parameters.

7.2.4 Transplant to Curved Background Surfaces

$$\begin{cases} \Delta\delta_x = x_1 - x_0 \\ \Delta\delta_y = y_1 - y_0 \\ \Delta\delta_z = z_1 - z_0 \end{cases} \quad (7.3)$$

Equation 7.3 presents the implementation of seamless transplant process. In this Laplace equation, Δ is the differential coordinates of the real 3D model, (x_0, y_0, z_0) is the original boundaries while (x_1, y_1, z_1) is the topological matching boundary of the curved background surface. The solution of this equation is the displacement of the deformation. By adding $(\delta_x, \delta_y, \delta_z)$ back to the input model, we generate the transplanted model. This solution is physically-based gradual change. Results are shown in Figure 7.3 and 7.4.

Figure 7.3: Transplant the Dragon relief to a curved background surface.

Figure 7.4: (a) The result generated by bas-relief modeling techniques. (b) Transplant a high-relief to a curved background surface.

7.3 Results

We implemented our algorithm in C++ and evaluated it on a laptop with an Intel i7 CPU 2.90 GHz. Our method allows user to generate contrast-enhanced high-relief from real 3D models.

Figure 7.6 show several high-reliefs generated by our method. The computed variables for the depth compression function are shown in Table 7.2

The Loin model is a real 3D model which has multiple z value for an (x, y) position. The chin of the Lion model has undercut part from the background.

Figure 7.5: Distribution of z -coordinate before and after depth compression. Distributions indicate that our method maintains the geometry details of the input real 3D models.

Figure 7.6: Contrast-enhanced high-reliefs generated by our method. The first and second columns are input 3D models and the third and fourth columns are high-reliefs produced by our method.

Existing bas-relief modeling techniques are unavailable to compress this real 3D model.

As shown in Table 7.2, the compression ratio is around 40%, means the volume is reduced by around 60%. This reduction is meaningful in producing

Table 7.2: Variables s and d in depth compression function generated by solving an optimization problem.

Model	s	d
Lion	0.395935	0.190371
Buddha	0.399966	0.107864
Hearno	0.388387	0.260911
Seeno	0.399935	0.0276465
Girl	0.39875	0.512438
Einstein	0.399164	0.236545
Man	0.378335	0.133773
King	0.460769	0.0285154
LongDress	0.64869	1.86382
Loot	0.568587	2.75415

3D printable high-reliefs since the 3D printing materials are saved effectively. We show the side view before and after depth compression in Figure 7.6.

In addition to volume reduction, our method also shows advantages in geometry details preservation. Figure 7.6 show modeling results on different type of geometry details, for example, the hair details in Einstein, Girl, Lion and Man. The clothing details are also maintained after the depth compression, for instance the clothing parts in Buddha, Hearno, Seeno and King.

We also analyse the distribution of z value before and after depth compression. Figure 7.5 illustrates the histogram of z -coordinates of all results. Histograms show that our method reduce the volume of high-reliefs and maintain the geometric details of the input models.

We also test our method on scanned models. We use the model constructed in Chapter 5, calculate mean curvatures and normals, then calculate the value of variables s and d . The values are shown in Table 7.2. Figure 7.7

Figure 7.7: Contrast-enhanced high-reliefs generated from scanned models by our method.

shows the contrast-enhanced high-reliefs generated by our method.

7.4 Summary

In conclusion, we present an optimization framework for high-relief modeling. we propose a depth compression function to reduce volume and maintain contrast at the same time. There are only two variables in the depth compression function, we apply an optimization framework to calculate the value of these two variables. The optimal value is the trade-off point of volume minimization and contrast maximization. Our method is easy to implement and produces good-quality high-reliefs. We show results on a range of real 3D models include real-world and synthetic models.

There are a few interesting directions for future work. First, combining the high-relief with a curved background surface with complex geometry details is an interesting problem. Second, it is highly desired to develop a method which allows multiple layers of high-relief modeling.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

This dissertation tackle four challenges in architectural geometry, they are maintaining semantic information across generic 3D model editing operations, constructing self-supporting surfaces with planar quadrilateral elements, computing field-coherent quad layout on triangle meshes and contrast-enhanced high-reliefs modeling. Effective solutions are proposed for these problems. These solutions covers different directions in architectural geometry including architecture maintenance, architecture construction, architecture texture mapping and architecture decoration.

In the first part of this dissertation, we introduced a data model synchronization method which preserves semantic information across editing operation. It is independent of the edit operation and depends only on geometry, UV mappings, and materials. This enables easy integration of existing and future 3D editing tools with rich data models in a broad range of 3D applications. The method was implemented in a LoD3 building editor for the Virtual Singapore project, including interactive push-pull and procedural generation

of façade provided by 3rd party libraries. The quality of the method was verified with 30 common editing tasks and compared with four commercial tools.

In the second part of this dissertation, we developed a simple yet effective method for generating 3D self-supporting surfaces with planar quadrilateral elements. Our method takes a triangular discretization of a 3D self-supporting surface as input and generates PQ mesh via constructing a conjugate direction field. In contrast to the existing methods that compute conjugate direction fields using global optimization, our method is a lightweight approach since it computes a local stretch matrix for each triangular face with small principal curvature difference that turns principal directions into conjugate directions. Computational results show that the generated PQ meshes approximate the input shape very well, hereby satisfying the balance constraints. Comparisons with the existing methods demonstrate the effectiveness of our method.

In the third part of this dissertation, we proposed a simple method for computing field-coherent quad layout from a given cross field. Each edge of the layout terminates either at a singularity of the input cross field or a T-junction. Unlike the existing method, our method does not generate any new singularities. Moreover, the layout edge aligns with the field directions well. Therefore, the resulting layout preserves the geometrical and topological features of the input cross field. Evaluation shows that our results have fewer quad patches than those of the existing method, and with comparable or higher quality in terms of field-coherence and patch regularity. Last but not the least, our method does not require any numerical solver, and is easy to implement and computationally efficient.

In the forth part of this dissertation, we presented research in contrast-

enhanced high-relief modeling. we propose a depth compression function to reduce volume and maintain contrast at the same time. There are only two variables in the depth compression function, we apply an optimization framework to calculate the value of these two variables. The variables are obtained when volume minimization and contrast maximization reach a balance point. Our method is easy to implement and produces good-quality high-reliefs. We show results on a range of real 3D models include real-world and synthetic models.

8.2 Future Work

This dissertation introduces several solutions for practical problems, however, there is still space to improve these methods. All these four methods apply local and discrete differential geometry driven methods to tackle the challenges. Although local methods have advantages in computational efficiency and easy to implement, how to include advantages in global method is an interesting direction for future work.

In the first part of this dissertation, the focus is on interactive editing operations only. It would be interesting to see how accuracy our labeling process will be in more practice cases and how the method performs in batch processing of 3D geometry, e.g., for change detection and labeling, an important topic for maintaining and updating 3D city models.

In the second part of this dissertation, There are a few interesting directions for future work. First, our method requires the input shape to be self-supporting. From the application point of view, it is highly desirable to either construct PQ meshes directly from user-specified boundary curves or to take a reference surface as input and then optimize its geometry to

fulfil the physical requirements of being self-supported. To make the method more flexible, we will also take the user-specified soft and/or hard directional constraints into consideration.

In the third part of this dissertation, our method can guarantee a singularity with index k has exactly $4 - 4k$ separatrices. However, our method does not solve the drifting issue: if the initial direction of a separatrix drifts a little bit, the separatrix, which is supposed to terminate at another singularity, may terminate at a T-junction instead. As a result, the quad layout contains skinny rectangles. A possible way to fix this issue is to develop a post-processing step to shrink the skinny rectangles. In our current implementation, we consider only singularities with indices $\pm\frac{1}{4}$. However, the proposed method is general in that it can also deal with other type of singularities. In the future, we will also extend it to extract the layout for N -RoSy fields [89].

In the forth part of this dissertation, how to transplant the high-relief to a curved background surface with complex topology structures and geometry details is still an open problem. In addition, a user-friendly graphical user interface is in demand for artists to design high-reliefs using our algorithm.

References

- [1] 3Dis. “CityEditor”. In: (2019). URL: <https://www.3dis.de/cityeditor/>.
- [2] Tobias Achterberg. “SCIP: Solving constraint interger programs”. In: *Mathematical Programming Computation* 1.1 (2009).
- [3] Fernando Alegre and Frank Dellaert. “A probabilistic approach to the semantic interpretation of building facades”. In: (2004).
- [4] Pierre Alliez et al. “Anisotropic polygonal remeshing”. In: *ACM SIGGRAPH 2003 Papers*. 2003, pp. 485–493.
- [5] AUTODESK. “3dMax”. In: (2020). URL: <https://www.autodesk.com/3ds-max>.
- [6] AUTODESK. “Autocad”. In: (2020). URL: <http://www.autodesk.com/autocad>.
- [7] AUTODESK. “AutoCAD Map 3D”. In: (2020). URL: <https://knowledge.autodesk.com/support/autocad-map-3d>.
- [8] AUTODESK. “InfraWorks”. In: (2020). URL: <https://www.autodesk.com/products/infraworks/overview>.
- [9] AUTODESK. “Maya”. In: (2020). URL: <http://www.autodesk.com/maya>.

- [10] BentleySystems. “Bentley Map”. In: (2019). URL: <https://www.bentley.com/en/products/product-line/asset-performance/opencities-map>.
- [11] Paul J Besl and Neil D McKay. “Method for registration of 3-D shapes”. In: *Sensor fusion IV: control paradigms and data structures*. Vol. 1611. International Society for Optics and Photonics. 1992, pp. 586–606.
- [12] Filip Biljecki et al. “Formalisation of the level of detail in 3D city modelling”. In: *Computers, Environment and Urban Systems* 48 (2014), pp. 1–15.
- [13] Tolga Birdal and Slobodan Ilic. “A point sampling algorithm for 3d matching of irregular geometries”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 6871–6878.
- [14] Bitmanagement. “BS Contact Geo”. In: (2019). URL: <http://www.bitmanagement.de/en/products/interactive-3d-clients/bs-contact-geo>.
- [15] Philippe Block and Lorenz Lachauer. “Closest-Fit, Compression-Only Solutions for Freeform Shells”. In: *Proceedings of the IABSE-IASS Symposium 2011*. 2011.
- [16] Philippe Block and John Ochsendorf. “Thrust network analysis: A new methodology for three-dimensional equilibrium”. In: *Journal of the International Association for shell and spatial structures* 48.3 (2007), pp. 167–173.
- [17] Alexander I Bobenko et al. *Discrete differential geometry*. Vol. 38. Springer, 2008.

- [18] Ioana Boier-Martin, Holly Rushmeier, and Jingyi Jin. “Parameterization of triangle meshes over quadrilateral domains”. In: *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. 2004, pp. 193–203.
- [19] David Bommes, Timm Lempfer, and Leif Kobbelt. “Global structure optimization of quadrilateral meshes”. In: *Computer Graphics Forum*. Vol. 30. 2. Wiley Online Library. 2011, pp. 375–384.
- [20] David Bommes, Henrik Zimmer, and Leif Kobbelt. “Mixed-integer quadrangulation”. In: *ACM Transactions On Graphics (TOG)* 28.3 (2009), pp. 1–10.
- [21] David Bommes et al. “Integer-grid maps for reliable quad meshing”. In: *ACM Transactions on Graphics (TOG)* 32.4 (2013), pp. 1–12.
- [22] David Bommes et al. “Quad-mesh generation and processing: A survey”. In: *Computer Graphics Forum*. Vol. 32. 6. Wiley Online Library. 2013, pp. 51–76.
- [23] Kenneth A Brakke. “The surface evolver”. In: *Experimental mathematics* 1.2 (1992), pp. 141–165.
- [24] Building and Construction Authority. “Code of Practice on Buildable Design”. In: (2000). URL: <https://www.bca.gov.sg/BuildableDesign/others/copbddec00.pdf>.
- [25] Marcel Campen, David Bommes, and Leif Kobbelt. “Quantized global parametrization”. In: *Acm Transactions On Graphics (tog)* 34.6 (2015), pp. 1–12.
- [26] Marcel Campen and Leif Kobbelt. “Dual strip weaving: Interactive design of quad layouts using elastica strips”. In: *ACM Transactions on Graphics (TOG)* 33.6 (2014), pp. 1–10.

- [27] Hugo Casquero et al. “Arbitrary-degree T-splines for isogeometric analysis of fully nonlinear Kirchhoff–Love shells”. In: *Computer-Aided Design* 82 (2017), pp. 140–153.
- [28] Hugo Casquero et al. “Seamless integration of design and Kirchhoff–Love shell analysis using analysis-suitable unstructured T-splines”. In: *Computer Methods in Applied Mechanics and Engineering* 360 (2020), p. 112765.
- [29] Hugo Casquero et al. “The divergence-conforming immersed boundary method: Application to vesicle and capsule dynamics”. In: *Journal of Computational Physics* 425 (2021), p. 109872.
- [30] Paolo Cignoni, Claudio Montani, and Roberto Scopigno. “Computer-assisted generation of bas-and high-reliefs”. In: *Journal of graphics tools* 2.3 (1997), pp. 15–28.
- [31] David Cohen-Steiner and Jean-Marie Morvan. “Restricted delaunay triangulations and normal cycle”. In: *Proceedings of the nineteenth annual symposium on Computational geometry*. 2003, pp. 312–321.
- [32] Keenan Crane, Mathieu Desbrun, and Peter Schröder. “Trivial connections on discrete surfaces”. In: *Computer Graphics Forum*. Vol. 29. 5. Wiley Online Library. 2010, pp. 1525–1533.
- [33] Fernando De Goes et al. “On the equilibrium of simplicial masonry structures”. In: *ACM Transactions on Graphics (TOG)* 32.4 (2013), pp. 1–10.
- [34] Tony DeRose, Mark Meyer, and Sanjay Bakshi. *Mesh transfer using UV-space*. US Patent 8,482,569. July 2013.
- [35] Shen Dong et al. “Spectral surface quadrangulation”. In: *ACM SIGGRAPH 2006 Papers*. 2006, pp. 1057–1066.

- [36] Michael Eigensatz et al. “Paneling architectural freeform surfaces”. In: *ACM SIGGRAPH 2010 papers*. 2010, pp. 1–10.
- [37] Esri. “ArcGIS Pro”. In: (2019). URL: <https://www.esri.com/zh-cn/arcgis/products/arcgis-pro/resources>.
- [38] Esri. “CityEngine”. In: (2019). URL: <https://www.esri.com/en-us/arcgis/products/esri-cityengine/overview>.
- [39] Esri. “Procedural Runtime Whitepaper”. In: (2013).
- [40] d’ Eon Eugene et al. “8i voxelized full bodies - a voxelized point cloud dataset”. In: *ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) input document WG11M40059/WG1M74006* (2017).
- [41] Andrew W Fitzgibbon. “Robust registration of 2D and 3D point sets”. In: *Image and vision computing* 21.13-14 (2003), pp. 1145–1153.
- [42] James D Foley et al. *Computer graphics: principles and practice*. Vol. 12110. Addison-Wesley Professional, 1996.
- [43] Song Ge, Guoliang Fan, and Meng Ding. “Non-rigid point set registration with global-local topology preservation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2014, pp. 245–251.
- [44] James Glymph et al. “A parametric strategy for free-form glass structures using quadrilateral planar facets”. In: *Automation in construction* 13.2 (2004), pp. 187–202.
- [45] Albert Edward Green and Wolfgang Zerna. *Theoretical elasticity*. Courier Corporation, 1992.
- [46] Gerhard Gröger et al. “OGC city geography markup language (CityGML) encoding standard”. In: (2012).

- [47] Benjamin Hagedorn and Jürgen Döllner. “High-level web service for 3D building information visualization and analysis”. In: *Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems*. ACM. 2007, p. 8.
- [48] Aaron Hertzmann and Denis Zorin. “Illustrating smooth surfaces”. In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. 2000, pp. 517–526.
- [49] IGD. “CityServer3D”. In: (2019). URL: <http://www.cityserver3d.de/en/>.
- [50] Zhongping Ji, Ligang Liu, and Yigang Wang. “B-Mesh: a modeling system for base meshes of 3D articulated shapes”. In: *Computer Graphics Forum*. Vol. 29. 7. Wiley Online Library. 2010, pp. 2169–2177.
- [51] Lalana Kagal et al. “A policy-based approach to governing autonomous behavior in distributed environments”. In: (2004).
- [52] Felix Kälberer, Matthias Nieser, and Konrad Polthier. “Quadcover-surface parameterization using branched coverings”. In: *Computer graphics forum*. Vol. 26. 3. Wiley Online Library. 2007, pp. 375–384.
- [53] KIT. *FZKViewer*. Available online: <https://www.iai.kit.edu/1302.php> (accessed on 10 August 2019). 2019. URL: <https://www.iai.kit.edu/1302.php>.
- [54] Felix Knöppel et al. “Globally optimal direction fields”. In: *ACM Transactions on Graphics (ToG)* 32.4 (2013), pp. 1–10.
- [55] Warner Tjardus Koiter and AMA Van Der Heijden. *WT Koiter’s elastic stability of solids and structures*. Cambridge University Press Cambridge, UK; New York, NY, USA, 2009.

- [56] Thomas H Kolbe, Gerhard Gröger, and Lutz Plümer. “CityGML: Interoperable access to 3D city models”. In: *Geo-information for disaster management*. Springer, 2005, pp. 883–899.
- [57] Nicolas Kowalski, Franck Ledoux, and Pascal Frey. “A PDE based approach to multidomain partitioning and quadrilateral meshing”. In: *Proceedings of the 21st international meshing roundtable*. Springer, 2013, pp. 137–154.
- [58] Yu-Kun Lai et al. “Metric-driven rosy field design and remeshing”. In: *IEEE Transactions on Visualization and Computer Graphics* 16.1 (2009), pp. 95–108.
- [59] Sylvain Lefebvre and Hugues Hoppe. “Appearance-space texture synthesis”. In: *ACM Transactions on Graphics (TOG)* 25.3 (2006), pp. 541–548.
- [60] Xin Li, Xiaodong Wei, and Yongjie Jessica Zhang. “Hybrid non-uniform recursive subdivision with improved convergence rates”. In: *Computer Methods in Applied Mechanics and Engineering* 352 (2019), pp. 606–624.
- [61] Xinghua Liang, Mohamed S Ebeida, and Yongjie Zhang. “Guaranteed-quality all-quadrilateral mesh generation with feature preservation”. In: *Computer Methods in Applied Mechanics and Engineering* 199.29-32 (2010), pp. 2072–2083.
- [62] Xinghua Liang and Yongjie Zhang. “Hexagon-based all-quadrilateral mesh generation with guaranteed angle bounds”. In: *Computer Methods in Applied Mechanics and Engineering* 200.23-24 (2011), pp. 2005–2020.

- [63] Xinghua Liang and Yongjie Zhang. “Matching interior and exterior all-quadrilateral meshes with guaranteed angle bounds”. In: *Engineering with Computers* 28.4 (2012), pp. 375–389.
- [64] Tao Liao, Guoliang Xu, and Yongjie Jessica Zhang. “Atom simplification and quality T-mesh generation for multi-resolution biomolecular surfaces”. In: *Isogeometric Analysis and Applications 2014*. Springer, 2015, pp. 157–182.
- [65] Tao Liao, Guoliang Xu, and Yongjie Jessica Zhang. “Structure-aligned guidance estimation in surface parameterization using eigenfunction-based cross field”. In: *Graphical Models* 76.6 (2014), pp. 691–705.
- [66] Markus Lipp, Peter Wonka, and Pascal Müller. “PushPull++”. In: *ACM Transactions on Graphics (TOG)* 33.4 (2014), p. 130.
- [67] Dong C Liu and Jorge Nocedal. “On the limited memory BFGS method for large scale optimization”. In: *Mathematical programming* 45.1 (1989), pp. 503–528.
- [68] Yang Liu et al. “Computing self-supporting surfaces by regular triangulation”. In: *ACM Transactions on Graphics (TOG)* 32.4 (2013), pp. 1–10.
- [69] Yang Liu et al. “General planar quadrilateral mesh design using conjugate direction field”. In: *ACM Transactions on Graphics (TOG)* 30.6 (2011), pp. 1–10.
- [70] Yang Liu et al. “Geometric modeling with conical meshes and developable surfaces”. In: *ACM SIGGRAPH 2006 Papers*. 2006, pp. 681–689.

- [71] Conglin Lu, Yan Cao, and David Mumford. “Surface evolution under curvature flows”. In: *Journal of Visual Communication and Image Representation* 13.1-2 (2002), pp. 65–81.
- [72] Jiayi Ma et al. “Robust point matching via vector field consensus”. In: *IEEE Transactions on Image Processing* 23.4 (2014), pp. 1706–1721.
- [73] Long Ma et al. “A framework for modeling high quality tension-determined surfaces”. In: *Computers & Graphics* 61 (2016), pp. 50–59.
- [74] Long Ma et al. “Constructing 3D Self-Supporting Surfaces with Isotropic Stress Using 4D Minimal Hypersurfaces of Revolution”. In: *ACM Transactions on Graphics (TOG)* 38.5 (2019), pp. 1–13.
- [75] Giorgio Marcias et al. “Data-driven interactive quadrangulation”. In: *ACM Transactions on Graphics (TOG)* 34.4 (2015), pp. 1–10.
- [76] Mark Meyer et al. “Discrete differential-geometry operators for triangulated 2-manifolds”. In: *Visualization and mathematics III*. Springer, 2003, pp. 35–57.
- [77] Masaaki Miki, Takeo Igarashi, and Philippe Block. “Parametric self-supporting surfaces via direct computation of airy stress functions”. In: *ACM Transactions on Graphics (TOG)* 34.4 (2015), p. 89.
- [78] Pascal Müller et al. “Procedural modeling of buildings”. In: *Acm Transactions On Graphics (Tog)*. Vol. 25. 3. ACM. 2006, pp. 614–623.
- [79] Ashish Myles et al. “Feature-aligned T-meshes”. In: *ACM Transactions on Graphics (TOG)* 29.4 (2010), pp. 1–11.

- [80] Andriy Myronenko and Xubo Song. “Point set registration: Coherent point drift”. In: *IEEE transactions on pattern analysis and machine intelligence* 32.12 (2010), pp. 2262–2275.
- [81] Jonathan Palacios and Eugene Zhang. “Rotational symmetry field design on surfaces”. In: *ACM Transactions on Graphics (TOG)* 26.3 (2007), 55–es.
- [82] Yoav IH Parish and Pascal Müller. “Procedural modeling of cities”. In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM. 2001, pp. 301–308.
- [83] Nico Pietroni et al. “Global parametrization of range image sets”. In: *Proceedings of the 2011 SIGGRAPH Asia Conference*. 2011, pp. 1–10.
- [84] Nico Pietroni et al. “Tracing field-coherent quad layouts”. In: *Computer Graphics Forum*. Vol. 35. 7. Wiley Online Library. 2016, pp. 485–496.
- [85] Helmut Pottmann et al. “Architectural geometry”. In: *Computers & graphics* 47 (2015), pp. 145–164.
- [86] F Prandi et al. “Using CityGML to deploy smart-city services for urban ecosystems”. In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 4 (2013), W1.
- [87] Achilleas Psyllidis et al. “A platform for urban analytics and semantic data integration in city planning”. In: *International conference on computer-aided architectural design futures*. Springer. 2015, pp. 21–36.
- [88] Nicolas Ray et al. “Geometry-aware direction field processing”. In: *ACM Transactions on Graphics (TOG)* 29.1 (2009), pp. 1–11.

- [89] Nicolas Ray et al. “N-symmetry direction field design”. In: *ACM Transactions on Graphics (TOG)* 27.2 (2008), pp. 1–13.
- [90] Nicolas Ray et al. “Periodic global parameterization”. In: *ACM Transactions on Graphics (TOG)* 25.4 (2006), pp. 1460–1485.
- [91] Faniry H Razafindrazaka, Ulrich Reitebuch, and Konrad Polthier. “Perfect matching quad layouts for manifold meshes”. In: *Computer Graphics Forum*. Vol. 34. 5. Wiley Online Library. 2015, pp. 219–228.
- [92] RhinoTerrain. “RhinoCity”. In: (2019). URL: <https://www.rhinoterrain.com/en/rhinocity.html>.
- [93] Robert. “Rhino”. In: (2019). URL: <https://www.rhino3d.com>.
- [94] Emanuele Rodolà et al. “A simple and effective relevance-based point sampling for 3D shapes”. In: *Pattern Recognition Letters* 59 (2015), pp. 41–47.
- [95] Merwin Rook, F Biljecki, and AA Diakité. “Towards Automatic Semantic Labelling of 3D City Models.” In: *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences* 4 (2016).
- [96] Szymon Rusinkiewicz. “Estimating Curvatures and Their Derivatives on Triangle Meshes”. In: (2004).
- [97] Szymon Rusinkiewicz and Marc Levoy. “Efficient variants of the ICP algorithm.” In: *3dim*. Vol. 1. 2001, pp. 145–152.
- [98] SafeSoftware. “CityGML Importer”. In: (2019). URL: <https://www.safe.com/citygml-importer>.
- [99] SAFESOFTWARE. “FME”. In: (2019). URL: <https://www.safe.com/fme/>.
- [100] Robert Sauer. *Differenzengeometrie*. Springer, 1970.

- [101] Xianfang Sun et al. “Bas-relief generation using adaptive histogram equalization”. In: *IEEE transactions on visualization and computer graphics* 15.4 (2009), pp. 642–653.
- [102] Hari Sundar et al. “Skeleton based shape matching and retrieval”. In: *2003 Shape Modeling International*. IEEE. 2003, pp. 130–139.
- [103] Code Synthesis. “CodeSynthesis XSD”. In: (2019). URL: <https://www.codesynthesis.com/>.
- [104] Kenshi Takayama et al. “Sketch-based generation and editing of quad meshes”. In: *ACM Transactions on Graphics (TOG)* 32.4 (2013), pp. 1–8.
- [105] Marco Tarini et al. “Simple quad domains for field aligned mesh parametrization”. In: *Proceedings of the 2011 SIGGRAPH Asia Conference*. 2011, pp. 1–12.
- [106] Stephen P Timoshenko and James M Gere. *Theory of elastic stability*. Courier Corporation, 2009.
- [107] TRIMBLE. “Sketchup”. In: (2019). URL: www.sketchup.com.
- [108] C Truesdell et al. “FD Murnaghan, finite deformation of an elastic solid”. In: *Bulletin of the American Mathematical Society* 58.5 (1952), pp. 577–579.
- [109] UVMSystems. “CityGrid”. In: (2019). URL: <http://www.uvmsystems.com/index.php/en/>.
- [110] Yannick Verdie, Florent Lafarge, and Pierre Alliez. “Lod generation for urban scenes”. In: *ACM Transactions on Graphics* 34.ARTICLE (2015), p. 30.

- [111] Michele Volpi and Vittorio Ferrari. “Structured prediction for urban scene semantic segmentation with geographic context”. In: *2015 Joint Urban Remote Sensing Event (JURSE)*. IEEE. 2015, pp. 1–4.
- [112] Etienne Vouga et al. “Design of self-supporting surfaces”. In: *ACM Transactions on Graphics (TOG)* 31.4 (2012), p. 87.
- [113] Etienne Vouga et al. “Design of self-supporting surfaces”. In: *ACM Trans. Graph.* 31.4 (2012), 87:1–87:11.
- [114] Xiaodong Wei et al. “Blended B-spline construction on unstructured quadrilateral and hexahedral meshes with optimal convergence rates in isogeometric analysis”. In: *Computer Methods in Applied Mechanics and Engineering* 341 (2018), pp. 609–639.
- [115] Xiaodong Wei et al. “Extended truncated hierarchical Catmull–Clark subdivision”. In: *Computer Methods in Applied Mechanics and Engineering* 299 (2016), pp. 316–336.
- [116] Xiaodong Wei et al. “Truncated hierarchical Catmull–Clark subdivision with local refinement”. In: *Computer Methods in Applied Mechanics and Engineering* 291 (2015), pp. 1–20.
- [117] Xiaodong Wei et al. “Truncated T-splines: fundamentals and methods”. In: *Computer Methods in Applied Mechanics and Engineering* 316 (2017), pp. 349–372.
- [118] Xiaodong Wei et al. “Tuned hybrid nonuniform subdivision surfaces with optimal convergence rates”. In: *International Journal for Numerical Methods in Engineering* 122.9 (2021), pp. 2117–2144.
- [119] Chenxia Wu, Ian Lenz, and Ashutosh Saxena. “Hierarchical Semantic Labeling for Task-Relevant RGB-D Perception.” In: *Robotics: Science and systems*. 2014.

- [120] Jin Xie et al. “Deepshape: Deep-learned shape descriptor for 3D shape retrieval”. In: *IEEE transactions on pattern analysis and machine intelligence* 39.7 (2016), pp. 1335–1345.
- [121] Mirko Zadavec, Alexander Schiftner, and Johannes Wallner. “Designing quad-dominant meshes with planar faces”. In: *Computer Graphics Forum*. Vol. 29. 5. Wiley Online Library. 2010, pp. 1671–1679.
- [122] Sen Zhang, Hui Zhang, and Jun-Hai Yong. “Automatic quad patch layout extraction for quadrilateral meshes”. In: *Computer-Aided Design and Applications* 13.3 (2016), pp. 409–416.
- [123] Yu-Wei Zhang et al. “Line-based sunken relief generation from a 3D mesh”. In: *Graphical models* 75.6 (2013), pp. 297–304.
- [124] Yu-Wei Zhang et al. “Real-time bas-relief generation from a 3D mesh”. In: *Graphical Models* 75.1 (2013), pp. 2–9.
- [125] Yongjie Zhang. “Challenges and advances in image-based geometric modeling and mesh generation”. In: *Image-Based Geometric Modeling and Mesh Generation*. Springer, 2013, pp. 1–10.
- [126] Yongjie Jessica Zhang. *Geometric modeling and mesh generation from scanned images*. Chapman and Hall/CRC, 2018.
- [127] Yongjie Zhang and Chandrajit Bajaj. “Adaptive and quality quadrilateral/hexahedral meshing from volumetric data”. In: *Computer methods in applied mechanics and engineering* 195.9-12 (2006), pp. 942–960.
- [128] Zhengyou Zhang. “Iterative point matching for registration of free-form curves and surfaces”. In: *International journal of computer vision* 13.2 (1994), pp. 119–152.

- [129] Qing Zhu et al. “Robust point cloud classification based on multi-level semantic relationships for urban scenes”. In: *ISPRS journal of photogrammetry and remote sensing* 129 (2017), pp. 86–102.