Shape reconstruction of meshed smooth surfaces equipped with inertial sensors

Tibor Stanko

Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Mathématiques–Informatique**

Arrêté ministériel : 25 mai 2016

Présentée par

Tibor STANKO

Thèse dirigée par **Stefanie HAHMANN**,
co-dirigée par **Georges-Pierre BONNEAU**
et co-encadrée par **Nathalie SAGUIN-SPRYNSKI**

préparée au sein du **CEA-Leti** et **LJK / INRIA**
dans **l'École Doctorale MSTII**

Reconstruction de surfaces lisses maillées à partir de capteurs inertiels

Shape reconstruction of meshed smooth surfaces equipped with inertial sensors

Thèse soutenue publiquement le 8 décembre 2017
devant le jury composé de :

Marc DANIEL
Professeur, Université d'Aix-Marseille, président

Mario BOTSCH
Professeur, Universität Bielefeld, rapporteur

Adrien BOUSSEAU
Chargé de recherche, HDR, Inria Sophia-Antipolis, rapporteur

Stefanie HAHMANN
Professeure, Grenoble INP, directrice de thèse

Georges-Pierre BONNEAU
Professeur, Université Grenoble Alpes, co-directeur de thèse

Nathalie SAGUIN-SPRYNSKI
Docteure, CEA-Leti, co-encadrante de thèse

Université Grenoble Alpes

Shape reconstruction of meshed smooth surfaces equipped with inertial sensors

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

by

Tibor Stanko

December 2017

To my family

Acknowledgements

Whatever it is you're seeking won't come in the form you're expecting.

— H. Murakami

Three years ago I left my home country, seeking new challenges in foreign lands. Those three years turned out to be an extraordinary experience, and I have many people to thank for it.

Un grand merci goes to my advisors. Stefanie, I remember our first discussion on Skype like it was yesterday. You always had your door open for me – thank you for all the advice and your constant support, even that time I missed my plane to San Francisco! Georges-Pierre, thank you for being so eloquent and rigorous while keeping the mood light – no meeting is dull if you're there! Nathalie, thank you for all your help, especially within CEA. Without you, we would never have had the real lilium!

I am grateful to Mario and Adrien, who took the time to review the manuscript. I also thank Marc for his role as the president of the jury.

During my PhD, I had the privilege of being a part of two research teams, each of them unique in its own way.

At CEA, I came to know to the world of inertial sensors and signal processing, and spent countless coffee breaks with my colleagues and fellow PhD students: Andres, Arun, Edgar, Ibou, Kersane, Matthias (*ça va, la thèse?*), Matthias, Maxime, Nils, Régis, Victor, Vita. A special thanks goes to Rémy and Tina, for all the hikes we went to and the dinners we shared.

My time spent at Inria was limited, but always inspiring – being a part of Imagine under Marie-Paule's leadership was a priceless experience. My thanks go to Camille and Thomas, my office mates; Robin, with whom I co-organized the external seminars; Maxime, for his help with the moving; Amélie, my fellow Berkeley traveler; and Lucas, for his work on *ShapeIt*.

My family is the main reason I love coming back to Slovakia. I am forever indebted to my parents, who have supported my efforts ever since I can remember.

I couldn't have done this without the love and understanding of Lujza, and Janko, who has the ability to lighten even the darkest of days.

Abstract

This thesis presents a complete framework for 3D shape reconstruction using inertial and magnetic sensors. When placed onto a shape, these sensors provide local surface orientations along a curve network on the shape, but their absolute position in the world space is unknown. The challenges with this type of 3D acquisition are threefold. First, sensor measurements are noisy and inconsistent. Second, since positions are unknown, the acquired curve network has to be reconstructed from orientations. Finally, the smooth surface needs to be inferred from a collection of curves with normals. To compute the shape from measured data, our main insight is to formulate the reconstruction as a set of optimization problems. Using discrete representations, these optimization problems are resolved efficiently and at interactive time rates.

We present two main contributions. First, we introduce a novel method for creating well-connected networks with cell-complex topology using only orientation and distance measurements and a set of user-defined constraints. By working directly with orientations, our method robustly resolves problems arising from data inconsistency and sensor noise. Our approach is driven by a simple principle mostly overlooked in previous works: at each intersection in a curve network, the positions and the normals of two intersecting curves have to coincide.

Second, we address the problem of surfacing a closed 3D curve network with given surface normals. Thanks to the normal vector input, the patch-finding problem can be solved unambiguously and an initial piecewise smooth triangle mesh is computed. The input normals are propagated throughout the mesh. Together with the initial mesh, the propagated normals are used to estimate mean curvature vectors. We then compute the final mesh by combining the standard Laplacian-based variational methods with the curvature information extracted from the input normals. The normal input increases shape fidelity and allows to achieve globally smooth and visually pleasing shapes.

Previous approaches used static devices placed along a network with fixed connectivity between the sensors (ribbon, grid). We explore a new dynamic setup, which uses a single mobile node of sensors. As a consequence, a dense set of data can be acquired along an arbitrary smooth curve network on a surface.

The proposed framework was tested on real-world data acquired using two devices equipped with mobile sensors. A quantitative evaluation was performed by computing the error of reconstruction for fabricated surfaces with known ground truth. Even for complex shapes, the mean error remains around 1%.

keywords: *3D shape reconstruction — inertial and magnetic sensors — curve networks — smooth surfaces — variational modeling — mesh processing*

Résumé

Cette thèse porte sur le développement de méthodes pour la reconstruction de formes 3D à l'aide de capteurs inertiels et magnétiques. Lorsqu'ils sont placés sur une forme, ces capteurs fournissent des orientations locales de surface mais leur position absolue dans l'espace 3D est inconnue. Les dispositifs que nous considérons dans cette thèse produisent des orientations locales de surface le long d'un réseau de courbes. Reconstruire des formes 3D à l'aide de telles données pose trois types de défis. Tout d'abord, les mesures des capteurs sont bruitées et incohérentes. Deuxièmement, comme les positions sont inconnues, le réseau de courbes acquis doit être reconstruit à partir des orientations. Enfin, une fois le réseau de courbes reconstruit, il est nécessaire de calculer une surface lisse interpolant ce réseau de courbes et les orientations associées. Pour relever ces défis, on formule les différentes étapes de reconstruction comme un ensemble de problèmes d'optimisation. En utilisant des représentations discrètes, ces problèmes sont résolus efficacement et interactivement.

Nous présentons deux contributions principales. Tout d'abord, nous introduisons une méthode produisant un réseau de courbes lisses et cohérentes en utilisant les mesures d'orientation et de distance, ainsi qu'un ensemble de contraintes topologiques fournies par l'utilisateur. Notre méthode se base notamment sur une procédure de lissage des orientations motivée par un principe simple: les positions et les normales des courbes doivent coïncider en chaque intersection d'un réseau.

Une fois le réseau de courbes reconstruit, nous proposons une méthode permettant de calculer une surface lisse interpolant ce réseau de courbes, ainsi que les orientations associées. Cette méthode a trois étapes. Tout d'abord grâce aux orientations, les cycles de courbes entourant les patchs surfaciques sont déterminés sans ambiguïté. Ensuite les orientations connues le long des courbes sont propagées à travers le maillage initial et utilisées pour estimer la courbure moyenne. Enfin le maillage final est calculé par une méthode basée sur le Laplacien et utilisant l'information de courbure. Les orientations connues sur le réseau de courbes permettent d'obtenir des maillages lisses et de diminuer les erreurs de reconstruction.

Les approches précédentes utilisaient des dispositifs statiques placés le long d'un réseau de connectivité fixe entre les capteurs (ruban, grille). Nous explorons dans cette thèse une nouvelle configuration dynamique, consistant à déplacer un dispositif ponctuel sur la surface. En conséquence, il est possible d'acquérir des données le long d'un réseau arbitraire de courbes lisses sur une surface. Les méthodes proposées dans cette thèse ont été testées sur des données réelles acquises avec ces dispositifs mobiles. Des surfaces physiques fabriquées à partir de modèles numériques nous ont permis de faire une évaluation quantitative en calculant l'erreur de reconstruction entre la vraie surface et notre modèle reconstruit. Même pour des formes complexes, l'erreur moyenne reste autour de 1%.

mots clés: *reconstruction des formes 3D — capteurs inertiels et magnétiques — réseaux de courbes — surfaces lisses — modélisation variationnelle — traitement de maillages*

x

Contents

1 Introduction **1**
1.1 Shape acquisition 2
1.1.1 Issues of optical scanners 3
1.2 Shape from sensors 5
1.3 State of the art 8
1.3.1 Curves 8
1.3.2 Surfaces 12
1.3.3 Limitations 13
1.4 Context, input data & scanning devices 14
1.5 Contributions 15

2 Foundations **19**
2.1 Space curves 22
2.2 Regular surfaces 23
2.3 Curves on surfaces 26
2.4 Smooth manifolds 29
2.5 Rotations in 3D 34
2.5.1 Rotations as special orthonormal matrices 34
2.5.2 Riemannian structure of SO(3) 35

2.5.3 Rotations as unit quaternions . . . 36
2.5.4 Spherical linear interpolation . . . 38
2.6 Cell complexes . . . 38
2.7 Discrete Laplacian . . . 41
2.7.1 Discretization via finite elements . . . 41
2.7.2 Alternative definitions . . . 46
2.7.3 Matrix form and higher-order Laplacians . . . 46
2.8 Variational modeling of shapes . . . 47

3 Curve networks from orientations . . . 51
3.1 Related work . . . 53
3.1.1 Shape from curves . . . 53
3.1.2 Orientation estimation and filtering . . . 54
3.1.3 Poisson reconstruction . . . 55
3.2 Problem statement . . . 55
3.3 Method overview . . . 57
3.4 Estimation of orientations . . . 59
3.4.1 Davenport's q method . . . 60
3.4.2 Application to data from IMUs . . . 63
3.5 Filtering of orientations . . . 63
3.5.1 Moving average filter . . . 65
3.5.2 Algorithm for computing the average orientation . . . 65
3.5.3 Pre-filtering by Gaussian convolution . . . 69
3.5.4 Smoothing via regression on SO(3) . . . 71
3.6 Poisson network reconstruction . . . 75
3.7 Evaluation . . . 77
3.7.1 Performance . . . 78
3.7.2 Filtering . . . 81
3.7.3 Convergence & error measurements . . . 83
3.7.4 Morphorider vs. smartphone . . . 87
3.8 Conclusion . . . 88
Figures & Tables: Network reconstruction error . . . 89

4 Surfacing networks with normals . . . 103
4.1 Related work . . . 105
4.1.1 Variational modeling with normals . . . 105
4.1.2 Surfacing curve networks . . . 108
4.1.3 Multi-sided patches . . . 110
4.2 Problem statement . . . 111
4.3 Method overview . . . 112

4.4 Computing the topology . . . 114
4.4.1 Detection of network cycles . . . 115
4.4.2 Tessellation of cycles . . . 116
4.5 Computing the geometry . . . 119
4.5.1 Initial mesh and propagation of normals . . . 120
4.5.2 Mean curvature estimation . . . 121
4.5.3 Optimization . . . 124
4.5.4 Hard constraints vs. soft constraints . . . 125
4.6 Evaluation . . . 127
4.6.1 Normal control . . . 127
4.6.2 Convergence & error measurements . . . 129
4.6.3 Comparison with Laplacian methods . . . 131
4.6.4 Comparison with the flow-aligned surfacing . . . 133
4.7 Conclusion . . . 134

5 Applications . . . 137
5.1 Experiments with fabricated surfaces . . . 139
5.1.1 Fabrication of lilium . . . 140
5.2 Acquisition & implementation . . . 141
5.2.1 Morphorider . . . 141
5.2.2 Reconstruction GUI . . . 145
5.2.3 Smartphone . . . 145
5.2.4 ShapeIt . . . 146
5.2.5 Sketching of virtual 3D objects . . . 147
5.3 Reconstruction of physical surfaces . . . 149
5.3.1 Acquisition of everyday objects . . . 149
5.3.2 Measuring the error . . . 153
5.3.3 Conclusion . . . 154
Figures & Tables: Surface reconstruction error . . . 156

6 Conclusion . . . 167
6.1 Summary of contributions . . . 168
6.2 Future work . . . 169

A Riemannian connection and covariant derivative . . . 171

B Notes on quaternion filtering . . . 173

C G^1 interpolation of discrete networks with normals . . . 177
C.1 Hermite interpolation and curvature energy . . . 178
C.2 Network of Hermite splines . . . 179

C.3 Optimization 180
C.4 Constraining the normals 183

Publications 187

Bibliography 189

Figures, Tables & Code

1.1 Taxonomy of shape acquisition techniques 2
1.2 Acquisition of a car body using a portable handheld scanner . . . 4
1.3 Real-time scene reconstruction using a single RGB-D camera . . . 4
1.4 Personalized anatomy model from medical imaging data 4
1.5 Reconstruction of a Buddha statue using a smartphone 4
1.6 Issues of optical scanners 5
1.7 Earth's gravitational field and magnetic field 6
1.8 Gravity variation. Magnetic field direction 6
1.9 Razor Inertial Measurement Unit (IMU) 7
1.10 *(Table)* Overview of shape-from-sensors devices and algorithms . . 9
1.11 Schema of 2D curve reconstruction from sensor data 11
1.12 Demonstrators developed at CEA-Leti [Spr+07a; Sag+14] 12
1.13 Triangulation of n-sided developable surface patch [Hua+13] . . . 13
1.14 Acquisition devices: Morphorider and smartphone 15

1.15 Morphorider setup and limitations 15

2.1 Parametrization of a regular surface 24
2.2 Sphere with six charts. Gauss map, elliptic and hyperbolic surface . 25
2.3 Frenet and Darboux frames 27
2.4 Exponential map 33
2.5 *(Table)* Toolbox for the special orthogonal group SO(3) 35
2.6 Antipodal points on the 3-sphere 38
2.7 Curve network with topology of a cell complex 40
2.8 Notation for triangle meshes 41
2.9 Hat function and gradient 43
2.10 Mixed Voronoi area 43
2.11 *(Algorithm)* Computation of the mixed Voronoi area 43
2.12 Gradient discretization on triangle mesh 44

3.1 Recent curve acquisition devices using inertial sensors 54
3.2 Examples of invalid input networks 56
3.3 Discontinuity of a curve network orientation function 57
3.4 Overview of our network reconstruction method 58
3.5 *(Code)* Computation of orientation from measured vectors 64
3.6 Gaussian kernel for orientation pre-filtering 70
3.7 Euclidean smoothing spline with varying stretching and bending . 72
3.8 Matrix of the discrete Poisson system for a cone network 76
3.9 *(Table)* Performance stats 79
3.10 *(Code)* Regression on SO(3) implemented using Manopt 80
3.11 Lilium from raw, pre-filtered, filtered orientations 81
3.12 Varying filtering weights, lilium network 82
3.13 Chair reconstructed with different sets of weights 83
3.14 Comparison of three acquisition setups, lilium network 86
3.15 Network error, sphere & torus, synthetic 89

3.16 Network error, bowl & bumpycube, synthetic 90
3.17 Network error, gamepad & lilium, synthetic 91
3.18 Network error, cone, Morphorider 92
3.19 Network error, lilium (5 curves), smartphone, estimated lengths 94
3.20 Network error, lilium (5 curves), smartphone, measured lengths 96
3.21 Network error, lilium (5 curves), Morphorider 98
3.22 Network error, lilium (7 curves), Morphorider 100

4.1 Surfacing of curve networks is an ill-posed problem 104
4.2 Three pipes [BK04] 107
4.3 Surface deformation via mixed finite elements [Jac+10] 107
4.4 Methods for surfacing sketched curve networks [Abb+13] 109
4.5 Our setup for surfacing of a curve network 111
4.6 Overview of our network surfacing method 112
4.7 Discrete representation for surfacing 113
4.8 Illustration of the cycles detection algorithm 115
4.9 *(Algorithm)* Cycles detection 115
4.10 Detected cycles in beetle 116
4.11 Initial tessellation 118
4.12 Comparison of three base geometries 121
4.13 Integral definition of the mean curvature normal 122
4.14 Mean curvature of a horse mesh, comparison 124
4.15 Surfacing noisy torus network, hard vs. soft constraints 126
4.16 Smooth surfaces reconstructed using our method 126
4.17 Surfacing of networks with winding curves 126
4.18 Influence of input normals 127
4.19 Surfacing error for lilium networks from Fig. 3.14 128
4.20 Convergence of our surfacing method upon refinement 129
4.21 Comparison with related methods 130

4.22 Reconstruction of sphere and torus 130
4.23 Comparison with [Pan+15] 132
4.24 Surface quality inspection with normal mapping 133

5.1 Example results, reconstruction & sketching 138
5.2 *(Table)* Dimensions of physical ground truth surfaces 140
5.3 Lilium B-spline surface from triangle mesh 140
5.4 Physical surfaces used as ground truth in our tests 141
5.5 *(Code)* Opening a serial connection in Matlab under Linux 142
5.6 *(Code)* Rawnet file for storing network orientations and distances . . 142
5.7 Matlab acquisition and reconstruction programs 144
5.8 C++ implementation of our reconstruction framework 144
5.9 Android applications for data acquisition with a smartphone . . . 144
5.10 Example of a surface acquired with ShapeIt 147
5.11 Mushroom sketched with a smartphone 148
5.12 Sailboat sketched with a smartphone 148
5.13 Surfaces interpolating the chair networks from Fig. 3.13 150
5.14 Guitar, Morphorider reconstruction 151
5.15 Baby bathtub, Morphorider reconstruction 151
5.16 Roof box reconstructed with different sets of curves 152
5.17 Surface error, cone, Morphorider 156
5.18 Surface error, lilium (5 curves), smartphone, estimated lengths 158
5.19 Surface error, lilium (5 curves), smartphone, measured lengths 160
5.20 Surface error, lilium (5 curves), Morphorider 162
5.21 Surface error, lilium (7 curves), Morphorider 164

B.1 Quaternions with fixed normal, parametric space 175

C.1 Examples of input data for Hermite spline interpolation 184
C.2 Hermite spline networks with tangent plane continuity 185

1 Introduction

We describe a framework for 3D shape reconstruction using micro-sensors. Recent research enabled novel techniques for 3D acquisition of physical objects based on inertial sensors (accelerometers, gyroscopes) and magnetic sensors (magnetometers). Today, these techniques already serve as an alternative to traditional acquisition methods, forming the basis for a recently launched start-up [Morphosense].

Most state-of-the-art methods in sensor shape reconstruction focus on a *static* setup, in which a fixed array of sensors is placed on the surface of the scanned object. In this thesis, we explore a *dynamic* setup – a fixed array of static sensors is replaced by a single moving sensor node which acts as a *virtual sensor network*. Such dynamic setup is versatile and free of assumptions that would considerably limit topology and geometry of the scanned shape.

Using new kinds of acquisition devices, we propose novel algorithms for reconstruction of scanned curves and surfaces. Our *shape-from-sensors* framework is designed for use with dynamic devices. Nevertheless, the proposed algorithms are backed by rigorous mathematical formulations and are directly extensible to data coming from static devices.

In the dynamic setup, we suppose the scanned object is *rigid*, i.e. it is not being deformed during acquisition – otherwise, the dynamic setup makes no sense. All the algorithms are formulated with this hypothesis in mind, and we do not explore what happens when the object deforms over time.

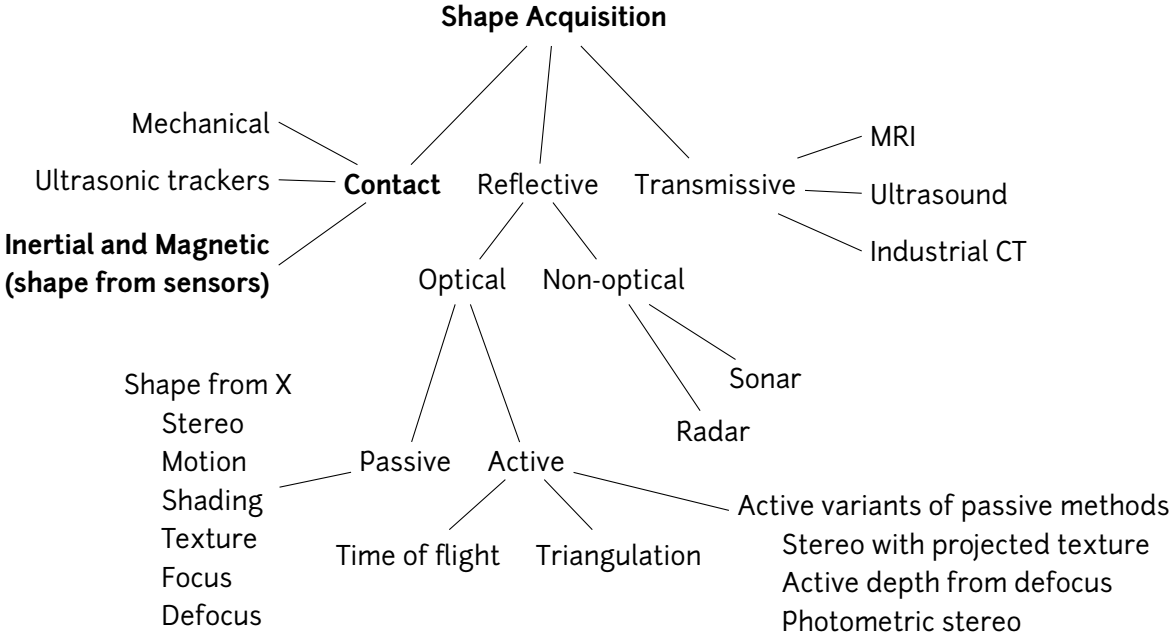


Figure 1.1: *Shape from sensors* in the taxonomy of 3D acquisition techniques [Cur97; CS00; Rus16; Wim16].

1.1 Shape acquisition

Physical properties of real-world objects (shape, color, texture) are acquired using *3D scanners*. This umbrella term encompasses a multitude of devices, ranging from coordinate measuring machines (CMMs) to magnetic resonance imaging (MRI). 3D reconstruction has numerous applications, for instance digitization of cultural heritage [Lev+03; Tau08; Ike+07; Stanford3D], medical scanning [Bau+16; ABP16], monitoring of large structures (e.g. buildings or bridges) [Sag+16; Morphosense], and real-time geometry reconstruction for augmented/mixed/virtual reality [Dou+16; Ort+16; New+11; Guo+17; Tka+17]. See the examples on p. 4.

Most 3D scanners used in vision and graphics are *optical*. Optical scanners use data from optical sensors to measure shapes, and typically employ a set of cameras, projectors and lasers [Roc+01]. A particular configuration of scanner's components is tightly linked with the scanning technique used for inferring the depth information.

See Fig. 1.1 for the taxonomy of scanning techniques and Fig. 1.2 for an example of a structured light scanner.

Regardless of which device setup and acquisition technique are used, the output of the scanning process is either a point cloud or a range image – the latter is a height function (a set of depth values) sampled over a regular lattice (typically Euclidean or cylindrical). Data processing pipeline is similar in both cases. First, multiple scans taken from different directions and/or with varying object orientation need to be aligned or *registered*. The registration can be done semi-manually, using a set of markers (Fig. 1.2); or automatically, via a registration algorithm such as the iterative closest point (ICP) [BM92]. Second, reconstruction algorithms are applied on the registered data to compute the final shape. For range images, common reconstruction algorithms are the zippering [TL94] and the volumetric range image processing (VRIP) [CL96]. Poisson surface reconstruction [KBH06; KH13] is commonly used with point clouds. For a detailed comparison of point-cloud-based algorithms, see the recent survey of Berger et al. [Ber+14].

With rise of affordable devices such as Kinect [New+11], shape acquisition became ubiquitous in recent years (Figs. 1.3 and 1.4). Researchers now focus on the development of online calibration and reconstruction methods using data from RGB or RGB-D cameras [Guo+17; Tka+17]. Even smartphones are starting to act as 3D scanners, making shape acquisition more accessible than ever before (Fig. 1.5) [Tan+13; Mur+16].

Still, there is no universal 3D scanner suitable for every application. The choice of a device is influenced by the concrete application in mind, which explains the diversity of shape acquisition devices and techniques (Fig. 1.1). New kinds of acquisition devices [Abe+17] and surface reconstruction methods [BL17; Sch+17] are constantly emerging.

1.1.1 Issues of optical scanners

Though widely used, optical 3D scanners are not a universal tool – there are situations in which these devices cannot be used. This section summarizes common limitations, grouped into three categories – environment, object, and deformation.

Environment. Data measured by optical sensors strongly depend on the environment in which they are captured. First, appropriate light conditions are crucial for a correct acquisition. Excluding specialized scanners such as LiDAR, most optical devices are limited to indoor acquisition with a controlled lighting. Furthermore, since a scanner needs to be controlled by a human operator, traditional scanners do not enable acquisition in hostile environments: high altitudes, underground, dangerous places. Examples of acquisition in hostile environments include structural

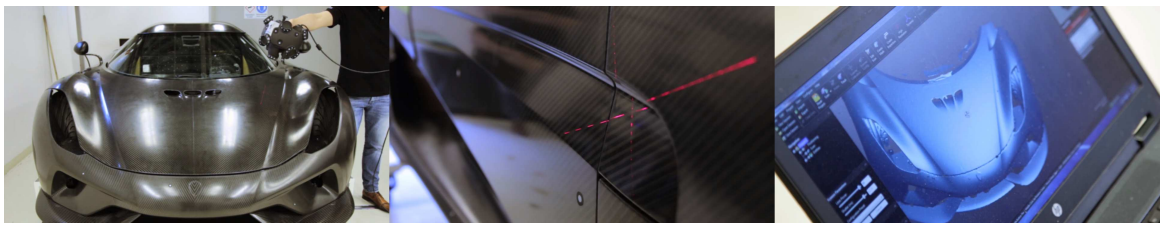


Figure 1.2: Acquisition of a car body using a portable handheld scanner. The scanner uses active stereo and a laser cross to reconstruct the depth. Individual scans are registered via manually-placed markers – white dots on the car body. Image from [Creaform].

Figure 1.3: Real-time scene reconstruction using a single RGB-D camera. Image from [Guo+17].

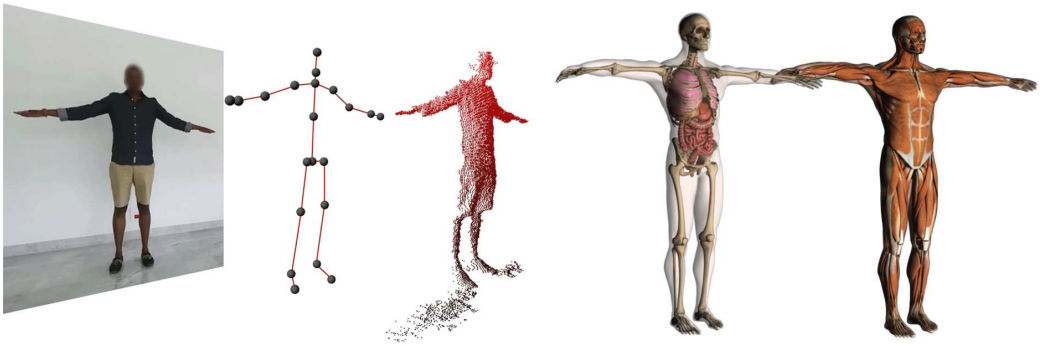


Figure 1.4: Personalized anatomy model computed using medical imaging data (MRI/CT scans). Image from [Bau+16].

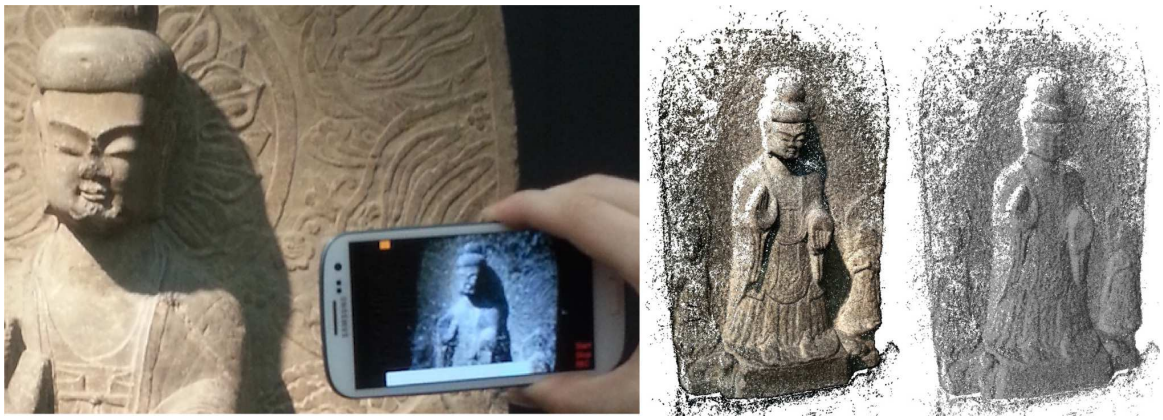


Figure 1.5: Reconstruction of a Shakyamuni Buddha statue (height: 1.6 meters) using a smartphone. Image from [Tan+13].

(a) ©pgasston@flickr **(b)** ©pasfam@flickr **(c)** Image from [Abe+17].

Figure 1.6: Issues of optical scanners. **(a-b)** Traditional 3D scanning methods cannot be used with large, inaccessible, moving and/or deforming objects. **(c)** Optical scanner reconstruction of a self-occluding object with high genus fails to recover the interior part.

health monitoring of a wind turbine (Fig. 1.6a) or leak detection in underground pipes [Sag+16].

Object. Optical scanners do not scale well. Small objects are not difficult to acquire, but the acquisition gets more complicated and time-consuming with increasing object size. Furthermore, there are limitations on materials that can be scanned. Diffuse materials work well, specular and transparent materials are challenging or even impossible to handle. Problems due to self-occlusions are minimized by scanning the object from various angles, but often cannot be resolved completely (Fig. 1.6c).

Deformation. Though some optical systems are capable of reconstructing deforming and non-rigid geometry (Fig. 1.3), deformation tracking remains a challenging task. For an example, consider the problem of reconstructing the surface of a sail moving and deforming in the wind (Fig. 1.6b). Due to many self-occlusions and varying lighting, this problem is unsolvable using optical methods. It requires a different approach – for instance, by instrumenting the fabric of the sail with sensors.

To overcome the limitations of traditional 3D scanners, one typically needs to use alternative shape acquisition tools (Fig. 1.1). One such alternative, which we introduce in the following section, is to exploit orientation data from inertial and magnetic sensors.

1.2 Shape from sensors

This section describes how inertial and magnetic sensors can be used for 3D acquisition of shapes.

Principle. Attached to an object, inertial and magnetic sensors measure vectors in a local coordinate frame. *Accelerometer* – when stationary – measures the direction

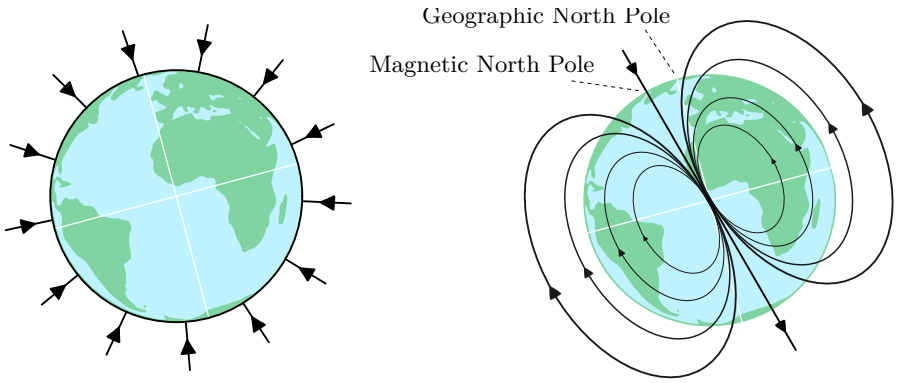
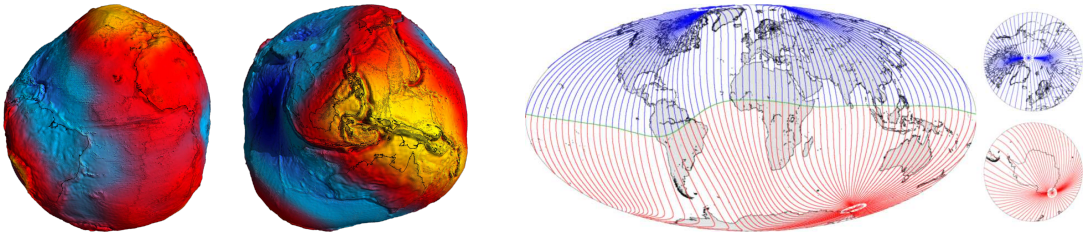


Figure 1.7: Earth's gravitational field (*left*) and magnetic field (*right*).



(a) Image from [ESA17]. **(b)** Image from [Mau10].

Figure 1.8: (a) Variation in the strength of the gravitational field and **(b)** the direction of the magnetic field, data from 2010.

$\mathbf{e}_{\text{acc}}$ of acceleration due to Earth's gravitational field, which is the vector pointing straight upwards (Fig. 1.7 left). *Magnetometer* positioned far enough from any ferromagnetic object measures the direction $\mathbf{e}_{\text{mag}}$ of the geomagnetic field, which is the vector pointing towards the magnetic North pole (Fig. 1.7 right). *Gyroscope* measures object's angular velocity.

Note that the vectors $\mathbf{e}_{\text{acc}}$ and $\mathbf{e}_{\text{mag}}$ depend on the location and the time of measurement. The geomagnetic field and the gravitational field vary around the Earth due to various anomalies and constant movement of magnetic poles.[1]

Fig. 1.8a shows the variation in the gravitational field visualized as a *geoid*. Using this technique, the magnitude of the relative deviation is visualized as a colored height field over Earth's surface. For practical purposes, gravitational variations are negligible, and the gravity vector is assumed to be constant.

Fig. 1.8b shows how the direction of the magnetic field varies over the surface of the Earth. These variations are not negligible, and data measured by magnetometers have to be calibrated in order to consider the location of measurement. At the

[1] According to recent surveys, the North Magnetic Pole is moving approximately north-northwest at 55 km per year [Nai06].

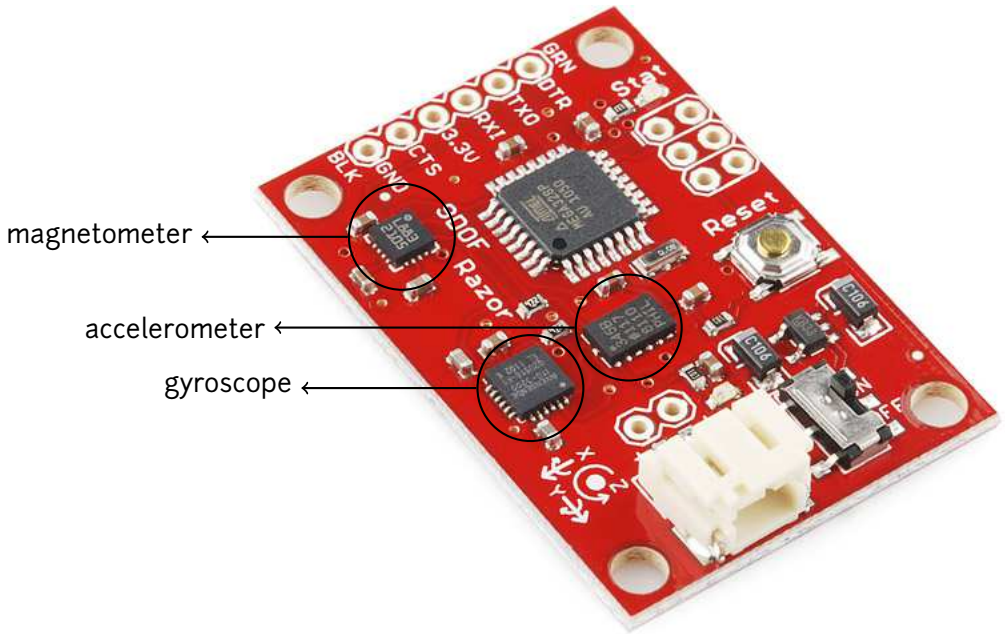


Figure 1.9: The 9DOF Razor Inertial Measurement Unit (IMU) featuring a triple-axis gyroscope, a triple-axis accelerometer and a triple-axis magnetometer. Image from [Sparkfun].

end of Section 3.4, we describe the specific reference vectors used for orientation acquisition in Grenoble, France.

Hypotheses. To reliably measure local orientation of the object for the purposes of shape reconstruction, the following two conditions are assumed to be true [HS08]:

1. *Device acceleration is negligible with respect to gravity acceleration.*

2. *There is no considerable magnetic field except for the Earth's magnetic field.*

Inertial measurement unit (IMU). In 3D, a single measured vector is not sufficient to determine object's orientation. Two linearly independent vectors are needed to fix all rotational degrees of freedom. Accelerometers cannot determine rotation around the vertical axis (vector $\mathbf{e}_{\text{acc}}$). Similarly, magnetometers cannot determine rotation around the vector $\mathbf{e}_{\text{mag}}$. To estimate object's full rotation, the sensor measurements have to be combined.

In practice, sensors are assembled into a microelectromechanical system (MEMS) called *inertial measurement unit* or IMU (Fig. 1.9). A typical IMU features a combination of a triple-axis accelerometer, a triple-axis gyroscope and a triple-axis magnetometer. Here, *triple-axis* refers to the fact that the sensor measures data along three orthogonal axes, and outputs a single three-dimensional vector. We refer to

devices equipped with IMUs as *sensor devices*. An example of a sensor device is a smartphone.

Shape acquisition with sensors. Unlike traditional 3D scanners, sensor devices do not measure the spatial position of points on a shape. Rather, they measure the local *orientation* of a shape. In order to retrieve the position, orientations have to be coupled with geodesic distance between adjacent measurements.

We refer to this as the problem of *shape from sensors*. The main goal of this thesis is to adopt a novel view on this problem and present new reconstruction algorithms. Before introducing our setup, let us have look at the state of the art.

1.3 State of the art

In this section, we look at previous methods that focus on resolving the shape-from-sensors problem introduced in the previous section.

The pioneering work in this domain was done in the thesis of Nathalie Sprynski [Spr07]. Since then, many other methods have followed, improving on the initial work or introducing new approaches; see the overview in Table 1.10. We present a summary of the state of the art, along with technical challenges and proposed solutions.

Certain parts of the exposition are formulated using mathematical notions defined in the next chapter, specifically in Sections 2.1 to 2.5. We believe this is not an issue; methods presented in this section serve as an overview of what has already been done, and we skip most technical details.

This section only summarizes methods dealing with the problem of shape from sensors. Other methods related to the individual parts of our framework are included in the subsequent chapters (Section 3.1 and Section 4.1).

1.3.1 Curves

The use of sensors for shape acquisition was first explored by Sprynski [Spr07]. Since the sensors do *not* measure the absolute position of points in the world space, the reconstruction algorithms need to be formulated in terms of orientations provided by sensors and geodesic distances between samples (known *a priori* or measured). Curves are represented using natural parametrization and reconstructed via numerical integration. Surfaces are defined via geodesic interpolation [Spr+08] or using parallel ribbons of sensors [Sag+14]. Huard et al. [Hua+13] introduced a method for computing smooth patches from a given piecewise geodesic boundary curve. Hoshi and Shinoda [HS08] reconstructed the target surface using two

Shape from sensors: state of the art

Acquisition

	device	sensors	structure
[Spr+07a] [SLB11]	Morphosense	fixed	ribbon
[Car+15] [Sag+16]	Morphopipe	fixed	ribbon
[ABP16]	spine tracker	fixed	ribbon
[HS08]	3DCS	fixed	lattice
[Sag+14]	Morphoshape	fixed	lattice
[HCG16]	instrumented fabric	fixed	lattice
Chapter 5	Morphorider, smartphone	dynamic	single IMU, virtual network

Curve reconstruction

	topology	principle
[Spr+07b] [Spr+07a]	single curve	spherical interpolation of tangents, num. integration
[Hua+14]	single curve	Pyhagorean-hodograph curves
Chapter 3	curve network	filtering in SO(3), Poisson rec. with fixed topology

Surface reconstruction

	topology	principle
[Spr+08]	'parallel' curves	geodesic interp. along parallel curv. directions
[HS08]	lattice	surface as lattice, fixed link length and flexible joints
[SLB11]	lattice	Coons patches
[Sag+14]	lattice	same as [Spr+08]
[HCG16]	lattice	piecewise-linear approximation of surface patches
[Hua+13]	n-sided patch	geodesic interpolation in parametric space
Chapter 4	curve network	mesh-based variational approach with Laplacian energy

Table 1.10: State of the art in 3D shape acquisition using inertial and magnetic sensors. Prior research focused on devices with sensors organized in fixed structures (ribbon, lattice) . The algorithms presented in this thesis (last rows) enable acquisition with dynamic sensors, e.g. by sliding a smartphone along curves on the scanned surface.

families of sensors placed in orthogonal directions. Hermanis et al. [HCG16] constructed sensor-instrumented fabric and compared their reconstruction results with a Kinect reconstruction. Antonya et al. [ABP16] used an array of sensors for real-time tracking of the human spine.

In this line of research, the first and foremost problem that needs to be addressed is the following: given a sequence of orthonormal frames along a curve with known geodesic distances between adjacent frames, how to reconstruct the curve? The basic idea is to represent the curve via arc-length parametrization, do the reconstruction in the tangent space, and perform numerical integration to get positions in the world space. Building on the curve reconstruction, surfaces are specified via networks of curves.

The following approach is due to Sprynski [Spr07]. Consider a planar curve γ with length L, parametrized by arc length as

$$\mathbf{x} : [0, L] \to \mathbb{R}^2.$$

Since $\mathbf{x}$ is the arc-length parametrization, the tangent vector $\mathbf{t} = \mathbf{x}'$ is unit and can be represented by the oriented angle α between $\mathbf{t}$ and the horizontal axis:

$$\mathbf{x}' = \begin{pmatrix} \cos\alpha \\ \sin\alpha \end{pmatrix}.$$

The basic problem of curve reconstruction from sensor data is formulated as follows. Let $\alpha_i \in [0, 2\pi)$ be a sequence of $k+1$ tangent angles sampled from an unknown planar curve along a sequence of parameters (Fig. 1.11 top)

$$0 = s_0 < s_1 < \cdots < s_k = L.$$

Find a differentiable planar curve $\mathbf{x} : [0, L] \to \mathbb{R}^2$ such that

$$\mathbf{x}'(s_i) = (\cos\alpha_i,\ \sin\alpha_i)^\mathsf{T}. \tag{1.1}$$

Sprynski [Spr07] proposed to find a continuous angle function $\alpha : [0, L] \to \mathbb{R}$, which interpolates the given samples: $\alpha(s_i) = \alpha_i$. The curve $\mathbf{x}$, which satisfies Eq. (1.1), is defined via

$$\mathbf{x}'(s) = (\cos\alpha(s),\ \sin\alpha(s))^\mathsf{T}.$$

The continuous function $\alpha(s)$ is computed using a natural cubic spline (Fig. 1.11 bottom) [HF02]. In the additional pre-processing step, one needs to make sure the neighboring angles are not too distant from each other in the Euclidean sense: this is done by requiring $|\alpha_{i+1} - \alpha_i| < \pi$ and adding an integer multiple of 2π to α_{i+1} if needed.

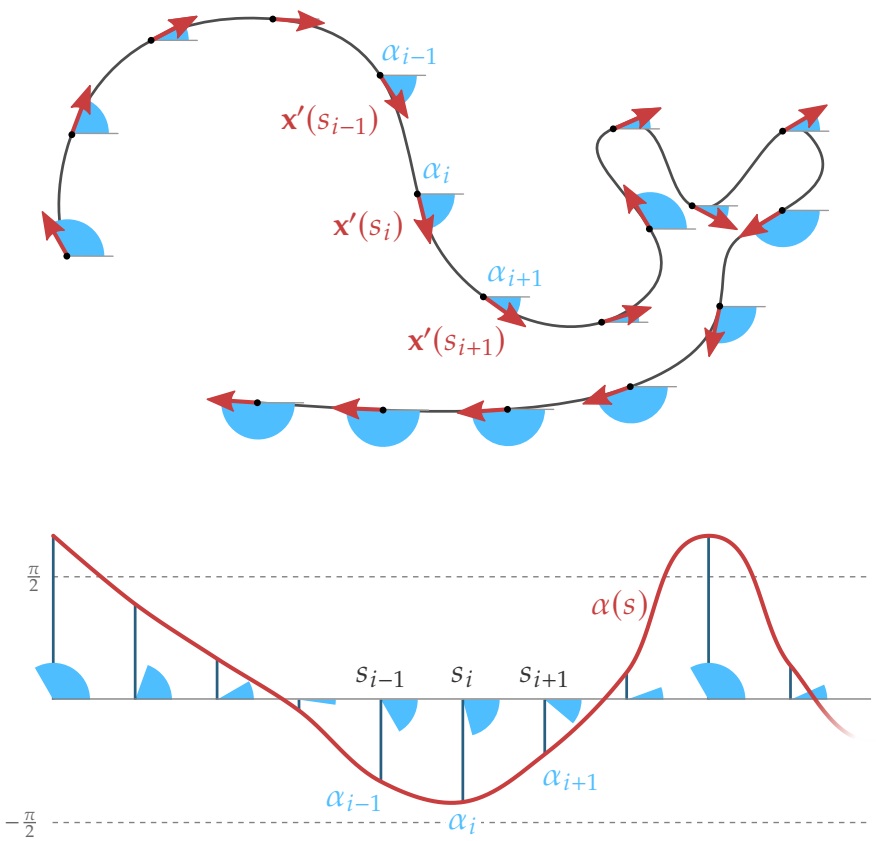


Figure 1.11: Schema of 2D curve reconstruction from sensor data

After the angle function $\alpha(s)$ has been estimated, the translational degree of freedom is fixed by setting the initial position $\mathbf{x}_0 = \mathbf{x}(0)$ and the discrete curve points are reconstructed by numerical integration:

$$\mathbf{x}(s) = \mathbf{x}_0 + \begin{pmatrix} \int_0^s \cos\alpha(s)\ \mathrm{d}s \\ \int_0^s \sin\alpha(s)\ \mathrm{d}s \end{pmatrix}.$$

The above approach can be directly extended to space curves by representing the tangent $\mathbf{x}' = \mathbf{t}$ using spherical coordinates and reconstructing both angle functions. This however does not yield satisfactory results. Better results are obtained using splines on the unit sphere [Nie04]. These splines can be evaluated efficiently using Bézier representation and a modified version of the De Casteljau's algorithm, replacing the linear interpolation with the *spherical linear interpolation* [Sho85]:

$$\mathrm{Slerp}(\mathbf{t}_0, \mathbf{t}_1, t) = \frac{\sin\left[(1-t)\theta\right]}{\sin\theta}\mathbf{t}_0 + \frac{\sin\left[t\theta\right]}{\sin\theta}\mathbf{t}_1, \tag{1.2}$$

where $\mathbf{t}_0, \mathbf{t}_1$ are unit tangent vectors with $\cos\theta = \mathbf{t}_0 \cdot \mathbf{t}_1$. Huard [Hua13] introduced

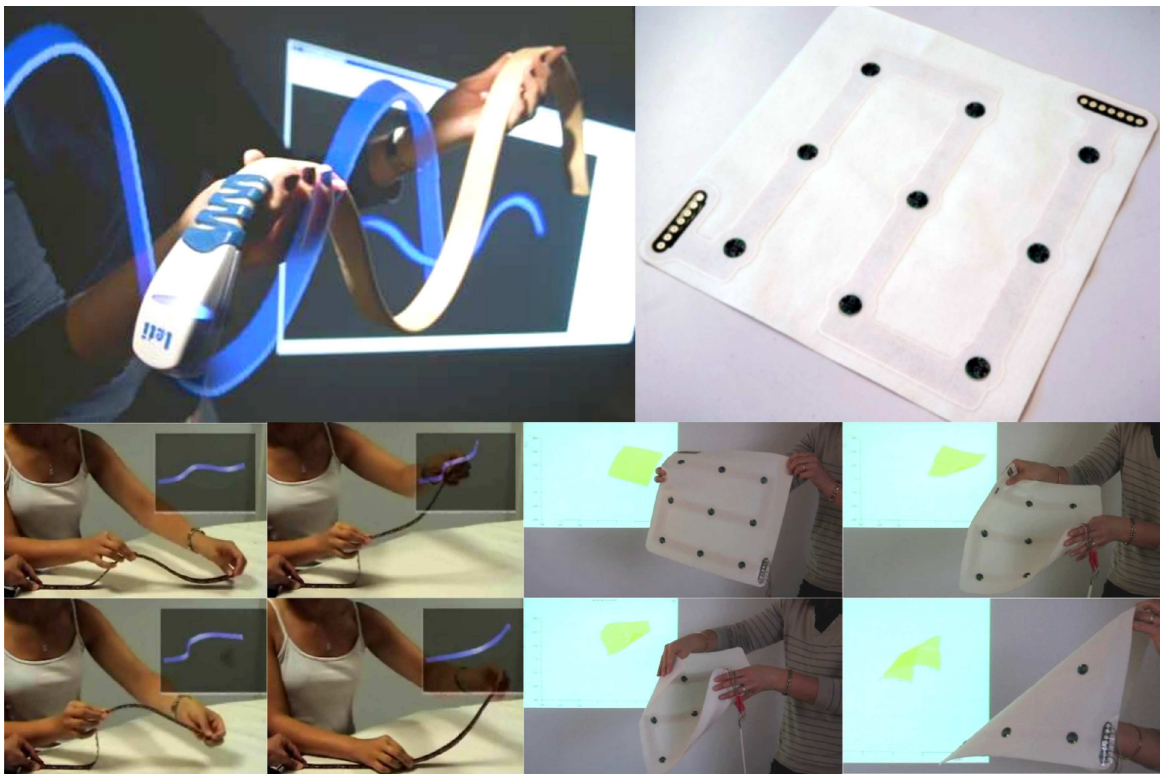


Figure 1.12: Some of the demonstrators developed at CEA-Leti: the Morphosense ribbon [Spr+07a] (*left*) and the Morphoshape patch [Sag+14] (*right*).

a modified algorithm of Nielson [Nie04] using Hermite interpolation on the sphere exploiting the knowledge of surface normals along the acquired curves.

Methods that reconstruct the curve in the tangent space (angle function $\alpha(s)$, spherical splines) require numerical integration to retrieve the actual curve points in $\mathbb{R}^2$ or $\mathbb{R}^3$. Huard et al. [Hua+14] proposed an alternative formulation for space curves using *Pythagorean-hodograph* quintic splines. A Pythagorean-Hodograph curve $\mathbf{x}$ is characterized by the fact that its hodograph $\mathbf{x}' = \mathbf{x}'(s)$ satisfies the Pythagorean relation $\|\mathbf{x}'\| = \sigma$ for some polynomial $\sigma = \sigma(s)$. This formulation allows reconstructing the curve in its analytic form, but is significantly slower compared to the methods using numerical integration.

1.3.2 Surfaces

Sprynski [Spr07] also introduced the first algorithms for surface reconstruction and deformation in the context of shape from sensors. She considered curve networks consisting of two families of parallel curves orthogonal to each other (Fig. 1.12 (*right*)). Such networks define a collection of quadrilateral surface regions, which can be modeled via Coons patches [Coo67; Far02]. When working with actual

sensor-acquired curves, obtaining a closed network with proper curve intersections is challenging if each curve is reconstructed individually. Various heuristics for aligning the curves were proposed by Sprynski [Spr07].

Huard [Hua13] focused on reconstruction of *developable surfaces*, which are isometric to a planar region (and thus have zero Gaussian curvature). First, he presented an algorithm for reconstruction of a developable surface $\mathcal{S}$ from a single geodesic curve γ, exploiting the fact that a developable surface $\mathcal{S}$ is also a ruled surface [Hua+12]. A *ruled surface* is parametrized by

$$\mathbf{x}(t, v) = \mathbf{p}(t) + v\mathbf{d}(t), \quad t \in I,\ v \in \mathbb{R},$$

where $\{\mathbf{p}(t) \in \mathbb{R}^3,\ \mathbf{d}(t) \in V(\mathbb{R}^3)\}_{t \in I}$ is a one-parameter family of lines [doCa16]. For a fixed t, $L_t = \mathbf{x}(t, v)$ is a straight line called a *ruling*; the curve $\mathbf{p}(t)$ is called a *directrix* of $\mathcal{S}$. The knowledge of a single geodesic curve $\gamma \subset \mathcal{S}$ enables reconstruction of the portion of $\mathcal{S}$ whose rulings intersect γ.

Second, Huard [Hua13] presented a method for reconstruction of an n-sided region on a quasi developable surface, delimited by a closed sequence of geodesic curves (Fig. 1.13) [Hua+13]. The main idea is to split the n-sided patch into a collection of triangular patches by estimating the missing geodesics. The missing geodesics are estimated using the fact that a geodesic on a developable surface corresponds to a straight line in plane under isometric mapping. The resulting triangular patches are interpolated with C^1 parametric interpolants.

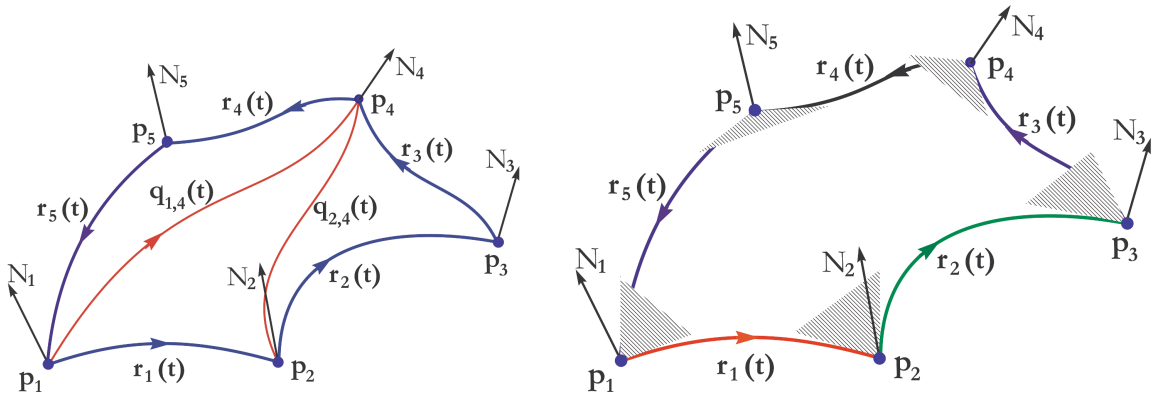


Figure 1.13: In Huard et al. [Hua+13], the n-sided region on a developable surface is split into triangular patches by estimating interior geodesics. Image from [Hua+13].

1.3.3 Limitations

Previous works on shape from sensors used devices with a fixed connectivity and known distances between sensor nodes. The sensors were organized uniformly on a 1D ribbon (Fig. 1.12 left) or on a regular 2D grid (Fig. 1.12 right). Multiple ribbons

were employed to acquire networks of curves on a surface, using a comb structure [SLB11] or patches with boundary from geodesic curves [Hua+13]. However, these ribbons cannot be placed on the surface arbitrarily as they will inevitably follow geodesic curves. Grids of sensor nodes are limited to specific types of surfaces (e.g. developable surfaces isometric to planar sheets). While most existing methods work in real time, they adopt various heuristics to glue the acquired curves together in order to have a closed network with proper topology.

1.4 Context, input data & scanning devices

We explore a novel *dynamic* setup for acquisition of orientations with sensor-instrumented devices. In our setup, a device with a single IMU is moved along a *virtual* network of curves on the scanned surface, and the data are acquired interactively. Until now, this type of acquisition setup using sensors has not been studied.

Our dynamic setup has several advantages compared to the existing methods. Since a dynamic device can be moved freely on the surface, the topology of the curve network is no longer fixed, in contrast to a static setup. This allows us to acquire more curve networks and reconstruct a broader family of shapes than what is possible with the existing methods. Moreover, in our dynamic setup, the size of the scanned object is not limited by the acquisition device.

Data acquired with dynamic devices typically have much higher sampling density than data acquired using static devices. We exploit this fact to improve the estimated orientations by filtering the dense sample with our custom methods. Our reconstruction algorithms benefit from this and result in accurate reconstruction of the acquired shape.

The two devices that we use are a standard smartphone, and a custom-made device called Morphorider. In this section, a short description is given for each of the two devices. Concrete details on how the devices work and how the data are acquired will be given in Chapter 5.

Morphorider (Fig. 1.14 left and Fig. 1.15) is a wireless MEMS-based device for measuring local orientation, containing a single IMU (with a tri-axial accelerometer and a tri-axial magnetometer) and an odometer for tracking distance. It is a prototype specifically designed for dynamic acquisition of curves on surfaces.

Similarly, orientation data can be measured with a smartphone (Fig. 1.14 right). In addition to an accelerometer and a magnetometer, a smartphone also contains a gyroscope. This enables orientation acquisition that is more robust to magnetic perturbations. We use a state of the art sensor fusion algorithm to robustly estimate orientation from sensor measurements [MHV11; Pac13]. On the other hand, measuring distances with a smartphone is difficult. GPS does not provide the stability

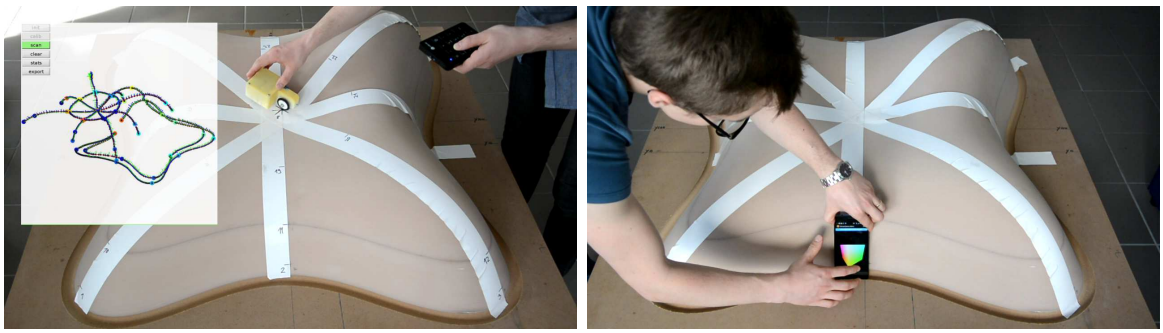


Figure 1.14: Two dynamic devices used for data acquisition in this thesis. (*Left*) a custom-made device called Morphorider and a screenshot of our MATLAB acquisition interface. (*Right*) a standard smartphone, showing the Android application with simple acquisition interface.

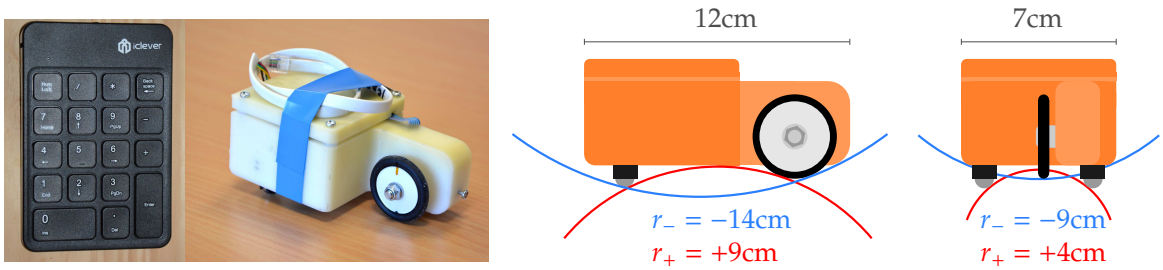


Figure 1.15: (*Left*) Morphorider with a wireless numpad used for recording the topology (indices of nodes) during acquisition. (*Right*) limitations of the device with respect to radius of curvature at locally convex (red) and concave (blue) surface points.

and the precision needed for our algorithms. To overcome this limitation, we assume the speed of acquisition is constant, then we parametrize each curve using acquisition timestamps. Optionally, manual distance measurements can be used to increase the precision of reconstruction.

1.5 Contributions

In this thesis, we propose a novel *shape-from-sensors* framework. To the best of our knowledge, we present the first reconstruction framework developed for data from dynamic sensor devices (Section 1.4). In this section we outline the core ideas that guide our approach, and we give an overview of the main contributions of this thesis.

Our approach for computing shapes from the measured data is to formulate the reconstruction as a set of optimization problems. Using discrete representations of shapes (polylines, meshes), the optimization problems are resolved efficiently and

at interactive time rates. Constrained optimization and discrete formulation are the two main ingredients, differentiating our methods from the existing approaches (Section 1.3). Existing methods mostly use continuous representation and spline-based techniques, which limits the family of shapes that can be reconstructed (see the summary of limitations in Section 1.3.3).

The result is a unified reconstruction framework with two main parts described in the core chapters of this thesis.

In Chapter 3, we describe the first part of the framework, which is dedicated to reconstruction of a smooth curve network from acquired orientations and distances. By working directly with orientations, our method robustly resolves problems arising from data inconsistency and sensor noise. Our approach is driven by a simple principle mostly overlooked in previous works: at each intersection in a curve network, the positions and normals of the two intersecting curves have to coincide. Guided by this idea, we develop automatic network reconstruction procedures that enforce the constraints at intersections by design.

In Chapter 4, we present the second part of the framework, which is an original method for surfacing a well-connected and smooth curve network with surface normals.

Thanks to the normal vector input, the patch-finding problem can be solved unambiguously and an initial piecewise smooth triangle mesh is computed. The input normals are propagated throughout the mesh. Together with the initial mesh, the propagated normals are used to estimate mean curvature vectors. We then compute the final mesh by combining the standard Laplacian-based variational methods with the curvature information extracted from the input normals. The normal input increases shape fidelity and allows to achieve globally smooth and visually pleasing shapes.

In Chapter 5, we show how we combine the algorithms for network reconstruction (Chapter 3) and network surfacing (Chapter 4) in order to acquire digital models of physical shapes. We describe the test surfaces used for the evaluation, as well as the process of data acquisition. In addition to reconstruction, we demonstrate how the same framework can be used to sketch 3D shapes using nothing but a smartphone.

The reconstruction methods that we introduce were originally designed for use with dynamic devices (Section 1.4). Nevertheless, our framework is not limited to the dynamic setup, and could also be used in combination with fixed-topology devices without modification. Possible applications are countless and include smart materials [Hua+13], shape digitization [Sta+17a], medical scanning [HCG16] and Structural Health Monitoring (SHM) [Sag+14].

Remarks on terminology. Throughout the thesis, we sometimes use the term *scanning* to describe the acquisition of shapes. By this, we do not mean the usual 3D acquisition associated with optical 3D scanners, but rather the acquisition of orientations and distances using one of our devices.

Likewise, the meaning of the term *reconstruction* might slightly vary, but the meaning is always clear from the context. In Chapter 3, reconstruction refers to the computation of a curve network. In Chapter 5, reconstruction refers to the whole process of computing a smooth surface from sensor data.

2

Foundations

We focus on curves on surfaces immersed in the Euclidean space $\mathbb{R}^3$. This chapter establishes theoretical background for working with such objects. The common denominator is the concept of a smooth manifold: Darboux frames from the group of rotations (3-manifold) are used to describe a curve network (collection of 1-manifolds) on a surface (2-manifold).

This chapter is organized as follows. Sections 2.1 to 2.3 summarize important facts about curves and surfaces in $\mathbb{R}^3$, respectively. Section 2.4 generalizes the concept of manifold immersed in n-dimensional Euclidean space and introduces essential notions from Riemannian geometry. Section 2.5 describes rotations in three dimensions as points on the manifold SO(3). Section 2.6 introduces the concept of a cell complex, which we use to define curve networks and triangle meshes. Section 2.7 describes discretization of the Laplace-Beltrami operator for triangle meshes. Section 2.8 summarizes variational methods for shape modeling.

Let us start by introducing the notation and conventions used throughout the thesis. $\mathbb{R}^n$ denotes both the n-dimensional vector space over real numbers and the n-dimensional Euclidean space with the usual metric. We adopt the column vector convention: a point $\mathbf{x} \in \mathbb{R}^n$ is a n-tuple $(x_1, \ldots, x_n)^\intercal$. The standard dot product

$\mathbf{u}^\intercal \mathbf{v}$ for two vectors in $\mathbb{R}^n$ is denoted by a dot $\mathbf{u} \cdot \mathbf{v}$ or by angle brackets $\langle \mathbf{u}, \mathbf{v} \rangle$. The canonical basis in $\mathbb{R}^3$ is denoted by

$$\mathbf{e}_1 = (1,0,0)^\intercal, \qquad \mathbf{e}_2 = (0,1,0)^\intercal, \qquad \mathbf{e}_3 = (0,0,1)^\intercal.$$

Let $\mathbf{u} = (u_x, u_y, u_z)^\intercal, \mathbf{v} = (v_x, v_y, v_z)^\intercal, \mathbf{w} = (w_x, w_y, w_z)^\intercal \in \mathbb{R}^3$. The cross product of two vectors is defined as

$$\mathbf{u} \times \mathbf{v} = \det \begin{pmatrix} \mathbf{e}_1 & u_x & v_x \\ \mathbf{e}_2 & u_y & v_y \\ \mathbf{e}_3 & u_z & v_z \end{pmatrix} = \mathbf{e}_1 \det \begin{pmatrix} u_y & v_x \\ u_z & v_z \end{pmatrix} - \mathbf{e}_2 \det \begin{pmatrix} u_x & v_x \\ u_z & v_z \end{pmatrix} + \mathbf{e}_3 \det \begin{pmatrix} u_x & v_x \\ u_y & v_y \end{pmatrix}.$$

It follows that $\mathbf{e}_1 \times \mathbf{e}_2 = \mathbf{e}_3$. The mixed product of three vectors is defined as

$$\mathbf{w} \cdot \mathbf{u} \times \mathbf{v} = \det \begin{pmatrix} w_x & u_x & v_x \\ w_y & u_y & v_y \\ w_z & u_z & v_z \end{pmatrix} = w_x \det \begin{pmatrix} u_y & v_x \\ u_z & v_z \end{pmatrix} - w_y \det \begin{pmatrix} u_x & v_x \\ u_z & v_z \end{pmatrix} + w_z \det \begin{pmatrix} u_x & v_x \\ u_y & v_y \end{pmatrix}.$$

If the vectors $\mathbf{u}$ and $\mathbf{v}$ are unit and orthogonal, the orthonormal frame $\{\mathbf{u}, \mathbf{v}, \mathbf{u} \times \mathbf{v}\}$ is right-handed and positively oriented.

The trace of a square $n \times n$ matrix $\mathrm{A} = (a_{ij})$ is defined as the sum of elements on the main diagonal, and is equal to the sum of eigenvalues,

$$\operatorname{trace}(\mathrm{A}) = \sum_{i=1}^{n} a_{ii} = \sum_{i=1}^{n} \sigma_i.$$

For two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, the standard dot product is related to the matrix trace by the following relations:

$$\operatorname{trace}\left(\mathbf{x}\mathbf{y}^\intercal\right) = \mathbf{x}^\intercal \mathbf{y} = \mathbf{x} \cdot \mathbf{y} = \mathbf{y}^\intercal \mathbf{x} = \operatorname{trace}\left(\mathbf{y}\mathbf{x}^\intercal\right).$$

The cross product matrix operator $[\mathbf{u}\times]$ is defined via the relation $\mathbf{u} \times \mathbf{v} = [\mathbf{u}\times]\,\mathbf{v}$. It holds that

$$[\mathbf{u}\times] = \begin{pmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{pmatrix}.$$

The matrix $[\mathbf{u}\times]$ is an example of a *skew-symmetric* or *antisymmetric* matrix. In general, a square matrix A is skew-symmetric if its transpose equals its negative,

$$\mathrm{A} - \mathrm{A}^\intercal = 0.$$

Note that the trace of a skew-symmetric matrix always vanishes since its diagonal is zero.

The Frobenius inner product of two $m \times n$ matrices $\mathrm{A} = (a_{ij}), \mathrm{B} = (b_{ij}) \in \mathbb{R}^{m \times n}$ is

$$\langle \mathrm{A}, \mathrm{B} \rangle_{\mathrm{F}} = \operatorname{trace}\left(\mathrm{A}^{\mathsf{T}} \mathrm{B}\right) = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}\, b_{ij} \tag{2.1}$$

and induces the Frobenius norm:

$$\|\mathrm{A}\|_{\mathrm{F}} = \sqrt{\langle \mathrm{A}, \mathrm{A} \rangle_{\mathrm{F}}} = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2}. \tag{2.2}$$

Both I and $\mathrm{I_n}$ will denote the $n \times n$ identity matrix. $0_{m \times n}$ or 0 denotes the $m \times n$ zero matrix and $\mathbf{0} = 0_{m \times 1}$ is the zero vector.

Let $U \subset \mathbb{R}^m$ and $V \subset \mathbb{R}^n$ be open sets. Consider a mapping f from U to V:

$$\begin{array}{rccc} f & : & U \subset \mathbb{R}^m & \longrightarrow & V \subset \mathbb{R}^n \\ & & \mathbf{x} = (x_1, \ldots, x_m)^{\mathsf{T}} & \longmapsto & f(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_n(\mathbf{x}))^{\mathsf{T}} \end{array}$$

- f is *continuous* or *of class* C^0 if all components f_i are continuous.
- f is *differentiable* if all partial derivatives $\partial f / \partial x_j$ exist.
- f is *continuously differentiable* if f is differentiable and all partial derivatives $\partial f / \partial x_j$ are continuous.
- f is *of class* C^k if all partial derivatives $\partial f / \partial x_j$ are of class C^{k-1}.
- f is *of class* C^∞ if all partial derivatives $\partial^k f / \partial x_{i_1} \ldots \partial x_{i_k}$ exist and are continuous for any k.

For a one-parameter curve $\mathbf{x} : I \subset \mathbb{R} \to \mathbb{R}^3$, the prime and the double prime denote first and second spatial derivatives:

$$\mathbf{x}' = \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}s}, \qquad \mathbf{x}'' = \frac{\mathrm{d}^2\mathbf{x}}{\mathrm{d}s^2}.$$

For a two-parameter surface $\mathbf{x} : \Omega \subset \mathbb{R}^2 \to \mathbb{R}^3$, the partial derivatives are denoted by subscripts:

$$\mathbf{x}_u = \frac{\partial \mathbf{x}}{\partial u}, \qquad \mathbf{x}_{uv} = \frac{\partial^2 \mathbf{x}}{\partial u \partial v}.$$

2.1 Space curves

Definition 2.1 (Curve, length)**.** *A* parametric space curve γ *is an image of a differentiable map*

$$\mathbf{x} : [a, b] \subset \mathbb{R} \to \mathbb{R}^3 : \quad s \mapsto \mathbf{x}(s) = \big(x(s), y(s), z(s)\big)^\intercal .$$

The map $\mathbf{x}$ *is called a* parametrization *of* γ*. The derivative of the curve defines the tangent field* $\mathbf{t} = \mathbf{x}'$*. The curve* γ *is called* regular *iff the tangent field never vanishes, i.e. iff* $\mathbf{x}' \neq 0$ *for all* $s \in [a, b]$*.*

For a regular curve, the length of the segment between the starting point $\mathbf{x}(a)$ *and a point* $\mathbf{x}(s)$ *on the curve is*

$$l(s) = \int_a^s \big\|\mathbf{x}'(t)\big\| \, \mathrm{d}t.$$

The total length of γ *is* $L = l(b)$*.*

Definition 2.2 (Natural parametrization)**.** *A regular curve* γ *is parametrized by its arc length if* $l(s) = s - a$ *for all* $s \in [a, b]$*. Setting* $a = 0$ *then yields the unique* natural parametrization

$$\mathbf{x} : [0, L] \subset \mathbb{R} \to \gamma \subset \mathbb{R}^3.$$

Since $l(s) = \int_0^s \big\|\mathbf{x}'(t)\big\| \, \mathrm{d}t = s$, the derivative of the natural parametrization has unit length everywhere:

$$\big\|\mathbf{x}'(s)\big\| = \big\|\mathbf{t}(s)\big\| \equiv 1 \quad \text{for all } s \in [0, L].$$

The natural parametrization is an isomorphism and is also called the *arc-length* or the *unit-speed* parametrization. Intuitively, a curve parametrized by the arc length is an image of the line segment $[0, L]$ with no stretching or squishing. Any regular curve can be reparametrized by the arc length.

Unless stated otherwise, we assume that all curves are regular and given by the natural parametrization.

Proposition 2.3. *For an arc-length parametrized curve* γ*, the vectors* $\mathbf{t} = \mathbf{x}'$ *and* $\mathbf{t}' = \mathbf{x}''$ *are orthogonal.*

Proof. Differentiating the relation $\|\mathbf{t}\|^2 = \mathbf{t} \cdot \mathbf{t} = 1$ once yields $\mathbf{t} \cdot \mathbf{t}' = 0$. □

Definition 2.4 (Frame)**.** A frame of the curve γ *is an orthonormal frame* $\mathcal{A} = \{\mathbf{t}, \mathbf{n}_1, \mathbf{n}_2\}$ *defined by the unit tangent* $\mathbf{t} = \mathbf{x}'$ *and the choice of the vector* $\mathbf{n}_1 \perp \mathbf{t}$ *varying smoothly along the curve. The frame is then completed by* $\mathbf{n}_2 = \mathbf{t} \times \mathbf{n}_1$*.*

Proposition 2.5 (Framing formulas). *The following relations hold:*

$$\begin{pmatrix} \mathbf{t}' \\ \mathbf{n}_1' \\ \mathbf{n}_2' \end{pmatrix} = \begin{pmatrix} 0 & \kappa_1 & \kappa_2 \\ -\kappa_1 & 0 & \tau \\ -\kappa_2 & -\tau & 0 \end{pmatrix} \begin{pmatrix} \mathbf{t} \\ \mathbf{n}_1 \\ \mathbf{n}_2 \end{pmatrix}, \tag{2.3}$$

where $\mathbf{t}' = \kappa_1 \mathbf{n}_1 + \kappa_2 \mathbf{n}_2$ *is the* curvature vector *of* γ*, and the* torsion τ *measures the twisting of the orthonormal frame* $\mathcal{A}$*.*

If the curvature vector $\mathbf{t}' = \mathbf{x}''$ never vanishes, we can write it in terms of the *principal normal* $\mathbf{n}$ and the curvature $\kappa = 1/r$ as $\mathbf{t}' = \kappa \mathbf{n}$. Here, r is the radius of the osculating circle at $\gamma(s)$, which is locally the closest second-order approximation of the curve (see Fig. 2.3 left, p. 27). Intuitively, the curvature κ measures the deviation of γ from a straight line; the torsion τ measures the deviation of γ from a planar curve.

Definition 2.6 (Frenet-Serret frame). The Frenet-Serret frame $\mathcal{F} = \{\mathbf{t}, \mathbf{n}, \mathbf{b}\}$ *is given by*

$$\mathbf{t} = \mathbf{x}', \qquad \mathbf{n} = \mathbf{x}''/\left\|\mathbf{x}''\right\|, \qquad \mathbf{b} = \mathbf{t} \times \mathbf{n}.$$

The vector $\mathbf{b}$ *is called the* binormal. *Substitution into Eq.* (2.3) *yields the* Frenet-Serret formulas

$$\begin{pmatrix} \mathbf{t}' \\ \mathbf{n}' \\ \mathbf{b}' \end{pmatrix} = \begin{pmatrix} 0 & \kappa & 0 \\ -\kappa & 0 & \tau \\ 0 & -\tau & 0 \end{pmatrix} \begin{pmatrix} \mathbf{t} \\ \mathbf{n} \\ \mathbf{b} \end{pmatrix}. \tag{2.4}$$

At inflection points ($\kappa = 0$), the Frenet-Serret frame is not defined. Among the infinity of frames which exist for the curve γ, the Frenet-Serret frame arises from the orthogonality of the first two derivatives of the curve ($\mathbf{x}' \cdot \mathbf{x}'' = 0$, cf. Proposition 2.3) and naturally reflects the local differential properties of the curve.

One of the central concepts in this thesis is the *Darboux frame*, which is useful for studying curves on surfaces. Before we define it in Section 2.3, let us formalize the notion of a regular surface.

2.2 Regular surfaces

Definition 2.7 (Regular surface in $\mathbb{R}^3$). *A subset* $\mathcal{S} \subset \mathbb{R}^3$ *is called a* regular surface *if for each point* $\mathbf{p} \in \mathcal{S}$*, there exists a neighborhood* $V \subset \mathbb{R}^3$ *of* $\mathbf{p}$ *and a differentiable homeomorphism of an open set* $U \subset \mathbb{R}^2$ *(Fig. 2.1)*

$$\mathbf{x} : U \to V \cap \mathcal{S} : \quad (u, v) \mapsto \mathbf{x}(u, v),$$

such that the Jacobian matrix

$$\mathrm{J}(\mathbf{x})(u, v) = \begin{pmatrix} \frac{\partial \mathbf{x}}{\partial u} & \frac{\partial \mathbf{x}}{\partial v} \end{pmatrix} = \begin{pmatrix} \mathbf{x}_u & \mathbf{x}_v \end{pmatrix}$$

has rank 2 for all $(u, v) \in U$. *The map* $\mathbf{x}$ *is called a* parametrization *of the region* $V \cap \mathcal{S}$.

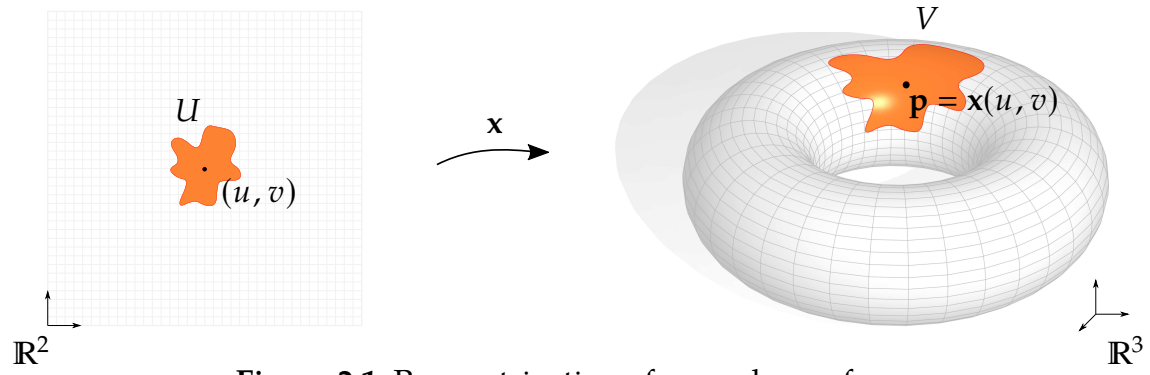


Figure 2.1: Parametrization of a regular surface

Fixing a point $\mathbf{x} \in \mathcal{S}$, the local properties of the surface are studied via the Gauss map $\mathbf{N}$.

Definition 2.8 (Gauss map. Tangent plane. Shape operator)**.** Gauss map *associates a surface point* $\mathbf{x}$ *with the unit surface normal* $\mathbf{N}$ *at* $\mathbf{x}$, *represented by a point on the unit sphere* $\mathbb{S}^2$ *(Fig. 2.2 right). If the surface is locally parametrized by* $\mathbf{x} = \mathbf{x}(u, v)$, *Gauss map can be written explicitly as*

$$\mathbf{N} : \mathcal{S} \to \mathbb{S}^2 : \quad \mathbf{x} \mapsto \mathbf{N}(\mathbf{x}) = \frac{\mathbf{x}_u \times \mathbf{x}_v}{\|\mathbf{x}_u \times \mathbf{x}_v\|}. \tag{2.5}$$

The tangent plane *of* $\mathcal{S}$ *at* $\mathbf{x}$, *denoted by* $\mathrm{T}_\mathbf{x}\mathcal{S}$, *is the two-dimensional orthogonal complement of the one-dimensional vector space induced by* $\mathbf{N}$*:*

$$\mathrm{T}_\mathbf{x}\mathcal{S} = \left\{ \mathbf{t} \in \mathbb{R}^3 \ : \ \mathbf{t} \perp \mathbf{N}(\mathbf{x}) \right\}$$

The shape operator *– also called the Weingarten map – is the differential of the Gauss map* $\mathbf{N} = \mathbf{N}(\mathbf{x})$ *with negative sign:*

$$\mathbf{S} : \mathrm{T}_\mathbf{x}\mathcal{S} \to \mathrm{T}_\mathbf{N}\mathbb{S}^2 : \quad \mathbf{x} \mapsto -\mathrm{d}\mathbf{N}. \tag{2.6}$$

The tangent spaces $\mathrm{T}_\mathbf{x}\mathcal{S}$ and $\mathrm{T}_\mathbf{N}\mathbb{S}^2$ are naturally identified since they are parallel planes in $\mathbb{R}^3$. Therefore, the shape operator at $\mathbf{x}$ can be seen as an *endomorphism* acting on the tangent space $\mathrm{T}_\mathbf{x}\mathcal{S}$ – it is a linear map from $\mathrm{T}_\mathbf{x}\mathcal{S}$ to itself.

Local bending of the surface is measured by curvatures.

Definition 2.9 (Normal curvature. Principal curvatures and directions)**.** *For every unit direction* $\mathbf{t}$ *in the tangent plane* $\mathrm{T}_\mathbf{x}\mathcal{S}$ *at a fixed point* $\mathbf{x}$, *the* normal curvature $\kappa_n(\mathbf{t})$ *is defined as the curvature of the curve that is the intersection of* $\mathcal{S}$ *and the plane defined by* $\mathbf{N}$ *and* $\mathbf{t}$. *The* principal curvatures κ_1, κ_2 *are the extrema of* κ_n *and the* principal directions *are the corresponding tangent vectors* $\mathbf{t}_1, \mathbf{t}_2$.

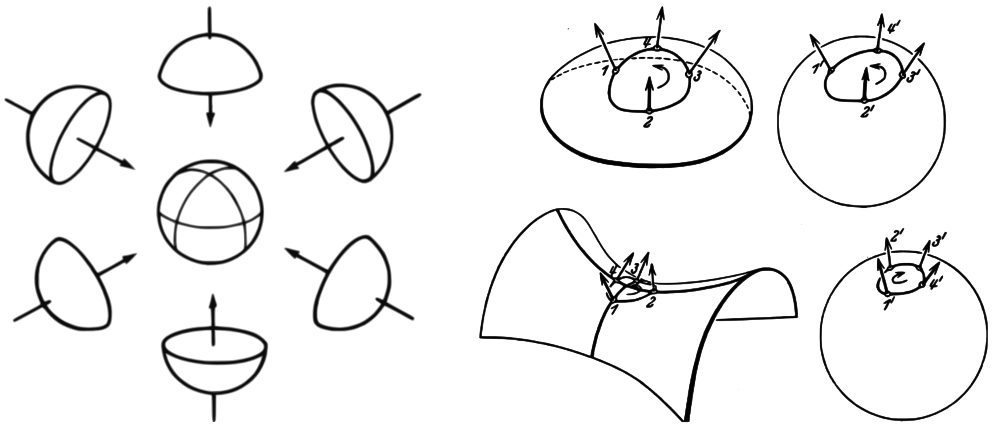


Figure 2.2: (*Left*) sphere $\mathbb{S}^2$ as the union of six parametrized hemispheres. Image from [doCa92]. (*Right*) Gauss map of locally elliptic and hyperbolic surfaces. Image from [HC52].

Alternatively, the principal curvatures and directions can be seen as the eigenvalues and eigenvectors of the shape operator $\mathbf{S}$. Two important quantities are associated with the principal curvatures.

Definition 2.10 (Gaussian curvature. Mean curvature)**.** *The intrinsic* Gaussian curvature *is the product of principal curvatures*

$$\mathrm{K} = \kappa_1 \kappa_2 \tag{2.7}$$

The non-intrinsic mean curvature *is defined as*

$$\mathrm{H} = \frac{1}{2\pi} \int_0^{2\pi} \kappa_n(\mathbf{t}) \, \mathrm{d}\phi, \tag{2.8}$$

where ϕ is the angle between $\mathbf{t}$ and $\mathbf{t}_1$.

Euler's theorem states that $\kappa_n(\mathbf{t}) = \kappa_1 \cos^2 \phi + \kappa_2 \sin^2 \phi$. Using this relation, the mean curvature H can be written as the arithmetic mean of principal curvatures:

$$\mathrm{H} = \tfrac{1}{2} (\kappa_1 + \kappa_2) \, . \tag{2.9}$$

The Gaussian curvature K is intrinsic to the surface and is invariant under isometric deformations of the surface. This was first proven by Gauss in his Theorema Egregium [doCa16, p. 237].

On the other hand, the mean curvature H is *not* intrinsic. To see why, take a flat sheet of paper, whose curvature is zero in all directions – this implies that the mean curvature is zero everywhere. Now, isometrically deform this sheet into a cylinder. The principal curvature along the axis of the cylinder remains zero, but the principal

curvature along the circular cross-section of the cylinder is *positive*. Since the mean curvature changed from zero (flat sheet) to a positive number (cylinder) under an isometric deformation, it cannot be intrinsic to the surface.

The sign of H depends on the choice of orientation; therefore, it is more natural to work with the *mean curvature normal*

$$\mathbf{H} = -2\mathrm{H}\mathbf{N} \tag{2.10}$$

where $\mathbf{N}$ is the unit outward normal.

Let us now discuss the relation between the mean curvature normal and the Laplace operator. For a sufficiently differentiable scalar function

$$f : \Omega \subset \mathbb{R}^2 \to \mathbb{R} : (u, v) \mapsto f(u, v),$$

the Laplace operator is defined as the divergence of the gradient,

$$\Delta f = \nabla \cdot \nabla f = f_{uu} + f_{vv},$$

where we used the notation $f_{uu} = \partial^2 f / \partial u^2$, $f_{vv} = \partial^2 f / \partial v^2$. Mean curvature H is the geometric interpretation of the Laplace operator Δ for surfaces; this is analogical to curves where the curvature κ is the geometric version of the second derivative. Fixing a point $\mathbf{x}$ on $\mathcal{S}$ and, the surface $\mathcal{S}$ can locally be seen as the graph of the height function $f_h : \mathrm{T}_{\mathbf{x}}\mathcal{S} \to \mathbb{R}$ defined over the tangent plane at $\mathbf{x}$. Applying the Laplacian to the height function yields the mean curvature at $\mathbf{x}$:

$$\Delta f_h = \mathrm{H}.$$

Extending the Laplacian to functions defined on the surface $f : \mathcal{S} \to \mathbb{R}^d$, the *Laplace-Beltrami operator*, the intrinsic surface Laplacian, is defined as the divergence of the gradient with respect to the metric of the surface:

$$\Delta_{\mathcal{S}} f = \nabla_{\mathcal{S}} \cdot \nabla_{\mathcal{S}} f.$$

Substituting the coordinate function $f = \mathbf{x}$, we get the following connection between Laplace-Beltrami and the mean curvature normal [doCa16]:

$$\Delta_{\mathcal{S}} \mathbf{x} = \mathbf{H}. \tag{2.11}$$

2.3 Curves on surfaces

The Frenet-Serret frame defined in Section 2.1 is one example of a frame along a curve. When studying curves on surfaces, it is convenient to define another type of frame, which carries the properties of the surface.

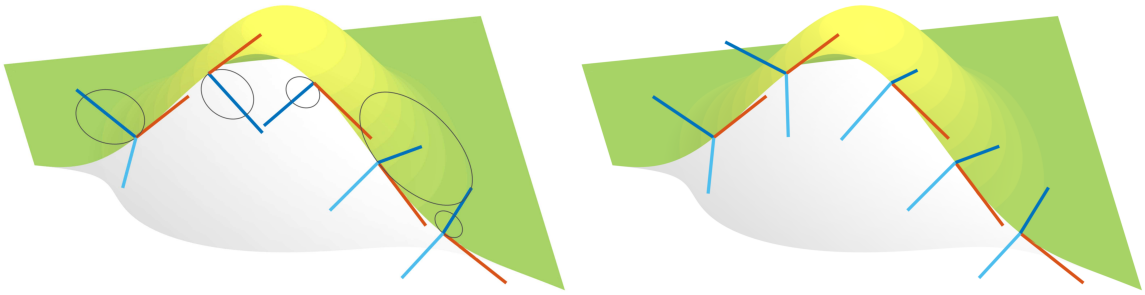


Figure 2.3: (*Left*) the Frenet-Serret frame and osculating circles along a space curve on a Bézier surface and (*right*) the Darboux frame along the same curve. [*red – tangent* $\mathbf{t}$, *dark blue – principal normal* $\mathbf{n}$ / *surface normal* $\mathbf{N}$, *light blue – binormal* $\mathbf{b}$ / *conormal* $\mathbf{B}$]

Definition 2.11 (Darboux frame)**.** *Suppose the curve* γ *is lying on a surface,* $\gamma \subset \mathcal{S} \subset \mathbb{R}^3$ *(Fig. 2.3). The* Darboux frame $\mathcal{D} = \{\mathbf{t}, \mathbf{N}, \mathbf{B}\}$ *is defined by the unit tangent* $\mathbf{t}$*, the unit surface normal* $\mathbf{N}$*, and the* conormal *or the* tangent normal $\mathbf{B} = \mathbf{t} \times \mathbf{N}$.

Note that unlike Frenet-Serret frame, Darboux frame is defined even at inflection points where the curvature κ vanishes. Since the curvature vector $\mathbf{t}' = \kappa\mathbf{n}$ lies in the normal plane spanned by $\mathbf{N}$ and $\mathbf{B}$ (Proposition 2.3), $\mathbf{t}'$ decomposes into the normal component and the binormal component as

$$\mathbf{t}' = \kappa\mathbf{n} = \kappa \left(\cos\psi\, \mathbf{N} - \sin\psi\, \mathbf{B}\right),$$

where $\psi = \sphericalangle(\mathbf{n}, \mathbf{N})$ is the (oriented) angle between the principal normal $\mathbf{n}$ and the surface normal $\mathbf{N}$. This leads to the following differential characterization of the Darboux frame.

Proposition 2.12. *The change of the Darboux frame* $\mathcal{D} = \{\mathbf{t}, \mathbf{N}, \mathbf{B}\}$ *is given by*

$$\begin{pmatrix} \mathbf{t}' \\ \mathbf{N}' \\ \mathbf{B}' \end{pmatrix} = \begin{pmatrix} 0 & \kappa_n & \kappa_g \\ -\kappa_n & 0 & \tau_g \\ -\kappa_g & -\tau_g & 0 \end{pmatrix} \begin{pmatrix} \mathbf{t} \\ \mathbf{N} \\ \mathbf{B} \end{pmatrix} \tag{2.12}$$

where κ_n *is the* normal curvature*,* κ_g *is the* geodesic curvature*, and* τ_g *is the* geodesic torsion.

Proof. Since both the Frenet-Serret frame $\mathcal{F} = \{\mathbf{t}, \mathbf{n}, \mathbf{b}\}$ and the Darboux frame $\mathcal{D} = \{\mathbf{t}, \mathbf{N}, \mathbf{B}\}$ are right-handed and share the same tangent $\mathbf{t}$, $\mathcal{D}$ can be obtained by rotating $\mathcal{F}$ around $\mathbf{t}(s)$ by some angle $\psi = \psi(s)$ (cf. Fig. 2.3)

$$\begin{pmatrix} \mathbf{t} \\ \mathbf{N} \\ \mathbf{B} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & \sin\psi \\ 0 & -\sin\psi & \cos\psi \end{pmatrix} \begin{pmatrix} \mathbf{t} \\ \mathbf{n} \\ \mathbf{b} \end{pmatrix}.$$

Differentiating this relation yields

$$\begin{aligned}\mathbf{N}' = \frac{\mathrm{d}\mathbf{N}}{\mathrm{d}s} &= \frac{\mathrm{d}\cos\psi}{\mathrm{d}\psi}\frac{\mathrm{d}\psi}{\mathrm{d}s}\mathbf{n} + \cos\psi\,\frac{\mathrm{d}\mathbf{n}}{\mathrm{d}s} + \frac{\mathrm{d}\sin\psi}{\mathrm{d}\psi}\frac{\mathrm{d}\psi}{\mathrm{d}s}\mathbf{b} + \sin\psi\,\frac{\mathrm{d}\mathbf{b}}{\mathrm{d}s},\\ \mathbf{B}' = \frac{\mathrm{d}\mathbf{B}}{\mathrm{d}s} &= -\frac{\mathrm{d}\sin\psi}{\mathrm{d}\psi}\frac{\mathrm{d}\psi}{\mathrm{d}s}\mathbf{n} - \sin\psi\,\frac{\mathrm{d}\mathbf{n}}{\mathrm{d}s} + \frac{\mathrm{d}\cos\psi}{\mathrm{d}\psi}\frac{\mathrm{d}\psi}{\mathrm{d}s}\mathbf{b} + \cos\psi\,\frac{\mathrm{d}\mathbf{b}}{\mathrm{d}s}.\end{aligned} \tag{2.13}$$

Using the Frenet-Serret formulas (2.4) and multiplying the above identities by the length element $\mathrm{d}s$ gives

$$\begin{aligned}\mathrm{d}\mathbf{N} &= -\sin\psi\;\mathrm{d}\psi\,\mathbf{n} + \cos\psi\;(-\kappa\mathbf{t}+\tau\mathbf{b})\,\mathrm{d}s + \cos\psi\;\mathrm{d}\psi\,\mathbf{b} + \sin\psi\;(-\tau\mathbf{n})\,\mathrm{d}s\\ &= \mathbf{t}\,(-\kappa\cos\psi)\,\mathrm{d}s + (\mathrm{d}\psi+\tau\,\mathrm{d}s)\underbrace{(-\sin\psi\,\mathbf{n}+\cos\psi\,\mathbf{b})}_{\mathbf{B}},\\ \mathrm{d}\mathbf{B} &= -\cos\psi\;\mathrm{d}\psi\,\mathbf{n} - \sin\psi\;(-\kappa\mathbf{t}+\tau\mathbf{b})\,\mathrm{d}s - \sin\psi\;\mathrm{d}\psi\,\mathbf{b} + \cos\psi\;(-\tau\mathbf{n})\,\mathrm{d}s\\ &= \mathbf{t}\,(\kappa\sin\psi)\,\mathrm{d}s - (\mathrm{d}\psi+\tau\,\mathrm{d}s)\underbrace{(\cos\psi\,\mathbf{n}+\sin\psi\,\mathbf{b})}_{\mathbf{N}}.\end{aligned} \tag{2.14}$$

Recall that $\mathbf{t}' = \frac{\mathrm{d}\mathbf{t}}{\mathrm{d}s} = \kappa\mathbf{n} = \kappa\,(\cos\psi\,\mathbf{N} - \sin\psi\,\mathbf{B})$. In matrix notation, we therefore have

$$\begin{pmatrix}\mathrm{d}\mathbf{t}\\ \mathrm{d}\mathbf{N}\\ \mathrm{d}\mathbf{B}\end{pmatrix} = \begin{pmatrix}0 & \kappa\cos\psi\;\mathrm{d}s & -\kappa\sin\psi\;\mathrm{d}s\\ -\kappa\cos\psi\;\mathrm{d}s & 0 & \tau\;\mathrm{d}s+\mathrm{d}\psi\\ \kappa\sin\psi\;\mathrm{d}s & -\tau\;\mathrm{d}s-\mathrm{d}\psi & 0\end{pmatrix}\begin{pmatrix}\mathbf{t}\\ \mathbf{N}\\ \mathbf{B}\end{pmatrix}.$$

Eq. (2.12) is obtained by multiplying the last identity by $1/\mathrm{d}s$ and setting $\kappa_n = \kappa\cos\psi$, $\kappa_g = -\kappa\sin\psi$, $\tau_g = \tau + \frac{\mathrm{d}\psi}{\mathrm{d}s}$. □

The normal curvature κ_n is intrinsic to the surface $\mathcal{S}$ and depends only on the direction of the tangent $\mathbf{t}$, not on the actual choice of the curve γ. The geodesic curvature κ_g is intrinsic to the curve γ and describes the embedding of γ into the surface $\mathcal{S}$. The geodesic torsion τ_g measures the twisting of the Darboux frame around the tangent vector. The following special cases are recognized when one of these quantities vanishes along γ:

- γ is a *geodesic curve* if $\kappa_g \equiv 0$.
- γ is an *asymptotic curve* if $\kappa_n \equiv 0$.
- γ is a *line of curvature* if $\tau_g \equiv 0$.

Note that Darboux frame is *rotation-minimizing* along lines of curvature – for such curves, the frame does not rotate around the instanteneous tangent [BFS10; Wan+08].

2.4 Smooth manifolds

Curves and surfaces in $\mathbb{R}^3$ introduced in previous sections are examples of *manifolds*. Informally, manifolds are objects that are locally Euclidean. In this section, we formalize the notion of a smooth manifold (embedded in $\mathbb{R}^n$) and introduce concepts from Riemannian geometry such as tangent bundle, inner product, scalar and vector fields on a manifold, distance, geodesic curve, exponential map, logarithmic map. These concepts will serve us for generalization of energy-minimizing splines from Euclidean spaces to Riemannian manifolds in Section 3.5.4.

We closely follow Boumal [Bou10] and Absil et al. [AMS08]; some definitions are inspired by Milnor [Mil65], Berger [Ber03] and do Carmo [doCa16; doCa92]. All results are given without proofs.

Definition 2.13 (Diffeomorphism). *A map $f : X \to Y$ is called a* diffeomorphism *if f carries X homeomorphically onto Y and both f and its inverse f^{-1} are C^∞.*

Note that the above definition implies that f is a one-to-one mapping.

Definition 2.14 (Smooth manifold of dimension k). *A subset $\mathcal{M} \subset \mathbb{R}^n$ is called a* smooth manifold *of* dimension k *if each $\mathbf{x} \in \mathcal{M}$ has a neighborhood $V \cap \mathcal{M}$ that is diffeomorphic to an open subset U of the Euclidean space $\mathbb{R}^k$.*

Any particular diffeomorphism $g : U \to V \cap \mathcal{M}$ is called a parametrization *of the region $V \cap \mathcal{M}$. The inverse diffeomorphism $g^{-1} : V \cap \mathcal{M} \to U$ is called a* system of coordinates *on $V \cap \mathcal{M}$ or a* chart.

An example of a parametrization (of a regular surface) is shown in Fig. 2.1.

Similarly to vector spaces, the tangent space to a manifold $\mathcal{M}$ at $\mathbf{x}$ is intuitively the best linear approximation of $\mathcal{M}$ in the neighborhood of $\mathbf{x}$ – it is the vector space induced by the hyperplane which is tangent to $\mathcal{M}$ at $\mathbf{x}$. Vectors in the tangent space are defined via equivalence classes of curves passing through $\mathbf{x}$. Here, a *curve* is a differentiable map $\mathbf{c} : (-\epsilon, \epsilon) \to \mathcal{M}$. Consider the set of all such differentiable curves $\mathbf{c}(t)$ passing through a fixed point $\mathbf{x} \in \mathcal{M}$ at $t = 0$:

$$C_{\mathbf{x}} = \left\{ \mathbf{c} : (-\epsilon, \epsilon) \to \mathcal{M} \;\text{ such that }\; \mathbf{c} \in \mathrm{C}^1 \text{ and } \mathbf{c}(0) = \mathbf{x} \right\}$$

and an equivalence relation $\sim$ on $C_{\mathbf{x}}$ defined by

$$\mathbf{c}_1 \sim \mathbf{c}_2 \Leftrightarrow \tfrac{\mathrm{d}}{\mathrm{d}t}\, g^{-1}\left(\mathbf{c}_1(t)\right)\Big|_{t=0} = \tfrac{\mathrm{d}}{\mathrm{d}t}\, g^{-1}\left(\mathbf{c}_2(t)\right)\Big|_{t=0} \tag{2.15}$$

where $\mathbf{c}_1, \mathbf{c}_2 \in C_{\mathbf{x}}$ are two curves passing through $\mathbf{x}$ and $g^{-1} : V \cap \mathcal{M} \to U \subset \mathbb{R}^k$ is a chart (a system of coordinates) on $\mathcal{M}$ such that $\mathbf{x}$ is contained in V. The definition of $\sim$ is independent from the choice of a chart g^{-1}.

Definition 2.15 (Tangent space $\mathrm{T}_{\mathbf{x}}\mathcal{M}$)**.** *The* tangent space *to the smooth manifold $\mathcal{M}$ of dimension k at a fixed point $\mathbf{x}$ is a (k-dimensional) vector space denoted by $\mathrm{T}_{\mathbf{x}}\mathcal{M}$ and defined as the quotient space*

$$\mathrm{T}_{\mathbf{x}}\mathcal{M} = C_{\mathbf{x}} / \sim$$

For a curve $\mathbf{c} \in C_{\mathbf{x}}$, the equivalence class $[\mathbf{c}] \in \mathrm{T}_{\mathbf{x}}\mathcal{M}$ is called a tangent vector *to $\mathcal{M}$ at $\mathbf{x}$.*

Definition 2.16 (Tangent bundle $\mathrm{T}\mathcal{M}$)**.** *The* tangent bundle $\mathrm{T}\mathcal{M}$ *of a smooth manifold $\mathcal{M}$ is the disjoint union of tangent spaces $\mathrm{T}_{\mathbf{x}}\mathcal{M}$,*

$$\mathrm{T}\mathcal{M} = \coprod_{\mathbf{x}\in\mathcal{M}} \mathrm{T}_{\mathbf{x}}\mathcal{M}.$$

We define the natural projection on the roots of vectors as the mapping $\pi : \mathrm{T}\mathcal{M} \to \mathcal{M}$ such that

$$\pi(\mathbf{u}) = \mathbf{x} \quad \textit{iff} \quad \mathbf{u} \in \mathrm{T}_{\mathbf{x}}\mathcal{M}.$$

Since each tangent space $\mathrm{T}_{\mathbf{x}}\mathcal{M}$ is a vector space, we can associate an inner product to $\mathrm{T}_{\mathbf{x}}\mathcal{M}$. A collection of such inner products with special properties defines a Riemannian metric.

Definition 2.17 (Inner product on $\mathrm{T}_{\mathbf{x}}\mathcal{M}$)**.** *An* inner product $\langle\cdot,\cdot\rangle_{\mathbf{x}}$ *on a tangent space $\mathrm{T}_{\mathbf{x}}\mathcal{M}$ is a bilinear, symmetric, positive-definite form. For any $\mathbf{u}, \mathbf{v}, \mathbf{w} \in \mathrm{T}_{\mathbf{x}}\mathcal{M}$ and $\alpha, \beta \in \mathbb{R}$,*

(i) $\left\langle \alpha\mathbf{u} + \beta\mathbf{v}, \mathbf{w} \right\rangle_{\mathbf{x}} = \alpha \langle \mathbf{u}, \mathbf{w} \rangle_{\mathbf{x}} + \beta \langle \mathbf{v}, \mathbf{w} \rangle_{\mathbf{x}}$ *(bilinearity)*

(ii) $\langle \mathbf{u}, \mathbf{v} \rangle_{\mathbf{x}} = \langle \mathbf{v}, \mathbf{u} \rangle_{\mathbf{x}}$ *(symmetry)*

(iii) $\langle \mathbf{u}, \mathbf{u} \rangle_{\mathbf{x}} \geq 0$ *and* $\langle \mathbf{u}, \mathbf{u} \rangle_{\mathbf{x}} = 0$ *iff* $\mathbf{u} = \mathbf{0}$ *(positive-definitness)*

We now describe how a family of inner products defined on the tangent bundle induces a Riemannian structure on $\mathcal{M}$.

Fix a point $\mathbf{x} \in \mathcal{M}$ and let $f : V \cap \mathcal{M} \to U \subset \mathbb{R}^k$ be a chart such that $\mathbf{x}$ is contained in V. Let $\{\mathbf{e}_1, \ldots, \mathbf{e}_k\}$ be a basis of $\mathbb{R}^k$. Denote by $\mathbf{E}_i = f^{-1}\mathbf{e}_i$ its inverse image, which is a basis of $\mathrm{T}_{\mathbf{x}}\mathcal{M}$. Fixing two tangent vectors $\mathbf{u}, \mathbf{v} \in \mathrm{T}_{\mathbf{x}}\mathcal{M}$, we can therefore write

$$\mathbf{u} = \sum_{i=1}^{k} u_i \mathbf{E}_i, \qquad \mathbf{v} = \sum_{j=1}^{k} v_j \mathbf{E}_j,$$

or, using the vector notation: $\mathbf{u} = (u_1, \ldots, u_k)^\mathsf{T}$ and $\mathbf{v} = (v_1, \ldots, v_k)^\mathsf{T}$. The inner product of $\mathbf{u}$ and $\mathbf{v}$ is

$$\langle \mathbf{u}, \mathbf{v} \rangle_{\mathbf{x}} = \left\langle \sum_{i=1}^{k} u_i \mathbf{E}_i, \sum_{j=1}^{k} v_j \mathbf{E}_j \right\rangle_{\mathbf{x}} = \sum_{i=1}^{k} \sum_{j=1}^{k} u_i v_j \left\langle \mathbf{E}_i, \mathbf{E}_j \right\rangle_{\mathbf{x}} = \mathbf{u}^\mathsf{T} \mathrm{G}_{\mathbf{x}} \mathbf{v}$$

where $G_{\mathbf{x}}$ is a symmetric positive-definite matrix whose elements are given by $(G_{\mathbf{x}})_{ij} = \langle \mathbf{E}_i, \mathbf{E}_j \rangle_{\mathbf{x}}$. A family of smoothly-varying matrices $G_{\mathbf{x}}$ induces a Riemannian structure on $\mathcal{M}$.

Definition 2.18 (Riemannian manifold)**.** A Riemannian manifold *is a smooth manifold $\mathcal{M}$ equipped with a Riemannian metric g.* A Riemannian metric *on $\mathcal{M}$ is a family of inner products on $\mathrm{T}_{\mathbf{x}}\mathcal{M}$ such that the corresponding matrices $G_{\mathbf{x}}$ are smoothly-varying, meaning the maps $(G_{\mathbf{x}})_{ij} : U \to \mathbb{R}$ are C^∞.*

Definition 2.19 (Riemannian submanifold of $\mathbb{R}^n$)**.** A Riemannian submanifold $\mathcal{M}$ *of $\mathbb{R}^n$ is a Riemannian manifold immersed in $\mathbb{R}^n$ equipped with the Riemannian metric inherited from $\mathbb{R}^n$.*

Definition 2.20 (Scalar and vector fields, directional derivative)**.** *A* scalar field *on a smooth manifold $\mathcal{M}$ is a C^1 function $f : \mathcal{M} \to \mathbb{R}$.*

A directional derivative *of a scalar field f on $\mathcal{M}$ at $\mathbf{x}$ in the direction $\mathbf{u} = [\mathbf{c}] \in \mathrm{T}_{\mathbf{x}}\mathcal{M}$ is the scalar*

$$\mathrm{D}f(\mathbf{x})[\mathbf{u}] = \tfrac{\mathrm{d}}{\mathrm{d}t} f(\mathbf{c}(t))\Big|_{t=0}$$

A vector field *on a smooth manifold $\mathcal{M}$ is a mapping which associates to each point $\mathbf{x}$ a vector $\mathbf{X}(\mathbf{x}) = \mathbf{X}_{\mathbf{x}}$ from the tangent space $\mathrm{T}_{\mathbf{x}}\mathcal{M}$:*

$$\mathbf{X} : \quad \mathcal{M} \to \mathrm{T}\mathcal{M} : \quad \mathbf{x} \mapsto \mathbf{X}_{\mathbf{x}}$$

Moreover, $\mathbf{X}$ satisfies the condition that the composition $\pi \circ \mathbf{X}$ is the identity map (each $\mathbf{x} \in \mathcal{M}$ is mapped to itself).

An important vector field is the gradient of a scalar field f on $\mathcal{M}$. Analogously to Euclidean spaces, the gradient at a fixed point $\mathbf{x}$ is the direction of the steepest ascent of f at $\mathbf{x}$. Note that for scalar fields on $\mathcal{M} = \mathbb{R}^n$, the Riemannian gradient agrees with the usual Euclidean gradient $\nabla f = (\partial f/\partial x_1, \ldots, \partial f/\partial x_n)$.

Definition 2.21 (Riemannian gradient)**.** *The* gradient *of a scalar field f on $\mathcal{M}$ at $\mathbf{x}$, denoted by* $\operatorname{grad} f(\mathbf{x})$*, is the unique element of $\mathrm{T}_{\mathbf{x}}\mathcal{M}$ that satisfies*

$$\mathrm{D}f(\mathbf{x})[\mathbf{u}] = \big\langle \operatorname{grad} f(\mathbf{x}), \mathbf{u} \big\rangle \qquad \text{for all } \mathbf{u} \in \mathrm{T}_{\mathbf{x}}\mathcal{M}.$$

Computationally, this definition is often inconvenient. For a submanifold $\mathcal{M}$ of $\mathbb{R}^n$, the Riemannian gradient can be obtained by projecting the Euclidean gradient to the tangent bundle $\mathrm{T}\mathcal{M}$. For a fixed point $\mathbf{x} \in \mathcal{M}$, let us define the (unique) orthogonal projectors which decompose any $\mathbf{v} \in \mathbb{R}^n$ into the tangent component in $\mathrm{T}_{\mathbf{x}}\mathcal{M}$ and the orthogonal component in $\mathrm{T}_{\mathbf{x}}^{\perp}\mathcal{M}$:

$$\begin{aligned} \mathrm{P}_{\mathbf{x}} &: \mathbb{R}^n \to \mathrm{T}_{\mathbf{x}}\mathcal{M} : \mathbf{v} \mapsto \mathrm{P}_{\mathbf{x}}\mathbf{v}, \\ \mathrm{P}_{\mathbf{x}}^{\perp} &: \mathbb{R}^n \to \mathrm{T}_{\mathbf{x}}^{\perp}\mathcal{M} : \mathbf{v} \mapsto \mathrm{P}_{\mathbf{x}}^{\perp}\mathbf{v}. \end{aligned}$$

Lemma 2.22 (Riemannian gradient as projection). *Let f be a scalar field on the Riemannian submanifold $\mathcal{M}$ of $\mathbb{R}^n$ defined as the restriction $f = \phi\big|_{\mathcal{M}}$ of a scalar field $\phi : \mathbb{R}^n \to \mathbb{R}$ to the manifold $\mathcal{M}$. Then*

$$\operatorname{grad} f(\mathbf{x}) = \mathrm{P}_{\mathbf{x}} \nabla \phi(\mathbf{x}).$$

Let us repeat the purpose of this section: our goal is to introduce basic notions from Riemannian geometry that we later use to define energy-minimizing splines in the space of orientations in $\mathbb{R}^3$ – this space is indeed a 3-manifold (see the next section). Energy-minimizing splines on Riemannian manifolds can be seen as a generalization of classical (Euclidean) splines to curved spaces. In order to define such splines, we need to define the concepts of velocity $\dot{\mathbf{c}}$ and acceleration $\ddot{\mathbf{c}}$ for a curve lying on a manifold.

Definition 2.23 (Velocity along a curve). *Let $\mathbf{c} : [a, b] \to \mathcal{M}$ be a C^1 curve on the manifold $\mathcal{M} \subset \mathbb{R}^n$. The* velocity *along $\mathbf{c}$ is given by*

$$\dot{\mathbf{c}}(t) = [\mathbf{c}^t]$$

where $\mathbf{c}^t : [a - t, b - t] \to \mathcal{M} : \mathbf{c}^t(s) = \mathbf{c}(s + t)$ is the reparametrization of $\mathbf{c}$ such that $\mathbf{c}^t(0) = \mathbf{c}(t)$. Recall that the tangent vector $[\mathbf{c}^t]$ is defined in Definition 2.15 via the equivalence relation from Eq. (2.15).

For the definition of acceleration $\mathrm{D}^2/\mathrm{d}t^2\mathbf{c}$ using the concept of covariant derivative, see Appendix A. Note that if $\mathcal{M}$ is a submanifold of $\mathbb{R}^n$, the velocity $\dot{\mathbf{c}}$ reduces to the usual derivative $\mathbf{c}'$. The relation to the usual acceleration $\mathbf{c}''$ is almost as straightforward – similarly to the computation of gradient, the curvature vector $\mathbf{c}''$ has to be reprojected back onto the manifold, as given by the following proposition.

Proposition 2.24 (Acceleration along a curve as projection). *Let $\mathbf{c} : \mathbb{R} \to \mathcal{M}$ be a C^2 curve on the submanifold $\mathcal{M} \subset \mathbb{R}^n$. The* acceleration *along $\mathbf{c}$ is given by*

$$\frac{\mathrm{D}^2}{\mathrm{d}t^2}\mathbf{c}(t) = \mathrm{P}_{\mathbf{c}(t)} \frac{\mathrm{d}^2}{\mathrm{d}t^2}\mathbf{c}(t) = \mathrm{P}_{\mathbf{c}(t)}\mathbf{c}''(t).$$

Having introduced velocity along a curve, we can proceed to definitions of curve length and geodesic distance on $\mathcal{M}$. As before, these are analogical to the Euclidean case – length is defined as integral of velocity, and geodesic distance is the length of the shortest path $\mathbf{c} \subset \mathcal{M}$ between two points.

Definition 2.25 (Length of a curve). *The length of a C^1 curve $\mathbf{c} : [a, b] \to \mathcal{M}$ on a Riemannian manifold is $\mathcal{M}$*

$$L(\mathbf{c}) = \int_a^b \sqrt{\left\langle \dot{\mathbf{c}}(t), \dot{\mathbf{c}}(t) \right\rangle_{\mathbf{c}(t)}}\, \mathrm{d}t = \int_a^b \left\| \dot{\mathbf{c}}(t) \right\|_{\mathbf{c}(t)} \mathrm{d}t.$$

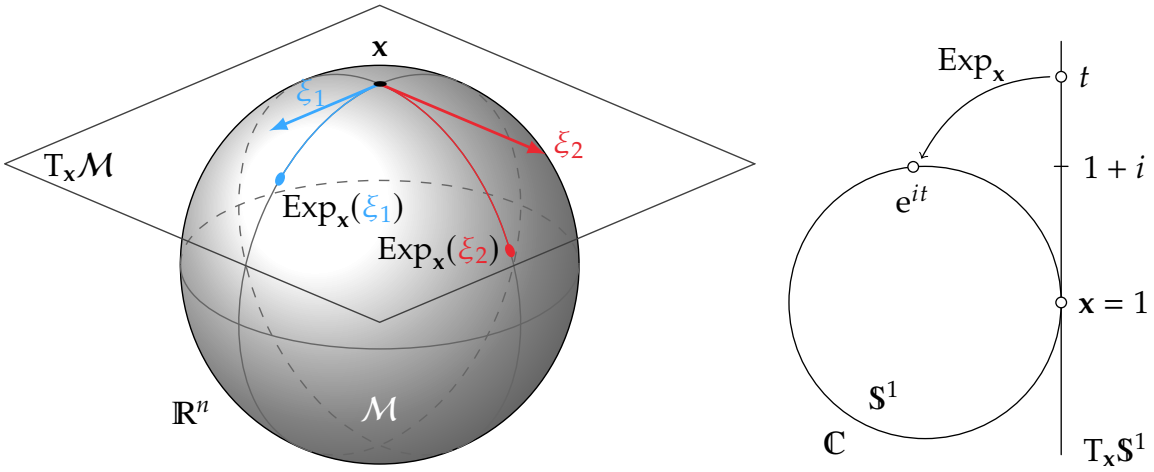


Figure 2.4: Exponential map. On the right, an example on the unit circle $\mathbb{S}^1 \subset \mathbb{C}$ viewed as a 1-manifold in the complex plane.

When $\mathcal{M}$ is embedded in $\mathbb{R}^n$, this reduces to the computation of length from Definition 2.1 for regular curves, i.e. $\dot{\mathbf{c}}$ is replaced by $\mathbf{c}'$.

Definition 2.26 (Riemannian distance)**.** Riemannian distance *or* geodesic distance *on the manifold $\mathcal{M}$ is given by*

$$\mathrm{dist} : \mathcal{M} \times \mathcal{M} \to \mathbb{R}^+ : (\mathbf{x}, \mathbf{y}) \mapsto \mathrm{dist}\,(\mathbf{x}, \mathbf{y}) = \inf_{\mathbf{c} \in \Gamma} L(\mathbf{c}).$$

Here, Γ is the set of all C^1 curves $\mathbf{c} : [0, 1] \to \mathcal{M}$ connecting $\mathbf{x}$ and $\mathbf{y}$ i.e. $\mathbf{c}(0) = \mathbf{x}$ and $\mathbf{c}(1) = \mathbf{y}$.

Definition 2.27 (Geodesic curve)**.** *A curve $\mathbf{c} : [a, b] \to \mathcal{M}$ is geodesic iff it has zero acceleration on all its domain.*

Distance, velocity and acceleration respectively control data interpolation, stretching, and bending of a smoothing spline on a Riemannian manifold. In Chapter 3 we work with *discrete* splines on Riemannian manifolds: in such setting, a continuous curve $\mathbf{c} \in \mathcal{M}$ is approximated by a finite sequence of points $\mathbf{x}_i \in \mathcal{M}$. To discretize distance, velocity and acceleration, we need the concepts of *exponential map*, and its inverse, the *logarithmic map*. Loosely speaking, the exponential map generalizes addition in vector spaces, while the logarithmic map generalizes difference.

Definition 2.28 (Exponential map)**.** *Let $\mathcal{M}$ be a Riemannian manifold and $\mathbf{x} \in \mathcal{M}$. For every tangent vector $\xi \in T_x\mathcal{M}$, there exists an open interval $I \ni 0$ and a unique geodesic $\mathbf{c}(t; \mathbf{x}; \xi) : I \to \mathcal{M}$ such that $\mathbf{c}(0) = \mathbf{x}$ and $\dot{\mathbf{c}}(0) = \xi$. Moreover, we have the homogeneity property*

$$\mathbf{c}(t; \mathbf{x}; a\xi) = \mathbf{c}(at; \mathbf{x}; \xi).$$

The exponential map *at* $\mathbf{x}$ *is defined as (Fig. 2.4)*

$$\mathrm{Exp}_{\mathbf{x}} : \mathrm{T}_{\mathbf{x}}\mathcal{M} \to \mathcal{M} : \xi \mapsto \mathrm{Exp}_{\mathbf{x}}(\xi) = \mathbf{c}(1; \mathbf{x}; \xi).$$

Definition 2.29 (Logarithmic map)**.** *The inverse of the exponential map is called the* logarithmic map *at* $\mathbf{x}$*; it is defined as*

$$\mathrm{Log}_{\mathbf{x}} : \mathcal{M} \to \mathrm{T}_{\mathbf{x}}\mathcal{M} : \mathbf{y} \mapsto \mathrm{Log}_{\mathbf{x}}(\mathbf{y}) = \xi \text{ such that } \mathrm{Exp}_{\mathbf{x}}(\xi) = \mathbf{y} \text{ and } \|\xi\|_{\mathbf{x}} = \mathrm{dist}(\mathbf{x}, \mathbf{y}).$$

2.5 Rotations in 3D

The notion of Darboux frame $\mathcal{D} = \{\mathbf{t}, \mathbf{N}, \mathbf{B}\}$, introduced in Section 2.3, is convenient for relating surfaces with curves lying on them. The Darboux frame is defined using the tangent vector $\mathbf{t}$ of the curve, the normal vector $\mathbf{N}$ of the surface, and the binormal $\mathbf{B} = \mathbf{t} \times \mathbf{N}$. Later, we will see that it is convenient to represent the Darboux frame $\mathcal{D}$ using the *rotation* between the canonical frame $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$ and $\mathcal{D}$.

2.5.1 Rotations as special orthonormal matrices

A rotation is a linear transformation of a vector space, which preserves lengths (no stretching) and orientation (no flipping). Mathematically, this corresponds to a change of basis that does not change the orientation; it transforms a right-handed frame into another right-handed frame. Conversely, any rotation R can be thought of as a transformation, which takes the canonical frame $\mathcal{E} = \{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$ and transforms it into a different (right-handed) frame

$$\mathcal{A} = \{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\}, \ \mathbf{a}_i \in \mathbb{R}^3 \ : \ \mathbf{a}_i \cdot \mathbf{a}_i = 1, \ \ \mathbf{a}_i \cdot \mathbf{a}_j = 0 \ \text{ for } \ i \neq j, \ \ \mathbf{a}_1 \times \mathbf{a}_2 = \mathbf{a}_3,$$

so that

$$\mathrm{R}(\mathbf{e}_1) = \mathbf{a}_1, \quad \mathrm{R}(\mathbf{e}_2) = \mathbf{a}_2, \quad \mathrm{R}(\mathbf{e}_3) = \mathbf{a}_3.$$

Such linear transformation R can be represented by an orthogonal matrix A with the vectors $\mathbf{a}_i$ in columns:

$$\mathrm{A} = \begin{pmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3 \end{pmatrix}. \tag{2.16}$$

Using this representation, the transformation R is computed via the matrix-vector multiplication:

$$\mathrm{R}(\mathbf{v}) = \mathrm{A}\mathbf{v}.$$

The inverse rotation is given by $\mathrm{A}^{-1} = \mathrm{A}^{\mathsf{T}}$. Indeed,

$$\mathrm{A}^{\mathsf{T}}\mathrm{A} = \begin{pmatrix} \mathbf{a}_1^{\mathsf{T}} \\ \mathbf{a}_2^{\mathsf{T}} \\ \mathbf{a}_3^{\mathsf{T}} \end{pmatrix} \begin{pmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3 \end{pmatrix} = \mathrm{I}_3. \tag{2.17}$$

Set:	$\mathrm{SO}(3)$	$=$	$\left\{\mathrm{A} \in \mathbb{R}^{3\times 3} \,:\, \mathrm{A}^\intercal \mathrm{A} = \mathrm{I}_3 \text{ and } \det(\mathrm{A}) = 1\right\}$
Tangent spaces:	$\mathrm{T}_\mathrm{A}\mathrm{SO}(3)$	$=$	$\left\{\mathrm{A}\Omega \in \mathbb{R}^{3\times 3} \,:\, \Omega + \Omega^\intercal = 0\right\}$
Inner product:	$\langle \mathrm{A}\Omega_1, \mathrm{A}\Omega_2 \rangle$	$=$	$\operatorname{trace}\left(\Omega_1^\intercal \Omega_2\right)$
Vector norm:	$\Vert \mathrm{A}\Omega \Vert$	$=$	$\sqrt{\langle \mathrm{A}\Omega, \mathrm{A}\Omega \rangle}$
Distance:	$\operatorname{dist}(\mathrm{A}, \mathrm{B})$	$=$	$\left\Vert \log\left(\mathrm{A}^\intercal \mathrm{B}\right) \right\Vert_\mathrm{F}$
Exponential:	$\operatorname{Exp}_\mathrm{A}(\mathrm{A}\Omega)$	$=$	$\mathrm{A} \exp(\Omega)$
Logarithm:	$\operatorname{Log}_\mathrm{A}(\mathrm{B})$	$=$	$\mathrm{A} \log\left(\mathrm{A}^\intercal \mathrm{B}\right)$
Projector:	$\mathrm{P}_\mathrm{A}(\mathrm{H})$	$=$	$\mathrm{A} \operatorname{skew}\left(\mathrm{A}^\intercal \mathrm{H}\right)$

Table 2.5: Toolbox for the special orthogonal group SO(3) [Bou13].

The determinant of A is computed as the mixed product of the frame vectors:

$$\det(\mathrm{A}) = \mathbf{a}_3 \cdot (\mathbf{a}_1 \times \mathbf{a}_2) = \mathbf{a}_3 \cdot \mathbf{a}_3 = 1. \tag{2.18}$$

Note that Eq. (2.18) does not hold if the frame $\mathcal{A}$ is left-handed, i.e. if $\mathbf{a}_1 \times \mathbf{a}_2 = -\mathbf{a}_3$: in this case, the determinant is negative. This corresponds to the change of orientation and the resulting transformation is no longer a rotation, even though it preserves lengths.

The natural bijection between the set of rotations in $\mathbb{R}^3$ and the set of right-handed orthonormal frames in $\mathbb{R}^3$ means that rotations can be represented by orthogonal matrices with unit determinant. The set of all such matrices is the *special orthogonal group* of dimension 3:

$$\mathrm{SO}(3) = \left\{\mathrm{A} \in \mathbb{R}^{3\times 3} \,:\, \mathrm{A}^\intercal \mathrm{A} = \text{identity and } \det(\mathrm{A}) = 1\right\}. \tag{2.19}$$

2.5.2 Riemannian structure of SO(3)

SO(3) is a Lie group: it has a structure of a compact closed 3-manifold. The usual Riemannian structure on SO(3) is that of a Riemannian submanifold of $\mathbb{R}^{3\times 3}$ with the Frobenius inner product defined as $\langle \mathrm{A}, \mathrm{B} \rangle = \operatorname{trace}\left(\mathrm{A}^\intercal \mathrm{B}\right)$. Using this structure, the exponential and logarithmic maps reduce to matrix exponential and logarithm. The matrix exponential is defined via the following power series:

$$\exp(\mathrm{A}) = \sum_{n=0}^{\infty} \frac{\mathrm{A}^n}{n!}.$$

Matrix logarithm is defined as the inverse of matrix exponential: we say that A is a matrix logarithm of B, $\log(\mathrm{B}) = \mathrm{A}$, if $\exp(\mathrm{A}) = \mathrm{B}$. Riemannian toolbox for SO(3) is summarized in Table 2.5, with the skew matrix operator defined as

$$\operatorname{skew}(\mathrm{M}) = \frac{1}{2}\left(\mathrm{M} - \mathrm{M}^\intercal\right). \tag{2.20}$$

The chordal (Frobenius) metric

$$d_F(A, B) = \|B - A\|_F$$

differs from the geodesic Riemannian metric on SO(3), which is defined as (cf. Table 2.5)

$$\text{dist}(A, B) = \left\|\log\left(A^T B\right)\right\|_F.$$

However, for rotations that are (geodesically) not too distant from each other, the chordal metric is a good approximation of the Riemannian metric [Bou13]. Indeed, let $\Omega \in \mathbb{R}^{3\times 3}$ be skew-symmetric with $\|\Omega\|_F = 1$ meaning that $A\Omega$ is an element of the tangent space $T_A SO(3)$. Moreover, let $B = \text{Exp}_A(A\Omega) = A\exp(t\Omega)$. Then the geodesic distance of A and B is

$$\begin{aligned}\text{dist}^2(A, B) &= \left\|\log\left(A^T B\right)\right\|_F^2 = \left\|\log\left(A^T A\exp(t\Omega)\right)\right\|_F^2 = \left\|\log\left(\exp(t\Omega)\right)\right\|_F^2 \\ &= t^2\|\Omega\|_F^2 = t^2.\end{aligned} \tag{2.21}$$

For the chordal distance of A and B we have

$$\begin{aligned}\|B - A\|_F^2 &= \left\|A\exp(t\Omega) - A\right\|_F^2 = \left\|A\left(\exp(t\Omega) - I_3\right)\right\|_F^2 = \left\|\left(\exp(t\Omega) - I_3\right)\right\|_F^2 \\ &= \left\|\sum_{n=0}^{\infty}(t\Omega)^n/n! - I_3\right\|_F^2 = \left\|\Omega + \tfrac{1}{2}t^2\Omega^2 + O(t^3)\right\|_F^2 \\ &= t^2 + t^3\left\langle\Omega, \Omega^2\right\rangle + O(t^4) = t^2 + O(t^4) = \text{dist}^2(A, B) + O(t^4).\end{aligned} \tag{2.22}$$

Here we used the fact that $\left\langle\Omega, \Omega^2\right\rangle = \text{trace}\left(\Omega^3\right)$ vanishes since the matrix Ω^3 is skew-symmetric.

To conclude, the Frobenius metric is locally a cubic approximation of the Riemannian metric. The motivation for replacing the geodesic distance by the chordal distance is the computational efficiency – using the chordal distance takes less time since it avoids the computation of the logarithmic map. This important fact will be used later, in Section 3.5.4, in order to simplify a cost function for a smoothing spline defined on the manifold SO(3).

2.5.3 Rotations as unit quaternions

First described by Hamilton, quaternions $\mathbb{H}$ are a non-commutative algebra of dimension four well-suited for representing rotations in $\mathbb{R}^3$. A quaternion is a 4-tuple $\mathbf{q} = (x, y, z, w)$ of real numbers, which defines a point in $\mathbb{R}^4$. Denoting by $\mathbf{i} = (1, 0, 0)$, $\mathbf{j} = (0, 1, 0)$, $\mathbf{k} = (0, 0, 1)$ the canonical basis in $\mathbb{R}^3$, the quaternion $\mathbf{q}$ is sometimes symbolically written as the sum of the scalar part w and the vector part $\mathbf{v} = (x, y, z)$,

$$\mathbf{q} = w + \mathbf{v} = w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k}. \tag{2.23}$$

Quaternion algebra is defined by the equations

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1.$$

The rotation around a unit Euler axis $\mathbf{e} \in \mathbb{R}^3$ by the angle θ is represented by the *unit quaternion* $\mathbf{q} \in \mathbb{H}^*$:

$$\mathbf{q} = \cos\frac{\theta}{2} + \mathbf{e}\sin\frac{\theta}{2}. \tag{2.24}$$

The inverse rotation corresponds to the *conjugate* quaternion

$$\bar{\mathbf{q}} = \cos\frac{\theta}{2} - \mathbf{e}\sin\frac{\theta}{2}.$$

The operation of quaternion conjugation can be interpreted either as direction inversion $-\mathbf{e}$ or angle inversion $-\theta$, yielding the same $\bar{\mathbf{q}}$. Inverting *both* the direction and the angle gives the *antipodal* quaternion

$$-\mathbf{q} = -\cos\frac{\theta}{2} - \mathbf{e}\sin\frac{\theta}{2},$$

which represents the rotation around $-\mathbf{e}$ by the angle $-\theta$. In essence, this is equivalent to the rotation represented by $\mathbf{q}$.

Any unit quaternion $\mathbf{q} = w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k} \in \mathbb{H}^*$ satisfies the relation $x^2+y^2+z^2+w^2 = 1$. It follows that there is a bijection between the unit quaternions $\mathbb{H}^*$ and the 3-sphere

$$\mathbb{S}^3 = \left\{(x, y, z, w) \;:\; x^2 + y^2 + z^2 + w^2 = 1\right\} \subset \mathbb{R}^4. \tag{2.25}$$

The 3-sphere is a useful topological model of rotations in $\mathbb{R}^3$. We must however not forget that unit quaternions are a 2:1-covering of SO(3) since antipodal quaternions $\mathbf{q}$ and $-\mathbf{q}$ represent the same rotation. Therefore, the manifold SO(3) is not homeomorphic to the 3-sphere itself, but rather to the quotient group

$$\mathbb{S}^3/\{I, -I\} \approx \mathrm{SO}(3),$$

where $I = (1, 0, 0, 0)$ is the north pole and $-I$ is the south pole; both poles correspond to the identity rotation. This means that conceptually, we can visualize a rotation – a point in SO(3) – as a *pair* of antipodal points on the 3-sphere (Fig. 2.6).

The conversion of a unit quaternion $\mathbf{q} = (x, y, z, w) \in \mathbb{S}^3$ to the corresponding rotation matrix $\mathrm{A}(\mathbf{q}) \in \mathrm{SO}(3)$ is described by the following relation [Sho85]:

$$\mathrm{A}(\mathbf{q}) = \begin{pmatrix} 2w^2 + 2x^2 - 1 & 2xy + 2wz & 2xz - 2wy \\ 2xy - 2wz & 2w^2 + 2y^2 - 1 & 2yz + 2wx \\ 2xz + 2wy & 2yz - 2wx & 2w^2 + 2z^2 - 1 \end{pmatrix}. \tag{2.26}$$

Note that all individual terms are quadratic in $\mathbf{q}$. It follows that $\mathrm{A}(-\mathbf{q}) = \mathrm{A}(\mathbf{q})$ and $\mathrm{A}(\bar{\mathbf{q}}) = \mathrm{A}(\mathbf{q})^\mathsf{T}$.

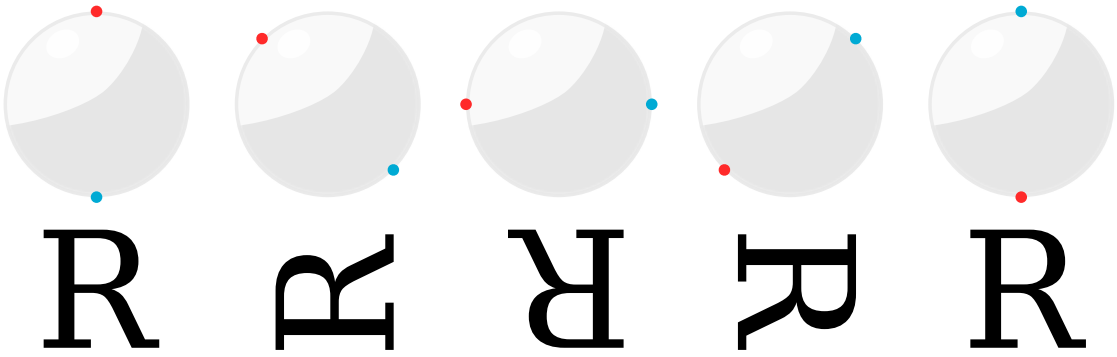


Figure 2.6: A pair of antipodal points on the 3-sphere (top) corresponds to a unique rotation (bottom). The above figure is a 2D illustration of this principle.

2.5.4 Spherical linear interpolation

In computer animation, animators define the rotation of an object in a series of keyframes, and the intermediate rotation is computed automatically by the animation system. The quaternion representation provides a direct way for interpolation between rotations using the spherical structure of the unit quaternions. Interpolation between two unit quaternions $\mathbf{q}_0, \mathbf{q}_1 \in \mathbb{S}^3$ essentially means drawing the (shortest) spherical arc between $\mathbf{q}_0$ and $\mathbf{q}_1$ plotted as points on the 3-sphere. If ϕ is the angle between the two quaternions given by $\mathbf{q}_0 \cdot \mathbf{q}_1 = \cos\phi$, such spherical arc is given by the spherical linear interpolation (Slerp) from $\mathbf{q}_0$ to $\mathbf{q}_1$ with a parameter $t \in [0, 1]$ (cf. Eq. (1.2) on p. 11) :

$$\operatorname{Slerp}(\mathbf{q}_0, \mathbf{q}_1; t) = \frac{\sin\left[(1-t)\phi\right]}{\sin\phi}\mathbf{q}_0 + \frac{\sin\left[t\phi\right]}{\sin\phi}\mathbf{q}_1. \tag{2.27}$$

Given an orientation in the matrix form, the choice of the representative quaternion is not unique – any one of the two antipodal quaternions can be picked. Generally, the quaternion is picked in a way that minimizes the length of the resulting spherical arc.

2.6 Cell complexes

In this section we define the notion of the 2-dimensional cell complex, a topological structure needed to formalize two important concepts: the triangle mesh, and the curve network. This section follows Leškovský [Leš02]. For the follow-up reading, see Hatcher [Hat02] and Berberich et al. [Ber+10].

Let X be a set and denote by X^k the Cartesian product of k copies of the set,

$$X^k = \underbrace{X \times \cdots \times X}_{k \text{ times}}.$$

The elements of X^k are k-tuple variations $(x_1, \ldots, x_k)$. To define a cell complex, we want some of these variations to be equivalent. For instance, if $X = \{0, 1, 2\}$ represents the vertices, the six triplets $(0, 1, 2)$, $(1, 2, 0)$, $(2, 0, 1)$, $(2, 1, 0)$, $(0, 2, 1)$, $(1, 0, 2)$ define the same triangle $\triangle$. This leads to the following definition.

Definition 2.30. *The set of all* cyclically and reverse unordered k-tuples *for a given set X is denoted by $X^{[k]}$ and defined as the factor set*

$$X^{[k]} = X^k/\sim$$

under the equivalence relation

$$(x_1, x_2, \ldots, x_{k-1}, x_k) \sim (x_k, x_1, \ldots, x_{k-1}) \sim (x_k, x_{k-1}, \ldots, x_2, x_1).$$

An element $a \in X^{[k]}$ is called a cell *and denoted by $x = [x_1, \ldots, x_k]$. A subelement $y = [y_1, \ldots, y_m] \in X^{[m]}$ of x, written as $y \subset x$, satisfies $x = [y_1, \ldots, y_m, z_1, \ldots, z_{k-m}]$ for some $z = [z_1, \ldots, z_{k-m}] \in X^{[k-m]}$. We say that y* belongs *to x and is* incident *to z.*

Definition 2.31 (Cell complex, simplicial complex)**.** *An* abstract 2-dimensional cell complex *or simply a* cell complex *is a topological structure $\mathcal{C} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$ which consists of 0-cells or vertices $\mathcal{V} \neq \emptyset$, 1-cells or edges $\mathcal{E} \subset \mathcal{V}^{[2]}$ and 2-cells or faces $\mathcal{F} \subset \bigcup_{k=3}^{\infty} \mathcal{V}^{[k]}$. For a vertex $v \in \mathcal{V}$, denote by* $\operatorname{link}(v)$ *the set of all vertices having a common face with v:*

$$\operatorname{link}(v) = \left\{ v_i \in \mathcal{V} : \quad v_i \neq v \text{ and } \; v, v_i \in f \in \mathcal{F} \right\}$$

The following conditions need to be satisfied:

(i) **No isolated vertices:** *each vertex $v \in \mathcal{V}$ belongs to some edge $e \in \mathcal{E}$.*

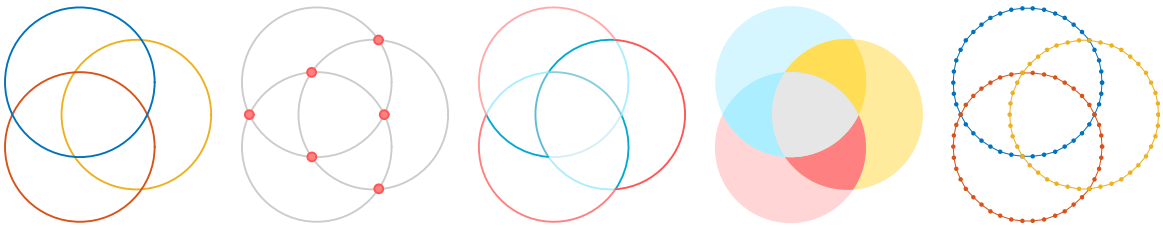
(ii) **No dangling edges:** *each edge $e \in \mathcal{E}$ belongs to exactly one face $f \in \mathcal{F}$ (boundary edge) or to exactly two faces $f_1, f_2 \in \mathcal{F}$ (interior edge).*

(iii) **All edges in $\mathcal{E}$:** *each subelement $e = [v_1, v_2] \subset f \in \mathcal{F}$ is an edge $e \in \mathcal{E}$.*

(iv) **No articulation vertices:** *If the set* $\operatorname{link}(v)$ *is finite with k vertices v_i, there exists an ordering $(v_1, \ldots, v_k)$ such that $[v_i, v_{i+1}] \in \mathcal{E}$ is an edge for all $1 \leq i \leq k-1$. Moreover, if $[v_k, v_1] \in \mathcal{E}$, then v is an inner vertex; otherwise v is a boundary vertex.*

A simplicial complex is a special case of a cell complex, in which all faces are triangular: $\mathcal{F} \subset \mathcal{V}^{[3]}$.

For our purposes, we also add the requirement that the cell complex needs to be connected (in the graph sense), meaning there is a path between each pair of vertices.



(a) curves γ **(b)** nodes Υ **(c)** segments Σ **(d)** cycles Ω **(e)** discrete $\mathcal{N}$

Figure 2.7: A planar curve network ($\mathcal{S} = \mathbb{R}^2$) with the topology of a cell complex. The network, given by three circular curves **(a)**, induces a cell complex **(b–d)** with six nodes, twelve segments (red: boundary, blue: interior), and seven cycles. A discrete curve network is obtained by sampling the smooth curves **(e)**.

Example 2.32 (Triangle mesh). *A* triangle mesh $\mathcal{T} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$ *is a piecewise-linear 2-manifold with the topology of a simplicial complex. The geometry of the mesh is given by the embedding of vertices* $\mathcal{V}$ *specified via a coordinate function*

$$\mathbf{v} : \mathcal{V} \to \mathbb{R}^3 : v_i \mapsto \mathbf{v}_i.$$

The embedding of an edge $e = (v_i, v_j)$ *is the line segment between* $\mathbf{v}_i$ *and* $\mathbf{v}_j$*. The embedding of a face* $f = (v_i, v_j, v_k)$ *is the triangle with vertices* $\mathbf{v}_i$, $\mathbf{v}_j$, $\mathbf{v}_k$.

Example 2.33 (Curve network on a surface). *A* curve network $\Gamma = (\Upsilon, \Sigma, \Omega)$ *on a regular surface* $\mathcal{S}$ *is a set of curves* γ *lying on* $\mathcal{S}$ *and inducing a cell complex structure on it (Fig. 2.7). The topology of a curve network is composed of* nodes Υ *(0-cells),* segments $\Sigma \subset \Upsilon^{[2]}$ *(1-cells), and* cycles $\Omega \subset \bigcup_{k=3}^{\infty} \Upsilon^{[k]}$ *(2-cells).*

A curve network Γ inherits the geometry of the surface region bounded by Γ (cf. Fig. 2.7d). In particular, each curve $\gamma \subset \mathcal{S}$ defines an embedding of a sequence of segments.

Example 2.34 (Discrete network). *A* discrete network $\mathcal{N}$ *associated to a curve network* Γ *is a set of vertices* $\mathbf{v}_i$ *sampled from* Γ *and connected by edges, having the same topology as* Γ*. Each segment* $\gamma_s : [0, L_s] \to \mathbb{R}^3$ *is sampled at parameter values* $0 = t_0 \leq t_1 \leq \cdots \leq t_k = L_s$*:*

$$\gamma(t_0) = \mathbf{v}_0, \ \gamma(t_1) = \mathbf{v}_1, \ \ldots \ \gamma(t_k) = \mathbf{v}_k.$$

Generally, a discrete curve network violates the condition *(iv)* from Definition 2.31 and therefore *does not* induce a cell complex – see the example in Fig. 2.7e.

2.7 Discrete Laplacian

The Laplace-Beltrami operator (or simply the Laplacian), which we introduced in Section 2.2, plays an important role in variational modeling of curves and surfaces to be described in the next section. In order to work with functions defined on triangle meshes (Section 2.6), the Laplace operator needs to be discretized.

Consider a triangle mesh $\mathcal{T} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$ (Fig. 2.8) with topology defined by a simplicial complex consisting of vertices, edges and triangular faces (Example 2.32):

$$v_i \in \mathcal{V}, \quad e_{ij} \in \mathcal{E} \subset \mathcal{V}^{[2]}, \quad f_{ijk} \in \mathcal{F} \subset \mathcal{V}^{[3]}.$$

The geometry of $\mathcal{T}$ is specified by the coordinate function $\mathbf{v}$ which associates each vertex with a position in $\mathbb{R}^3$:

$$\mathbf{v} : \mathcal{V} \to \mathbb{R}^3 : v_i \mapsto \mathbf{v}_i.$$

The edge vector corresponding to the oriented edge e_{ij} is indicated in bold:

$$\mathbf{e}_{ij} = -\mathbf{e}_{ji} = \mathbf{v}_j - \mathbf{v}_i.$$

When referring to a triangle T, we mean the embedding of the face f_{ijk} induced by $\mathbf{v}$. The interior angles in T are $\theta_i, \theta_j, \theta_k$, and the area of T is given by (Fig. 2.8a)

$$\mathrm{A_T} = \frac{1}{2} \sin\theta_i \left\|\mathbf{e}_{ji}\right\| \left\|\mathbf{e}_{ki}\right\| = \frac{1}{2} \sin\theta_j \left\|\mathbf{e}_{ij}\right\| \left\|\mathbf{e}_{kj}\right\| = \frac{1}{2} \sin\theta_k \left\|\mathbf{e}_{ik}\right\| \left\|\mathbf{e}_{jk}\right\|. \tag{2.28}$$

The vertices of T are oriented counter-clockwise with respect to the orientation defined by the outward pointing triangle normal.

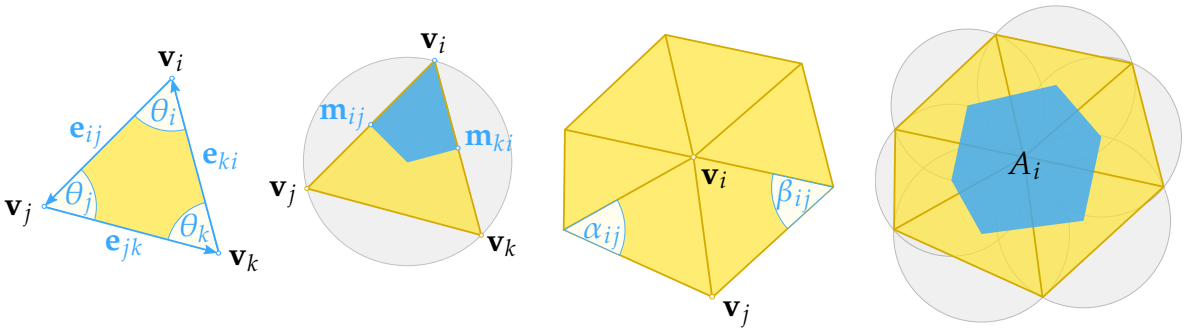


Figure 2.8: Notation for triangle meshes (Section 2.7)

2.7.1 Discretization via finite elements

To discretize differential operators such as $\nabla_{\mathcal{S}}, \Delta_{\mathcal{S}}$ on a piecewise-linear triangle mesh $\mathcal{T}$, we follow the finite element method of [Mey+03]; see also [Des+99; Bot+10; Cra+13; Jac13].

To discretize the Laplacian operator over a mesh $\mathcal{T}$, the idea is to integrate a piecewise-linear vector function

$$\mathbf{u} : \mathcal{T} \to \mathbb{R}^3$$

over a local averaging domain around the vertex v_i (Fig. 2.8 and Fig. 2.10):

$$\iint_{\mathrm{A}_i} \Delta_{\mathcal{S}} \mathbf{u} \, \mathrm{dA}. \tag{2.29}$$

Divergence theorem. Recall that *Stokes' theorem* from vector calculus states that integrating a differential form α over the boundary $\partial\mathcal{S}$ of an orientable manifold $\mathcal{S}$ is equal to integrating its exterior derivative $\mathrm{d}\alpha$ over the entire domain:

$$\int_{\mathcal{S}} \mathrm{d}\alpha = \int_{\partial\mathcal{S}} \alpha. \tag{2.30}$$

In a special case when $\mathcal{S}$ is a 2-manifold surface, integrating the divergence of a vector field $\mathbf{X}$ over a 2-dimensional averaging domain $\mathrm{A} \subset \mathcal{S}$ can be expressed as integration over the closed boundary $\partial\mathrm{A}$. This is expressed in the *divergence theorem*:

$$\iint_{\mathrm{A}} \nabla \cdot \mathbf{X} \, \mathrm{dA} = \oint_{\partial \mathrm{A}} \mathbf{N} \cdot \mathbf{X} \, \mathrm{d}s, \tag{2.31}$$

where $\mathbf{N}$ is the unit normal field along $\partial\mathrm{A}$, and $\oint$ denotes integration along a closed curve. Using the divergence theorem, Eq. (2.29) can be simplified as

$$\iint_{\mathrm{A}_i} \Delta_{\mathcal{S}} \mathbf{u} \, \mathrm{dA} = \iint_{\mathrm{A}_i} \nabla_{\mathcal{S}} \cdot \nabla_{\mathcal{S}} \mathbf{u} \, \mathrm{dA} = \oint_{\partial \mathrm{A}_i} \nabla_{\mathcal{S}} \mathbf{u} \cdot \mathbf{N} \, \mathrm{d}s, \tag{2.32}$$

where $\mathbf{N}$ represents the outward-pointing normal along the boundary $\partial\mathrm{A}_i$.

Finite elements. To evaluate Eq. (2.32), we first discretize the gradient $\nabla_{\mathcal{S}}\mathbf{u}$. To this end, the piecewise linear *hat functions* φ_n over the vertices of $\mathcal{T}$ (Fig. 2.9) are defined via

$$\varphi_n(v_m) = \begin{cases} 1 & n = m, \\ 0 & n \neq m. \end{cases}$$

The piecewise linear function $\mathbf{u}$ can then be expressed in terms of the hat functions φ_n as

$$\mathbf{u} = \sum_n u_n \varphi_n.$$

As each φ_n is piecewise-linear, its gradient $\nabla_{\mathcal{S}}\varphi_n$ is *piecewise-constant* and vanishing on all triangles not containing the vertex v_n. Inside the triangle T with vertices $\mathbf{v}_i, \mathbf{v}_j, \mathbf{v}_k$, edges $\mathbf{e}_{ij}, \mathbf{e}_{jk}, \mathbf{e}_{kj}$ and interior angles $\theta_i, \theta_j, \theta_k$ (Fig. 2.8), we have

$$\nabla_{\mathcal{S}}\mathbf{u} = \nabla_{\mathcal{S}} \sum_n u_n \varphi_n = \sum_n u_n \nabla_{\mathcal{S}} \varphi_n = u_i \nabla_{\mathcal{S}} \varphi_i + u_j \nabla_{\mathcal{S}} \varphi_j + u_k \nabla_{\mathcal{S}} \varphi_k + \sum_{n \neq i,j,k} u_n \underbrace{\nabla_{\mathcal{S}} \varphi_n}_{=\mathbf{0}}.$$

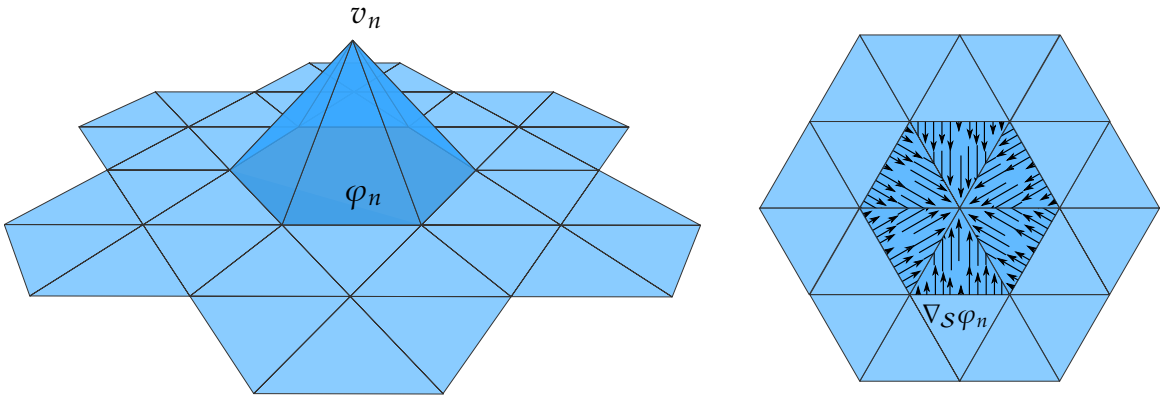


Figure 2.9: Hat function φ_n and the direction of the gradient vector field $\nabla_S \varphi_n$, which vanishes on triangles not incident to v_n.

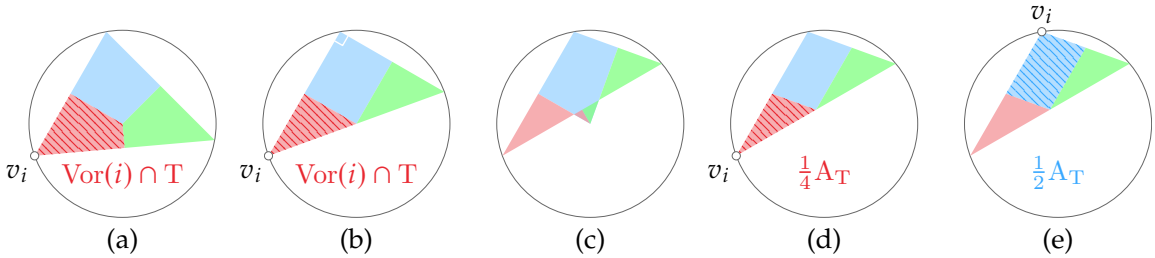


Figure 2.10: (a) Voronoi areas of the vertices in a triangle are obtained by connecting edge midpoints to the center of the circumcircle and computing the areas of the formed quadrilaterals. For a right triangle (b), one of the midpoints is identical to circle's center, and the two quads collapse to triangles. To avoid problems with possible negative areas (c), a modified version is used in obtuse triangles: instead of connecting the midpoints to circle's center, the midpoint opposite to the obtuse angle is connected to the other two midpoints (d-e). This yields a continuous definition of the *mixed Voronoi area* A_i. See also Algorithm 2.11.

```
A_i = 0
for each triangle T incident to the vertex v_i do
    if θ_i is non-obtuse then                  # use the Voronoi area, Fig. 2.10 (a-b)
        A_i += Area(Voronoi(i) ∩ T)
    else                                       # alternative definition, Fig. 2.10 (d-e)
        if θ_i is obtuse then
            A_i += A_T/2                       # blue quadrilateral
        else
            A_i += A_T/4                       # red triangle
        end if
    end if
end for
```

Algorithm 2.11: Computation of the mixed Voronoi area A_i [Mey+03]

The direction of the gradient $\nabla_{\mathcal{S}}\varphi_i$ in the triangle T is parallel to T and perpendicular to the opposite edge $\mathbf{e}_{jk}$ (Fig. 2.9 right). The magnitude $\left\|\nabla_{\mathcal{S}}\varphi_i\right\|$ is inversely proportional to the height $\mathbf{h}_{jk}$ of T with $\mathbf{e}_{jk}$ as a base (Fig. 2.12):

$$\left\|\nabla_{\mathcal{S}}\varphi_i\right\| = \frac{1}{\left\|\mathbf{h}_{jk}\right\|} = \frac{1}{\sin\theta_k \left\|\mathbf{e}_{ik}\right\|} = \frac{\left\|\mathbf{e}_{jk}\right\|}{\sin\theta_k \left\|\mathbf{e}_{jk}\right\| \left\|\mathbf{e}_{ik}\right\|} = \frac{\left\|\mathbf{e}_{jk}\right\|}{2\mathrm{A_T}}.$$

This means that the gradient can be written as $\nabla_{\mathcal{S}}\varphi_i = \frac{\mathbf{e}_{jk}^{\perp}}{2\mathrm{A_T}}$ where $\mathbf{e}^{\perp}$ denotes the vector obtained by rotating $\mathbf{e}$ by 90 degrees counterclockwise in the plane of T (Fig. 2.12) and $\mathrm{A_T}$ denotes the area of T.

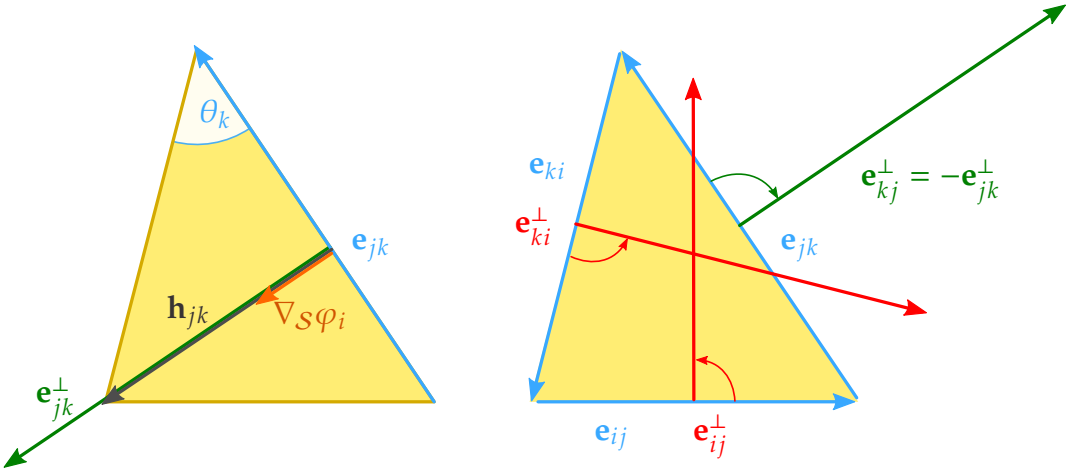


Figure 2.12: Vectors used for computing $\nabla_{\mathcal{S}}\mathbf{u}$ in Eq. (2.35)

Discrete gradient. For a fixed parameter inside T, the hat functions satisfy the partition of unity: $\varphi_i + \varphi_j + \varphi_k = 1$. Differentiating this equality yields $\nabla_{\mathcal{S}}\varphi_i + \nabla_{\mathcal{S}}\varphi_j + \nabla_{\mathcal{S}}\varphi_k = \mathbf{0}$. Therefore, the constant expression for the gradient $\nabla_{\mathcal{S}}\mathbf{u}$ on the triangle T is

$$\nabla_{\mathcal{S}}\mathbf{u} = \nabla_{\mathcal{S}}\varphi_j(u_j - u_i) + \nabla_{\mathcal{S}}\varphi_k(u_k - u_i) = \frac{1}{2\mathrm{A_T}}\left((u_j - u_i)\,\mathbf{e}_{ki}^{\perp} + (u_k - u_i)\,\mathbf{e}_{ij}^{\perp}\right). \quad (2.33)$$

Discrete Laplacian. Now, consider the integration in Eq. (2.32) for each triangle individually. Denoting by $\mathbf{m}_{ij} = \frac{1}{2}(\mathbf{v}_i + \mathbf{v}_j)$ the midpoint of the edge $\mathbf{e}_{ij}$ (Fig. 2.8), we can write

$$\int_{\partial \mathrm{A}_i \cap \mathrm{T}} \nabla_{\mathcal{S}}\mathbf{u} \cdot \mathbf{N}\, ds = \nabla_{\mathcal{S}}\mathbf{u} \cdot \left(\mathbf{m}_{ij} - \mathbf{m}_{ki}\right)^{\perp} = \frac{1}{2}\nabla_{\mathcal{S}}\mathbf{u} \cdot \mathbf{e}_{kj}^{\perp}. \quad (2.34)$$

Substituting for $\nabla_{\mathcal{S}}\mathbf{u}$ by Eq. (2.33) gives

$$\begin{aligned}\int_{\partial \mathrm{A}_i \cap \mathrm{T}} \nabla_{\mathcal{S}}\mathbf{u} \cdot \mathbf{N}\, \mathrm{d}s &= (u_j - u_i)\frac{\mathbf{e}_{ki}^{\perp} \cdot \mathbf{e}_{kj}^{\perp}}{4\mathrm{A}_\mathrm{T}} + (u_k - u_i)\frac{\mathbf{e}_{ij}^{\perp} \cdot \mathbf{e}_{kj}^{\perp}}{4\mathrm{A}_\mathrm{T}}. \\ &= (u_j - u_i)\frac{\mathbf{e}_{ki} \cdot \mathbf{e}_{kj}}{4\mathrm{A}_\mathrm{T}} + (u_k - u_i)\frac{\mathbf{e}_{ij} \cdot \mathbf{e}_{kj}}{4\mathrm{A}_\mathrm{T}}.\end{aligned} \tag{2.35}$$

The area of T is given by Eq. (2.28). Since

$$\cos\theta_j = \frac{\mathbf{e}_{ij} \cdot \mathbf{e}_{kj}}{\left\|\mathbf{e}_{ij}\right\| \left\|\mathbf{e}_{kj}\right\|}, \quad \cos\theta_k = \frac{\mathbf{e}_{ik} \cdot \mathbf{e}_{jk}}{\left\|\mathbf{e}_{ik}\right\| \left\|\mathbf{e}_{jk}\right\|},$$

the integral from Eq. (2.35) is equal to

$$\begin{aligned}\int_{\partial \mathrm{A}_i \cap \mathrm{T}} \nabla_{\mathcal{S}}\mathbf{u} \cdot \mathbf{N}\, \mathrm{d}s &= (u_j - u_i)\, \frac{\mathbf{e}_{ki} \cdot \mathbf{e}_{kj}}{2\sin\theta_k \left\|\mathbf{e}_{ki}\right\| \left\|\mathbf{e}_{kj}\right\|} + (u_k - u_i)\, \frac{\mathbf{e}_{ij} \cdot \mathbf{e}_{kj}}{2\sin\theta_j \left\|\mathbf{e}_{ij}\right\| \left\|\mathbf{e}_{kj}\right\|} \\ &= \frac{1}{2}\left((u_j - u_i)\, \frac{\cos\theta_j}{\sin\theta_j} + (u_k - u_i)\, \frac{\cos\theta_k}{\sin\theta_k}\right) \\ &= \frac{1}{2}\left((u_j - u_i)\cot\theta_j + (u_k - u_i)\cot\theta_k\right).\end{aligned}$$

Integrating over the whole domain A_i and rearranging the terms yields the *cotangent formula*

$$\int_{\partial \mathrm{A}_i} \nabla_{\mathcal{S}}\mathbf{u} \cdot \mathbf{N}\, \mathrm{d}s = \iint_{\mathrm{A}_i} \Delta_{\mathcal{S}}\mathbf{u}\, \mathrm{d}A = \sum_{v_j \in \operatorname{link}(v_i)} \frac{1}{2}\left(\cot\alpha_{ij} + \cot\beta_{ij}\right)\left(u_j - u_i\right) \tag{2.36}$$

where α_{ij}, β_{ij} are the angles opposite to the edge e_{ij} (Fig. 2.8). Averaging by the Voronoi area A_i (Fig. 2.10) finally gives the discrete Laplacian at the vertex v_i

$$\Delta_{\mathcal{S}} u_i = (\Delta_{\mathcal{S}}\mathbf{u})_i = \frac{1}{2\mathrm{A}_i} \sum_{v_j \in \operatorname{link}(v_i)} \left(\cot\alpha_{ij} + \cot\beta_{ij}\right)\left(u_j - u_i\right) \tag{2.37}$$

where the set $\operatorname{link}(v_i)$ denotes the one-ring neighborhood of v_i.

Discrete mean curvature. Using the discrete Laplacian (2.37) in connection with Eq. (2.11) yields a discretization of the mean curvature vector $\mathbf{H}$ on a triangle mesh:

$$\mathbf{H}_i = \Delta_{\mathcal{S}}\mathbf{v}_i = \frac{1}{2\mathrm{A}_i} \sum_{v_j \in \operatorname{link}(v_i)} \left(\cot\alpha_{ij} + \cot\beta_{ij}\right)\left(\mathbf{v}_j - \mathbf{v}_i\right). \tag{2.38}$$

Consequently, the absolute mean curvature at the vertex v_i is (cf. Eq. (2.10), p. 26)

$$\mathrm{H}_i = \frac{1}{2}\left\|\mathbf{H}_i\right\|. \tag{2.39}$$

2.7.2 Alternative definitions

Though widely used, Eq. (2.37) is only one possible discretization of $\Delta_{\mathcal{S}}$. The general form is

$$\Delta_{\mathcal{S}} u_i = w_i \sum_{v_j \in \operatorname{link}(v_i)} w_{ij}(u_j - u_i),$$

where w_i, w_{ij} are the vertex and edge weights respectively, with $\sum_j w_{ij} = 1$. The cotangent Laplacian thus uses the weights

$$w_i = 1/\mathrm{A}_i, \quad w_{ij} = \frac{1}{2}\left(\cot \alpha_{ij} + \cot \beta_{ij}\right).$$

Another discretization is the uniform (graph) Laplacian which does not account for the geometry of the mesh [Fie73; Tau95]

$$w_i = 1, \quad w_{ij} = 1/\left|\operatorname{link}(v_i)\right|.$$

2.7.3 Matrix form and higher-order Laplacians

For efficient manipulation, the discrete Laplacian $\Delta_{\mathcal{S}}$ is usually represented in matrix form. Vertex masses are arranged into a diagonal matrix M called the *mass matrix*: $\mathrm{M}_{ii} = 1/w_i$. Edge weights are stored in a symmetric matrix C called the *cotangent matrix*:

$$\mathrm{C}_{ij} = \begin{cases} -\sum_{k \in \operatorname{link}(v_i)} w_{ik}, & i = j, \\ w_{ij}, & j \in \operatorname{link}(v_i), \\ 0, & \text{otherwise.} \end{cases}$$

Discrete Laplacian is given by the matrix

$$\mathrm{L} = \mathrm{M}^{-1}\mathrm{C}. \tag{2.40}$$

Higher-order Laplacians are defined recursively

$$\begin{aligned} \Delta^0_{\mathcal{S}} f_i &= f_i, \\ \Delta^k_{\mathcal{S}} f_i &= w_i \sum_{v_j \in \operatorname{link}(v_i)} w_{ij}(\Delta^{k-1}_{\mathcal{S}} f_j - \Delta^{k-1}_{\mathcal{S}} f_i). \end{aligned}$$

In matrix form, this corresponds to matrix multiplication $\Delta^k_{\mathcal{S}} = \mathrm{L}^k$.

2.8 Variational modeling of shapes

Variational modeling is a popular approach to shape fitting, reconstruction and deformation, due to its theoretical guarantees and straightforward interpretation. In a variational framework, the shape $\mathcal{S}$ is computed as the minimizer of some energy E subject to boundary conditions. The energy functional E typically involves quantities such as area and curvature.

Minimal surfaces locally minimize the surface area; this property is expressed via the membrane energy

$$\mathrm{E}_{\text{membrane}} = \iint_{\mathcal{S}} \mathrm{dA}, \tag{2.41}$$

where dA denotes the surface area element. Minimizing the total curvature, we get the thin-plate energy [Duc77]

$$\mathrm{E}_{\text{thinPlate}} = \iint_{\mathcal{S}} \kappa_1^2 + \kappa_2^2 \ \mathrm{dA}. \tag{2.42}$$

Finally, minimizing the variation of curvature yields the minimum variation surface [MS92]

$$\mathrm{E}_{\text{minVar}} = \iint_{\mathcal{S}} \left\| \frac{\partial \kappa_1}{\partial \mathbf{t}_1} \right\|^2 + \left\| \frac{\partial \kappa_2}{\partial \mathbf{t}_2} \right\|^2 \mathrm{dA}. \tag{2.43}$$

These energies are nonlinear and their minimization is computationally expensive, which makes them unsuitable for interactive applications. In practice, these energies are approximated by the linearized functionals: the Dirichlet energy E_1, the Laplacian energy E_2, and the Laplacian gradient energy E_3 [Jac13]:

$$\mathrm{E}_1 = \iint_{\Omega} \|\nabla \mathbf{x}\|^2 \, \mathrm{dA}, \tag{2.44a}$$

$$\mathrm{E}_2 = \iint_{\Omega} \|\Delta \mathbf{x}\|^2 \, \mathrm{dA}, \tag{2.44b}$$

$$\mathrm{E}_3 = \iint_{\Omega} \|\nabla \Delta \mathbf{x}\|^2 \, \mathrm{dA}. \tag{2.44c}$$

Proposition 2.35. *Minimizers of the linearized energies (2.44) correspond to the solutions of the Euler-Lagrange equations*

$$\Delta^k \mathbf{x} = \mathbf{0}, \quad k = 1, 2, 3. \tag{2.45}$$

Proof. Let us derive the Euler-Lagrange equation of the Dirichlet boundary value problem

$$\min_{\mathbf{x}} \mathrm{E}_1(\mathbf{x}) \quad \text{s.t.} \quad \mathbf{x}|_{\partial\Omega} = \mathbf{x}_0. \tag{2.46}$$

To this end, we use the calculus of variations [Kob97; Bot+10]. The derivation works analogically for the other two energies.

Let us assume that $\mathbf{x} : \Omega \to \mathbb{R}^3$ is the minimizer of E_1. Then it holds that $\mathrm{E}_1(\mathbf{x}) \leq \mathrm{E}_1(\mathbf{x} + \lambda\mathbf{f})$ for an arbitrary test function $\mathbf{f}$ which vanishes on the boundary of Ω:

$$\mathbf{f} : \Omega \to \mathbb{R}^3, \quad \mathbf{f}|_{\partial\Omega} = \mathbf{0}. \tag{2.47}$$

As a function of λ, the energy E_1 therefore attains its minimum at $\lambda = 0$:

$$\frac{\partial}{\partial\lambda}\mathrm{E}_1\left(\mathbf{x} + \lambda\mathbf{f}\right)\Big|_{\lambda=0} = 0. \tag{2.48}$$

Simplifying the left side yields

$$\frac{\partial}{\partial\lambda}\iint_\Omega \left\|\nabla\left(\mathbf{x} + \lambda\mathbf{f}\right)\right\|_{\mathrm{F}}^2 \mathrm{dA} = 2\iint_\Omega \langle\nabla\mathbf{f}, \nabla\mathbf{x}\rangle_{\mathrm{F}} + \lambda\left\|\nabla\mathbf{f}\right\|_{\mathrm{F}} \mathrm{dA}. \tag{2.49}$$

The last term vanishes for $\lambda = 0$, so the substitution into Eq. (2.48) gives

$$\iint_\Omega \langle\nabla\mathbf{f}, \nabla\mathbf{x}\rangle_{\mathrm{F}} \,\mathrm{dA} = 0. \tag{2.50}$$

Recall that the divergence product rule states that given a scalar function $f : \Omega \to \mathbb{R}$ and a vector function $\mathbf{g} : \Omega \to \mathbb{R}^2$, the following identity holds:

$$\nabla \cdot (f\mathbf{g}) = (\nabla f) \cdot \mathbf{g} + f(\nabla \cdot \mathbf{g}). \tag{2.51}$$

Denote by f_i the elements of $\mathbf{f}$ and by x_i the elements of $\mathbf{x}$, $i = 1, 2, 3$. Applying the product rule from Eq. (2.51) for $f = f_i, \mathbf{g} = \nabla x_i$ yields

$$\nabla \cdot (f_i \nabla x_i) = \nabla f_i \cdot \nabla x_i + f_i \underbrace{(\nabla \cdot \nabla x_i)}_{\Delta x_i}.$$

Equivalently,

$$\iint_\Omega \nabla \cdot (f_i \nabla x_i) \,\mathrm{dA} = \iint_\Omega \nabla f_i \cdot \nabla x_i \,\mathrm{dA} + \iint_\Omega f_i \Delta x_i \,\mathrm{dA}. \tag{2.52}$$

Using the divergence theorem from Eq. (2.31), the left side is equal to

$$\iint_\Omega \nabla \cdot (f_i \nabla x_i) \,\mathrm{dA} = \oint_{\partial\Omega} \mathbf{N} \cdot (f_i \nabla x_i) \,\mathrm{d}s = 0,$$

where the last identity follows from the definition of $\mathbf{f}$ in Eq. (2.47) since $f_i|_{\partial\Omega} = 0$.

$$0 = \iint_\Omega \sum_{i=1}^{3} \nabla f_i \cdot \nabla x_i \,\mathrm{dA} + \iint_\Omega \sum_{i=1}^{3} f_i \Delta x_i \,\mathrm{dA} = \iint_\Omega \langle\nabla\mathbf{f}, \nabla\mathbf{x}\rangle_{\mathrm{F}} \,\mathrm{dA} + \iint_\Omega \mathbf{f} \cdot \Delta\mathbf{x} \,\mathrm{dA}.$$

The first integral is zero, see Eq. (2.50); the second integral vanishes if and only if $\Delta\mathbf{x} = \mathbf{0}$ since the choice of $\mathbf{f}$ is arbitrary. □

A triangle mesh discretization of Eq. (2.45) is obtained by using the discrete Laplacian from Section 2.7. This yields the linear system

$$\mathrm{L}^k \mathrm{V} = 0 \tag{2.53}$$

where the matrix $\mathrm{V} = (\mathbf{v}_1, \ldots, \mathbf{v}_n)^\mathsf{T}$ contains vertex positions in rows. To get a non-trivial solution of this system, suitable boundary conditions need to be imposed. In the context of surface modeling, first-order positional constraints are the most common. Higher-order constraints are sometimes needed, including prescription of normal or tangent vectors, often in combination with positional constraints.

Two principal approaches are adopted in order to integrate the positional constraints into the system (2.53). We introduce the two approaches on the example of $\Delta\mathbf{x} = 0$. Suppose $\mathcal{V}_c = \{v_1, \ldots, v_c\}$ is the set of *constrained vertices* and $\mathcal{V}_f = \{v_{c+1}, \ldots, v_{c+f} = v_n\}$ is the set of remaining *free vertices*.

Hard constraints are imposed by eliminating the rows and columns corresponding to the fixed vertices from L^k – the positions are therefore exactly interpolated. This is equivalent to solving the $n \times n$ system

$$\begin{bmatrix} \mathrm{L}_f \\ \mathrm{I_c} \quad 0 \end{bmatrix} \begin{bmatrix} \mathrm{V}_c \\ \mathrm{V}_f \end{bmatrix} = \begin{bmatrix} 0 \\ \mathrm{V}_c^* \end{bmatrix} \tag{2.54}$$

where $\mathrm{I_c}$ is the $c \times c$ identity matrix, L_f are the f rows of the Laplacian matrix corresponding to the free vertices $\mathcal{V}_f$, and $\mathrm{V}_c, \mathrm{V}_f$ are the computed positions of constrained and free vertices, respectively; * denotes the fixed positions. Splitting the matrix $\mathrm{L}_f = \begin{bmatrix} \mathrm{L}_{fc} & \mathrm{L}_{ff} \end{bmatrix}$ into two blocks corresponding to constrained vertices and free vertices, the system (2.54) is equivalent to

$$\mathrm{L}_{ff}\mathrm{V}_f = -\mathrm{L}_{fc}\mathrm{V}_c^*.$$

On the other hand, *soft constraints* are not satisfied exactly – they are approximated by including additional rows in the system:

$$\underbrace{\begin{bmatrix} \mathrm{L} \\ \mathrm{I_c} \quad 0 \end{bmatrix}}_{\mathrm{A}} \underbrace{\begin{bmatrix} \mathrm{V}_c \\ \mathrm{V}_f \end{bmatrix}}_{\mathrm{X}} = \underbrace{\begin{bmatrix} 0 \\ \mathrm{V}_c^* \end{bmatrix}}_{\mathrm{B}}. \tag{2.55}$$

This over-determined $(n + c) \times n$ system is then solved in the least-squares sense, i.e. X is found by minimizing

$$\|\mathrm{AX} - \mathrm{B}\|^2 = \|\mathrm{LX}\|^2 + \sum_{v_i \in \mathcal{V}_c} \left\|\mathbf{v}_i - \mathbf{v}_i^*\right\|^2. \tag{2.56}$$

The explicit solution is given by the normal equations

$$\mathrm{A}^\mathsf{T}\mathrm{AX} = \mathrm{A}^\mathsf{T}\mathrm{B}. \tag{2.57}$$

3

Curve networks from orientations

First part of our framework is the reconstruction of a curve network. Chapter 1 introduced the problem of acquisition and reconstruction of shapes using sensors; Chapter 2 presented the mathematical foundations of this thesis, forming the theoretical basis upon which we will build our algorithms. In this chapter, we describe the first contribution of this thesis: a method for reconstructing smooth curve networks from orientation and distance data provided by sensors.

After scanning the surface of a physical object using one of our acquisition devices (Section 1.4), we obtain a sample of surface orientations with associated distance parameters. In our dynamic setup, these orientations are acquired along a network of smooth curves on the surface (Example 2.33, p. 40). The topology of the network is not limited by the devices that we use.

The goal of the algorithms presented in this chapter is to reconstruct a smooth curve network from the acquired orientations while respecting the topology and consistency of the network. Our approach is driven by a simple principle mostly overlooked in previous works: at each intersection in a curve network, the positions and normals of two intersecting curves have to coincide. Guided by this idea, we

develop automatic network reconstruction procedures that enforce the constraints at intersections by design.

Our method addresses the following three challenges, which we have identified prior to its development.

Unknown positions. Sensors measure local orientations of the surface – no absolute positions in the world space nor relative positions of adjacent sensors are known.

Inconsistent data. Intersecting curves often provide conflicting data, for instance two different normals for the same point in the world space.

Sensor noise. Raw data from inertial sensors are noisy and need to be pre-processed before they can be used to recover the positions.

In order to resolve these issues, our method works in two stages: filtering of orientations (Section 3.5), and reconstruction of positions (Section 3.6).

Filtering. Acquired orientations, which are sampled non-uniformly, are first pre-processed to obtain a uniform sampling without outliers (Section 3.5.3). The computed uniform samples are then smoothed using non-parametric regression in the space of orientations (Section 3.5.4). We formulate the regression as an optimization problem with a custom cost function, which respects the intersection constraints placed on the orientations.

Reconstruction. The smooth curve network is retrieved by integrating the filtered orientations. The integration is done by solving a global Poisson system. The system is formulated in such a way that the resulting network is well-connected and smooth.

The advantages of our approach are multiple. First, the resulting method is fully automatic and does not require manual corrections in order to obtain well-connected intersecting curves. In addition to satisfying the topological constraints, the reconstructed network has continuous normals at curve intersections. This is an important feature for a proper subsequent processing of the network.

Second, while the filtering is performed automatically, the user has some degree of control over the result via a small set of parameters. In Section 3.7 we discuss possible strategies for choosing the optimal parameters.

Third, the method is fast for a reasonably chosen sampling density. In practice, this means that the user can adjust the filtering weights interactively, and immediately gets the recomputed network as visual feedback. Results and performance statistics are also discussed in Section 3.7.

Note that prior to our two-stage reconstruction, we use the existing methods to estimate the orientations from raw sensors measurements. The theory behind these methods is summarized in Section 3.4.

We begin this chapter by reviewing the related methods. While in Section 1.3 we described the state-of-the-art methods dealing with sensors data, here we summarize methods that are directly related to our algorithms.

The work presented in this chapter was originally published in [Sta+17a; Sta+17b] and presented at [SMI17]. See the list of publications on p. 187.

3.1 Related work

Previous work related to this chapter is organized into three sections. Section 3.1.1 describes methods and devices used for reconstruction and sketching of 3D curves. Section 3.1.2 summarizes algorithms for estimation and filtering of orientations. Finally, Section 3.1.3 describes the use of Poisson's equation in the context of shape reconstruction.

3.1.1 Shape from curves

The problem of generating shapes from collections of curves has been well studied in computer-aided geometric design (CAGD) and all standard techniques can be found in Farin [Far02]. In the recent years, the interest in this problem was re-ignited due to its applications in sketch-based modeling [IMT99; Nea+07; BBS08; AJC11; Xu+14] and in virtual reality (VR) systems such as Google's TiltBrush [Goo] (Fig. 3.1 left).

Inspired by years of research in CAGD, the surface from curves paradigm is invaluable for shape design in modern sketching and VR systems. Recently, Arora et al. [Aro+17] have studied issues with accurate design of shapes in the existing VR systems. Their study suggests that users struggle even with simple tasks (drawing a closed circle) when sketching in three dimensions. A system like ours can help in solving such accuracy and consistency issues.

Although equally important, the problem of *reconstruction* of existing physical shapes from a collection of curves received less attention. 3D scanners usually provide large amounts of data and tend to ignore intrinsic structure of the scanned shape. Alternatively, objects can be defined by their characteristic curves as often done in perception and sketch-based modeling. Cao et al. [CNW16] detected characteristic curves in noisy point clouds, then use these curves for surface reconstruction.

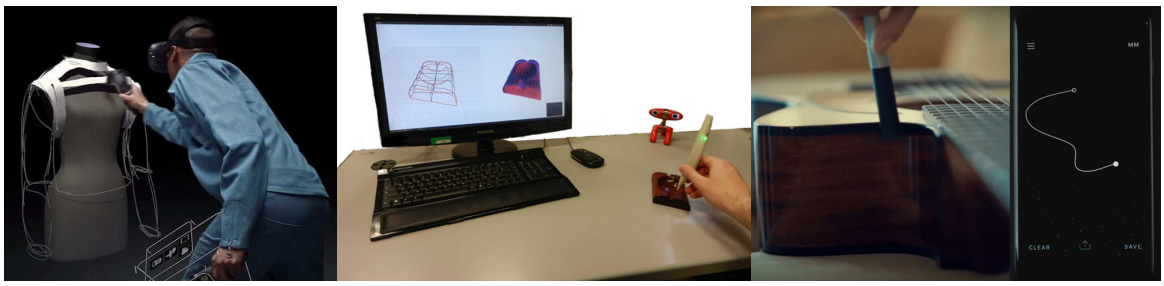


Figure 3.1: Recent curve acquisition devices that use inertial sensors. Left to right, Tilt Brush [Goo], SmartPen [Mil+16] and 01 by Instrumments [Ins].

The use of inertial measurement units (IMUs) for shape acquisition might provide a good alternative in situations for which the optical methods do not yield proper results, positioning sensors along object's characteristic curves. Milosevic et al. [Mil+16] introduced SmartPen, a low-cost system for capturing 3D curves, which combines an IMU with a stereo camera (Fig. 3.1 middle). By combining a stereo camera with a sensor unit, their system is a mixture of traditional 3D scanners and our shape from sensors setup. However, SmartPen's sensors only serve for determining device's orientation needed for estimating relative position of the tip of the pen. Much like traditional point-cloud scanners, the system relies on visual input to get 3D position of the device in world space; this limits the size of the scanned object.

A recent example of a curve acquisition device is the commercially available 01 by Instrumments [Ins], a dimensioning tool with an IMU and a laser (Fig. 3.1 right). Usage of this device for 3D curve reconstruction has yet to be demonstrated experimentally.

3.1.2 Orientation estimation and filtering

Orientation or *attitude* control has been studied extensively in aeronautics, where accurate algorithms are indispensable for correct estimation of vehicle's orientation with respect to celestial objects [MM00]. Noise in data is usually reduced using a Kalman filter – specific approaches depend on the representation used for orientations [CMC07] (Section 2.5). Markley et al. [Mar+07] described a classical algorithm for computing means in the group SO(3). An average rotation is defined as the minimizer of a weighted penalty function, and the corresponding unit quaternion is computed efficiently via eigendecomposition of a 4×4 matrix (Section 3.4).

Apart from statistical approaches such as Kalman filters, *geometric methods* can be used to denoise orientations. Manifold methods are often used for data regularization in image processing [Ros+14]. Shoemake [Sho85] introduced quaternion

splines using spherical Bézier curves; Nielson [Nie04] later extended this algorithm and introduced $v-$quaternion splines. Although spline methods work well for interpolation of rotations in a sequence of keyframes, the Bézier representation is not suitable for filtering noisy orientations. Energy-minimizing splines for data in Euclidean spaces have been well-studied. Reinsch [Rei67] introduced smoothing splines that minimize stretching and bending while approximating given data in a Euclidean space. Hofer and Pottmann [HP04] described a method for computing such splines for data on a Riemannian manifold $\mathcal{M}$. Data are first optimized in the ambient (Euclidean) space, then projected onto $\mathcal{M}$; these two steps are repeated until convergence. An equivalent result can be found by optimizing data directly on the manifold [BA11]. In this sense, classical splines are obtained for $\mathcal{M} = \mathbb{R}^d$. We use the latter approach for smoothing splines on the manifold SO(3) to filter raw orientation data from sensors (Section 3.5).

3.1.3 Poisson reconstruction

Poisson's equation often arises in the context of reconstruction of shapes:

$$\Delta\phi = \nabla \cdot \mathbf{f},$$

where ϕ is a scalar field and $\mathbf{f}$ is a vector field on a manifold. Kazhdan et al. [KBH06] resolved a Poisson problem $\Delta\phi = \nabla \cdot \mathbf{N}$ to find the indicator function ϕ, which implicitly defines the surface, from a sample of the surface's oriented normal field $\mathbf{N}$. This approach is a state-of-the-art method for surface reconstruction from unorganized points clouds. Our reconstruction algorithm is inspired by the work of Crane et al. [CPS13] who used Poisson's equation $\Delta x = \nabla \cdot \mathbf{T}$ to retrieve positions x of a closed planar curve from its tangent field $\mathbf{T}$ during isometric curvature flow. We extend this approach to closed curve networks on surfaces, and use it to reconstruct positions of the scanned curves (Section 3.6).

3.2 Problem statement

Given a smooth connected 2-manifold surface $\mathcal{S} \subset \mathbb{R}^3$, we consider a curve network Γ, which is a collection of G^1 smooth curves embedded on $\mathcal{S}$. The network Γ is therefore a topological subspace of $\mathcal{S}$. Curves in Γ divide the surface into a finite number of cells forming a two-dimensional *cell complex* $\mathcal{C} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$ on $\mathcal{S}$ (cf. Example 2.33 and Fig. 2.7 on p. 40), consisting of nodes $\mathcal{V}$ (0-cells), segments $\mathcal{E}$ (1-cells) and cycles $\mathcal{F}$ (2-cells). Moreover, we suppose the cell complex is connected and the cycles are contractible (Fig. 3.2). Each curve $\gamma \in \Gamma$ can be thought of as a *path* in $\mathcal{C}$. We say that two curves intersect at a node $v \in \mathcal{V}$ if their corresponding paths contain v.

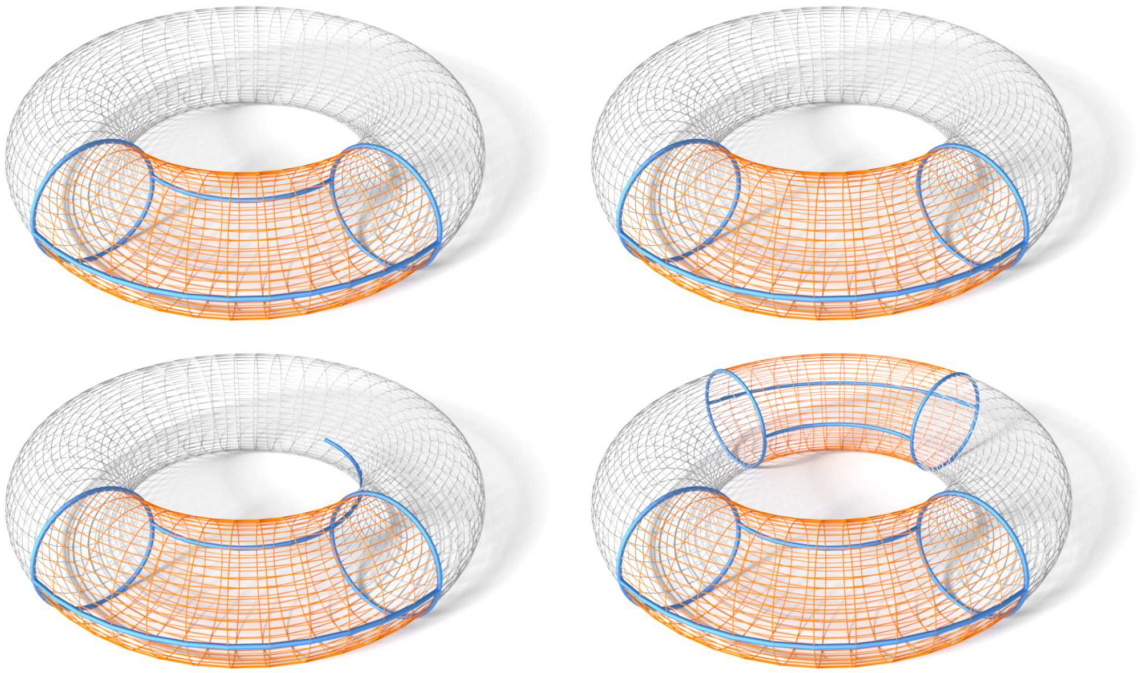


Figure 3.2: Curve networks: in this example, only the top left curve network is a valid input to our algorithm. The other three networks have non-contractible cycles (*top right*), violate the definition of a cell complex (*bottom left*) or they are disconnected (*bottom right*).

Denote by $\mathbf{x} : [0, L] \to \mathbb{R}^3$ the natural parametrization of $\gamma \in \Gamma$ where L is the length of γ. For a fixed point $\mathbf{x}$ on the curve, the orthonormal Darboux frame $\mathcal{D} = \{\mathbf{t}, \mathbf{N}, \mathbf{B}\}$ is defined as (Section 2.3)

$$\mathbf{t} = \mathbf{x}', \quad \mathbf{N} \perp \mathrm{T}_{\mathbf{x}}\mathcal{S}, \quad \mathbf{B} = \mathbf{t} \times \mathbf{N}.$$

Here, $\mathrm{T}_{\mathbf{x}}\mathcal{S}$ denotes the two-dimensional tangent space of $\mathcal{S}$ at the point $\mathbf{x} \in \mathcal{S}$ and $\mathbf{N}$ is chosen as the outward surface normal. We represent the Darboux frame $\mathcal{D}$ as an orientation matrix $\mathrm{A} = [\mathbf{t}\ \mathbf{N}\ \mathbf{B}] \in \mathrm{SO}(3)$, which is a member of the special orthogonal group (cf. Eq. (2.19) in Section 2.5)

$$\mathrm{SO}(3) = \left\{ \mathrm{A} \in \mathbb{R}^{3\times 3} \ : \ \mathrm{A}^{\mathsf{T}}\mathrm{A} = \text{identity and } \det(\mathrm{A}) = 1 \right\}.$$

When talking about an "orientation" or a "frame", we refer to the same concept – the Darboux frame of a curve point with respect to the underlying surface. The projections on the tangent and the normal component of the orientation matrix A are denoted by

$$\mathrm{A}|_{\mathbf{t}} : \ \mathrm{A} = [\mathbf{t}\ \mathbf{N}\ \mathbf{B}] \mapsto \mathbf{t}, \quad \mathrm{A}|_{\mathbf{N}} : \ \mathrm{A} = [\mathbf{t}\ \mathbf{N}\ \mathbf{B}] \mapsto \mathbf{N}.$$

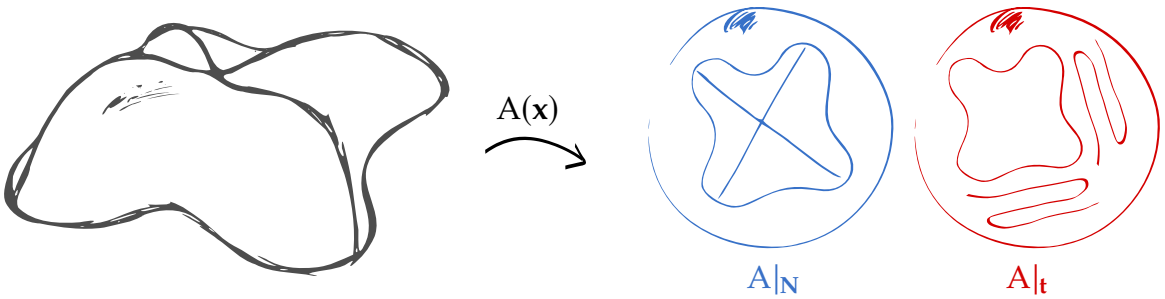


Figure 3.3: For a closed network, the orientation function $\mathbf{x} \mapsto \mathrm{A}(\mathbf{x})$ is *not* continuous (*right*) since the tangent vector is different for each curve passing through an intersection. Note that the projection on the normal component $\mathrm{A}|_{\mathbf{N}} : \Gamma \to \mathbb{S}^2$ (Gauss map of $\mathcal{S}$ restricted to curves) is closed as the surface normal varies continuously (*left*).

The function $\mathrm{A} : \Gamma \to \mathrm{SO}(3)$, which associates each curve point to its Darboux frame, is not continuous. Indeed, at intersections the tangent vector $\mathbf{t}$ differs for each adjacent curve (Fig. 3.3). Note however that the projection on the normal component $\mathrm{A}|_{\mathbf{N}}$ is continuous and is identical to the Gauss map of $\mathcal{S}$ restricted to the curves in Γ.

Our goal is to retrieve the unknown positions $\mathbf{x}$ provided a sample of orientations $\mathrm{A}_i = \mathrm{A}\left(\mathbf{x}(d_i)\right)$ for each curve γ at known distances $d_i \in [0, L]$. We suppose the topology of the underlying cell complex C is known.

The following section provides an overview of our method for solving this problem.

3.3 Method overview

In this section, we present the overview of our method for resolving the problem defined in Section 3.2. Given a set of noisy orientations acquired along a set of curves on a surface, our method ensures the reconstruction of a smooth curve network with normals. Fig. 3.4 summarizes the main steps.

First, we use the existing algorithms to estimate orientations from raw sensor measurements. As described in Section 1.2, each sensor measures a three-dimensional vector. To estimate the total orientation, the measured vectors are matched with the reference vectors, and the orientation, represented by a quaternion, is computed as a solution of an eigenproblem. More details on orientation estimation are given in Section 3.4.

Next, the orientations and distances acquired from the physical surface are prefiltered (Section 3.5.3) using efficient averaging schemes in the quaternion space

(Section 3.5.2). The pre-filtered data are smoothed using regression on the manifold of orientation matrices (Section 3.5.4) to obtain smoothly varying frames with consistent normals.

When working with orientations, we use two different representations (Section 2.5). *Quaternions* are used for estimation and for pre-filtering, leveraging efficient quaternion algorithms for estimation and averaging of orientations. For the subsequent regression, we represent the orientations by *rotation matrices* from SO(3). This choice facilitates the inclusion of normal consistency constraints, which are difficult to handle using the quaternion representation (Appendix B).

Finally, in Section 3.6, we formulate the reconstruction of positions from the filtered orientations as a Poisson problem. Poisson's equation is discretized using finite differences, while respecting the topology of the network. This results in a sparse global linear system for the whole network, which is efficiently resolved in order to retrieve the positions.

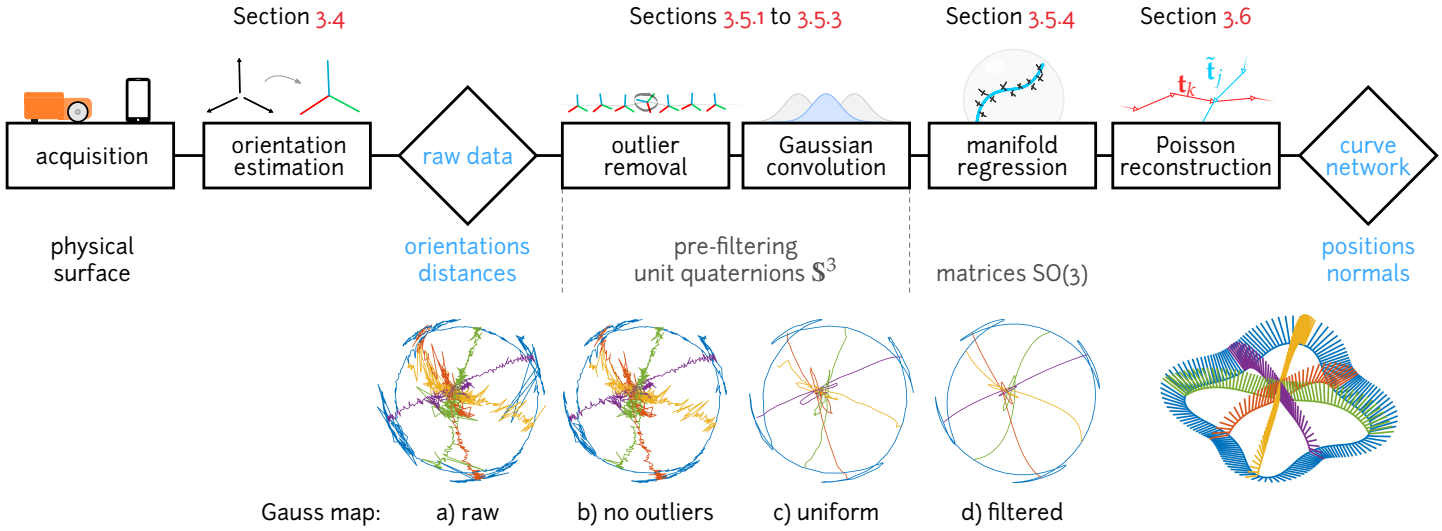


Figure 3.4: A schematic overview of the curve network reconstruction method presented in this chapter. Orientations after each step of the method are visualized via the Gauss map (normal component of the orientation plotted on the unit 2-sphere). Image on the right shows the reconstructed network.

3.4 Estimation of orientations

Our acquisition devices introduced in Section 1.4 are equipped with IMUs, which contain two sensors (Morphorider) or three sensors (smartphone). Each sensor measures a single *vector* in 3D. However, the problem defined in the previous section requires a sample of orientations $A_i \in SO(3)$ as the input.

In this section, we describe the pre-processing step, in which the individual sensor measurements are combined in order to determine the overall orientation of the object (Fig. 3.4). We use the existing methods to this end, which are formulated as solutions to the Wahba's problem [Wah65]:

Problem 3.1 (Wahba's problem)**.** *Given a set* $\{\mathbf{m}_i\}_{i=1}^n$ *of measured vectors with unit length and a set* $\{\mathbf{e}_i\}_{i=1}^n$ *of the corresponding unit reference vectors, find the orientation matrix* A *that minimizes the cost function* $J(\mathrm{A})$*:*

$$\min_{\mathrm{A}\in SO(3)} J(\mathrm{A}) \qquad \textit{with} \qquad J(\mathrm{A}) = \frac{1}{2}\sum_{i=1}^{n} \alpha_i \left\|\mathbf{m}_i - \mathrm{A}\mathbf{e}_i\right\|^2, \tag{3.1}$$

where α_i *are weights associated to each measurement.*

Loosely speaking, minimizing Wahba's energy means finding the rotation that provides the closest pairwise match between the set of measured vectors and the set of reference vectors. Note that in the case of Morphorider, two reference vectors are available ($n = 2$): $\mathbf{e}_1 = \mathbf{e}_{\mathrm{acc}}$ and $\mathbf{e}_2 = \mathbf{e}_{\mathrm{mag}}$ (see Section 1.2).

The following proposition clarifies the theoretical and practical aspects for solving Wahba's problem. Proofs in this section might be omitted on a first reading.

Proposition 3.2. *The cost function* $J(\mathrm{A})$ *from Eq.* (3.1) *is equal to*

$$J(\mathrm{A}) = \alpha - \operatorname{trace}\left(\mathrm{A}\mathrm{B}^{\mathsf{T}}\right) \tag{3.2}$$

where

$$\alpha = \sum_{i=1}^{n} \alpha_i, \qquad \mathrm{B} = \sum_{i=1}^{n} \alpha_i \, \mathbf{m}_i \, \mathbf{e}_i^{\mathsf{T}}.$$

Corollary 3.3. *Minimizing the cost* $J(\mathrm{A})$ *is equivalent to the maximization problem*

$$\max_{\mathrm{A}\in SO(3)} \operatorname{trace}\left(\mathrm{A}\mathrm{B}^{\mathsf{T}}\right). \tag{3.3}$$

Proof (Proof of Proposition 3.2). Simplifying the term inside the sum,

$$\left\|\mathbf{m}_i - \mathrm{A}\mathbf{e}_i\right\|^2 = \langle \mathbf{m}_i - \mathrm{A}\mathbf{e}_i, \mathbf{m}_i - \mathrm{A}\mathbf{e}_i\rangle = \underbrace{\left\|\mathbf{m}_i\right\|^2}_{=1} + \underbrace{\left\|\mathrm{A}\mathbf{e}_i\right\|^2}_{=1} - 2\langle \mathbf{m}_i, \mathrm{A}\mathbf{e}_i\rangle = 2(1 - \langle \mathbf{m}_i, \mathrm{A}\mathbf{e}_i\rangle).$$

Substituting into $J(\mathrm{A})$,

$$J(\mathrm{A}) = \frac{1}{2}\sum_{i=1}^{n} \alpha_i \left\|\mathbf{m}_i - \mathrm{A}\mathbf{e}_i\right\|^2 = \sum_{i=1}^{n} \alpha_i (1 - \langle \mathbf{m}_i, \mathrm{A}\mathbf{e}_i \rangle) = \underbrace{\sum_{i=1}^{n} \alpha_i}_{\alpha} - \sum_{i=1}^{n} \alpha_i \langle \mathbf{m}_i, \mathrm{A}\mathbf{e}_i \rangle .$$

Note that the standard dot product satisfies the relation

$$\left\langle \mathbf{x}, \mathbf{y} \right\rangle = \mathbf{x}^\mathsf{T}\mathbf{y} = \operatorname{trace}\left(\mathbf{y}\mathbf{x}^\mathsf{T}\right).$$

Therefore,

$$\langle \mathbf{m}_i, \mathrm{A}\mathbf{e}_i \rangle = \operatorname{trace}\left(\mathrm{A}\mathbf{e}_i\mathbf{m}_i^\mathsf{T}\right).$$

Denoting $\mathrm{B}_i = \mathbf{m}_i\mathbf{e}_i^\mathsf{T}$, the substitution of the above relation into the second sum in $J(\mathrm{A})$ yields

$$\sum_{i=1}^{n} \alpha_i \langle \mathbf{m}_i, \mathrm{A}\mathbf{e}_i \rangle = \sum_{i=1}^{n} \alpha_i \operatorname{trace}\left(\mathrm{A}\mathbf{e}_i\mathbf{m}_i^\mathsf{T}\right) = \sum_{i=1}^{n} \alpha_i \operatorname{trace}\left(\mathrm{A}\mathrm{B}_i^\mathsf{T}\right) = \operatorname{trace}\left(\mathrm{A}\sum_{i=1}^{n} \alpha_i \mathrm{B}_i^\mathsf{T}\right),$$

which completes the proof since $\mathrm{B} = \sum_{i=1}^{n} \alpha_i \mathrm{B}_i$. □

3.4.1 Davenport's q method

We now describe the *q method*, which is due to Davenport [Dav68]. This algorithm provides a closed-form solution to the maximization in Eq. (3.3) using the quaternion representation. Denote by $\mathbf{q} = (\mathbf{v}, w) = (x, y, z, w)$ a unit quaternion and recall that $\left\|\mathbf{q}\right\|^2 = \|\mathbf{v}\|^2 + w^2 = x^2 + y^2 + z^2 + w^2 = 1$; rewriting the Eq. (2.26) then gives the following relation between the quaternion $\mathbf{q}$ and the orientation matrix $\mathrm{A}(\mathbf{q})$:

$$\begin{aligned}\mathrm{A}(\mathbf{q}) &= (2w^2 - 1)\begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix} + 2\begin{pmatrix}x^2 & xy & xz\\ xy & y^2 & yz\\ xz & yz & z^2\end{pmatrix} - 2w\underbrace{\begin{pmatrix}0 & -z & y\\ z & 0 & -x\\ -y & x & 0\end{pmatrix}}_{[\mathbf{v}\times]} \\ &= (w^2 - \|\mathbf{v}\|^2)\,\mathrm{I}_3 + 2\mathbf{v}\mathbf{v}^\mathsf{T} - 2w[\mathbf{v}\times],\end{aligned} \tag{3.4}$$

where $[\mathbf{v}\times]$ is the cross-product matrix operator.

Proposition 3.4. *Parametrizing the orientation matrix* $\mathrm{A} = \mathrm{A}(\mathbf{q})$ *by the unit quaternion* $\mathbf{q} = (\mathbf{v}, w) = (x, y, z, w)$ *as in Eq. (3.4), the cost in the maximization problem (3.3) is equal to*

$$\operatorname{trace}\left(\mathrm{A}(\mathbf{q})\mathrm{B}^\mathsf{T}\right) = \mathbf{q}^\mathsf{T}\mathrm{K}\mathbf{q} \tag{3.5}$$

where the traceless (i.e. with zero trace) symmetric matrix K *is given by*

$$\mathrm{K} = \begin{pmatrix} \mathrm{B} + \mathrm{B}^\mathsf{T} - \operatorname{trace}(\mathrm{B})\,\mathrm{I}_3 & \mathbf{b} \\ \mathbf{b}^\mathsf{T} & \operatorname{trace}(\mathrm{B}) \end{pmatrix}. \tag{3.6}$$

The vector $\mathbf{b}$ *is defined via the relation* $[\mathbf{b}\times] = \mathrm{B}^\mathsf{T} - \mathrm{B}$.

Proof. Using the relation from Eq. (3.4) we can write

$$\mathrm{A}(\mathbf{q})\mathrm{B}^\mathsf{T} = (w^2 - \|\mathbf{v}\|^2)\mathrm{B}^\mathsf{T} + 2\mathbf{v}\mathbf{v}^\mathsf{T}\mathrm{B}^\mathsf{T} - 2w[\mathbf{v}\times]\mathrm{B}^\mathsf{T}.$$

Applying the trace operator on the above equation,

$$\operatorname{trace}\left(\mathrm{A}(\mathbf{q})\mathrm{B}^\mathsf{T}\right) = (w^2 - \|\mathbf{v}\|^2)\operatorname{trace}\left(\mathrm{B}^\mathsf{T}\right) + 2\operatorname{trace}\left(\mathbf{v}\,(\mathrm{B}\mathbf{v})^\mathsf{T}\right) - 2w\operatorname{trace}\left([\mathbf{v}\times]\mathrm{B}^\mathsf{T}\right). \tag{3.7}$$

The second term can be simplified using the following relation between the trace and the dot product:

$$\operatorname{trace}\left(\mathbf{x}\mathbf{y}^\mathsf{T}\right) = \mathbf{x}\cdot\mathbf{y} = \mathbf{x}^\mathsf{T}\mathbf{y} = \mathbf{y}^\mathsf{T}\mathbf{x}.$$

It follows that

$$2\operatorname{trace}\left(\mathbf{v}\,(\mathrm{B}\mathbf{v})^\mathsf{T}\right) = 2\mathbf{v}\cdot(\mathrm{B}\mathbf{v}) = \mathbf{v}^\mathsf{T}\mathrm{B}\mathbf{v} + \mathbf{v}^\mathsf{T}\mathrm{B}^\mathsf{T}\mathbf{v}. \tag{3.8}$$

To simplify the last term in Eq. (3.7), let us denote the elements of B by $b_1, \ldots, b_9$, so that

$$\mathrm{B} = \begin{pmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{pmatrix}, \qquad \mathrm{B}^\mathsf{T} = \begin{pmatrix} b_1 & b_4 & b_7 \\ b_2 & b_5 & b_8 \\ b_3 & b_6 & b_9 \end{pmatrix}.$$

Then

$$\begin{aligned} \operatorname{trace}\left([\mathbf{v}\times]\mathrm{B}^\mathsf{T}\right) &= \operatorname{trace}\left(\begin{pmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{pmatrix}\begin{pmatrix} b_1 & b_4 & b_7 \\ b_2 & b_5 & b_8 \\ b_3 & b_6 & b_9 \end{pmatrix}\right) \\ &= \operatorname{trace}\begin{pmatrix} b_3y - b_2z & \cdot & \cdot \\ \cdot & b_4z - b_6x & \cdot \\ \cdot & \cdot & b_8x - b_7y \end{pmatrix} \\ &= x(b_8 - b_6) + y(b_3 - b_7) + z(b_4 - b_2) = -\mathbf{v}\cdot\mathbf{b} \end{aligned}$$

where the vector $\mathbf{b} = -(b_8 - b_6, b_3 - b_7, b_4 - b_2)$ is defined via the relation $[\mathbf{b}\times] = \mathrm{B}^\mathsf{T} - \mathrm{B}$. Consequently,

$$-2w\operatorname{trace}\left([\mathbf{v}\times]\mathrm{B}^\mathsf{T}\right) = w\mathbf{v}^\mathsf{T}\mathbf{b} + w\mathbf{b}^\mathsf{T}\mathbf{v}. \tag{3.9}$$

Substituting Eq. (3.8) and Eq. (3.9) into trace (3.7) gives

$$\begin{aligned} \operatorname{trace}\left(\mathrm{A}(\mathbf{q})\mathrm{B}^\mathsf{T}\right) &= (w^2 - \|\mathbf{v}\|^2)\operatorname{trace}\left(\mathrm{B}^\mathsf{T}\right) + 2\operatorname{trace}\left(\mathbf{v}\,(\mathrm{B}\mathbf{v})^\mathsf{T}\right) - 2w\operatorname{trace}\left([\mathbf{v}\times]\mathrm{B}^\mathsf{T}\right) \\ &= (w^2 - \mathbf{v}^\mathsf{T}\mathbf{v})\operatorname{trace}\left(\mathrm{B}^\mathsf{T}\right) + \left(\mathbf{v}^\mathsf{T}\mathrm{B}\mathbf{v} + \mathbf{v}^\mathsf{T}\mathrm{B}^\mathsf{T}\mathbf{v}\right) + \left(w\mathbf{v}^\mathsf{T}\mathbf{b} + w\mathbf{b}^\mathsf{T}\mathbf{v}\right). \\ &= \mathbf{v}^\mathsf{T}\left(\mathrm{B} + \mathrm{B}^\mathsf{T} - \operatorname{trace}(\mathrm{B})\,\mathrm{I}_3\right)\mathbf{v} + w\mathbf{v}^\mathsf{T}\mathbf{b} + w\mathbf{b}^\mathsf{T}\mathbf{v} + w^2\operatorname{trace}\left(\mathrm{B}^\mathsf{T}\right). \end{aligned}$$

The last expression shows the trace is quadratic in $\mathbf{q} = (\mathbf{v}, w) = (x, y, z, w)$ and can be written as a quadratic form

$$\operatorname{trace}\left(\mathrm{A}\mathrm{B}^{\mathsf{T}}\right) = \begin{pmatrix}\mathbf{v}^{\mathsf{T}} & w\end{pmatrix} \mathrm{K} \begin{pmatrix}\mathbf{v} \\ w\end{pmatrix}$$

where the symmetric matrix K is precisely the matrix given in Eq. (3.6). □

At this point, a short recapitulation is in place. In the beginning of this section, we have defined closest rotation between measured and reference vectors via a weighted least-squares energy $J(\mathrm{A})$. The orientation matrix was parametrized by a quaternion as $\mathrm{A}(\mathbf{q})$ and the minimization was reformulated as the maximization problem:

$$\mathbf{q} = \operatorname*{arg\,min}_{\mathbf{q}\in\mathbb{S}^3} \sum_{i=1}^{n} \alpha_i \left\|\mathbf{m}_i - \mathrm{A}(\mathbf{q})\,\mathbf{e}_i\right\|_{\mathrm{F}}^2 = \operatorname*{arg\,max}_{\mathbf{q}\in\mathbb{S}^3} \operatorname{trace}\left(\mathrm{A}(\mathbf{q})\,\mathrm{B}^{\mathsf{T}}\right) = \operatorname*{arg\,max}_{\mathbf{q}\in\mathbb{S}^3} \mathbf{q}^{\mathsf{T}}\mathrm{K}\mathbf{q}.$$

The last step leads us to a constrained maximization whose solution is given by the following proposition.

Proposition 3.5. *Let* $\mathrm{K} \in \mathbb{R}^{n\times n}$ *be a symmetric real-valued matrix. Consider the constrained quadratic problem*

$$\max_{\mathbf{x}\in\mathbb{R}^n} \mathbf{x}^{\mathsf{T}}\mathrm{K}\,\mathbf{x} \quad \text{s.t.} \quad \|\mathbf{x}\| = 1, \tag{3.10a}$$

which is equivalent to the unconstrained maximization on a sphere,

$$\max_{\mathbf{x}\in\mathbb{S}^{n-1}} \mathbf{x}^{\mathsf{T}}\mathrm{K}\,\mathbf{x} \tag{3.10b}$$

Then the solution to both above problems is given by the (unit) dominant eigenvector *of the matrix* K*, which is the eigenvector corresponding to the largest eigenvalue.*

Proof. The matrix K is symmetric and real-valued. By spectral theorem, K has the spectral decomposition of the form

$$\mathrm{K} = \mathrm{Q}\Lambda\mathrm{Q}^{\mathsf{T}} = \begin{pmatrix}\mathbf{v}_1 & \dots & \mathbf{v}_n\end{pmatrix} \begin{pmatrix}\lambda_1 & & \\ & \ddots & \\ & & \lambda_n\end{pmatrix} \begin{pmatrix}\mathbf{v}_1^{\mathsf{T}} \\ \vdots \\ \mathbf{v}_n^{\mathsf{T}}\end{pmatrix} = \lambda_1\mathbf{v}_1\mathbf{v}_1^{\mathsf{T}} + \dots + \lambda_n\mathbf{v}_n\mathbf{v}_n^{\mathsf{T}}, \tag{3.11}$$

where $\lambda_{\max} = \lambda_1 \geq \dots \geq \lambda_n = \lambda_{\min}$ are the real eigenvalues of K ordered decreasingly and $\mathbf{v}_i$ are the corresponding eigenvectors. Then we can solve the original maximization problem using the change of parameters $\mathbf{y} = \mathrm{Q}\mathbf{x}$. Note that $\|\mathbf{y}\| = \|\mathrm{Q}\mathbf{x}\| = 1$ if and only if $\|\mathbf{x}\| = 1$. Consequently, solving the problem (3.10b)

is equivalent to maximizing the quadratic form $\mathbf{y}^\intercal \Lambda \mathbf{y}$ subject to the equality constraint $\|\mathbf{y}\| = 1$. Using the fact that $\lambda_1 = \lambda_{\max}$ is the largest eigenvalue, the following inequality provides an upper bound for this maximum:

$$\mathbf{y}^\intercal \Lambda \mathbf{y} = \lambda_1 y_1^2 + \cdots + \lambda_n y_n^2 \leq \lambda_{\max}(y_1^2 + \cdots + y_n^2) = \lambda_{\max}\|\mathbf{y}\|^2 = \lambda_{\max}. \tag{3.12}$$

It is not difficult to see that the maximum is attained iff $\mathbf{y} = (1, 0, \ldots, 0)$. This means that $\mathbf{x}^\intercal \mathrm{K} \mathbf{x}$ is maximized iff $\mathbf{x}$ is equal to $\mathbf{v}_1$, the dominant eigenvector of K, corresponding to the largest eigenvalue λ_1. □

An efficient algorithm for computing the matrix K is given by the following proposition, which we state without proof [CMC07].

Proposition 3.6. *The matrix* K *is given as* $\mathrm{K} = -\sum_{i=1}^{n} \mathrm{K}_i$, *where*

$$\mathrm{K}_i = \alpha_i \, \Omega(\mathbf{m}_i) \, \Theta(\mathbf{e}_i) = \alpha_i \begin{pmatrix} -[\mathbf{m}_i\times] & \mathbf{m}_i \\ -\mathbf{m}_i^\intercal & 0 \end{pmatrix} \begin{pmatrix} [\mathbf{e}_i\times] & \mathbf{e}_i \\ -\mathbf{e}_i^\intercal & 0 \end{pmatrix}. \tag{3.13}$$

3.4.2 Application to data from IMUs

For an example of the application of the q method, we describe how the orientations are acquired using Morphorider (Section 1.4). Morphorider measures two vectors: the accelerometer vector $\mathbf{m}_1 = \mathbf{m}_{\text{acc}}$ and the magnetometer vector $\mathbf{m}_2 = \mathbf{m}_{\text{mag}}$. For the measurements carried out in Grenoble, France, the normalized reference vectors are

$$\mathbf{e}_{\text{acc}} = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{e}_{\text{mag}} = \begin{pmatrix} \frac{1}{2} & 0 & -\frac{\sqrt{3}}{2} \end{pmatrix}.$$

For each datapoint, the measured vectors and the reference vectors are plugged into Eq. (3.13) in order to compute the matrix K. The orientation, represented by a quaternion $\mathbf{q}$, is estimated by computing the dominant eigenvector of K. In Matlab, this is done by calling the function `eig` (Code snippet 3.5). Alternatively, only the dominant eigenvector can be computed without computing the whole eigendecomposition (as done by `eig`) using a specialized algorithm such as the power iteration.

3.5 Filtering of orientations

In this section, we describe our approach for filtering raw orientation data estimated from sensor measurements (Section 3.4); see also the overview in Fig. 3.4. Sensor noise and limited scanning precision are the main problems that prevent using raw

```
% reference vectors are stored in R
% measured and calibrated data are stored in M
% mcross is the matrix cross operator
Omega = @(x) [ -mcross(x) x; -x' 0 ]; % define Ω()
Theta = @(x) [  mcross(x) x; -x' 0 ]; % define Θ()
K.Acc = Omega( M.Acc ) * Theta( R.Acc );
K.Mag = Omega( M.Mag ) * Theta( R.Mag );
% compute eigenvectors
[V,~] = eig( -K.Acc -K.Mag );
% retrieve the quaternion (dominant eigenvector)
q = V(:,end)';
```

Code snippet 3.5: Matlab code for computing the orientation (quaternion **q**) from a pair of measured vectors

orientations in the reconstruction of positions. By consequence, the acquired network of orientations is often *inconsistent*: at intersections, the normals are different for each passing curve (Fig. 3.13).

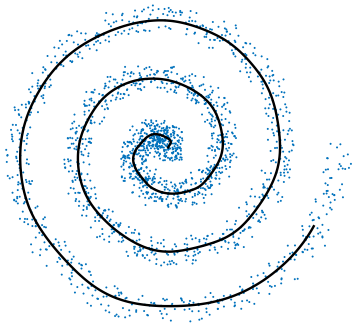
Our goal is to eliminate the noise and the inconsistency arising from acquisition. To this end, we use a two-step filtering process (Fig. 3.4). First, raw orientations are pre-filtered in the quaternion space using distance-weighted Gaussian convolution with fixed edge length. This way, we obtain uniform sampling of orientations with respect to arc length (Fig. 3.4b-c). Second, the uniform samples are smoothed and made consistent using regression on the manifold SO(3) (Fig. 3.4d).

This approach is motivated by the fact that existing algorithms for quaternion averaging are fast and efficient, but unable to resolve the additional constraints on normal consistency. On the other hand, expressing these constraints directly in SO(3) is straightforward. Naturally, we could skip the pre-filtering step completely and do the regression on SO(3) directly with raw data. However, due to the amount of data used, the resulting optimization would be much slower. By combining the two steps, our method is able to compute smooth and consistent orientations at interactive rates.

The input for this section is a raw sample of orientations $\mathrm{A}_i = \mathrm{A}(d_i) \in \mathrm{SO}(3)$ for each curve γ at known distances $d_i \in [0, L]$ where L is the length of γ. Orientations are represented by quaternions $\mathrm{A}_i = \mathrm{A}(\mathbf{q}_i)$ estimated from sensor measurements using the q method (Section 3.4).

We start this section by reviewing a common approach to smoothing: the moving average filter.

3.5.1 Moving average filter



In signal processing, many existing methods deal with smoothing of noisy discrete signals [Smi02]. A common technique for designing a low-pass filter is the *moving average*, which we introduce on an example in the following paragraph.

Consider a noisy set of datapoints $\mathbf{x}_k \in \mathbb{R}^2$ sampled from a planar curve $\gamma = \mathbf{x}(t)$. In order to filter out the noise in the data, each point $\mathbf{x}_k$ (blue points in the inset) is replaced by a weighted average $\bar{\mathbf{x}}_k$ of its neighbors (black curve in the inset). The weighted averages are computed via discrete convolution as

$$\bar{\mathbf{x}}_k = \operatorname*{average}_{-N \le i \le N} \mathbf{x}_{k+i} = \frac{\sum_{i=-N}^{N} w_{k,i}\, \mathbf{x}_{k+i}}{\sum_{i=-N}^{N} w_{k,i}}. \tag{3.14}$$

Here, N is the size of the moving window and $w_{k,i}$ are the weights, such as the uniform weights $w_{k,i} = 1$, or the inverse-distance Gaussian weights

$$w_{k,i} = \exp\left(-\frac{(k-i)^2}{2\sigma^2}\right).$$

A similar technique can be used to smooth a set of noisy orientations $\mathrm{A}_i \in \mathrm{SO}(3)$. Since the manifold of rotations SO(3) is not a Euclidean space, the averaging procedure from Eq. (3.14) cannot be used. Instead, weighted averages of orientations need to be computed in a way that respects the curved geometry of SO(3).

The spherical linear interpolation or Slerp (see Eq. (2.27) on p. 38) is not suitable for computing weighted averages of orientations for several reasons [Mar+07]. First of all, the result of Slerp depends on the choice of the representative quaternion. And second, Slerp is only defined for two input orientations; averaging three or more orientations would require a generalization of this approach, possibly leading to the so-called Karcher mean, which is computationally expensive [Kar77; JVV12].

Luckily, there is a simpler way to define a weighted average of orientations, using a cost function similar to the one in Wahba's problem in Eq. (3.1). Such averaging problem can be solved using a variation of the q method (Section 3.4.1). We explain the principle in the following section.

3.5.2 Algorithm for computing the average orientation

This section summarizes an efficient algorithm for computing weighted averages of orientations, as described by Markley et al. [Mar+07].

An intuitive way to define the average orientation is to use a cost similar to the one in Wahba's Problem 3.1 on p. 59:

Problem 3.7 (The average orientation). *Given a set of n orientation matrices $\mathrm{A}_i \in \mathrm{SO}(3), 1 \leq i \leq n$, with weights $\alpha_i \in \mathbb{R}$, the average orientation is defined as the minimizer of the energy*

$$\min_{\mathrm{A} \in \mathrm{SO}(3)} \sum_{i=1}^{n} \alpha_i \left\| \mathrm{A} - \mathrm{A}_i \right\|_{\mathrm{F}}^2 . \tag{3.15}$$

Using the unit quaternion parametrization $\mathrm{A} = \mathrm{A}(\mathbf{q})$, $\mathrm{A}_i = \mathrm{A}(\mathbf{q}_i)$, this is equivalent to

$$\min_{\mathbf{q} \in \mathbb{S}^3} \sum_{i=1}^{n} \alpha_i \left\| \mathrm{A}(\mathbf{q}) - \mathrm{A}(\mathbf{q}_i) \right\|_{\mathrm{F}}^2 . \tag{3.16}$$

The following proposition described an equivalent form of the optimization Eq. (3.16). This is analogical to Proposition 3.2 and Corollary 3.3.

Proposition 3.8. *The problem (3.15) is equivalent to*

$$\max_{\mathbf{q} \in \mathbb{S}^3} \operatorname{trace}\left(\mathrm{A}\mathrm{B}^{\mathsf{T}}\right) \qquad \textit{where} \qquad \mathrm{B} = \sum_{i=1}^{n} \alpha_i \mathrm{A}_i . \tag{3.17}$$

Proof. Using the definition of the Frobenius norm from Eq. (2.2), each penalty term in the energy (3.15) is written as

$$\begin{aligned} \left\| \mathrm{A} - \mathrm{A}_i \right\|_{\mathrm{F}}^2 &= \operatorname{trace}\left([\mathrm{A} - \mathrm{A}_i]^{\mathsf{T}} [\mathrm{A} - \mathrm{A}_i] \right) = \operatorname{trace}\left(\underbrace{\mathrm{A}^{\mathsf{T}}\mathrm{A}}_{\mathrm{I}_3} + \underbrace{\mathrm{A}_i^{\mathsf{T}}\mathrm{A}_i}_{\mathrm{I}_3} - \mathrm{A}^{\mathsf{T}}\mathrm{A}_i - \mathrm{A}\mathrm{A}_i^{\mathsf{T}} \right) \\ &= 6 - 2 \operatorname{trace}\left(\mathrm{A}\mathrm{A}_i^{\mathsf{T}} \right) . \end{aligned}$$

Clearly, $\left\| \mathrm{A} - \mathrm{A}_i \right\|_{\mathrm{F}}^2$ is minimized when $\operatorname{trace}\left(\mathrm{A}\mathrm{A}_i^{\mathsf{T}} \right)$ is maximized. Moreover,

$$\sum_{i=1}^{n} \alpha_i \operatorname{trace}\left(\mathrm{A}\mathrm{A}_i^{\mathsf{T}} \right) = \operatorname{trace}\left(\sum_{i=1}^{n} \alpha_i \mathrm{A}\mathrm{A}_i^{\mathsf{T}} \right) = \operatorname{trace}\left(\mathrm{A} \sum_{i=1}^{n} \alpha_i \mathrm{A}_i^{\mathsf{T}} \right) = \operatorname{trace}\left(\mathrm{A}\mathrm{B}^{\mathsf{T}} \right) . \qquad \square$$

Notice that the optimization (3.17) is analogical to Eq. (3.2) with different definition of B. Applying Proposition 3.4 therefore gives the following result.

Corollary 3.9. *The cost in (3.17) is equal to*

$$\operatorname{trace}\left(\mathrm{AB}^{\mathsf{T}}\right) = \mathbf{q}^{\mathsf{T}}\mathrm{K}\mathbf{q}, \tag{3.18}$$

where the matrix K *is given in Proposition 3.4.*

So far, the computations in this section mostly copied those in Section 3.4, but this is where the two approaches start to differ. The following proposition exploits the properties of the matrix B specific to this section. It forms the basis for efficient computation of the average quaternion.

Proposition 3.10. *The matrix* K *from the Eq. (3.18) can be expressed as*

$$\mathrm{K} = 4\mathrm{M} - \alpha \mathrm{I}_4, \tag{3.19}$$

where $\alpha = \sum_{i=1}^{n} \alpha_i$ *is the sum of all weights and the symmetric matrix* M *is the* structure tensor *defined as*

$$\mathrm{M} = \sum_{i=1}^{n} \alpha_i \, \mathbf{q}_i \mathbf{q}_i^{\mathsf{T}}. \tag{3.20}$$

Proof. Recall that $\mathrm{B} = \sum_{i=1}^{n} \alpha_i \mathrm{A}(\mathbf{q}_i)$ and

$$\mathrm{A}(\mathbf{q}_i) = \begin{pmatrix} 2w_i^2 + 2x_i^2 - 1 & 2x_i y_i + 2w_i z_i & 2x_i z_i - 2w_i y_i \\ 2x_i y_i - 2w_i z_i & 2w_i^2 + 2y_i^2 - 1 & 2y_i z_i + 2w_i x_i \\ 2x_i z_i + 2w_i y_i & 2y_i z_i - 2w_i x_i & 2w_i^2 + 2z_i^2 - 1 \end{pmatrix}.$$

Since $\operatorname{trace}\left(\mathrm{A}(\mathbf{q}_i)\right) = 6w_i^2 + 2x_i^2 + 2y_i^2 + 2z_i^2 - 3 = 4w_i^2 - 1$, we have

$$\operatorname{trace}(\mathrm{B}) = \sum_{i=1}^{n} \alpha_i (4w_i^2 - 1).$$

The upper left 3×3 block of matrix K is therefore

$$\begin{aligned} \mathrm{B} + \mathrm{B}^{\mathsf{T}} + \operatorname{trace}(\mathrm{B})\,\mathrm{I}_3 &= \sum_{i=1}^{n} \alpha_i \left(\mathrm{A}(\mathbf{q}_i) + \mathrm{A}^{\mathsf{T}}(\mathbf{q}_i) - (4w_i^2 - 1)\mathrm{I}_3\right) \\ &= \sum_{i=1}^{n} \alpha_i \begin{pmatrix} 4x_i^2 - 1 & 4x_i y_i & 4x_i z_i \\ 4x_i y_i & 4y_i^2 - 1 & 4y_i z_i \\ 4x_i z_i & 4y_i z_i & 4z_i^2 - 1 \end{pmatrix} = \sum_{i=1}^{n} \alpha_i (4\mathbf{v}_i \mathbf{v}_i^{\mathsf{T}} - \mathrm{I}_3). \end{aligned}$$

All that is left to do is to compute the vector $\mathbf{b}$ such that $[\mathbf{b}\times] = \mathrm{B}^{\mathsf{T}} - \mathrm{B}$. To do that, let us look at the following difference:

$$\mathrm{B}^{\mathsf{T}} - \mathrm{B} = \sum_{i=1}^{n} \alpha_i \left(\mathrm{A}^{\mathsf{T}}(\mathbf{q}_i) - \mathrm{A}(\mathbf{q}_i)\right) = 4 \sum_{i=1}^{n} \alpha_i \begin{pmatrix} 0 & -w_i z_i & w_i y_i \\ w_i z_i & 0 & -w_i x_i \\ -w_i y_i & w_i x_i & 0 \end{pmatrix},$$

which means that $\mathbf{b} = \sum_{i=1}^{n} \alpha_i \, 4w_i \mathbf{v}_i$. Altogether, we have

$$\mathrm{K} = \sum_{i=1}^{n} \alpha_i \begin{pmatrix} 4\mathbf{v}_i\mathbf{v}_i^\mathsf{T} - \mathrm{I}_3 4w_i\mathbf{v}_i & \\ 4w_i\mathbf{v}_i^\mathsf{T} & 4w_i^2 - 1 \end{pmatrix} = \sum_{i=1}^{n} \alpha_i \left(4\mathbf{q}_i\mathbf{q}_i^\mathsf{T} - \mathrm{I}_4 \right),$$

which completes the proof. □

Corollary 3.11. *The solution to the problem (3.16) is given by the (unit) dominant eigenvector of* M.

Proof. Let us denote the solution to the problem (3.16) by $\mathbf{q}$. By Proposition 3.5, $\mathbf{q}$ is the dominant eigenvector of K, corresponding to the largest eigenvalue $\lambda_{\max}$. Since K has the eigendecomposition $\mathrm{K} = \mathrm{Q}\Lambda\mathrm{Q}^\mathsf{T}$ (see proof of Proposition 3.5), we have

$$4\mathrm{M} - \alpha\mathrm{I}_4 = \mathrm{Q}\Lambda\mathrm{Q}^\mathsf{T},$$

which implies the set of eigenvectors of M is the same as that of K. Concretely, the eigendecomposition of M is

$$\mathrm{M} = \mathrm{Q}\bar{\Lambda}\mathrm{Q}^\mathsf{T},$$

where the eigenvalues of M are related to the eigenvalues of K by the linear equation

$$\bar{\Lambda} = \frac{1}{4}\left(\Lambda - \alpha\mathrm{I}_4\right).$$

The quaternion $\mathbf{q}$ is therefore the dominant eigenvector of M. □

Note that the definition of the matrix M is independent from the choice of the representative quaternion, which was one of the issues of using Slerp. This is because the sum in Eq. (3.20) does not change if the sign of $\mathbf{q}_i$ is flipped: $\mathbf{q}_i\mathbf{q}_i^\mathsf{T} = (-\mathbf{q}_i)(-\mathbf{q}_i^\mathsf{T})$.

As a conclusion, the recipe for computing the average orientation is the following: first, express the orientations as quaternions $\mathbf{q}_i$; second, compute the matrix M using Eq. (3.20); and third, compute the dominant eigenvector $\mathbf{q}$ of M – this is the average orientation. We will use the following notation for this procedure:

$$\mathbf{q} = \underset{i=1,\ldots,n}{\text{average}} \left(w_i \, \mathbf{q}_i \right), \tag{3.21}$$

where $w_i \geq 0$ are the weights. This process takes only a few milliseconds even if it is repeated several hundred or thousand times. We exploit this efficiency in the next section in order to compute a uniform sampling of the input data. (For exact timings, see Table 3.9, p. 79, column *convolution*.)

3.5.3 Pre-filtering by Gaussian convolution

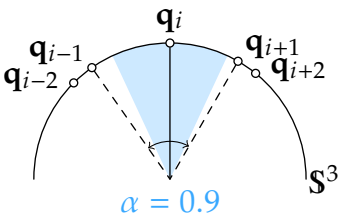
In this section we describe our method for computing the uniform sampling of orientations (Fig. 3.4c) from raw orientations (Fig. 3.4a), using the averaging algorithm from Section 3.5.2. Throughout this section, we use the quaternion representation of orientations.

Working with orientations that are sampled uniformly with respect to geodesic distance facilitates both the filtering (Section 3.5.4) and the reconstruction of positions (Section 3.6) – the uniform sampling is convenient for discretization of differential operators and integrals.

*Outlier removal (Fig. 3.4b).*Before computing the uniform sampling, we first remove outliers and process duplicate measures. Outliers are removed by looking at the angle of each quaternion $\mathbf{q}_i$ with its immediate neighbors:

$$\cos \sphericalangle(\mathbf{q}_i, \mathbf{q}_{i-1}) = \mathbf{q}_i \cdot \mathbf{q}_{i-1}, \qquad \cos \sphericalangle(\mathbf{q}_i, \mathbf{q}_{i+1}) = \mathbf{q}_i \cdot \mathbf{q}_{i+1}.$$

If both dot products are *smaller* than the specified threshold α, the angles between $\mathbf{q}_i$ and its neighbors are too big and $\mathbf{q}_i$ is removed – see the illustration in inset. For the examples in this thesis, we use the threshold $\alpha = 0.9$; this is consistent with the assumption that the adjacent datapoints are close in the space of orientations. In other words, the acquisition frequency is high enough. For sparsely sampled networks, a lower threshold or an adaptive scheme might be more suitable. Next we treat the duplicate measures. These are different orientations corresponding to the same parameter value d_i. Note that such duplicates are usually associated with network nodes. At nodes, the acquisition is temporarily stopped while the user indicates the index of the node to the system; as a result, the same point is acquired multiple times. The duplicate measures are averaged using the quaternion averaging scheme from Eq. (3.21) with $w_i \equiv 1$.



*Uniform parametrization.*To resample uniformly a given sample of orientations, we fix the edge length h_{fixed} and compute uniform parameters t_k for each curve γ by

$$t_k = \frac{kL}{N}, \quad k = 0, \dots, N, \qquad t_k - t_{k-1} = h, \tag{3.22}$$

where L is the length of γ and $N = \text{round}(L/h_{\text{fixed}})$.

We remark that while the parameter h_{fixed} is global and has the same value for all curves, h might slightly differ from curve to curve. In practice, this difference is negligible, and we consider the resulting parametrization t_k to be uniform. In the remainder of the thesis, we omit the subscript in h_{fixed} and simply write h.

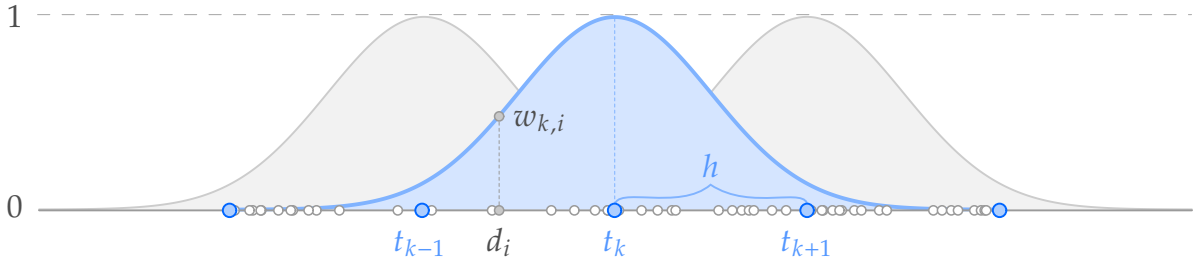


Figure 3.6: Schema of the pre-filtering with $\sigma = 0.2$. Raw orientations (represented by white dots) are convoluted with a Gaussian kernel to obtain uniform sampling (blue dots).

Gaussian convolution (Fig. 3.4c). For each t_k, we compute the corresponding orientation $\bar{\mathbf{q}}_k$ using convolution with distance weighting (Fig. 3.6):

$$\bar{\mathbf{q}}_k = \underset{\mathbf{q}_i \in \gamma}{\text{average}} \left(w_{k,i}\, \mathbf{q}_i \right).$$

Here, average( ) refers to the quaternion averaging scheme from Eq. (3.21). The Gaussian weights are computed as

$$w_{k,i} = \exp\left[-\frac{1}{2} \left(\frac{t_k - d_i}{\sigma h} \right)^2 \right],$$

where d_i is the input distance parameter associated to the measured orientation A_i. The parameter σ controls the radius of convolution; in general we use values between 0.2 and 0.5.

To some extent, the pre-filtering smooths out acquisition artifacts in the scanned data. The pre-filtered orientations $\bar{\mathbf{q}}_k$ are sampled uniformly and contain less noise than the raw data; compare Fig. 3.4 (a) and (c). By varying the parameter σ, it might be possible to sufficiently filter the data while preserving the acquired geometry. However, this approach has a major drawback: since the Gaussian convolution is applied to each curve individually, there is no guarantee that the filtered orientations will be consistent at intersections (see Section 4.2).

This type of coherence is however essential. The final reconstruction step of our network reconstruction method (Section 3.6) assumes consistent input. Furthermore, the consistency of normals is crucial to ensure correct subsequent processing of the network, for instance for surface fitting (Chapter 4).

The following section describes how we solve this problem.

3.5.4 Smoothing via regression on SO(3)

In the previous section, the noisy input orientations $A_i = A(\mathbf{q}_i)$ represented by quaternions $\mathbf{q}_i$ have been resampled and pre-filtered. The result is a uniform sample of orientations $\bar{A}_k = A(\bar{\mathbf{q}}_k)$.

Our goal in this section is to smooth the resampled orientations $\bar{A}_k$ while satisfying consistency constraints at intersections (Fig. 3.4d). To this end, the orientations are represented as rotation matrices $\bar{A}_k \in SO(3)$ instead of quaternions $\bar{\mathbf{q}}_k$, which was the case in the previous section.

This change of representation (from quaternions to rotation matrices) is motivated by our need to enforce consistent surface normals at intersections. In order to have a filtering framework with unified representation, our initial idea was to use quaternion splines [PR97; Sho85; Nie04]. However, expressing the consistency constraints in the quaternion space gives nonlinear equations. On the other hand, these constraints are linear in SO(3). Inspired by spline functions, which provide smooth curves that reasonably fit the input data by minimizing some energy functional, we propose in the following to formulate filtering of frames (orientations) as a customized energy minimization problem on SO(3).

Let us start by resuming the principles behind smoothing splines in the Euclidean space. Recall that a spline in tension $\mathbf{x}(t)$ in the d-dimensional Euclidean space $\mathbb{R}^d$ is the minimizer of the energy

$$\lambda \int_{t_0}^{t_N} \left\|\mathbf{x}'\right\|^2 \mathrm{d}t + \mu \int_{t_0}^{t_N} \left\|\mathbf{x}''\right\|^2 \mathrm{d}t$$

under the interpolation constraints $\mathbf{x}(t_k) = \mathbf{p}_k \in \mathbb{R}^d$. The weights λ and μ control stretching and bending of the spline, respectively. Incorporating positional constraints directly in the energy

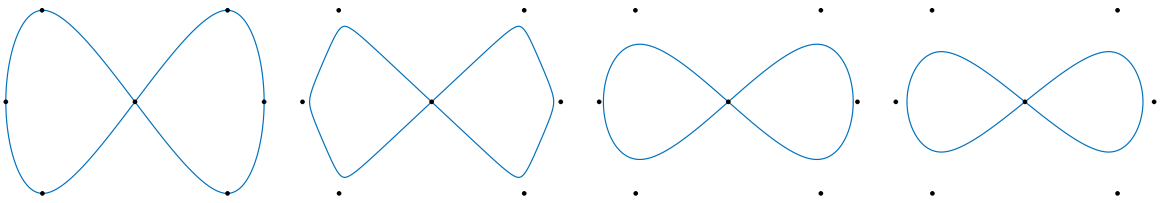
$$\sum_{k=0}^{N} \left\|\mathbf{x}(t_k) - \mathbf{p}_k\right\|^2 + \lambda \int_{t_0}^{t_N} \left\|\mathbf{x}'\right\|^2 \mathrm{d}t + \mu \int_{t_0}^{t_N} \left\|\mathbf{x}''\right\|^2 \mathrm{d}t \tag{3.23}$$

yields a smoothing spline, useful for non-parametric regression and data filtering in Euclidean spaces (Fig. 3.7).

Smoothing splines have been generalized for data on Riemannian manifolds by many authors [BA11; HP04; BC94]. We will now use such a manifold formulation to smooth data in the group SO(3) (Section 2.5, p. 34).

Analogously to Euclidean spaces, a smoothing spline $X(t)$ for orientation data $\bar{A}_k \in SO(3)$ is defined by minimizing the cost function

$$\sum_{k=0}^{N} \operatorname{dist}^2\left(X(t_k), \bar{A}_k\right) + \lambda \int_{t_0}^{t_N} \left\langle \dot{X}, \dot{X} \right\rangle \mathrm{d}t + \mu \int_{t_0}^{t_N} \left\langle \frac{\mathrm{D}^2 X}{\mathrm{d}t^2}, \frac{\mathrm{D}^2 X}{\mathrm{d}t^2} \right\rangle \mathrm{d}t, \tag{3.24}$$



$\lambda = 0.001$, $\mu = 0.001$ | $\lambda = 0.1$, $\mu = 0.001$ | $\lambda = 0.001$, $\mu = 0.1$ | $\lambda = 0.1$, $\mu = 0.1$

Figure 3.7: Euclidean smoothing spline with varying stretching λ and bending μ

where $\operatorname{dist}(\cdot,\cdot)$ is the geodesic distance on SO(3), $\langle\cdot,\cdot\rangle$ is the Riemannian metric, $\dot{X}$ is the velocity, and D^2X/dt^2 is the acceleration along X (cf. Section 2.4). We will write the above cost function as

$$E(X) = E_0(X) + \lambda E_1(X) + \mu E_2(X).$$

To discretize this problem, the continuous curve

$$X(t) : [0, L] \to SO(3)$$

is approximated by the tensor (a discrete curve)

$$\gamma = \left[X(t_0), \ldots, X(t_N)\right] = [X_0, \ldots, X_N] \in SO(3)^{N+1} \tag{3.25}$$

where t_k is the uniform parametrization defined in Eq. (3.22).

We use the method for spline fitting on manifolds of Boumal and Absil [BA11]. The differential and integral operators in Eq. (3.24) are approximated via *geometric* finite differences. Geometrically, the difference $\mathbf{b} - \mathbf{a}$ of two points $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$ in the Euclidean space is the vector starting at $\mathbf{a}$ and pointing towards $\mathbf{b}$ whose length is the distance between $\mathbf{a}$ and $\mathbf{b}$. On manifolds, specifically the manifold SO(3), the notion of difference is expressed using the logarithmic map (cf. Table 2.5 on p. 35). The distance between two rotation matrices $A, B \in SO(3)$ is

$$\operatorname{dist}(A, B) = \left\|\log\left(A^T B\right)\right\|_F.$$

For discretization step h, first and second geometric finite differences are

$$\begin{aligned}
\dot{X}_k &\approx 1/h \left(\log\left(X_k^T X_{k+1}\right)\right) \\
\ddot{X}_k &\approx 1/h^2 \left(\log\left(X_k^T X_{k+1}\right) + \log\left(X_k^T X_{k-1}\right)\right)
\end{aligned}$$

Integrals are also discretized using the step size h, for instance,

$$\int_{t_0}^{t_N} \left\langle \dot{X}, \dot{X} \right\rangle dt \approx \sum_{k=0}^{N-1} \underbrace{(t_{k+1} - t_k)}_{=h} \left\| \frac{1}{h} \log\left(X_k^T X_{k+1}\right) \right\|_F^2.$$

Consequently, for orientation data $\bar{A}_k \in SO(3)$, the discretization of the cost function from Eq. (3.24) has the form

$$\begin{aligned} E(\gamma) = \sum_{k=0}^{N} \left\| \log\left(X_k^\intercal \bar{A}_k\right) \right\|_F^2 + \frac{\lambda}{h} \sum_{k=0}^{N-1} \left\| \log\left(X_k^\intercal X_{k+1}\right) \right\|_F^2 \\ + \frac{\mu}{h^3} \sum_{k=1}^{N-1} \left\| \log\left(X_k^\intercal X_{k+1}\right) + \log\left(X_k^\intercal X_{k-1}\right) \right\|_F^2 . \end{aligned} \tag{3.26}$$

Boumal [Bou13] noticed that the discrete energy is well-approximated using the following simplified terms:

$$\begin{aligned} E_0(\gamma) &= \sum_{k=0}^{N} \left\| X_k - \bar{A}_k \right\|_F^2 , \\ E_1(\gamma) &= \frac{1}{h} \sum_{k=0}^{N-1} \left\| X_k - X_{k+1} \right\|_F^2 , \\ E_2(\gamma) &= \frac{1}{h^3} \sum_{k=1}^{N-1} \left\| \operatorname{skew}\left(X_k^\intercal \left(X_{k+1} + X_{k-1}\right)\right) \right\|_F^2 , \end{aligned} \tag{3.27}$$

where γ is the tensor of unknown orientations (discrete curve), $h = t_k - t_{k-1}$ is the uniform discretization step, and skew is the matrix operator from Eq. (2.20):

$$\operatorname{skew}(M) = \frac{1}{2}\left(M - M^\intercal\right). \tag{3.28}$$

For closed curves, all sums run from 0 to N due to periodicity.

Let us recall how the simplified energy in Eq. (3.27) is obtained. The terms E_0, E_1 result from the approximation of the geodesic distance on SO(3) by the chordal distance in the ambient space $\mathbb{R}^{3\times3}$ that we described in Eq. (2.22):

$$\operatorname{dist}^2(A, B) = \left\| \log\left(A^\intercal B\right) \right\|_F^2 \approx \|A - B\|_F^2 .$$

To obtain the simplified term E_2, consider the logarithmic map

$$B \mapsto \operatorname{Log}_A(B) = A \log\left(A^\intercal B\right).$$

This maps $B \in SO(3)$ to the tangent vector from $T_A SO(3)$ pointing towards B with length equal to the geodesic distance between A and B – cf. the Definition 2.29 of the Riemannian logaritmic map on p. 34. A similar vector is obtained by projecting the vector $(B - A)$ from the ambient space of $\mathbb{R}^{3\times3}$ to the tangent space $T_A SO(3)$. This projection is characterized by the operator $A \operatorname{skew}\left(A^\intercal B\right)$, cf. Fig. 2.5. The error of

this approximation is as follows. For an orientation matrix A, let $\Omega \in \mathbb{R}^{3\times 3}$ be skew-symmetric with $\|\Omega\|_F = 1$ and $B = A\exp(t\Omega)$. Since $A^T B = \exp(t\Omega)$, we have

$$\begin{aligned}\log\left(A^T B\right) &= t\Omega,\\ \operatorname{skew}\left(A^T B\right) &= \operatorname{skew}\left(I_3 + t\Omega + \tfrac{1}{2}t^2\Omega^2 + O(t^3)\right) = t\Omega + O(t^3).\end{aligned}$$

Both $\operatorname{skew}(I_3)$ and $\operatorname{skew}\left(\Omega^2\right)$ vanish – the skew operator applied to symmetric matrices yields a zero matrix. Replacing the log operator with the skew operator is therefore a quadratic approximation:

$$\log(A) \approx \operatorname{skew}(A).$$

In our setup, the filtered data $\bar{A}_k$ is associated to a network Γ and the energy terms in Eq. (3.27) are summed over all curves $\gamma \in \Gamma$. Simultaneously with regression on SO(3), we solve the consistency constraints as follows. Let $X, \tilde{X}_j$ be the local frames of two intersecting curves at their common node $v \in \mathcal{V}$. Then the two frames are called *consistent*, if the first frame $X_k = [\mathbf{t}_k\ \mathbf{N}\ \mathbf{B}_k]$ is obtained by rotating the second frame $\tilde{X}_j = \left[\tilde{\mathbf{t}}_j\ \mathbf{N}\ \tilde{\mathbf{B}}_j\right]$ around the common surface normal $\mathbf{N}$ at the node v. Denote by Φ the set of all such pairs of frames $\left(X, \tilde{X}_j\right)$. To enforce the consistency of surface normals, we add a term penalizing the difference in projection on the normal component at all nodes v

$$E_N(\Gamma) = \sum_{(X,\tilde{X}_j)\in\Phi} \left\| X_k|_{\mathbf{N}} - \tilde{X}_j|_{\mathbf{N}} \right\|^2. \tag{3.29}$$

Note there might be multiple pairs per node, in case of multiple intersecting curves.

Finally, we compute the set of smooth orientations $[X_0, \ldots, X_N] \in \mathrm{SO}(3)^{N+1}$ by minimizing the energy

$$\boxed{E(\Gamma) = \xi E_N(\Gamma) + \sum_{\gamma\in\Gamma} E_0(\gamma) + \lambda E_1(\gamma) + \mu E_2(\gamma)} \tag{3.30}$$

where $\xi, \lambda, \mu \geq 0$ are the weights controlling consistency of surface normals, stretching, and bending, respectively. The energy is minimized using the Riemannian trust-region (RTR), a generalization of the classical trust-region in $\mathbb{R}^n$ to differentiable manifolds; see the paper of Absil et al. [ABG07] for the description of the algorithm and a detailed convergence analysis. In Section 3.7, we discuss the performance and time needed for minimizing the energy (3.30) on a standard machine.

The resulting orientations are smooth along the curve network, and the normals are consistent at intersections. For an example, see the Gauss map visualization in

Fig. 3.4, i.e. the normal component of the orientations $X_k|_{\mathbf{N}}$ plotted on the unit 2-sphere. Notice that the network of pre-filtered normals in Fig. 3.4c) is *not* closed – this is precisely because the normals differ at intersections for different curves. The normals in Fig. 3.4d) have been smoothed and made continuous by minimizing the energy from Eq. (3.30).

This concludes the filtering part of our method. The next sections describes the final part, in which the filtered orientations are integrated in order to retrieve the positions. We refer to it as the *Poisson network reconstruction*.

3.6 Poisson network reconstruction

In the previous section, we described our method for filtering of raw orientation data measured by sensors, which yields uniform, smooth and consistent orientation sampling $[X_0, \dots, X_N] \in \mathrm{SO}(3)^{N+1}$. Here, $X_k = X(t_k)$; the uniform parametrization t_k is defined in Eq. (3.22).

We now address the problem of transforming the orientations $X_k \in \mathrm{SO}(3)$ into actual positions $\mathbf{x}_k \in \mathbb{R}^3$ (Fig. 3.4 right). The positions have to be recovered in a way that respects the topology of the underlying network (Section 3.2). Recall that there are two types of topological constraints: either the intersections of multiple curves (e.g. the T-junctions in Fig. 3.8), or the closure of a single curve (e.g. the blue circles in Fig. 3.8).

To recover the positions, previous approaches treat each curve individually and employ various heuristics to glue the curves together to obtain the correct topology [Spr07; HCG16]. These methods generally suffer from convergence problems inherent to integration methods for ODEs. Moreover, the gluing process is limited to specific topologies (such as a grid) and often requires manual corrections to obtain proper intersections.

Our approach is different: we will show that using Poisson's equation, it is possible to reconstruct all curves simultaneously while guaranteeing proper intersections. In this section, we extend the discretization of the Poisson equation for a single closed curve described by [CPS13] to curve networks with the topology of a cell complex as defined in Section 2.6 and Section 3.2.

If $\mathbf{x}$ is the natural parametrization of the curve γ (one-dimensional Riemannian manifold), the unit tangent field $\mathbf{t}$ along γ is equal to the gradient of positions

$$\nabla \mathbf{x} = \mathbf{t}$$

The vector field $\mathbf{t}$ is generally not integrable [KBH06; CPS13] and the solution needs to be found in the least-squares sense. To that end, the divergence operator $\nabla\cdot$ is

applied to both sides and $\mathbf{x}$ is found by solving Poisson's equation

$$\Delta \mathbf{x} = \nabla \cdot \mathbf{t} \tag{3.31}$$

Even though the curve network Γ as a whole is *not* a Riemannian manifold, it can be viewed as a *collection* of one-dimensional Riemannian manifolds constrained at intersections (see the problem statement in Section 3.2). This allows us to apply the above Poisson's equation in order to retrieve the network positions from local orientations. With that in mind, we discretize the gradient ∇ and the Laplacian Δ individually for each curve and retrieve the positions by solving a single global linear system. The topological constraints are enforced by representing each node as a unique point in the system (Fig. 3.8).

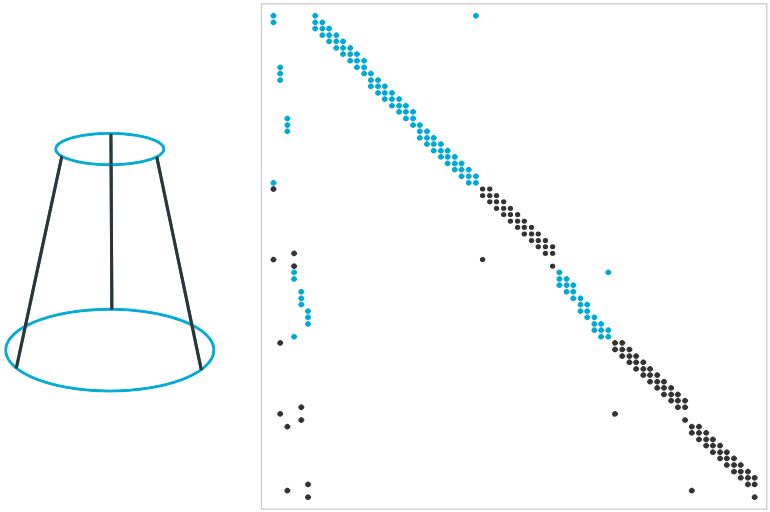


Figure 3.8: Matrix of the discrete Poisson system (3.31) for a cone network with two closed curves (blue) and three open curves (black). Dots indicate non-zero coefficients. First six columns correspond to network nodes.

Each curve is represented as a discrete collection of unknown points $\mathbf{x}_k = \mathbf{x}(t_k) \in \mathbb{R}^3, i = 0, \dots, N$, with known Darboux frames X_k, which are the outcome of the filtering from Section 3.5. We will denote by $\mathbf{t}_k = \mathrm{X}_k|_{\mathbf{t}}$ the (unit) tangent extracted from X_k. With uniform parametrization $h = t_k - t_{k-1}$, the differential operators are discretized via finite differences by

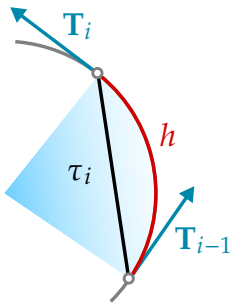
$$\Delta \mathbf{x}_k = 1/h^2 \left(\mathbf{x}_{k-1} - 2\mathbf{x}_k + \mathbf{x}_{k+1}\right), \quad \nabla \cdot \mathbf{t}_k = 1/h \left(\mathbf{t}_k - \mathbf{t}_{k-1}\right). \tag{3.32}$$

The above discretization is valid for closed curves (Fig. 3.8 blue circles) and interior vertices of open curves (Fig. 3.8 black lines). At endpoints of open curves, we directly impose the following boundary conditions

$$\begin{aligned} 1/h \left(\mathbf{x}_1 - \mathbf{x}_0\right) &= \mathbf{t}_0, \\ 1/h \left(\mathbf{x}_N - \mathbf{x}_{N-1}\right) &= \mathbf{t}_N. \end{aligned}$$

The resulting linear system (3.31) is sparse with at most three non-zero coefficients per row (Fig. 3.8). Thanks to the regression step described in Section 3.5.4, the tangent vectors of all curves meeting at a network node are co-planar at that node. The above boundary conditions therefore ensure the reconstruction of a G^1 curve network. Note that the discretization does not take into account the fact that $h = t_k - t_{k-1}$ is the *geodesic* distance along the reconstructed curve, which differs from the (Euclidean) norm $\tau_k = \|\mathbf{x}_{k-1} - \mathbf{x}_k\|$. In general, this is not an issue, as the least-squares minimization of the Poisson energy distributes the error evenly.

We remark that the Euclidean distance τ_k might be estimated from the geodesic distance h. This could be done for instance by approximating τ_k by the chordal length of the circular arc defined by the length h and angle between $\mathbf{t}_{k-1}$ and $\mathbf{t}_k$ (see inset).



This estimation comes at a price – the modified parametrization is no longer uniform. Our experiments have shown that the discretization error $|\tau_k - h|$ is generally too small to significantly influence the reconstruction; see the analysis in Section 3.7.3.

3.7 Evaluation

In this section we present the results that we have obtained using the reconstruction method introduced in Sections 3.5 to 3.6. The method itself is evaluated in three parts: analysis of performance (Section 3.7.1), analysis of filtering (Section 3.7.2), and analysis of convergence and reconstruction error (Section 3.7.3). In Section 3.7.4, we compare the results from three different acquisition schemes, one for Morphorider and two for smartphone (Section 1.4).

In the following examples, we use two types of data.

Synthetic data. We use six synthetic datasets for convergence analysis of the Poisson network reconstruction. This type of data is well-suited for analyzing the convergence since the synthetic data are not corrupted by sensor noise. Therefore, they do not need to be filtered, which provides a better understanding of how the Poisson reconstruction works.

Two synthetic curve networks were sampled from parametric surfaces (sphere and torus, Fig. 3.15). The other four synthetic curve networks were created in Blender (bowl, bumpy cube, gamepad, lilium) by tracing a network of polylines on a triangle mesh. The traced network (vertices and normals) was then exported using a python script. All synthetic data are stored in a text file with custom structure, containing the geometry (positions and normals) and the topology (edges) of the discrete network.

Acquired data. Data acquired from physical surfaces serve for evaluating the overall performance of the framework on real-world examples. Moreover, they enable quantitative comparison of the two acquisition methods. For some of the physical surfaces we dispose of ground truth models, which allows us to measure the error of reconstruction on real-world data (see Section 3.7.3).

In this section, we present results obtained on data acquired from two surfaces with known ground truths (cone and lilium) and one surface without a ground truth (chair). More results will be presented in Chapter 5, along with details on fabrication and acquisition of the physical surfaces.

3.7.1 Performance

Computationally, the most expensive part of the described algorithm is the minimization of the regression energy from Eq. (3.30). This minimization is carried out using the Riemannian trust region algorithm (RTR) implemented in modern libraries for optimization on manifolds. We have experimented with the Manopt toolbox [Bou+14] written in Matlab; and the ROPTLIB library [Hua+16] written in C++.

Both libraries provide similar interfaces for specification of a concrete optimization problem. Essentially, the user needs to define the domain – in our case, the product manifold $\mathrm{SO}(3)^N$ – and specify the cost function along with its gradient and Hessian (the latter is optional in Manopt). See the Code snippet 3.10 for an example. Note that it is enough for the user to specify Euclidean gradient and Hessian since the libraries are able to compute the Riemannian gradient and Hessian via projections on the manifold (cf. Lema 2.22).

The setup of optimization problem is straightforward in Manopt (see Code snippet 3.10) making Manopt more user-friendly and well-suited for prototyping than ROPTLIB. At the same time, ROPTLIB is faster when used within a C++ program. Note that it is possible to compile ROPTLIB as a mex file and use it as a Matlab library – the two libraries then provide comparable performance within Matlab in terms of computational time.

Table 3.9 shows timings for different parts of the algorithm implemented in C++, including pre-filtering and Poisson network reconstruction and using ROPTLIB for regression. Optimization is by far the most time-consuming part of the algorithm, although it usually converges in less than one second for reasonable sampling distances h. Speed of convergence depends on the choice of the weights λ and μ – the higher these weights are, the more time is needed for the optimization to converge. See the examples of chair and lilium in the table.

data	*# of samples*		*time (ms)*		
	raw	*uniform*	*convolution*	*regression*	*Poisson*
lilium Fig. 3.14 left	3243	96	3	51	1
		196	6	113	3
		382	6	234	7
		742	9	439	30
		1416	15	788	170
lilium Fig. 3.14 middle	531	99	3	28	2
		186	6	55	3
		362	8	83	10
		713	11	192	31
		1420	16	428	172
lilium Fig. 3.12	3243	428	7	125	10
				319	
				517	
				522	
cone Fig. 3.18	949	205	5	176	3
		398	3	328	8
		783	8	618	28
		1553	14	1169	136
		3094	22	2779	904
chair Fig. 5.13	584	247	3	592	6
				1315	
				1461	
mushroom Fig. 5.11	831	508	10	202	8

Table 3.9: Performance of the algorithm implemented in C++ using ROPTLIB on a standard quad-core CPU computer. The statistics include number of raw samples, number of uniform samples, Gaussian convolution time (Section 3.5.3), SO(3) regression time (Section 3.5.4), Poisson network reconstruction time (Section 3.6). All reported timings are in milliseconds.

```
% variables:
%   initial ... (size=3x3xn) orientation matrices along the network
%   options ... optimization settings (i.e. # iterations)
%         T  ... topology matrix, indices of neighboring vertices
%            ...
% get number of datapoints = rotation matrices
n = size(initial,3);
% construct the product manifold SO(3)^n = SO(3) x ... x SO(3)
manifold = rotationsfactory(3,n);
% problem definition - set the manifold
problem.M = manifold;
% set the cost function
problem.cost = @cost;
% set the (Euclidean) gradient of the cost function
problem.egrad = @grad;
% perform optimization via RTR
result = trustregions(problem,initial,options);
% definition of the cost function
function f = cost(X)
  % normal penalty: ||X_i|_N - X̃_j|_N||^2
  fn = 0;
  for pair = nodepairs,
    fn = fn + norm2(X(3,:,pair.first)-X(3,:,pair.second));
  end
  % interpolation:  ||X_k - A_k||_F^2
  diff = X - initial;
  f0 = arrayfun(@(k) norm2(diff(:,:,k)),1:n);
  % velocity:       ||X_k - X_{k+1}||_F^2
  i1 = find(T(:,3) > -1);
  f1 = arrayfun(@(k) dt(k) .* ...
       norm2(X(:,:,T(k,2)) - X(:,:,T(k,3))), i1);
  % acceleration:   ||skew(X_k^T (X_{k+1} + X_{k-1}))||_F^2
  i2 = find(T(:,1) > -1 & T(:,3) > -1);
  f2 = arrayfun(@(k) ddt(k) .* ...
       norm2(skew(X(:,:,T(k,2))'*(X(:,:,T(k,1))+X(:,:,T(k,3))))),i2);
  % overall cost
  f = wn*sum(fn) + w0*sum(f0) + w1*sum(f1) + w2*sum(f2);
end
% definition of the (Euclidean) gradient
function f = grad(X)
% ...
% ...
% ...
end
```

Code snippet 3.10: Regression on SO(3) implemented in Matlab using Manopt

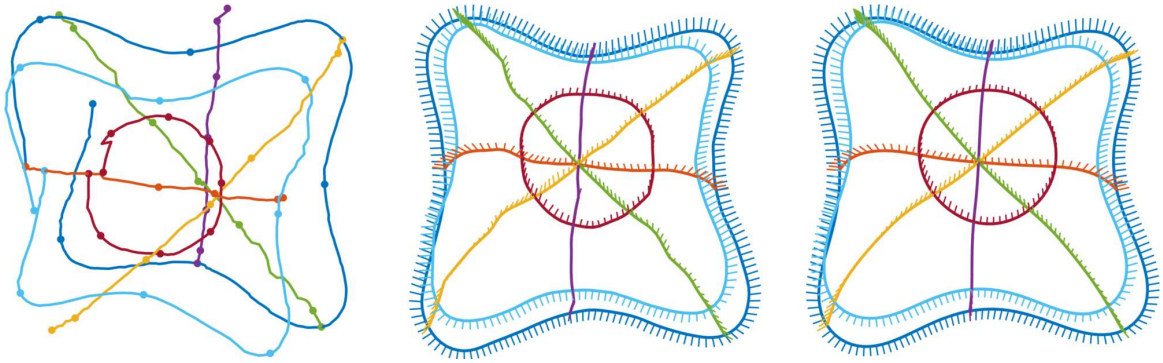


Figure 3.11: Curve network on lilium reconstructed using (*left*) raw orientations, (*middle*) pre-filtered orientations, (*right*) filtered orientations.

3.7.2 Filtering

The main goal of the filtering algorithm described in Section 3.5 is to obtain a clean and consistent network of orientations from raw sensor measures. The regression weights λ, μ in the filtering energy Eq. (3.30) influence the shape of the reconstructed network indirectly by controlling stretching and bending of splines on the manifold SO(3). In all examples, we use the weight $\xi = 10000$ to enforce the consistency of normals.

Fig. 3.11 shows reconstructions of lilium from the same orientation dataset at various filtering stages. The left network was reconstructed by direct integration of raw orientations using the forward Euler method. This naive approach completely fails to close the curves, and the resulting network has therefore incorrect topology. The other two networks were obtained using our Poisson network reconstruction from Section 3.6. Unlike the example on the left, this approach yields a network with proper topology. The middle network was reconstructed from pre-filtered orientations (Section 3.5.3) without the filtering on SO(3). Clearly, the pre-filtering is not sufficient to smooth the data, and the resulting curves contain unwanted wiggles. Finally, the right network was computed from filtered orientations ($\lambda = 1, \mu = 1$), providing a smooth and consistent network ready for surfacing.

The influence of the weights λ and μ on the Poisson-reconstructed positions is compared in Fig. 3.12. Interestingly, even though λ and μ control stretching and bending of *orientations* X_i, they also seem to control stretching and bending of reconstructed *positions* $\mathbf{x}_i$. The weights need to be chosen carefully, otherwise the network might end up over-filtered; see Fig. 3.12 where $\lambda = 100$.

Fig. 3.13 shows reconstruction of the chair network filtered with various sets of weights. It also illustrates the phenomenon of inconsistent orientations: pre-filtered normals do not agree at intersections, and naive averaging of each pair of conflicting normals results in a shape with poor fidelity to the scanned object (Fig. 3.13

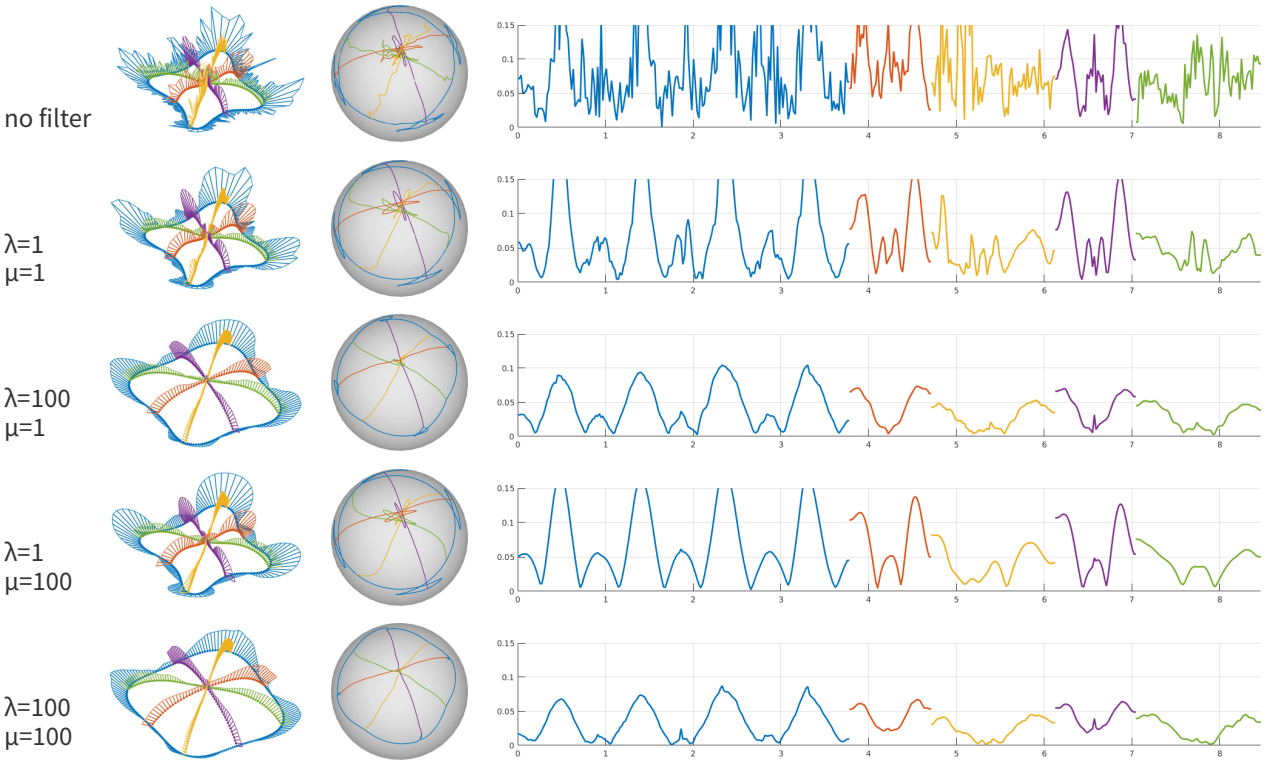


Figure 3.12: Reconstruction of lilium with various filtering weights. Left to right in each row: reconstructed network with porcupine plot (surface normal scaled by curvature); Gauss map; curvature plot.

left). Our regression filtering technique resolves this problem while preserving the underlying geometry (Fig. 3.13 right).

Finding the right filtering weights λ and μ is always a trade-off between fidelity to the input data and smoothness of the reconstructed network. In our experiments, there was no universal set of weights that would provide optimal results for all tested surfaces. This is partly due to the fact that the validation of the reconstructed network is often only visual, and picking the optimal result is a matter of subjective choice. In overall, we have found that the strategy for getting the best results is to prioritize bending over stretching by setting $\mu \gg \lambda$.

For the networks with known ground truth, it is possible to define the optimal result by measuring the error of reconstruction; we describe this in more detail in Section 3.7.3). In this sense, the weights that provide the smallest mean reconstruction error for the examples in Figures 3.18 to 3.22 are most often $\lambda = 1, \mu = 100$.

As a conclusion, the combination $\lambda = 1, \mu = 100$ seems like an ideal starting point for fine-tuning the weights in order to get the desired result. One of the advantages of our method is that the weights can be re-adjusted interactively, and the network is re-computed instantaneously.

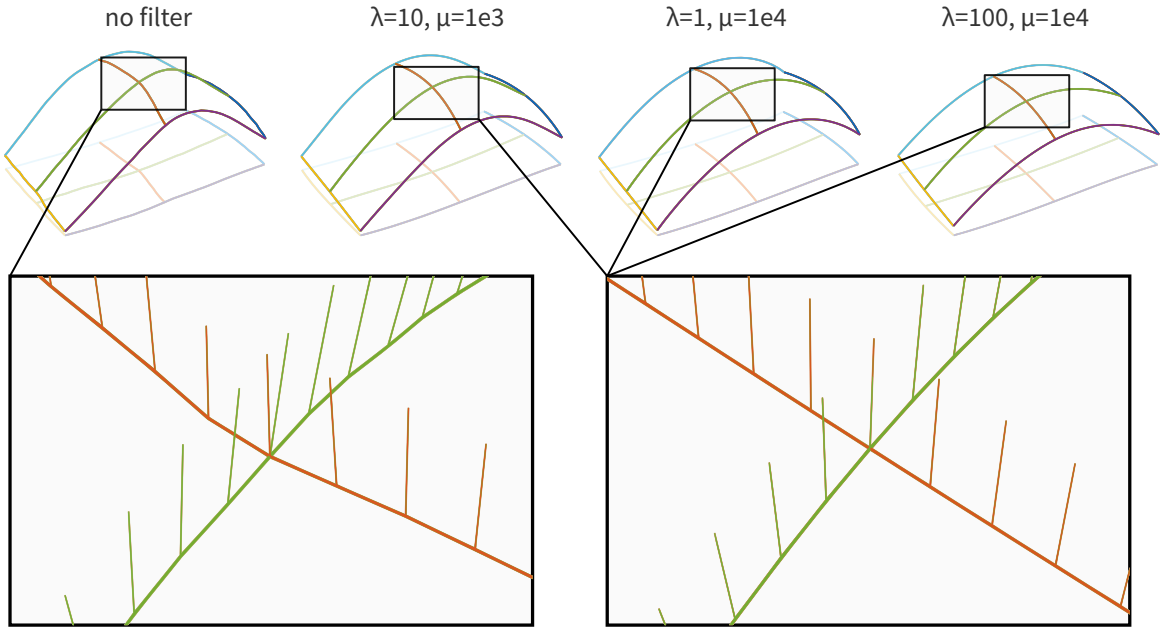


Figure 3.13: Scanned chair reconstructed with different sets of weights. After the pre-filtering, normals at nodes are not compatible ((*left*)). The normal penalty term in the filtering energy ensures the consistency of the reconstructed network ((*right*)). Photo of the physical chair is in Fig. 5.13 on p. 150.

3.7.3 Convergence & error measurements

In this section, we evaluate the proposed reconstruction method quantitatively. To this end, we compute the error of reconstruction for various synthetic and acquired datasets. We look at how the error evolves for decreasing discretization step h, which is fixed in the pre-filtering step; see Eq. (3.22).

Each dataset is tested using five different values, $h \in \{6.4\%, 3.2\%, 1.6\%, 0.8\%, 0.4\%\}$. These values are relative to the diameter $d_{\mathcal{S}}$ of the underlying surface $\mathcal{S}$. For instance, $h = 1.6\%$ means $h = 0.016\, d_{\mathcal{S}}$. The diameter $d_{\mathcal{S}}$ is computed as the length of the diagonal of the axis-aligned bounding box (AABB) of $\mathcal{S}$. Note that the diameters of the physical surfaces (lilium and cone) are given in Table 5.2, p. 140.

Measuring the error. The error metric associated with each point $\mathbf{x}$ in the reconstructed network is defined as the distance between $\mathbf{x}$ and its closest point $\tilde{\mathbf{x}}$ on the ground truth. For synthetic data, the closest point $\tilde{\mathbf{x}}$ is found on the ground truth network Γ.

For acquired data, the ground truth network is unknown; instead, the closest point is found on the underlying surface $\mathcal{S}$. In this case, before evaluating the error the acquired network needs to be registered to the surface since the two objects do not

live in the same coordinate systems. We use the iterative closest point (ICP) for the registration [BM92].

All tested datasets are shown with maximum, root mean square (rms), mean and minimum of the error. The statistics are computed over all vertices in the reconstructed network. Similarly to the specification of h, all of the lengths are relative to the diameter $d_{\mathcal{S}}$ of the corresponding ground truth surface $\mathcal{S}$.

Convergence. The measured error is used to study convergence of the method with respect to decreasing discretization step h.

We use synthetic data to study the convergence of Poisson network reconstruction with respect to the discretization step h. Synthetic data are well-suited for this task since they are not corrupted by noise and do not require smoothing. Note that the pre-filtering step is still needed in order to obtain a uniform sample of orientations. We measure the convergence by computing the error of reconstruction – we show that in most tested cases, the mean error approaches zero as h decreases.

We also look at how the error evolves for acquired data with known ground truth. In this case we do not expect the mean error of reconstruction to approach zero. Nevertheless, we expect the error to converge to a value, which is relatively close to zero.

Synthetic data. The results for synthetic networks are shown in Figures 3.15 to 3.17, p. 89–91. In almost all cases, the mean error approaches zero as h decreases. From this fact, we can conclude that Eq. (3.32) is a suitable discretization of the continuous Poisson equation (3.31).

We remark that for some networks, the maximal error slightly increases for $h = 0.4\%$ with respect to $h = 0.8\%$. This is because at this scale the input data are *upsampled* in the pre-filtering step – the uniform sample contains more datapoints than the raw sample. This difference is however small, and the mean error decreases or remains about the same.

Acquired data. We move on to the analysis of the results obtained with acquired data. The main difference with respect to synthetic data is that the acquired data are also filtered by regression on SO(3) (Section 3.5.4). For each dataset, we compare four filtering schemes – the four combinations with weights $\lambda = 1, 100$ and $\mu = 1, 100$.

The results for acquired data are shown in Figs. 3.18 to 3.22, p. 92–100. Overal, we conclude the error of reconstruction gets smaller with decreasing h. Not surprisingly, the only dataset that violates the convergence uses estimated lengths (Fig. 3.19). The use of exact, measured lengths is crucial to get an accurate reconstruction using our method.

We now discuss the results with individual datasets.

Cone (Fig. 3.18). This dataset was acquired with the Morphorider. In terms of error, the differences between the four schemes are subtle, and the mean error is always around 0.4%. The best choice of weights seems to be $\lambda = \mu = 100$.

Next, we study two curve networks on lilium: a network with 5 curves, and a network with 7 curves.

Lilium, 5 curves (Figs. 3.19 to 3.21). For this curve network, the data were acquired with both Morphorider and a smartphone. Both datasets follow the same set of curves traced on the surface using adhesive tape (see Fig. 3.14 top right).

The orientations from Morphorider are parametrized by arc-length, while the orientations from smartphone are parametrized by acquisition time. Let γ be a curve in the network with length L and total acquisition time T. Then the corresponding parametrizations of orientations are

$$\begin{aligned} [0, L] \to \mathrm{SO}(3): \quad & d_i \mapsto \mathrm{A}_i \quad \ldots \quad \text{Morphorider}, \\ [0, T] \to \mathrm{SO}(3): \quad & t_i \mapsto \mathrm{A}_i \quad \ldots \quad \text{smartphone}, \end{aligned}$$

where A_i are the scanned orientations, d_i represents an arc-length distance, and t_i represents a time instant during acquisition.

For smartphone, we compare the results obtained using two types of segment lengths. Let γ be a curve in the network with segments γ_i. Acquired orientations along the segment γ_i are time-parametrized: $[T_i, T_{i+1}] \to \mathrm{SO}(3)$. The two setups that we test are the following:

1. *Estimated segment lengths.* In this setup, time parametrization t_i is used as the arc-length parametrization d_i without modifications, $t_i = d_i$. This means that the length of a segment is estimated from its acquisition time.
2. *Measured segment lengths.* In this setup, we are given the exact length L_i of each segment γ_i. Time parameter t_i is scaled to estimate the arc-length parametrization d_i:
$$t_i \mapsto d_i = L_i \frac{t_i - t_{\text{start}}}{t_{\text{end}} - t_{\text{start}}}.$$

In order to compare the error, a network with estimated segment lengths is scaled to match the total length of the network with measured lengths.

The three setups are shown in Figs. 3.20 to 3.21. For the two networks using measured lengths, the best results are obtained using a small weight for stretching ($\lambda = 1$). On the other hand, the network with estimated lengths is improved if the stretching weight is higher ($\lambda = 100$). In terms of mean error, the best reconstruction is obtained using smartphone data with measured lengths. Overall, the mean error is relatively small, around 1.1% – 1.5% for reasonably chosen weights.

Note that the three setups are further compared in Section 3.7.4, see also Fig. 3.14.

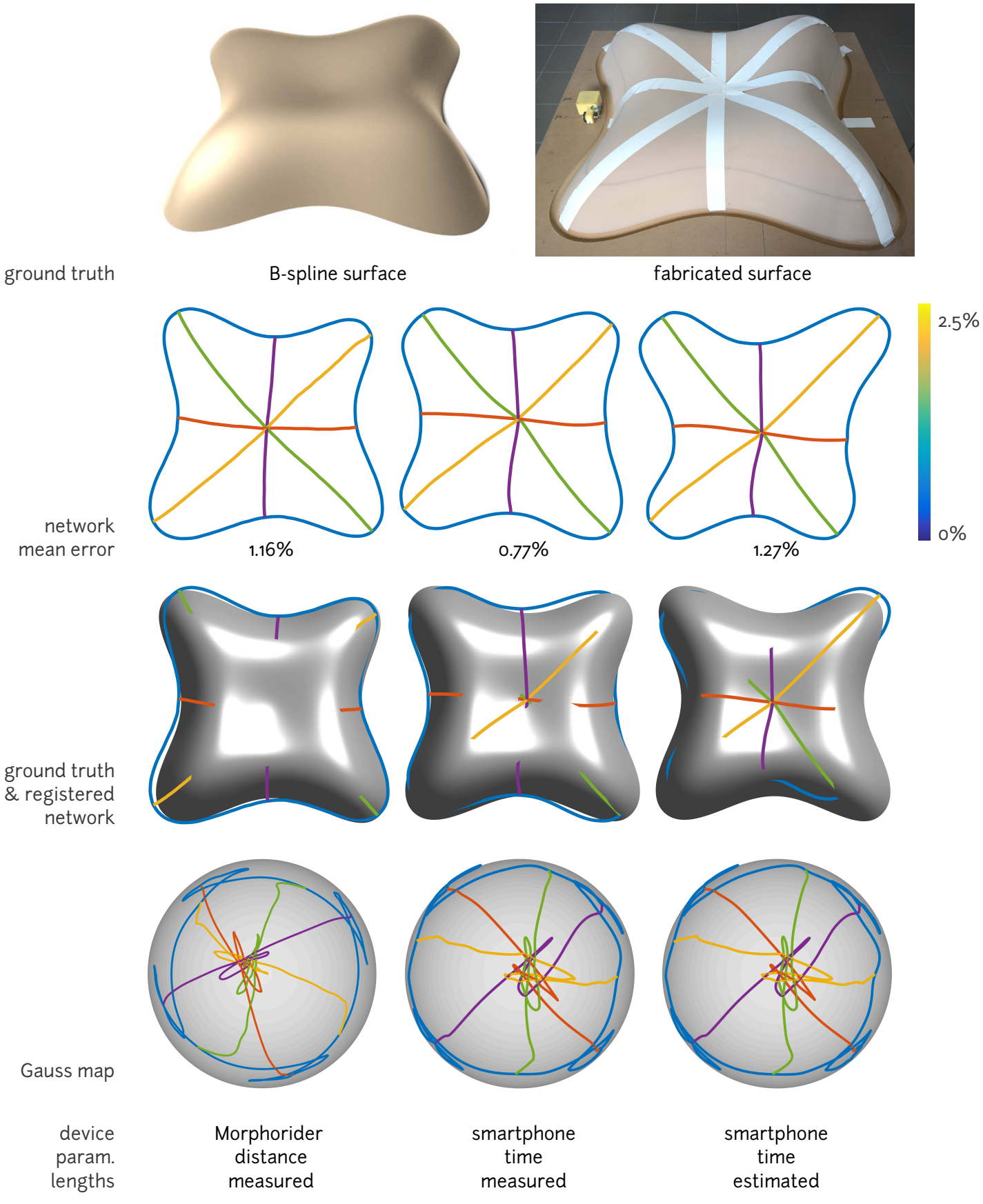


Figure 3.14: Comparison of three acquisition setups on lilium. For filtering, we have used the weights $\lambda = 1, \mu = 100$. All lengths are relative to the diameter (length of AABB diagonal) of the ground truth surface.

Lilium, 7 curves (Fig. 3.22). To acquire this dataset, we traced the same curves as for lilium with 5 curves, and we added two more closed curves in the interior of the surface. In this case we only used the Morphorider. Error of reconstruction is comparable to Fig. 3.21, with slight improvement for higher stretching weights ($\lambda = 100$).

3.7.4 Morphorider vs. smartphone

Fig. 3.14 shows a comparison of three acquisition setups tested on lilium with known ground truth. Top images show surfaced networks rendered with reflection lines; bottom images show the error of Poisson reconstruction for each network.

For the left network, the orientations were acquired using the Morphorider (Fig. 1.14 left) and parametrized by the arc-length – this is possible since we measure the exact distance from the beginning of the curve for each datapoint. For the networks in the middle and on the right, the orientations were acquired with a smartphone (Fig. 1.14 right) and parametrized by acquisition timestamps. In this case, we have no knowledge of exact distance traveled – instead, we assume that the speed of acquisition is constant.

The right network uses segment lengths automatically estimated from total acquisition time for each curve. For the middle network, we have manually measured and specified the length L_s of each segment s, which is then used to transform the time parameter $t \in [t_{\text{start}}, t_{\text{end}}]$ into the natural parameter $d \in [0, L_s]$. In this case, it is enough to assume that the speed of acquisition is constant-per-segment but does not need to be constant throughout the whole process of acquisition. In practice, the constant-speed-per-segment constraint is much easier to handle, and the manual distance measurement significantly improves the reconstruction – in this case, the mean error dropped by more than 1%.

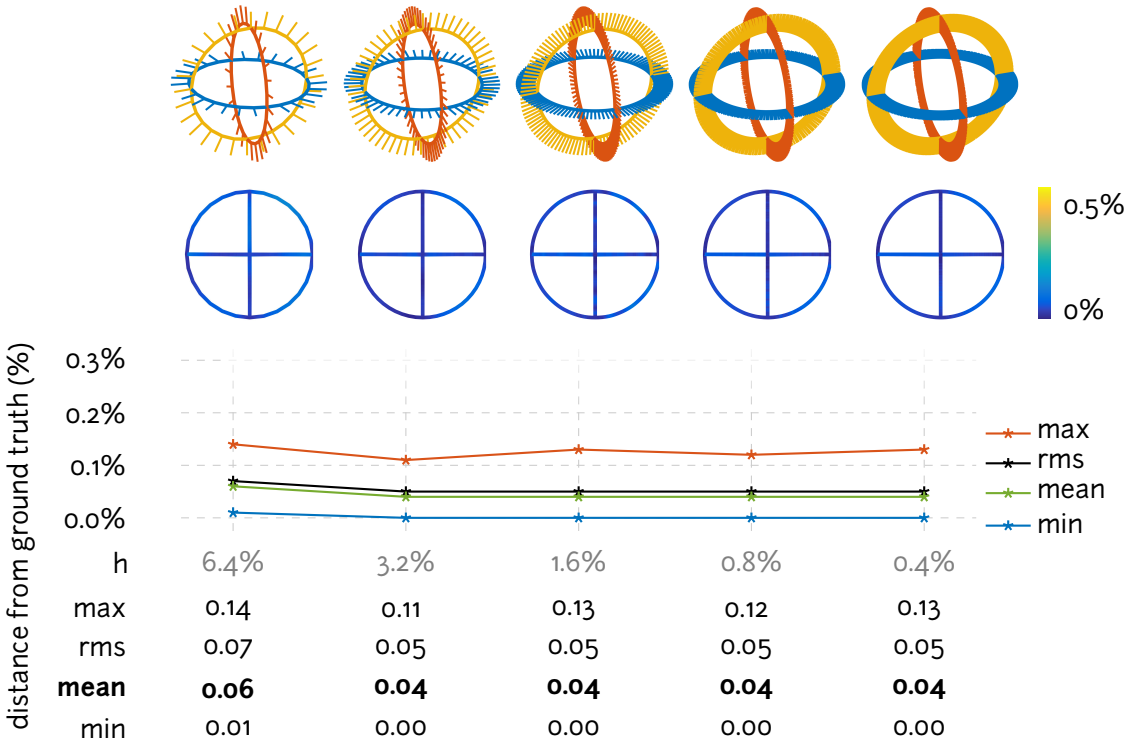
The mean error is about 1% using measured distances (both devices) and around 2% using time-estimated distances (smartphone). It is not surprising that the smartphone error decreases faster than the Morphorider error using the measured distances. One can further observe that even though the smartphone error is bigger than the Morphorider error at coarser resolutions, it becomes smaller at finer scales. We explain this behavior by the fact that the distance data are not the same for the two devices. For the Morphorider data, we have exact correspondences between distance and orientation measurements; for the smartphone data, only the total length of each segment is known and the correspondences are estimated from acquisition time. It seems like at finer scales, the error of this estimation is better-distributed, which results in a reconstruction with higher precision.

3.8 Conclusion

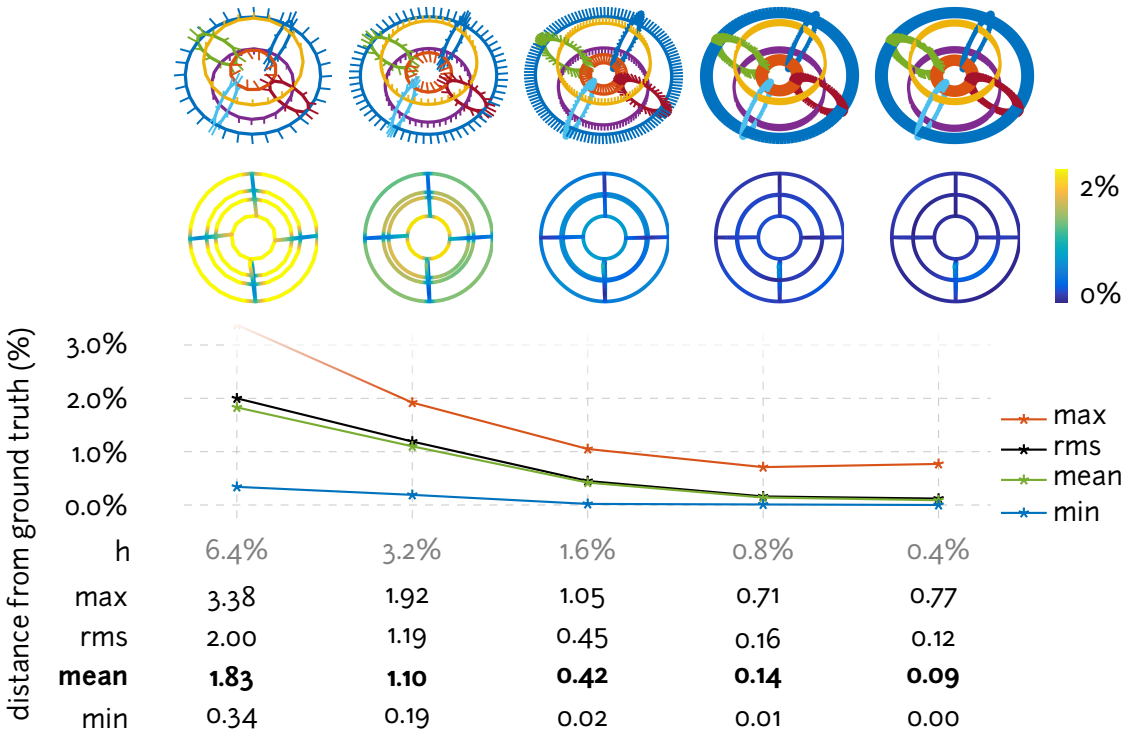
This chapter described a novel method for acquisition and reconstruction of curve networks on surfaces. The development of our method was guided by a simple observation, formulated mathematically via a set of normal consistency constraints and topological constraints. These constraints are the key ingredients in both parts of the method: filtering of orientations, and Poisson reconstruction of positions.

Results computed using our method are smooth and well-connected curve networks ready for further processing. Reconstruction is controlled by adjusting a small set of parameters.

Next chapter will present an algorithm for surfacing the networks obtained in this chapter.

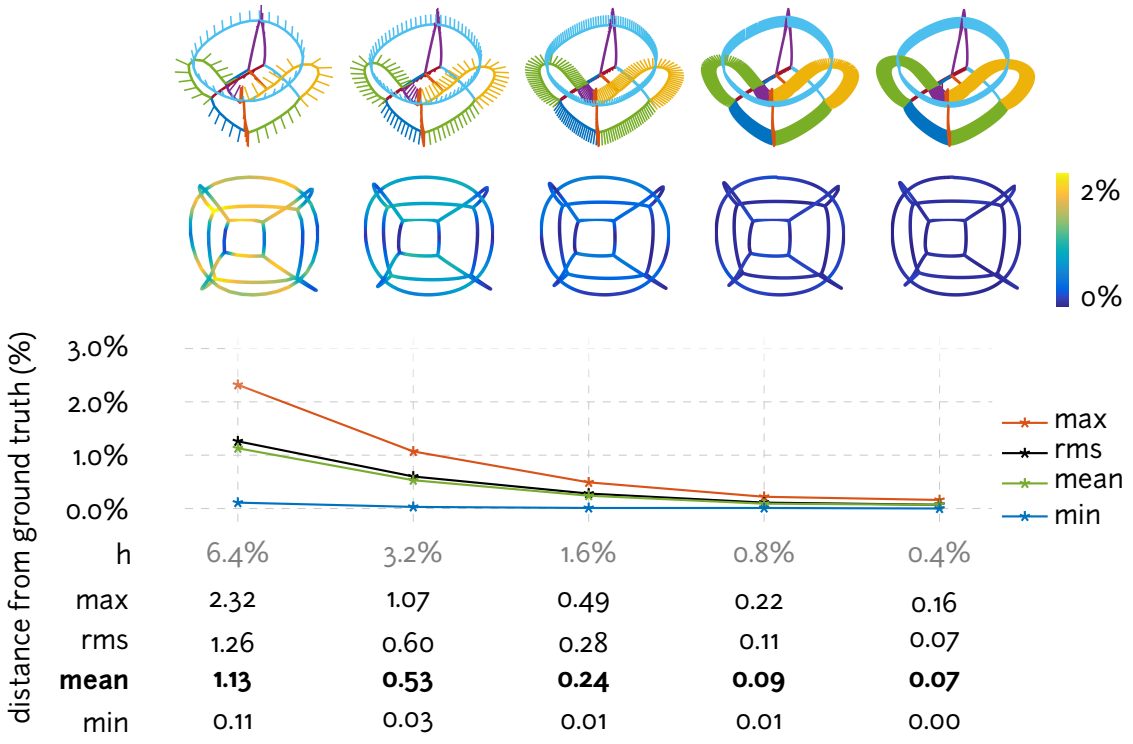


h	6.4%	3.2%	1.6%	0.8%	0.4%
max	0.14	0.11	0.13	0.12	0.13
rms	0.07	0.05	0.05	0.05	0.05
mean	**0.06**	**0.04**	**0.04**	**0.04**	**0.04**
min	0.01	0.00	0.00	0.00	0.00

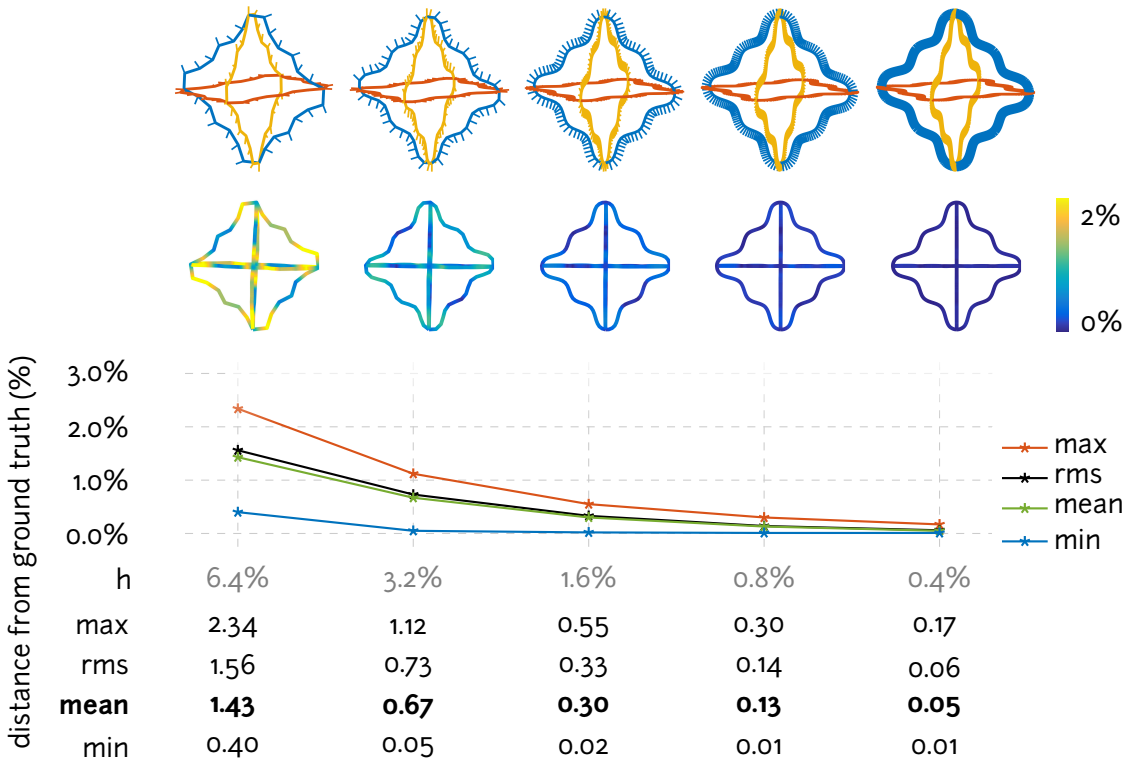


h	6.4%	3.2%	1.6%	0.8%	0.4%
max	3.38	1.92	1.05	0.71	0.77
rms	2.00	1.19	0.45	0.16	0.12
mean	**1.83**	**1.10**	**0.42**	**0.14**	**0.09**
min	0.34	0.19	0.02	0.01	0.00

Figure 3.15: Network error, sphere & torus, synthetic

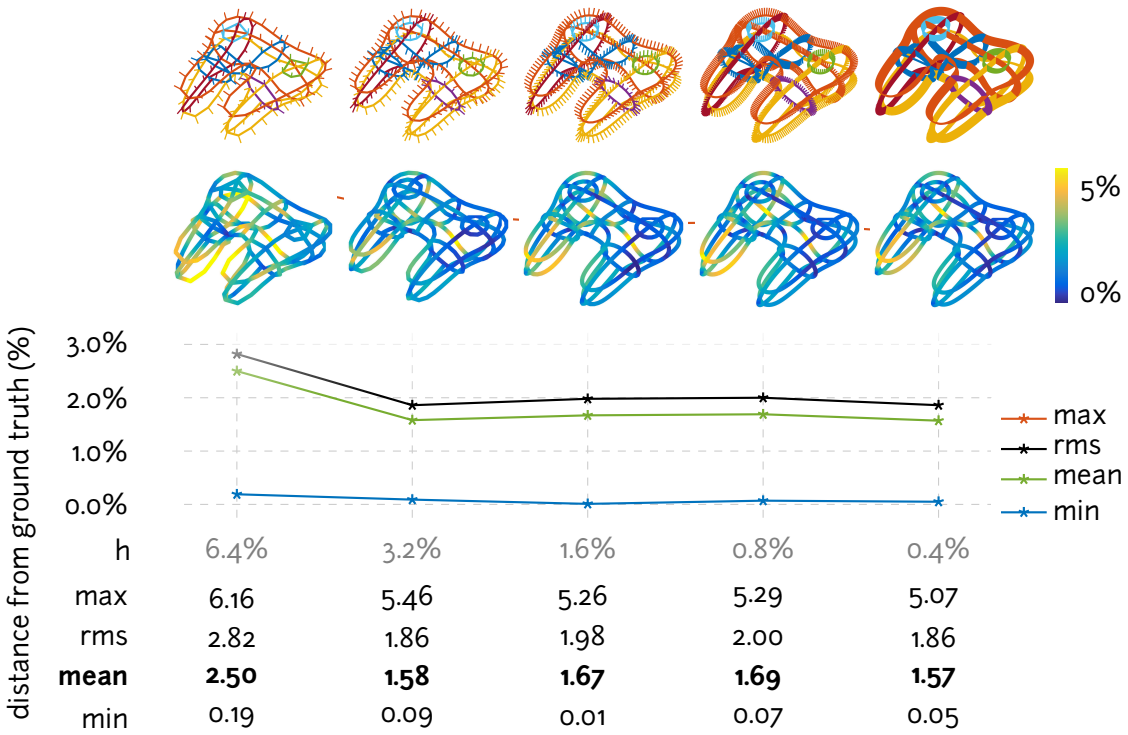


h	6.4%	3.2%	1.6%	0.8%	0.4%
max	2.32	1.07	0.49	0.22	0.16
rms	1.26	0.60	0.28	0.11	0.07
mean	**1.13**	**0.53**	**0.24**	**0.09**	**0.07**
min	0.11	0.03	0.01	0.01	0.00

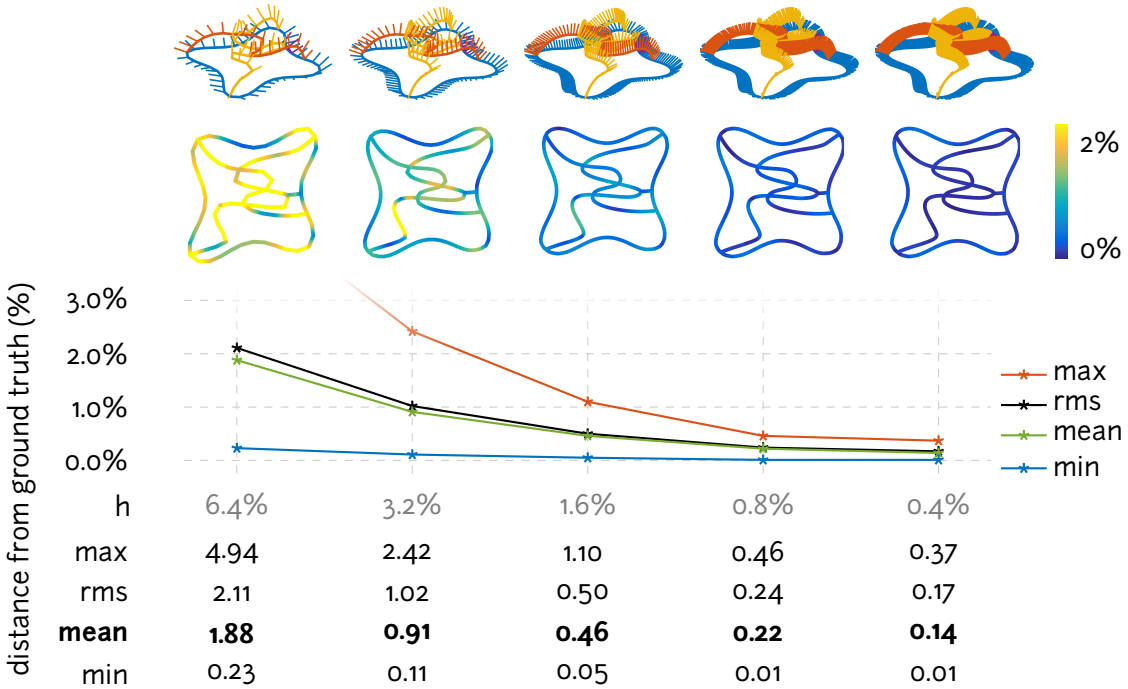


h	6.4%	3.2%	1.6%	0.8%	0.4%
max	2.34	1.12	0.55	0.30	0.17
rms	1.56	0.73	0.33	0.14	0.06
mean	**1.43**	**0.67**	**0.30**	**0.13**	**0.05**
min	0.40	0.05	0.02	0.01	0.01

Figure 3.16: Network error, bowl & bumpycube, synthetic



h	6.4%	3.2%	1.6%	0.8%	0.4%
max	6.16	5.46	5.26	5.29	5.07
rms	2.82	1.86	1.98	2.00	1.86
mean	**2.50**	**1.58**	**1.67**	**1.69**	**1.57**
min	0.19	0.09	0.01	0.07	0.05



h	6.4%	3.2%	1.6%	0.8%	0.4%
max	4.94	2.42	1.10	0.46	0.37
rms	2.11	1.02	0.50	0.24	0.17
mean	**1.88**	**0.91**	**0.46**	**0.22**	**0.14**
min	0.23	0.11	0.05	0.01	0.01

Figure 3.17: Network error, gamepad & lilium, synthetic

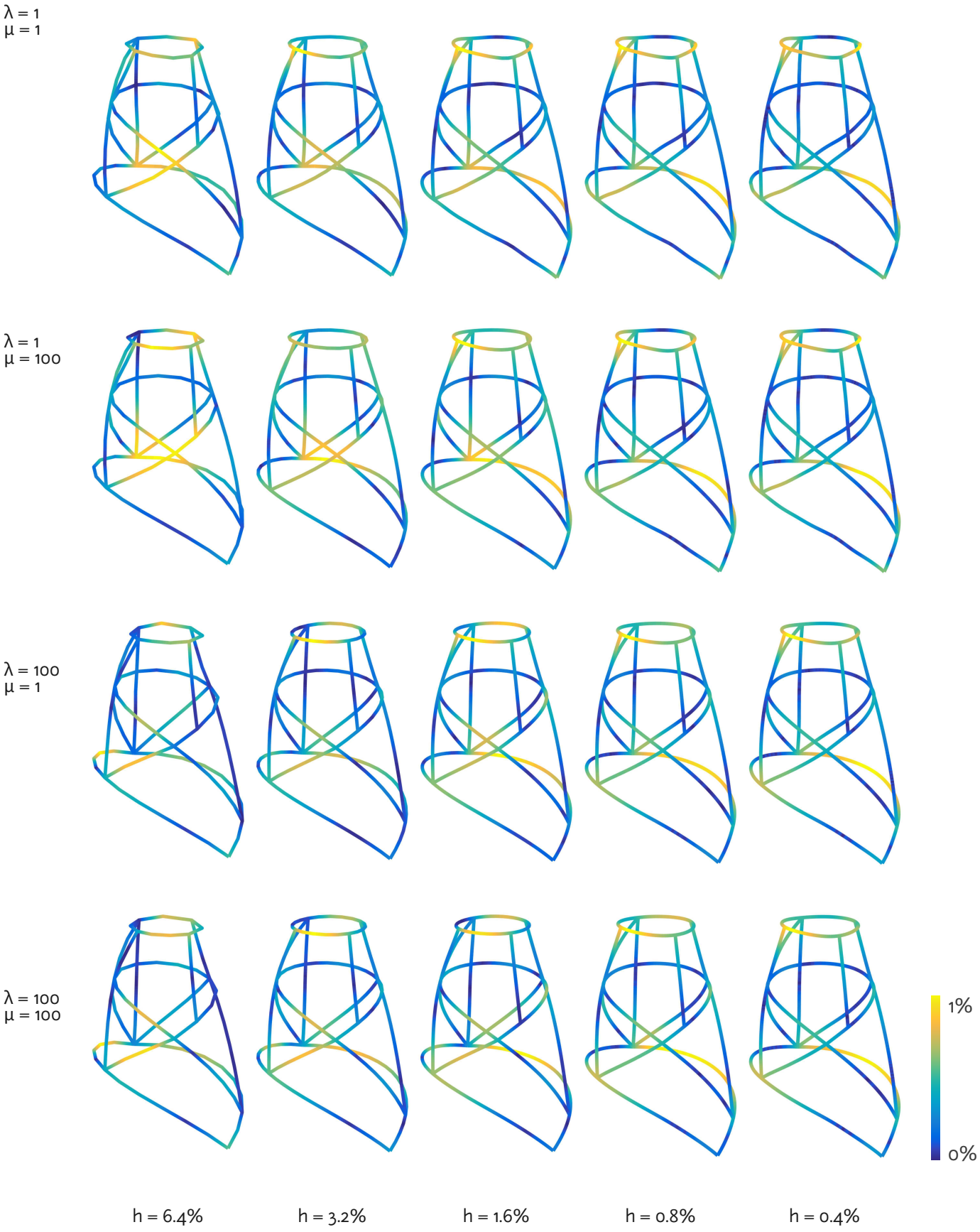
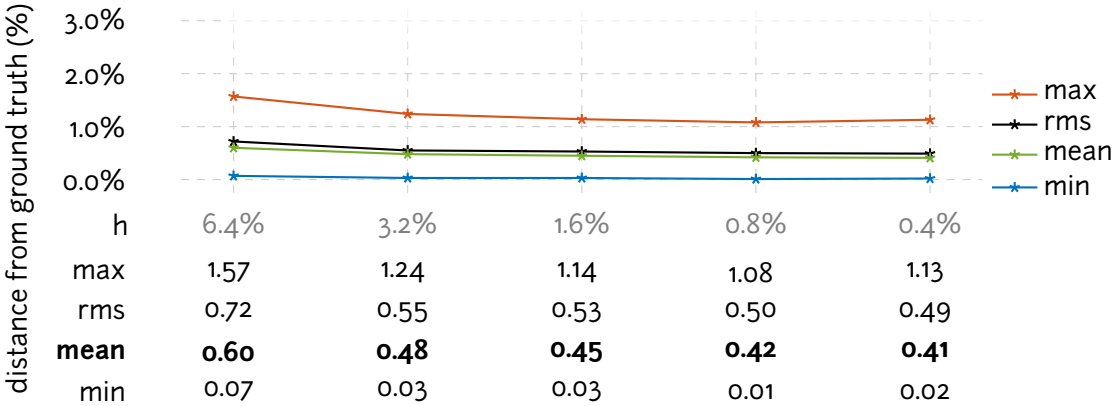


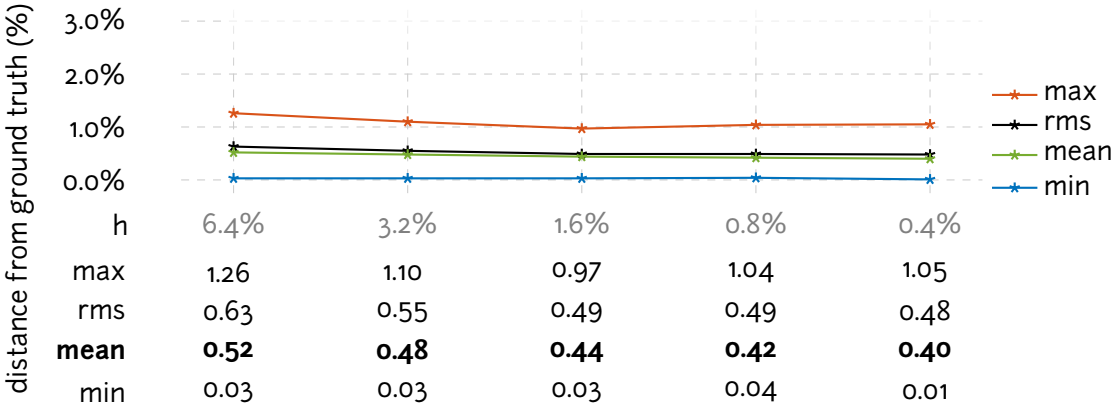
Figure 3.18: Network error, cone, Morphorider

λ = 1
μ = 1



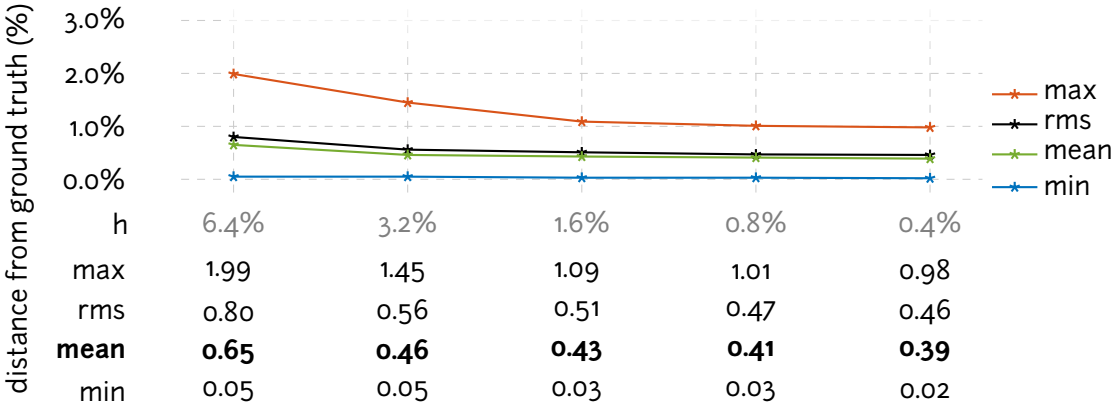
h	6.4%	3.2%	1.6%	0.8%	0.4%
max	1.57	1.24	1.14	1.08	1.13
rms	0.72	0.55	0.53	0.50	0.49
mean	**0.60**	**0.48**	**0.45**	**0.42**	**0.41**
min	0.07	0.03	0.03	0.01	0.02

λ = 1
μ = 100



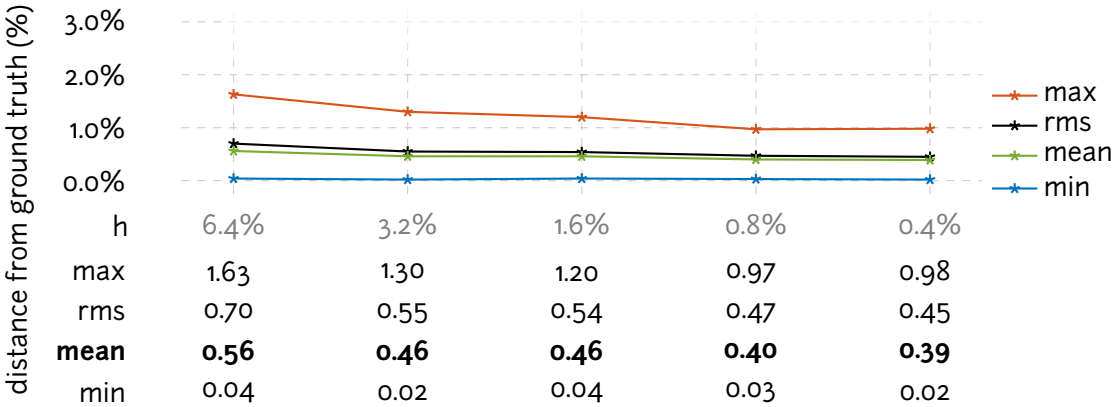
h	6.4%	3.2%	1.6%	0.8%	0.4%
max	1.26	1.10	0.97	1.04	1.05
rms	0.63	0.55	0.49	0.49	0.48
mean	**0.52**	**0.48**	**0.44**	**0.42**	**0.40**
min	0.03	0.03	0.03	0.04	0.01

λ = 100
μ = 1



h	6.4%	3.2%	1.6%	0.8%	0.4%
max	1.99	1.45	1.09	1.01	0.98
rms	0.80	0.56	0.51	0.47	0.46
mean	**0.65**	**0.46**	**0.43**	**0.41**	**0.39**
min	0.05	0.05	0.03	0.03	0.02

λ = 100
μ = 100



h	6.4%	3.2%	1.6%	0.8%	0.4%
max	1.63	1.30	1.20	0.97	0.98
rms	0.70	0.55	0.54	0.47	0.45
mean	**0.56**	**0.46**	**0.46**	**0.40**	**0.39**
min	0.04	0.02	0.04	0.03	0.02

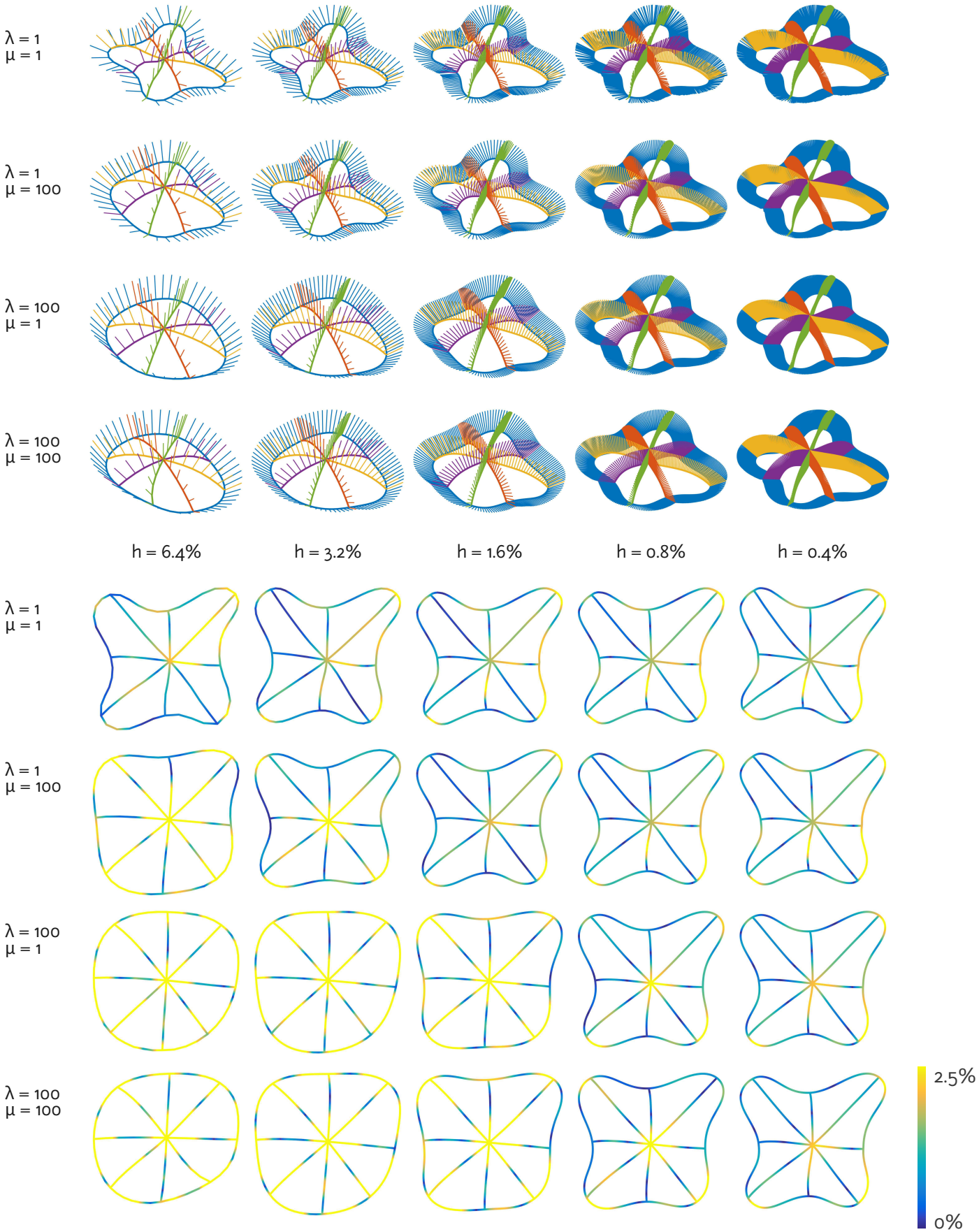
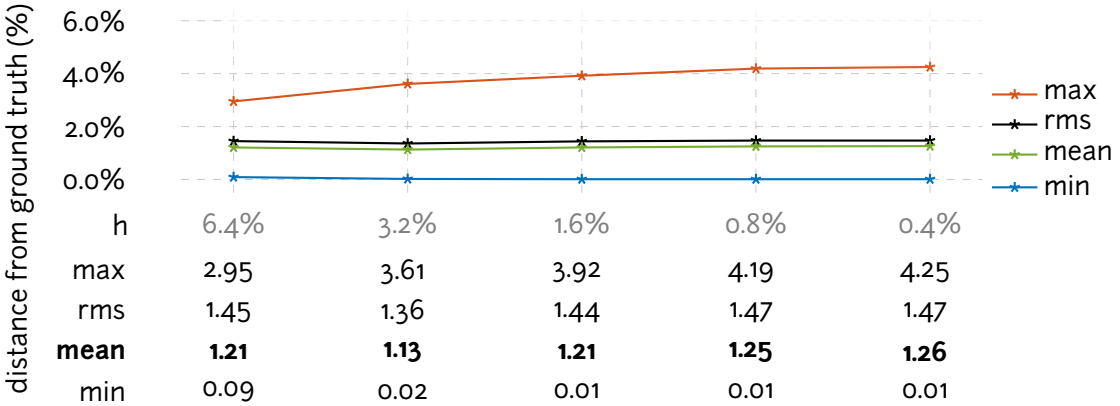


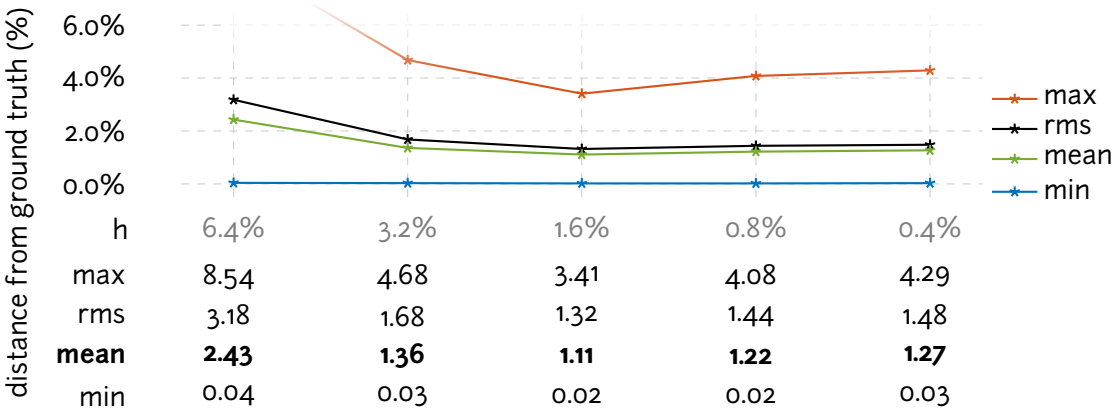
Figure 3.19: Network error, lilium (5 curves), smartphone, estimated lengths

λ = 1
μ = 1



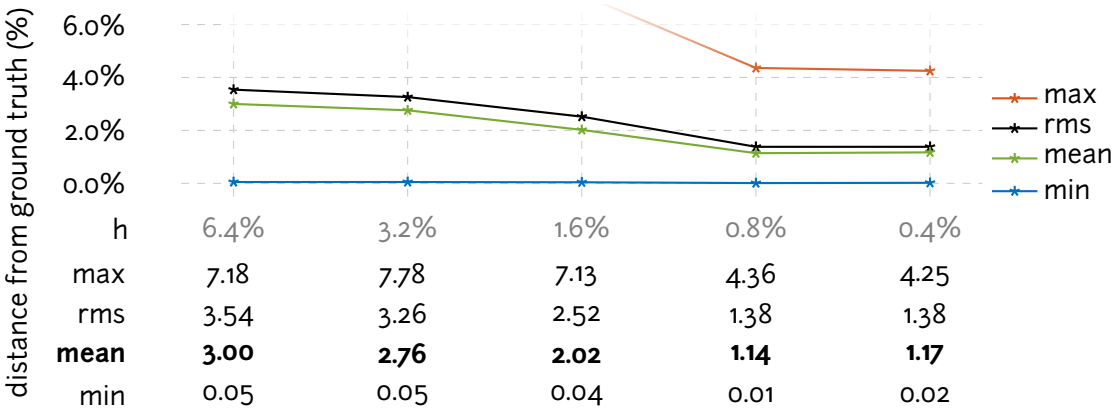
h	6.4%	3.2%	1.6%	0.8%	0.4%
max	2.95	3.61	3.92	4.19	4.25
rms	1.45	1.36	1.44	1.47	1.47
mean	**1.21**	**1.13**	**1.21**	**1.25**	**1.26**
min	0.09	0.02	0.01	0.01	0.01

λ = 1
μ = 100



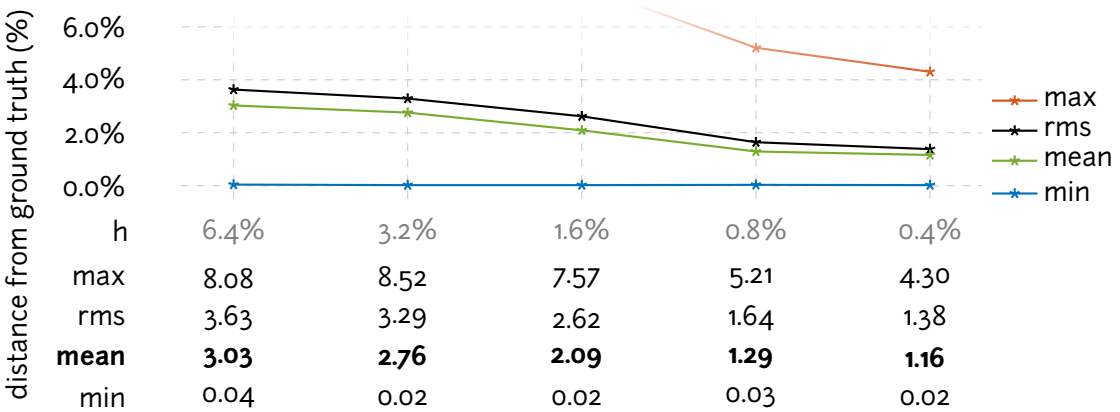
h	6.4%	3.2%	1.6%	0.8%	0.4%
max	8.54	4.68	3.41	4.08	4.29
rms	3.18	1.68	1.32	1.44	1.48
mean	**2.43**	**1.36**	**1.11**	**1.22**	**1.27**
min	0.04	0.03	0.02	0.02	0.03

λ = 100
μ = 1



h	6.4%	3.2%	1.6%	0.8%	0.4%
max	7.18	7.78	7.13	4.36	4.25
rms	3.54	3.26	2.52	1.38	1.38
mean	**3.00**	**2.76**	**2.02**	**1.14**	**1.17**
min	0.05	0.05	0.04	0.01	0.02

λ = 100
μ = 100



h	6.4%	3.2%	1.6%	0.8%	0.4%
max	8.08	8.52	7.57	5.21	4.30
rms	3.63	3.29	2.62	1.64	1.38
mean	**3.03**	**2.76**	**2.09**	**1.29**	**1.16**
min	0.04	0.02	0.02	0.03	0.02

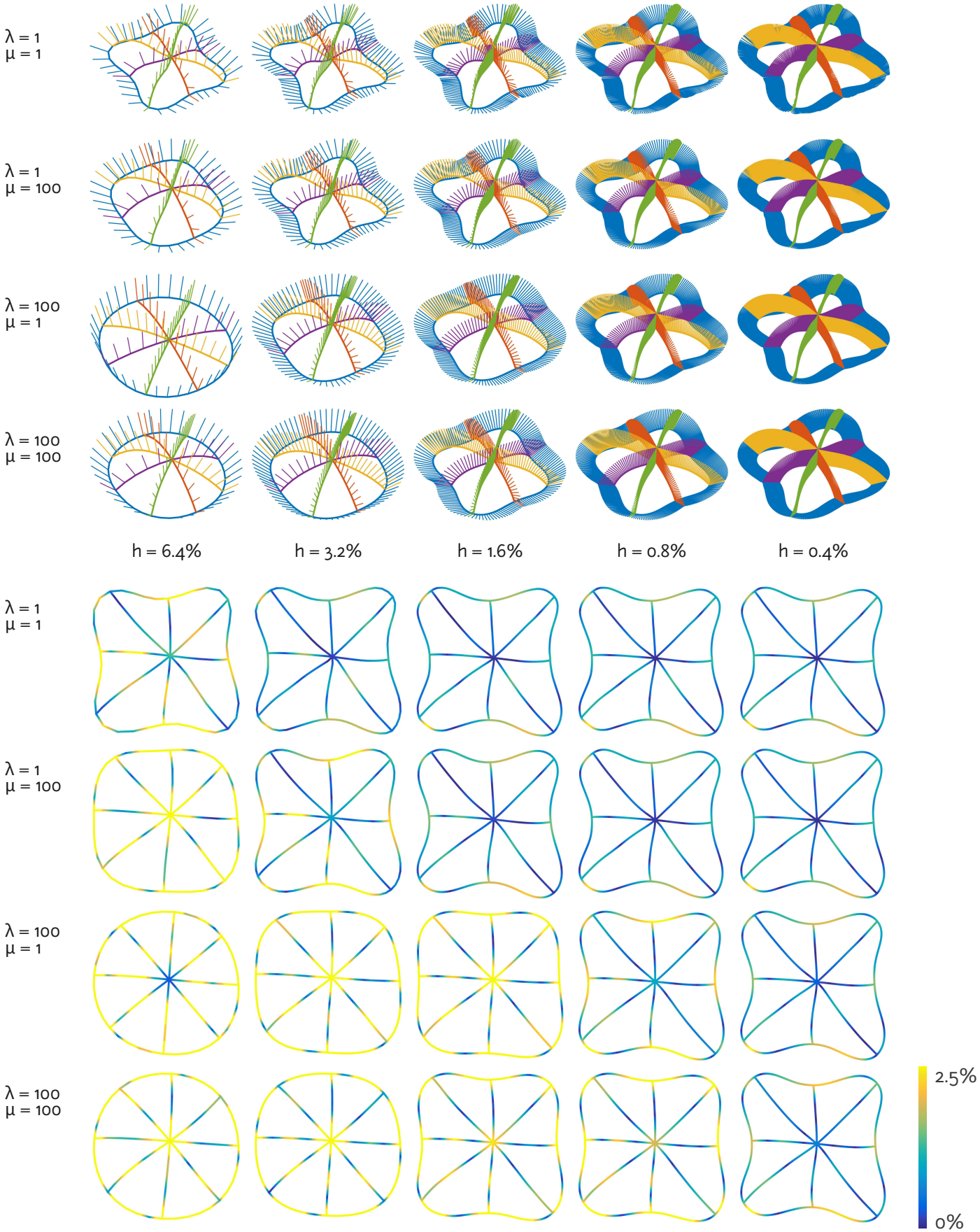
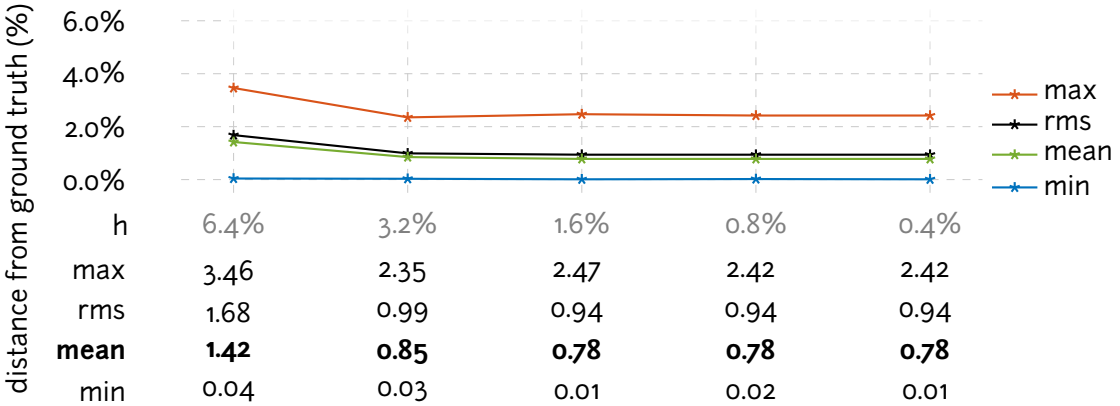


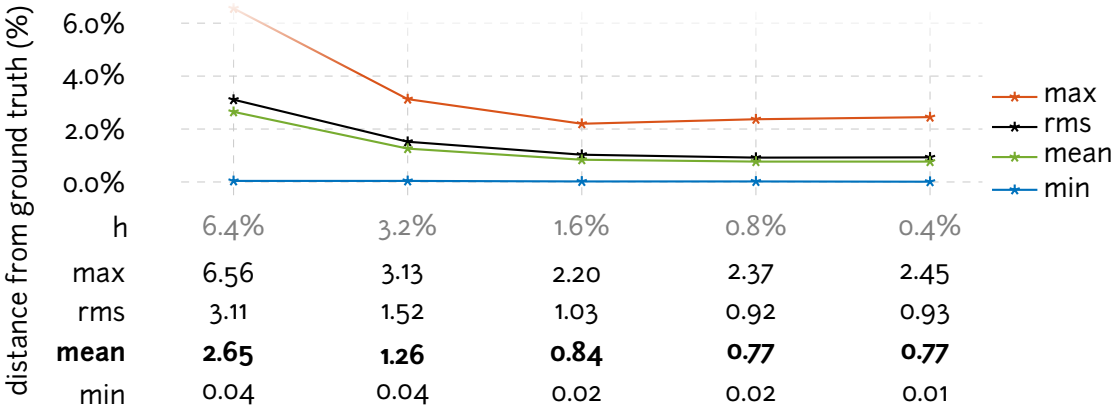
Figure 3.20: Network error, lilium (5 curves), smartphone, measured lengths

λ = 1
μ = 1



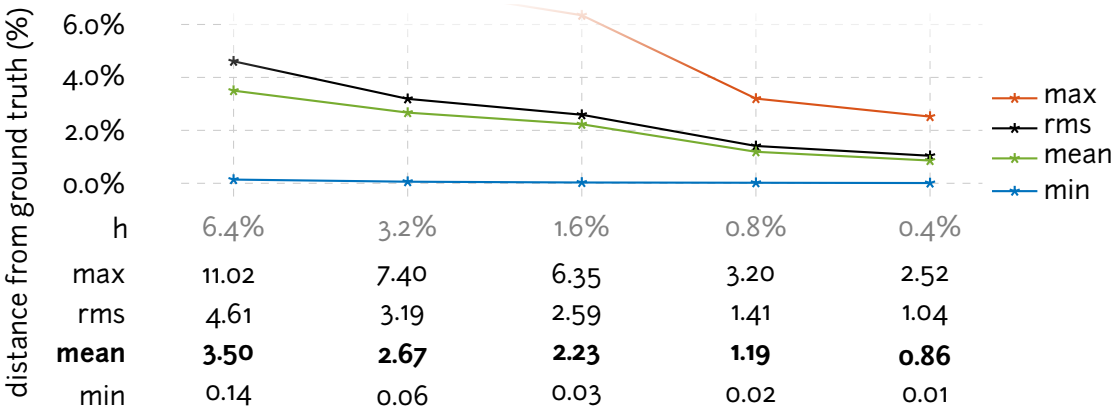
h	6.4%	3.2%	1.6%	0.8%	0.4%
max	3.46	2.35	2.47	2.42	2.42
rms	1.68	0.99	0.94	0.94	0.94
mean	**1.42**	**0.85**	**0.78**	**0.78**	**0.78**
min	0.04	0.03	0.01	0.02	0.01

λ = 1
μ = 100



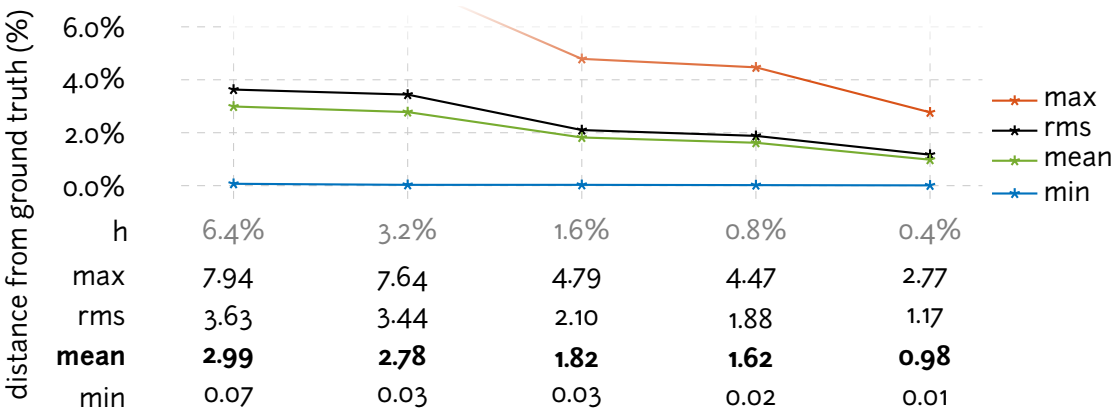
h	6.4%	3.2%	1.6%	0.8%	0.4%
max	6.56	3.13	2.20	2.37	2.45
rms	3.11	1.52	1.03	0.92	0.93
mean	**2.65**	**1.26**	**0.84**	**0.77**	**0.77**
min	0.04	0.04	0.02	0.02	0.01

λ = 100
μ = 1



h	6.4%	3.2%	1.6%	0.8%	0.4%
max	11.02	7.40	6.35	3.20	2.52
rms	4.61	3.19	2.59	1.41	1.04
mean	**3.50**	**2.67**	**2.23**	**1.19**	**0.86**
min	0.14	0.06	0.03	0.02	0.01

λ = 100
μ = 100



h	6.4%	3.2%	1.6%	0.8%	0.4%
max	7.94	7.64	4.79	4.47	2.77
rms	3.63	3.44	2.10	1.88	1.17
mean	**2.99**	**2.78**	**1.82**	**1.62**	**0.98**
min	0.07	0.03	0.03	0.02	0.01

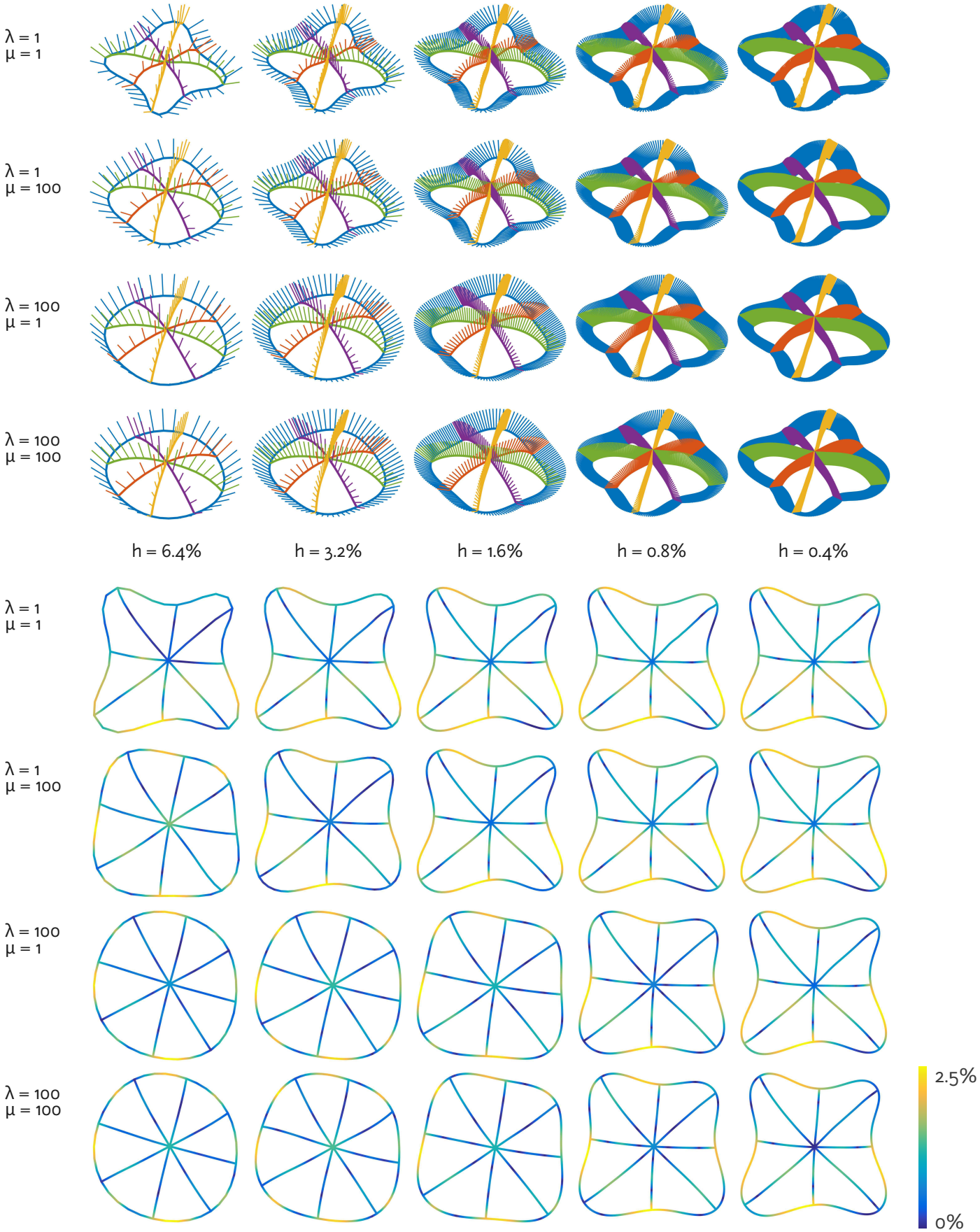
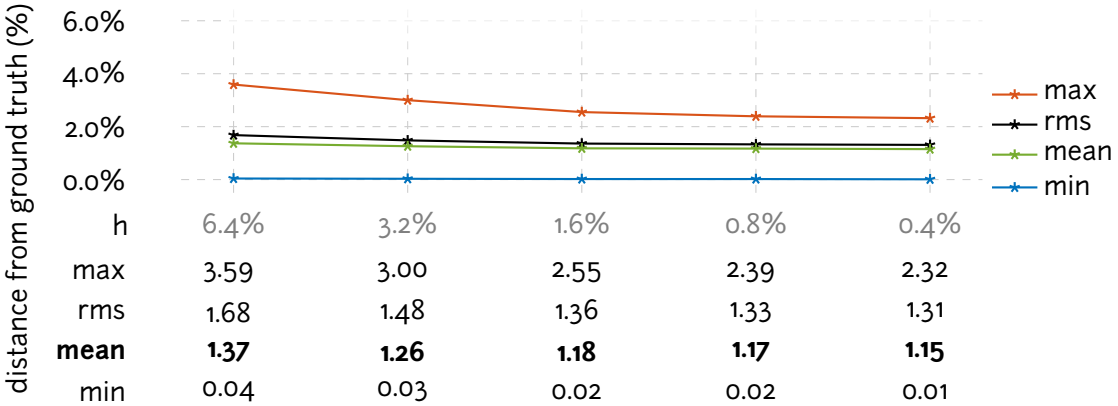


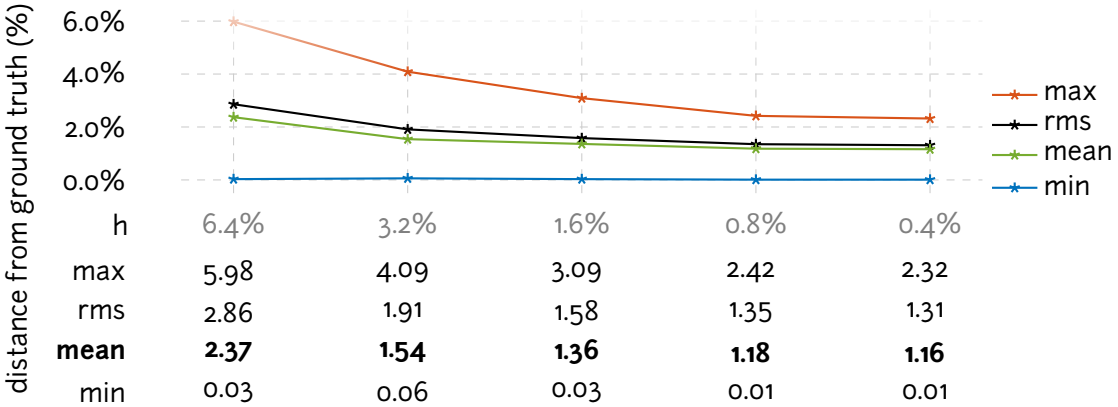
Figure 3.21: Network error, lilium (5 curves), Morphorider

λ = 1
μ = 1



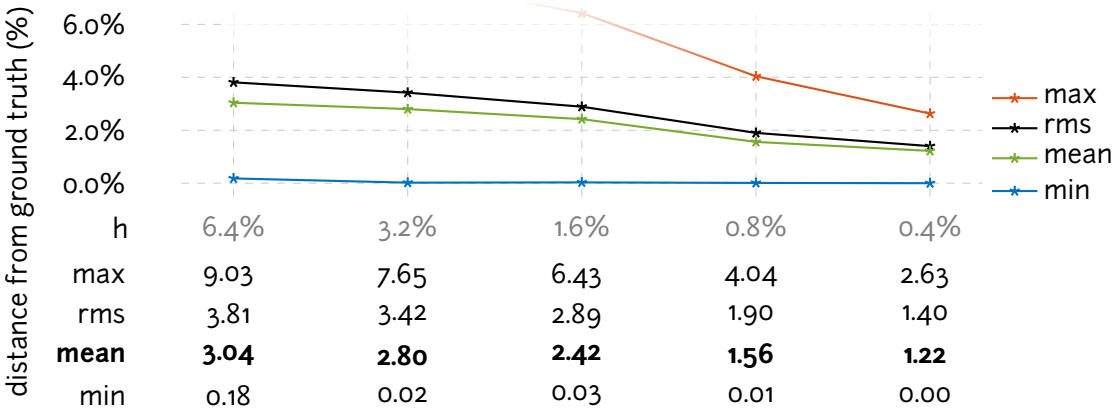
h	6.4%	3.2%	1.6%	0.8%	0.4%
max	3.59	3.00	2.55	2.39	2.32
rms	1.68	1.48	1.36	1.33	1.31
mean	**1.37**	**1.26**	**1.18**	**1.17**	**1.15**
min	0.04	0.03	0.02	0.02	0.01

λ = 1
μ = 100



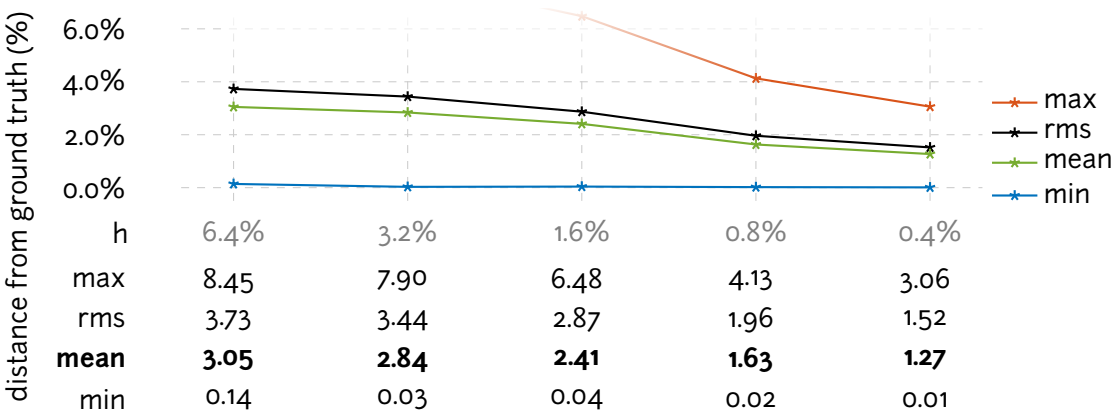
h	6.4%	3.2%	1.6%	0.8%	0.4%
max	5.98	4.09	3.09	2.42	2.32
rms	2.86	1.91	1.58	1.35	1.31
mean	**2.37**	**1.54**	**1.36**	**1.18**	**1.16**
min	0.03	0.06	0.03	0.01	0.01

λ = 100
μ = 1



h	6.4%	3.2%	1.6%	0.8%	0.4%
max	9.03	7.65	6.43	4.04	2.63
rms	3.81	3.42	2.89	1.90	1.40
mean	**3.04**	**2.80**	**2.42**	**1.56**	**1.22**
min	0.18	0.02	0.03	0.01	0.00

λ = 100
μ = 100



h	6.4%	3.2%	1.6%	0.8%	0.4%
max	8.45	7.90	6.48	4.13	3.06
rms	3.73	3.44	2.87	1.96	1.52
mean	**3.05**	**2.84**	**2.41**	**1.63**	**1.27**
min	0.14	0.03	0.04	0.02	0.01

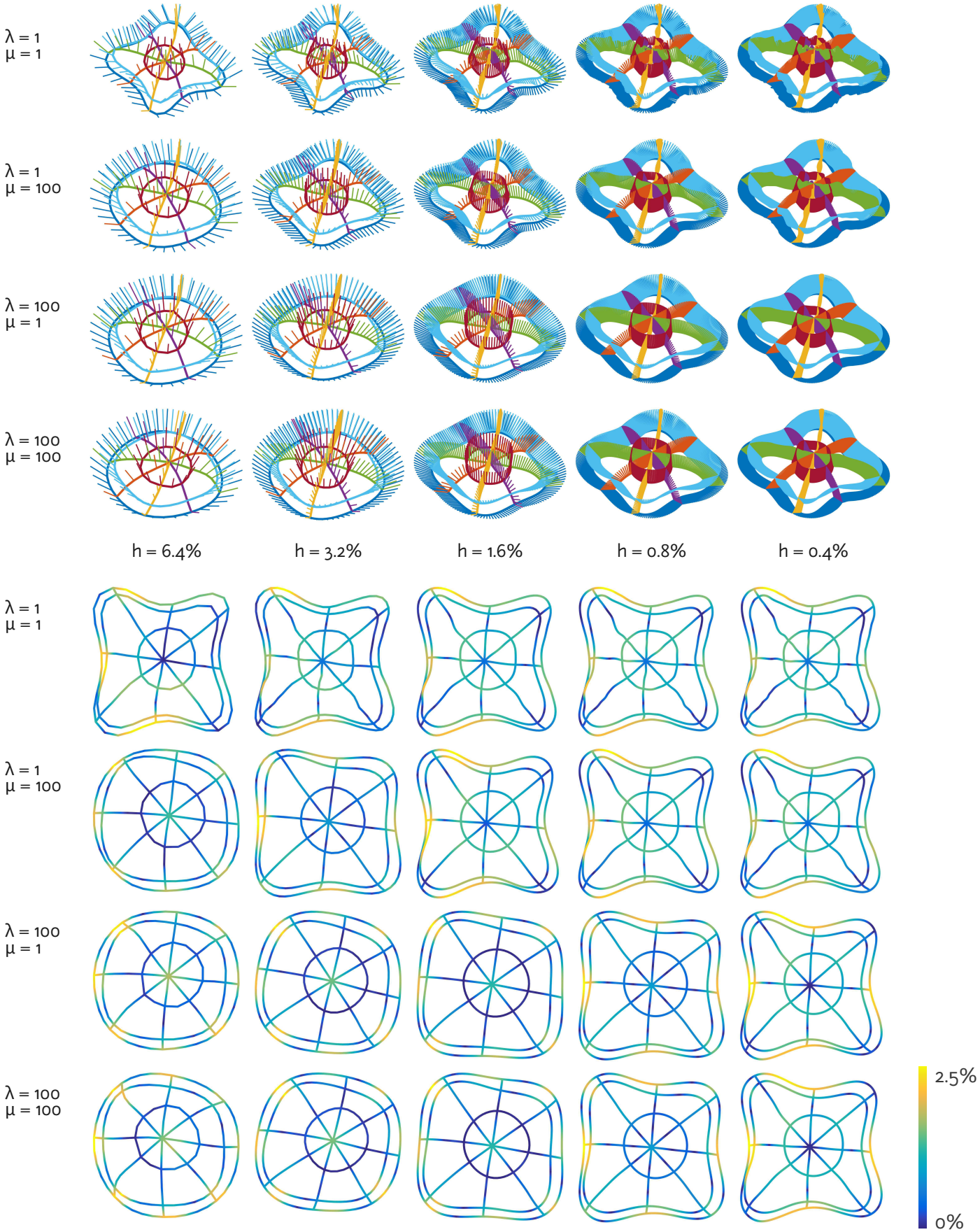
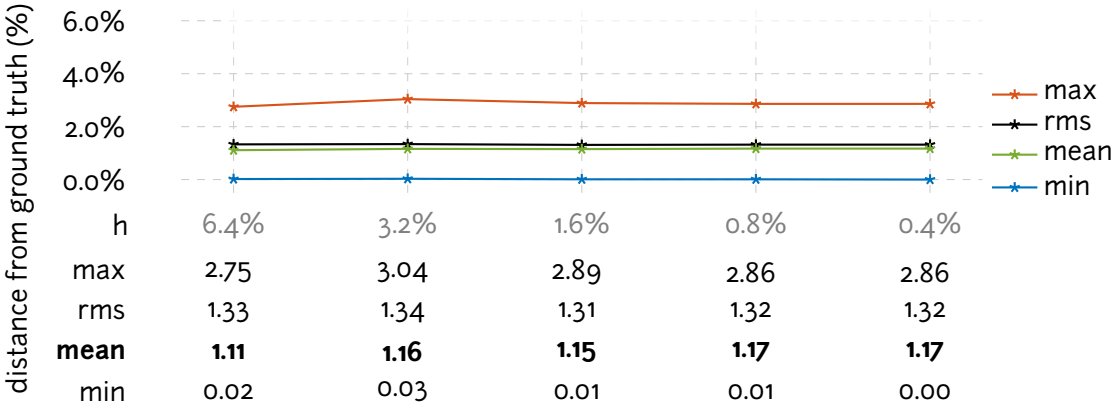


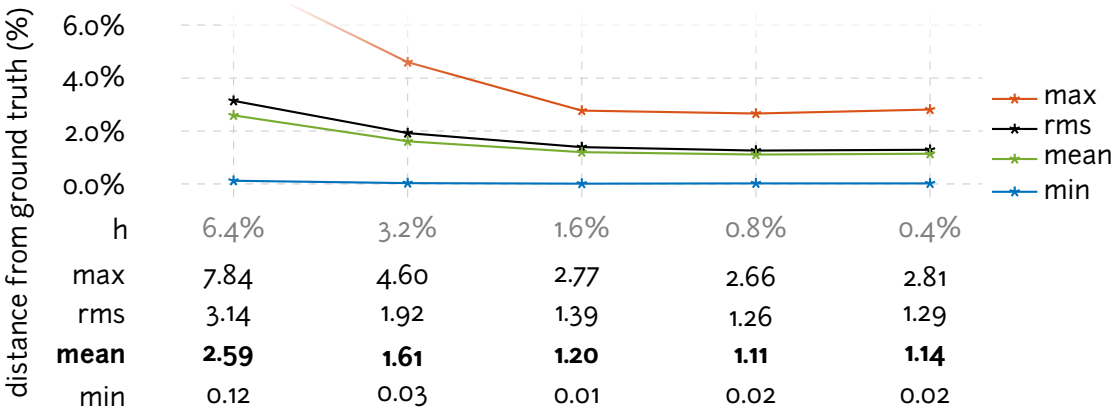
Figure 3.22: Network error, lilium (7 curves), Morphorider

λ = 1
μ = 1



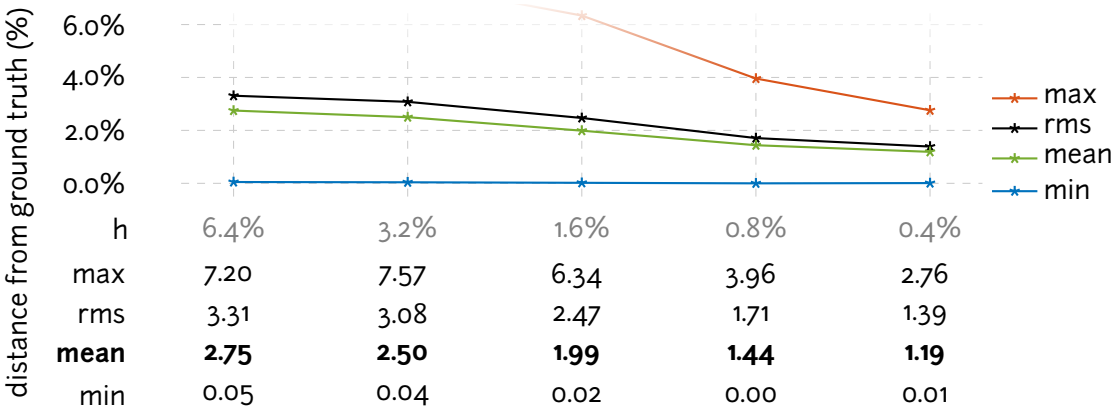
h	6.4%	3.2%	1.6%	0.8%	0.4%
max	2.75	3.04	2.89	2.86	2.86
rms	1.33	1.34	1.31	1.32	1.32
mean	**1.11**	**1.16**	**1.15**	**1.17**	**1.17**
min	0.02	0.03	0.01	0.01	0.00

λ = 1
μ = 100



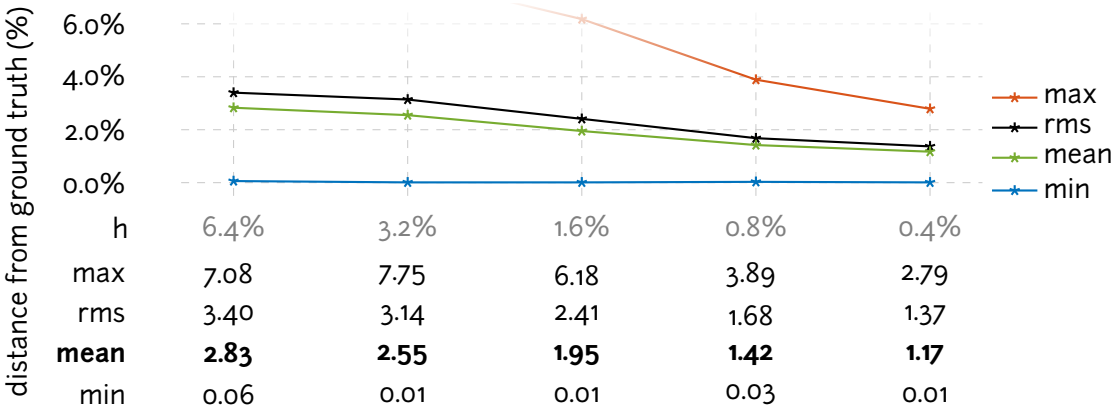
h	6.4%	3.2%	1.6%	0.8%	0.4%
max	7.84	4.60	2.77	2.66	2.81
rms	3.14	1.92	1.39	1.26	1.29
mean	**2.59**	**1.61**	**1.20**	**1.11**	**1.14**
min	0.12	0.03	0.01	0.02	0.02

λ = 100
μ = 1



h	6.4%	3.2%	1.6%	0.8%	0.4%
max	7.20	7.57	6.34	3.96	2.76
rms	3.31	3.08	2.47	1.71	1.39
mean	**2.75**	**2.50**	**1.99**	**1.44**	**1.19**
min	0.05	0.04	0.02	0.00	0.01

λ = 100
μ = 100



h	6.4%	3.2%	1.6%	0.8%	0.4%
max	7.08	7.75	6.18	3.89	2.79
rms	3.40	3.14	2.41	1.68	1.37
mean	**2.83**	**2.55**	**1.95**	**1.42**	**1.17**
min	0.06	0.01	0.01	0.03	0.01

4

Surfacing networks with normals

Second part of our framework is the surfacing of a curve network. Chapter 3 introduced our method for reconstruction of a curve network from orientations acquired along curves on a surface. This chapter describes the second contribution of this thesis: given a smooth network computed using our reconstruction method, we describe an algorithm for computing the underlying surface.

Application of algorithms from Chapter 3 to raw orientations results in a well-connected discrete curve network. Moreover, for each point in the network we also know the *surface normal* at that point: this is the normal component of the filtered orientations.

The goal of this chapter is to find a surface, which interpolates the discrete curve network and the surface normals. Section 4.2 gives a formal definition of what we mean by interpolation of positions and normals. This is an ill-posed problem: our attempt is to infer a (2-manifold) surface from a collection of (1-manifold) curves (Fig. 4.1). In order to solve it, we are forced to make additional assumptions about the surface (Section 4.3).

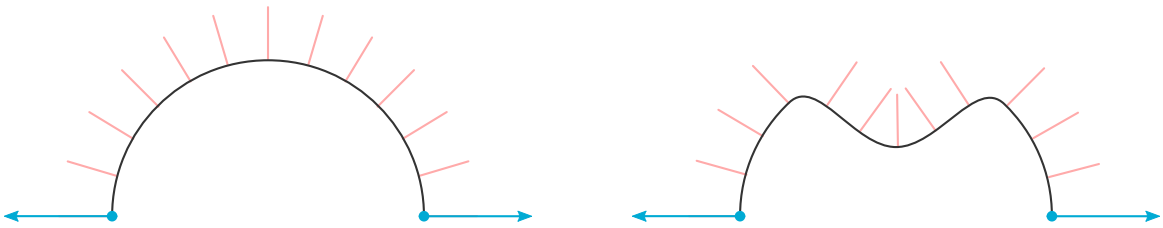


Figure 4.1: Surfacing of curve networks is an ill-posed problem, as shown in this 2D illustration. Both shapes interpolate the same input data (blue points and normals), but behave very differently in between.

Our method for computing the surface from a curve network with normals differs from previous sensor-related surfacing algorithms (Section 1.3.2). Indeed, in contrast to previous algorithms based on splines and Coons patches, we choose to work with triangle meshes, which can handle surfaces of arbitrary topological type. Another advantage is that our method exploits the knowledge of normals to compute the surface. Many previous methods only use positions, as it is more difficult to incorporate the normal constraints into a spline-based reconstruction.

In Section 4.6 we compare our method to related algorithms for surfacing networks, which are used in the context of variational surface modeling and sketch-based modeling. We show that our method provides surfaces whose quality is comparable to or better than the state-of-the-art algorithm of Pan et al. [Pan+15] while being an order of magnitude faster. Moreover, our method avoids manual specification of magnitude of normals, which is a limitation of most existing algorithms for surfacing curve networks with normals.

Our method for surfacing curve networks with normals has two main parts, which are summarized below.

Computation of topology. The surface is represented as a triangle mesh. To compute the topology of the mesh, we efficiently detect the cycles in the network (Section 4.4.1), which are then triangulated in plane (Section 4.4.2). The topology of the mesh is obtained by joining the triangulations of the individual cycles.

Computation of geometry. By solving two biharmonic systems with boundary constraints, we propagate the surface normals to the interior of patches, and obtain an initial guess for the vertex positions (Section 4.5.1). This allows us to compute discrete mean curvature for the whole mesh (Section 4.5.2). We then solve a constrained quadratic optimization, which minimizes an energy functional taking into account the previously estimated mean curvature (Section 4.5.3).

The structure of this chapter is similar to the previous chapter. We begin by reviewing the related work in Section 4.1. We then formally state the problem in Section 4.2 and give an overview of our method in Section 4.3. Technical parts of

the algorithm are described in Sections 4.4 to 4.5. Section 4.6 discusses the evaluation of the method and compares it to two state-of-the-art algorithms. Finally, Section 4.7 concludes the chapter.

The work presented in this chapter was originally published in [Sta+16a; Sta+16b] and presented at [GTMG16; EG16; jFIG17]. See the list of publications on p. 187.

4.1 Related work

The work related to this chapter is organized into three sections. Section 4.1.1 describes variational methods for surface modeling. Section 4.1.2 gives an overview of methods for surfacing of curve networks, often in the context of sketch-based modeling. Section 4.1.3 summarizes algorithms for computing multi-sided parametric patches.

4.1.1 Variational modeling with normals

We focus on variational methods that provide means for modeling and deformation of surfaces by specifying both positional and normal constraints along a network of curves. For an introduction to variational modeling of shapes, see Section 2.8.

Moreton and Séquin [MS92; MS91] presented a framework for surfacing of networks of curves with G^2 continuity across patches. Their method enables interpolation of a set of geometric constraints: positions, normals and curvatures. They introduce the important concepts of minimum variation curves (MVC) and minimum variation surfaces (MVS), obtained by minimizing integrals of squared magnitudes of curvatures:

$$\int \frac{\mathrm{d}\kappa^2}{\mathrm{d}s}\,\mathrm{d}s, \qquad\qquad \iint \left\|\frac{\partial \kappa_1}{\partial \mathbf{t}_1}\right\|^2 + \left\|\frac{\partial \kappa_2}{\partial \mathbf{t}_2}\right\|^2 \mathrm{dA}.$$

for curves and surfaces, respectively; cf. Eq. (2.43), p. 47. While this approach produces high-quality shapes and has desired theoretical properties, the resulting optimization is nonlinear and computationally expensive. The surface representation used in the implementation of Moreton and Séquin [MS92] are quintic triangular and quadrilateral Bézier patches. This places a limitation on the topology of the input curve network, unless a multi-sided generalization of Bézier patches is used (Section 4.1.3). Another limitation is that the constraints (positions, tangents, curvatures) can only be prescribed at patch corners, not along the patch boundary.

Linearized versions of nonlinear geometric functionals are more suitable for interactive applications, providing an appropriate trade-off between precision and computation speed. Minimizing such linearized functionals often comes down

to solving a simple linear system involving the discrete Laplace-Beltrami operator [PP93; Tau95; Kob97; Des+99; Mey+03; CPS13]. For more details, see Sections 2.7 and 2.8.

Schneider and Kobbelt [SK01] presented an approach to triangle mesh fairing subject to G^1 boundary conditions (vertex positions and unit normals). The surface $\mathcal{S}$ is computed using the mean curvature fairness criterion

$$\Delta_{\mathcal{S}} H = 0,$$

where $\Delta_{\mathcal{S}}$ is the Laplace-Beltrami operator of $\mathcal{S}$ and H is the mean curvature of $\mathcal{S}$. This approach is the surface analogue of a similar equation for curves $\kappa'' = 0$, which yields clothoid splines [Far+87; BLP10].

Botsch and Kobbelt [BK04] introduced an interactive framework for shape deformation by solving for the k^{th}-order Laplacian while imposing boundary conditions up to C^{k-1} (Fig. 4.2)

$$\Delta_{\mathcal{S}}^{k} \mathbf{x}(u,v) = \mathbf{0} \qquad (u,v) \in \Omega \backslash \partial\Omega, \tag{4.1a}$$

$$\Delta_{\mathcal{S}}^{j} \mathbf{x}(u,v) = \mathbf{b}_j(u,v) \qquad (u,v) \in \partial\Omega\,,\ j < k. \tag{4.1b}$$

Notice that Botsch and Kobbelt [BK04] did not implement the boundary constraints directly; instead, they fixed positions of $(k-1)$ rings of vertices to impose the C^{k-1} continuity, a common practice in discrete variational modeling [Bot+10]. However, this setting prevents dealing with arbitrary constrained curve networks without specifying the positions of $(k-1)$ rings of vertices.

Jacobson et al. [Jac+10] (see also Alec Jacobson's thesis [Jac13]) described a generalization of the method of Botsch and Kobbelt [BK04]. For a continuous deformation $\mathbf{u} : \Omega \to \mathbb{R}^3$ of a planar sheet over the domain Ω, they minimized Laplacian and Laplacian gradient energies (Fig. 4.3)

$$E_2 = \iint_{\Omega} \|\Delta \mathbf{u}\|^2 \, dA, \tag{4.2a}$$

$$E_3 = \iint_{\Omega} \|\nabla \Delta \mathbf{u}\|^2 \, dA. \tag{4.2b}$$

Instead of direct minimization of E_2 and E_3, the authors introduce a new variable $\mathbf{v} = \Delta \mathbf{u}$. This variable is then used to decompose the Euler-Lagrange equations $\Delta^k \mathbf{u} = 0, k = 2, 3$, into the following lower-order constrained optimization:

$$\min \iint_{\Omega} \|\mathbf{v}\|^2 \, dA \quad \text{s.t.} \quad \Delta \mathbf{u} = \mathbf{v}, \tag{4.3a}$$

$$\min \iint_{\Omega} \|\nabla \mathbf{v}\|^2 \, dA \quad \text{s.t.} \quad \Delta \mathbf{u} = \mathbf{v}. \tag{4.3b}$$

Figure 4.2: The framework of Botsch and Kobbelt [BK04] enables prescribing different levels of continuity. The three pipes were obtained by solving $\Delta^k \mathbf{v} = \mathbf{0}$, $k = 1, 2, 3$, with C^{k-1} continuity on the boundary. Image from [BK04].

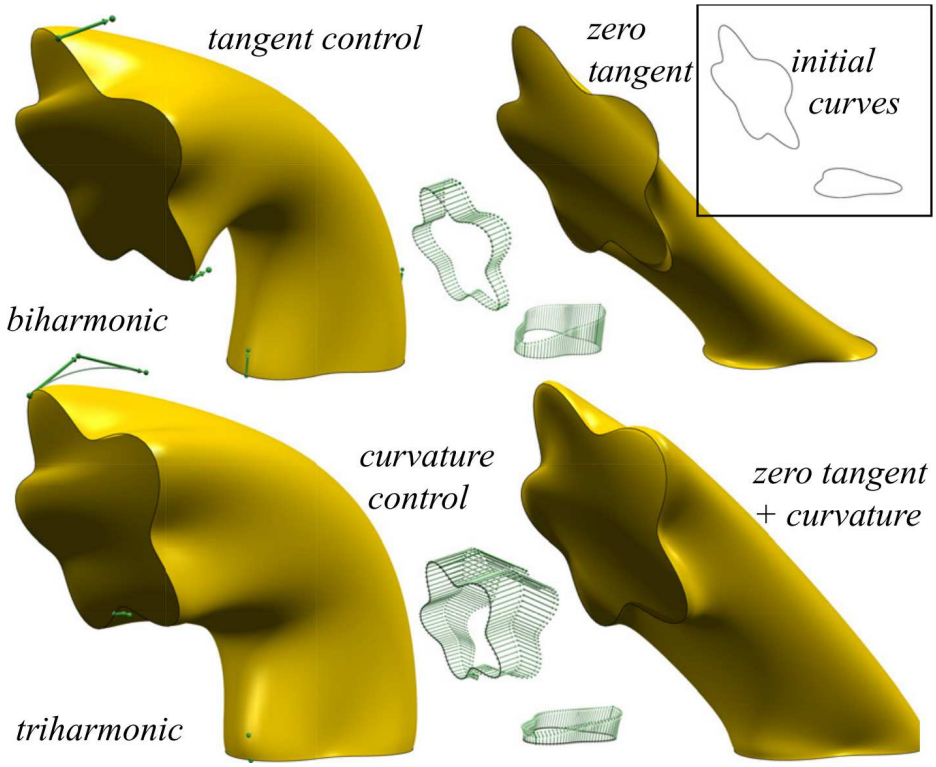


Figure 4.3: Mixed finite element method of Jacobson et al. [Jac+10] for surface deformation. Image from [Jac+10].

Three types of boundary conditions are considered: region, curve, and point. The discretization of the lower-order decomposition is carried out via mixed finite elements [CR74]. Similarly to the approach of Botsch and Kobbelt [BK04], higher-order constraints – tangents and second derivatives – are prescribed by fixing corresponding rings of vertices. Intersecting curve boundary conditions require special treatment, as they tend to produce over-determined systems; Jacobson et al. [Jac+10] solve this problem by averaging the conflicting values at vertices that cause problems.

In contrast to the above approaches, our method does not require fixing $(k-1)$ rings of vertices in order to enforce C^k continuity. Instead, we directly enforce the normal constraints available on the input. Another advantage of our method is that it does not need manual specification of magnitude of normals often required by previous methods.

4.1.2 Surfacing curve networks

Surfacing of a network of curves, also called lofting or skinning, is a classical problem in computer-aided geometric design (CAGD) [Far02]. Recently, it was revived thanks to the development in the field of sketch-based modeling. Interactive 3D sketching tools [Nea+07; Ros+07; KS07; BBS08; Sch+09] or methods for inferring 3D curves from 2D sketches [DDG05; Cor+11; Xu+14; Fon+17] provide raw curve networks, which serve as the input for surfacing algorithms. The common assumption in these works is that the underlying curve network was created with some design intent, and that the input information is minimal.

Most methods dealing with surfacing of 3D curves focus more on the detection of cycles in the input network, and less on the global quality of the final shape [SWZ04; AJA11; Bes+12; Abb+13; Zhu+13; SS14]. An exception is the state-of-the-art method of Pan et al. [Pan+15], which produces globally smooth output shapes of high quality.

Schaefer et al. [SWZ04] introduce a subdivision algorithm for interpolating a curve network with smooth limit surface. Their method requires the patches (i.e. cycles in the network) to be specified manually. Each cycle is first quadrangulated to obtain a base mesh $\mathcal{T}_0$ before applying a modified Catmull-Clark subdivision. The vertices of the base mesh are positioned in such a way that the limit surface interpolates the input curve network.

Bessmeltsev et al. [Bes+12] introduce a method for quadrangulation of artist-designed curve networks using computationally expensive segmentation and pairing of the input cycles, and apply a variety of heuristics to find the connectivity graph of the quadrangulation. They use discrete Coons patches to obtain the resulting quad mesh.

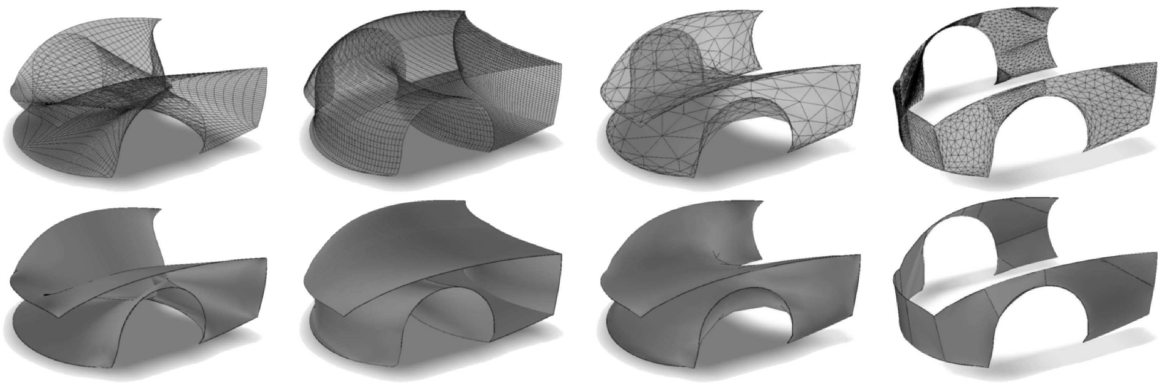


Figure 4.4: Comparison of methods for surfacing sketched curve networks. Left to right, subdivision lofting [SWZ04], design-driven quadrangulation [Bes+12], linearized Laplacian smoothing [AJA11], minimal ruled surface [Abb+13]. Image from [Abb+13].

Zhuang et al. [Zhu+13] present an efficient algorithm for detecting cycles in complex networks using a structure called the *routing system* – a collection of corners (two consecutive curves on one cycle) and bridges (two consecutive corners in one cycle). They search for the routing system, which minimizes two cost metrics: the intra-bridge cost, measured using the parallel-transport normals computed using the rotation minimizing frame [Wan+08]; and the inter-bridge cost, which forces the patches to be partitioned uniformly around a curve. To visualize the patches, each cycle is triangulated with help of the parallel-transport normals that minimize the intra-bridge cost. The triangulation is computed using the dynamic programming algorithm of Zou et al. [ZJC13] to match the normals as best as possible.

Sadri and Singh [SS14] present an approach to network surfacing using the *flow complex* [GJ08], a computational geometry structure similar to the Delaunay triangulation, but providing better insight to the topology of the underlying surface.

The flow-aligned surfacing algorithm of Pan et al. [Pan+15] attempts to mimic the human perception by computing a surface consistent with design intent of the input network. The input curves are split into two groups [Xu+14; Bes+12]. The first group are the *flow lines* (lines of curvature), which capture the curvature directions of the underlying shape (cf. end of Section 2.3). Surface is at least G^1 continuous across the flow lines. The second group are the *trim curves*, which define surface boundaries or sharp curves. Trim curves are G^0 across patches. Pan et al. [Pan+15] use the result of Zhuang et al. [Zhu+13] as the initial mesh, which is then iteratively refined by aligning the curvature tensor with the induced flow field. This method does not assume normal input known *a priori*, but estimates surface normals using the rotation-minimizing frame (RMF) [Wan+08; BFS10]. The rotation-minimizing frame is used to estimate surface normals and determine which curves are the flow

lines; the estimated normals are used as boundary conditions. The algorithm is not interactive – for a typical mesh with 12k vertices, it takes 7-8 seconds to converge (10-12 iterations with $\approx$ 0.7s per iteration).

Despite the differences in inputs for the method of Pan et al. [Pan+15] and for our method, flow-aligned surfacing is closest to our work in this branch of research. We compare our results with [Pan+15] in Section 4.6.4.

4.1.3 Multi-sided patches

Multi-sided patches generalize triangular interpolants ($n = 3$) and quadrangular interpolants ($n = 4$) to boundaries with higher number of curves ($n > 4$). They are usually computed with transfinite interpolation methods, possibly with prescribed tangent ribbons to achieve G^1-continuity across boundary curves. Examples of methods for computing multi-sided patches include generalized Coons patches [Far02], Gregory patches [GZ94], generalized Bézier patches [VSK16], and subdivision approaches [SWZ04].

Essentially, there are two main groups of methods. The first group of methods assumes a pre-segmentation of each input cycle into n curve segments with low-distortion mapping to a convex planar n-sided polygon. Prescribed tangent ribbons must be defined consistently and twists estimated accordingly. Methods in the second group quadrangulate the input cycles with topological guarantees on the extraordinary vertices and approximate the coarse mesh using well-known subdivision schemes.

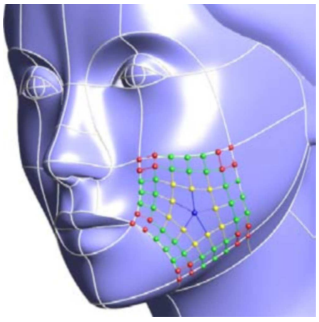


Image from [VSK16]

Instead of using multi-sided patches for curve network interpolation, we use the triangle mesh representation in our surfacing method. The advantage of triangle meshes is that they can handle surfaces of arbitrary topological type. Furthermore, triangle mesh representation is better-suited for prescribing surface normals along the input curve network.

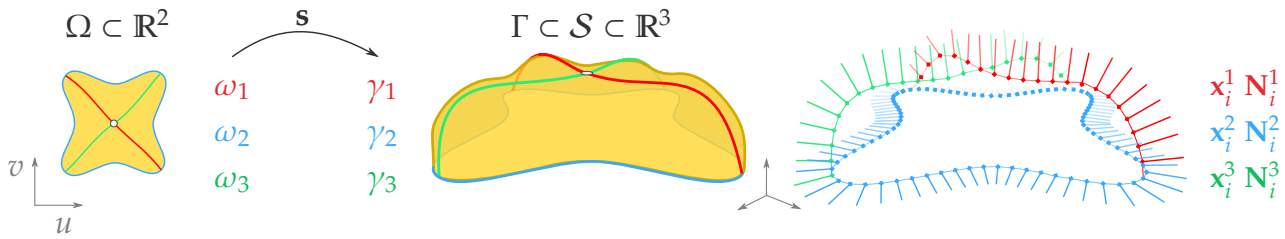


Figure 4.5: Our setup for surfacing of a curve network (Section 4.2)

4.2 Problem statement

This section formalizes the problem of surfacing of a curve network with normals that we consider in this chapter. General setting of the problem is similar to the previous chapter (see Section 3.2 on p. 55).

Let $\mathcal{S}$ be an unknown surface. We assume that $\mathcal{S}$ is smooth and connected, with or without boundary, locally parametrized by

$$\mathbf{s} : \Omega \subset \mathbb{R}^2 \to \mathcal{S} \subset \mathbb{R}^3 : (u, v) \mapsto \mathbf{s}(u, v). \tag{4.4}$$

We consider a curve network $\Gamma \subset \mathcal{S}$ on the surface, which induces a cell complex structure $\Gamma = (\Upsilon, \Sigma, \Omega)$ on $\mathcal{S}$. The cell complex consists of nodes Υ, segments Σ, and cycles Ω (Fig. 2.7, p. 40). A curve $\gamma \in \Gamma$ with length L is G^1 smooth and without self-intersections. Moreover, it corresponds to a sequence of segments in Σ. For instance, the blue curve in Fig. 4.5 contains four segments, delimited by four nodes (curve intersections).

A parametrization of a curve $\gamma \in \Gamma$ is a composition of two maps (Fig. 4.5). First, the interval $[0, L]$ is mapped to a planar curve in the domain Ω:

$$\begin{array}{rcccl} \omega & : & [0, L] & \longrightarrow & \Omega \subset \mathbb{R}^2 \\ & & t & \longmapsto & \omega(t) = \big(u(t), v(t)\big) \end{array} \tag{4.5}$$

The second map is the surface parametrization $\mathbf{s}$. Taking the composition of Eq. (4.4) and Eq. (4.5), a space curve $\gamma \in \Gamma$ is parametrized by

$$\begin{array}{rcccl} \mathbf{x} & : & [0, L] & \longrightarrow & \mathcal{S} \subset \mathbb{R}^3 \\ & & t & \longmapsto & \mathbf{x}(t) = \mathbf{s}(\omega(t)) \end{array} \tag{4.6}$$

Note that we use the above parametrizations only to state the problem formally. In practice, neither $\mathbf{s}$ nor ω are known.

The Darboux frame $\mathcal{D} = \{\mathbf{t}, \mathbf{N}, \mathbf{B}\}$ along the curve $\gamma \in \Gamma \subset \mathcal{S}$ at $\mathbf{x}(t) = \mathbf{s}(u, v)$ is given by (cf. Definition 2.11)

$$\mathbf{t} = \mathbf{x}' = \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}, \quad \mathbf{N} = \frac{\mathbf{s}_u \times \mathbf{s}_v}{\|\mathbf{s}_u \times \mathbf{s}_v\|}, \quad \mathbf{B} = \mathbf{t} \times \mathbf{N}.$$

Given a discrete network $\mathcal{N}$ of vertices $\mathbf{x}_i$ and surface normals $\mathbf{N}_i$ sampled uniformly from Γ (Fig. 4.5 right), our goal is to find a surface $\mathcal{S}$, which interpolates $\mathbf{x}_i$ and $\mathbf{N}_i$, and is visually smooth. More precisely, we seek to satisfy the following conditions:

$$\mathbf{x}_i \in \mathcal{S}, \qquad \mathbf{N}_i \perp \mathrm{T}_{\mathbf{x}_i}\mathcal{S}. \tag{4.7}$$

The topology of the discrete curve network $\mathcal{N}$ is known and given as a set of edges $e_{ij} \in \mathcal{E}_\mathcal{N}$, where e_{ij} is an edge connecting the vertices $\mathbf{x}_i$ and $\mathbf{x}_j$. See Section 2.6 for more details.

As we already stated in the introduction of this chapter, this problem does not have a unique solution (Fig. 4.1). In order to solve it, we need to make additional assumptions on the surface $\mathcal{S}$.

In the next section, we discuss the additional assumptions and summarize our approach for solving the surfacing problem.

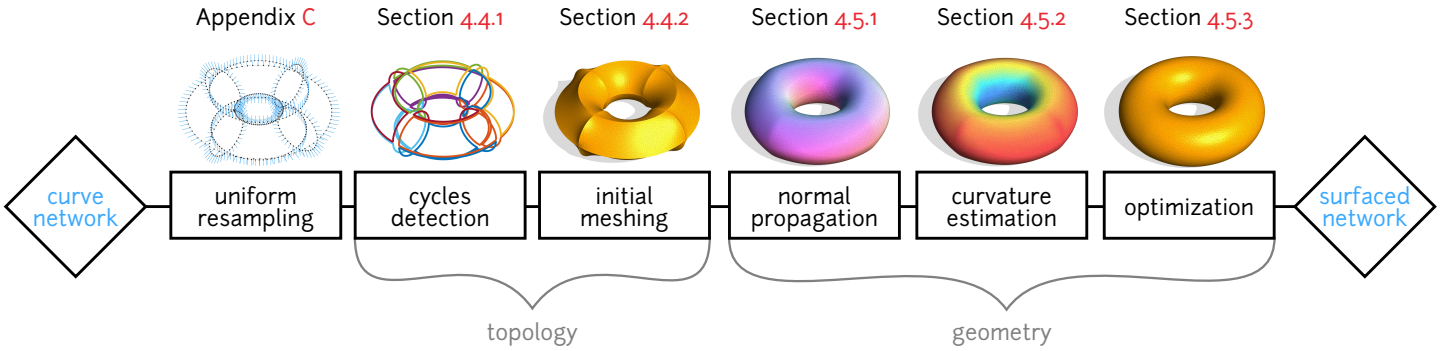
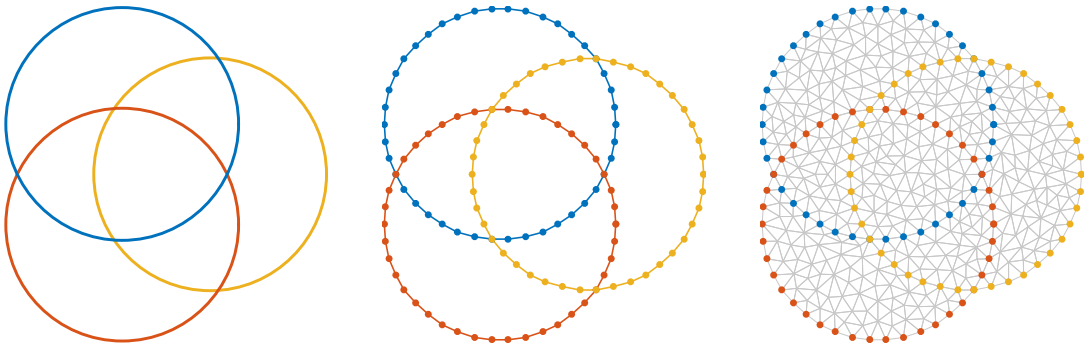


Figure 4.6: A schematic overview of our method for surfacing of curve networks. More details about the optional uniform resampling step are given in Appendix C.

4.3 Method overview

This section summarizes the main ideas of our method for computing a surface $\mathcal{S}$ that satisfies the conditions (4.7).

The overview of the method is given in Fig. 4.6. The input data is a discrete network $\mathcal{N}$ with vertices $\mathbf{x}_i$ and normals $\mathbf{N}_i$ sampled from the unknown surface $\mathcal{S}$ along a curve network Γ. The topology of $\mathcal{N}$ is also given as the input and specified as a set of edges $e_{ij} \in \mathcal{E}_\mathcal{N}$. We suppose the network is sampled uniformly; if this is not the case, the data are interpolated by a network of cubic splines and resampled. In Appendix C, we describe an algorithm for computing a G^1 interpolant of the input data $\mathbf{x}_i$, $\mathbf{N}_i$. Note that the networks reconstructed using or method from Chapter 3 are already sampled uniformly and do not need to be resampled (cf. Eq. (3.22)).



(a) network Γ **(b)** discrete network $\mathcal{N}$ **(c)** triangle mesh $\mathcal{T}$

Figure 4.7: We use a discrete representation to solve the problem of surfacing of a curve network, here illustrated on a planar network ($\mathcal{S} = \mathbb{R}^2$). The triangle mesh **(c)** interpolates the discrete network **(b)** sampled from a set of smooth curves **(a)**. See also Fig. 2.7.

The unknown surface $\mathcal{S}$ is represented as a triangle mesh $\mathcal{T} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$. The geometry of $\mathcal{T}$ is specified by the embedding of its vertices:

$$\mathbf{v} : \mathcal{V} \to \mathbb{R}^3 : v_k \in \mathcal{V} \mapsto \mathbf{v}_k = \mathbf{v}(v_k).$$

Our goal is to compute the mesh $\mathcal{T}$ (topology and geometry) so that the following conditions are satisfied

(i) **Topological subset** (Fig. 4.7). Edges of the discrete network form a subset of edges of the triangle mesh: $\mathcal{E}_\mathcal{N} \subset \mathcal{E}$.

(ii) **Interpolation of positions.** Vertices of the discrete network are a subset of vertices of the triangle mesh: $\mathbf{x}_i = \mathbf{v}_k$ for some $v_k \in \mathcal{V}$.

(iii) **Interpolation of normals.** Vertex normals along the discrete curve network match the input normals: if $\mathbf{x}_i = \mathbf{v}_k$, then $\mathbf{N}(v_k) = \mathbf{N}_i$.

The last condition requires a suitable definition of per-vertex normals $\mathbf{N}(v_k)$ for the mesh $\mathcal{T}$. We use the usual angle-weighted averaging of adjacent face normals [JLW05; Bot+10].

Our method for finding a mesh $\mathcal{T}$, satisfying the above conditions, has two parts: the computation of topology, and the computation of geometry (cf. Fig. 4.6).

Computation of topology (Section 4.4). To get the topology, we first find the boundaries of surface patches in the discrete network, by detecting the network cycles. Detection of cycles for a general curve network is a hard and ambiguous problem – usually, only the positions are given as the input. In our case, this ambiguity can be efficiently resolved thanks to our knowledge of input surface normals along $\mathcal{N}$. Our cycles detection algorithm is described in Section 4.4.1.

Each cycle is meshed individually by tessellating the closed boundary of the cycle. This tessellation is done by computing a constrained Delaunay triangulation (CDT) in the plane. Since the boundary of a cycle is generally a space curve, it has to be mapped to a plane before computing the CDT. We use the projection to a plane for this map.

In order for this approach to be valid, we require this planar projection to be injective. This places additional constraints on the input network. More details are given in Section 4.4.2, where we also compare the projection to other planar maps.

Triangulations of individual cycles are joined together to get the set of faces $\mathcal{F}$ of the mesh.

Computation of geometry (Section 4.5). After computing the topology of the mesh, we are interested in finding the embedding $\mathbf{v}_k$ of the vertices, which satisfies the conditions (ii) and (iii). To this end, we use the connection between the Laplace-Beltrami operator and the mean curvature normal of the mesh:

$$\Delta_{\mathcal{S}}\, \mathbf{v}_k = \mathbf{H}_k. \tag{4.8}$$

See Eq. (2.11) on p. 26. In order to exploit the input normals, the mean curvature normal is *not* computed using the cotangent formula as in Eq. (2.38). Instead, we use the following alternative definition:

$$\mathbf{H}_k = \mathrm{H}_k \mathbf{N}^*_k. \tag{4.9}$$

In this definition, H_k is the unknown per-vertex mean curvature, and $\mathbf{N}^*_k$ is the per-vertex normal obtained by propagating the input normals $\mathbf{N}_i$ to all vertices of the mesh. Section 4.5.1 describes how the propagated normals are computed along with initial positions $\mathbf{v}^*_k$ of the vertices of $\mathcal{T}$. Section 4.5.2 describes the computation of H_k by combining the initial vertices $\mathbf{v}^*_k$ and the propagated normals $\mathbf{N}^*_k$ into a compact curvature measure.

Finally in Section 4.5.3, we compute the vertex positions $\mathbf{v}_k$. This is done by minimizing an energy formulated using Eq. (4.8) and Eq. (4.9). The input vertex positions $\mathbf{x}_i$ are used as boundary constraints in this optimization.

4.4 Computing the topology

The objective of this chapter is to compute a mesh that interpolates the input positions and normals. In this section, we describe the first part of our surfacing method: computation of topology of the triangle mesh $\mathcal{T} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$.

We begin by detecting cycles in the input curve network (Section 4.4.1). Each cycle is then tessellated individually using the constrained Delaunay triangulation or CDT (Section 4.4.2). The topology of $\mathcal{T}$ is obtained by combining the individual triangulations into a single mesh.

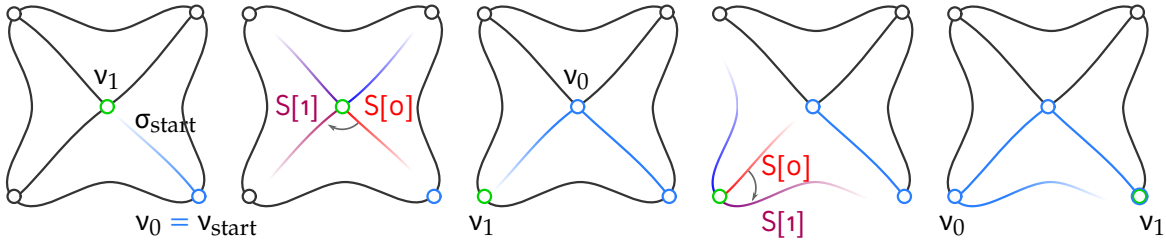


Figure 4.8: Illustration of the cycles detection algorithm (Code snippet 4.9)

```
function DetectCycle(v_start, σ_start)
    v_0 ⟵ v_start                    # initialize the first node
    v_1 ⟵ null                       # initialize the second node
    σ ⟵ σ_start                      # initialize the segment
    ω[0] ⟵ σ                         # initialize the cycle with the first segment
    while v_1 ≠ v_start do
        v_1 ⟵ AdjacentNode(v_0,σ)    # get the other endnode of σ
        segments ⟵ GetSegments(v_1)  # get oriented segments starting at v_1
        frame ⟵ LocalFrame(v_1)      # construct the local frame at v_1
        S ⟵ Sort(segments, frame)    # sort the adjacent segments by polar angle
        σ ⟵ S[1]                     # retrieve the next segment in the cycle
        ω[end+1] ⟵ σ                 # add the segment to the end of the current cycle
        v_0 ⟵ v_1                    # iterate
    end while
    return ω
end function
```

Algorithm 4.9: Cycles detection algorithm from Section 4.4.1. See also Fig. 4.8.

4.4.1 Detection of network cycles

The detection of cycles in a general curve network is a complex and ambiguous problem, often without a unique solution. In order to overcome this problem, methods for surfacing sketched networks adopt a variety of heuristics to mimic the human perception [Zhu+13].

In our specific setting, there is no ambiguity, and the problem has a unique and well-defined solution (Fig. 4.10). This is a consequence of two facts. First, the curve network has a structure of a cell complex. And second, the surface normals are available as the input: they locally determine surface orientation, which enables us to sort segments around their common node. This sorting is the crucial part of our cycles detection algorithm.

Since the discrete network inherits the topology of the smooth curve network, the solution can be found using the available input data: the positions $\mathbf{x}_i$, the normals

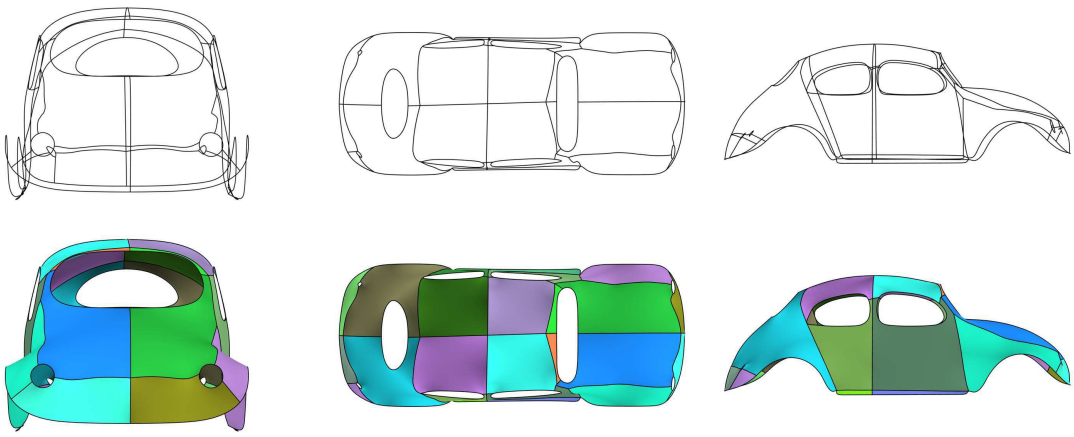


Figure 4.10: Detected cycles in beetle

$\mathbf{N}_i$, and the edges $e_{ij} \in \mathcal{E}_\mathcal{N}$. Our detection of cycles is inspired by face extraction in edge-based manifold data structures (e.g. halfedge). The algorithm is summarized in Code snippet 4.9 and illustrated in Fig. 4.8.

For each node v in the network, we sort the adjacent segments around v. This is done by projecting the segments to the tangent plane at v. If $\mathbf{x}$ is the position of v and $\mathbf{N}$ is the normal at v, the projection of a point $\mathbf{p}$ to the tangent plane at $\mathbf{x}$ is denoted by $\mathbf{p}'$ and computed as

$$\mathbf{p}' = \mathbf{p} - \langle \mathbf{p} - \mathbf{x}, \mathbf{N} \rangle \mathbf{N}. \tag{4.10}$$

In practice, we compute the projection $\mathbf{x}_i'$ of the point $\mathbf{x}_i$ that is closest to $\mathbf{x}$ on each adjacent segment. All projected points $\mathbf{x}_i'$ are then cyclically sorted in the tangent plane using polar coordinates. This gives an ordering of the segments around v.

As soon as we have the orderings for each node, we extract the cycles as follows (cf. Code snippet 4.9). Starting from any (Node,Segment) pair, we trace the unique cycle by choosing the next node as the other endpoint of the current segment. The next segment is then picked from the ordered set. To handle surfaces with boundary, we require the user to tag all boundary segments.

In the next section we describe how the detected cycles are used to obtain the topology of the mesh.

4.4.2 Tessellation of cycles

Cycles detected using the algorithm from Section 4.4.1 serve us to define the boundary curves of surface patches. In this section we compute the tessellations of individual patches – when joined together, these tessellations give us the topology (the faces $\mathcal{F}$) of the entire mesh $\mathcal{T}$.

A cycle ω is defined by a sequence of n segments, which form a closed discrete curve $C = \{\mathbf{x}_i\}_{i \in I}$. The discrete curve C is a boundary of an n-sided patch on the mesh $\mathcal{T}$. The barycenter of C is computed as

$$\mathbf{x}_C = \frac{1}{|I|} \sum_{i \in I} \mathbf{x}_i.$$

The average normal along C is computed as

$$\mathbf{N}_C = \frac{\sum_{i \in I} \mathbf{N}_i}{\left\| \sum_{i \in I} \mathbf{N}_i \right\|}.$$

If the sum $\sum_{i \in I} \mathbf{N}_i$ vanishes, the average normal is instead defined as the normal of the plane closest to the curve C in the least-squares sense.

To compute the tessellation, we map the discrete curve C to the plane $\mathbb{R}^2$ using an embedding

$$\pi : C \subset \mathbb{R}^3 \to \mathbb{R}^2 : \mathbf{x}_i \mapsto \pi(\mathbf{x}_i),$$

where π is required to be injective by definition. We define π as the projection to a plane given by $\mathbf{x}_C$ and $\mathbf{N}_C$. The projection is computed analogically to Eq. (4.10).

While such a projection is not injective in general, we restrict ourselves to input data, for which the projection is always injective (Section 4.3). This restriction is a trade-off between generality and precision. We have found in our experiments that the planar projection leads to a smaller conformal distortion between the planar triangulation and the mesh triangulation, in comparison with other planar embeddings of π that we have tested. Small conformal distortion is crucial for correct computation of weights for the discrete Laplacian, which is used to infer the geometry of the mesh in Section 4.5.

Besides the planar projection, we have also experimented with other embeddings π: isometric map to a circle, and piecewise-isometric map to a polygon. Fig. 4.11 shows the comparison of all three approaches. The planar projection yields valid results even for the bunny network on lilium (bottom right) while the other two approaches fail to capture the high variation of curvature in the network. For the network on sphere (top left), all three methods produce a valid triangulation. However, the surface computed with planar projection has the highest quality among the three.

Next, we compute the tessellation of the planar curve $\pi(C)$. Since the curve C is closed and the map π is injective, the planar embedding $\pi(C)$ is a Jordan curve, meaning it has a well-defined interior and exterior. To get a tessellation $\mathcal{T}_\omega$ of C, we triangulate the interior region using *constrained Delaunay triangulation* (CDT) with *Steiner points* [CDS12, ch. 2]. CDT is similar to Delaunay triangulation, except every edge in C is present as a single edge in $\mathcal{T}_\omega$. Steiner points are additional vertices not present in C, but included in the computation of $\mathcal{T}_\omega$. They are introduced in

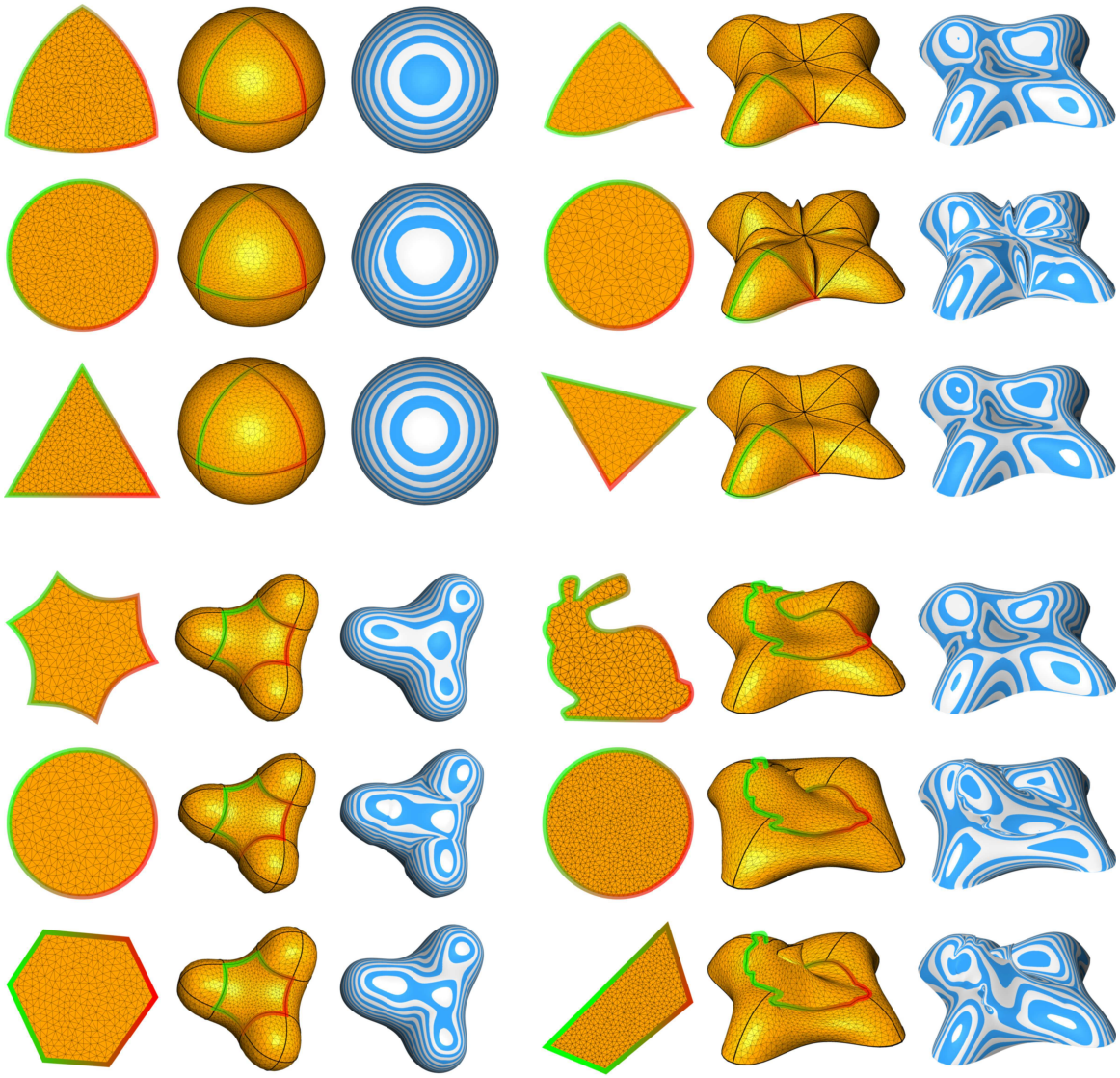


Figure 4.11: Various approaches for computing the initial tessellation. For each of the four curve networks, the planar tessellation of every cycle is computed via (*top to bottom*) projection to a plane, mapping to a circle, mapping to a closed polygon with preserved lengths. Left to right for each of the four datasets, we show the planar tessellation of one cycle, the same cycle on the final surface, and the isophotes of the final surface.

order to satisfy the constraints on minimum angle $\alpha_{\min}$ and maximum triangle area $A_{\max}$ within the tessellation. To get a high-quality triangulation with triangles of approximately the same size (Fig. 4.7), we use $\alpha_{\min} = 20^\circ$ and $A_{\max} = \frac{1}{2}h^2$, where h is the average edge length in C. The triangulation is computed using Shewchuk's Triangle [She96].

An issue that is not often discussed in the literature is the metric of this parametrization, and its impact on the resulting surface. To get the best results with Laplacian-based methods, the metric of the planar tessellation should be distorted as-least-as-possible with respect to the surface metric. In practice, it is hard to have a full isometry between the two – all the more for reconstruction problems where the surface is unknown *a priori*. Some authors address this problem by iterative refinement of the surface [SK01; Pan+15].

Finally, we remark that computing the tessellation directly in three dimensions instead of in-plane could provide interesting results. An example is the dynamic programming method of Zou et al. [ZJC13] which is also used by Zhuang et al. [Zhu+13] and Pan et al. [Pan+15]. Naturally, this comes at a price of higher computational complexity. In future, we plan to test this kind of tessellation in order to overcome some of the limitations of the planar projection.

4.5 Computing the geometry

The previous section described the computation of topology for the mesh $\mathcal{T}$. Network cycles are first detected, then individually triangulated in plane, and finally combined to get the sets of vertices, edges, and faces of the mesh $\mathcal{T}$.

In this section, we describe how we compute the geometry of $\mathcal{T}$ so that the embedding of the mesh in three dimensions satisfies the interpolation constraints (ii-iii) from Section 4.3. The result is a visually smooth surface.

We formulate the computation of geometry as an optimization problem. In order to match the input positions and input normals, we minimize the following mean curvature energy functional (cf. Eq. (2.11) and Eq. (4.8))

$$\boxed{\mathrm{E}_{\mathrm{mean}}(\mathcal{S}) = \iint_{\mathcal{S}} \|\Delta_{\mathcal{S}}\mathbf{x} - \mathbf{H}\|^2 \,\mathrm{d}A} \tag{4.11}$$

where $\Delta_{\mathcal{S}}$ is the Laplace-Beltrami operator, $\mathbf{x}$ is the coordinate function, and $\mathbf{H}$ is the mean curvature normal.

Since we represent the surface $\mathcal{S}$ as a triangle mesh, this energy is discretized over the vertices of the mesh $\mathcal{T}$. The discretization is described in Section 4.5.3. Section 4.5.2 describes our estimation of the mean curvature normal $\mathbf{H}$, which is needed to evaluate the energy (4.11).

Our curvature estimation needs an initial guess for the geometry of the mesh (vertex positions and normals). The following section describes how the initial geometry is obtained.

4.5.1 Initial mesh and propagation of normals

We now proceed to the computation of the base geometry of the mesh $\mathcal{T}$. These are the *initial positions* $\mathbf{v}_k^*$ and the *propagated normals* $\mathbf{N}_k^*$.

Two types of vertices are distinguished for the mesh $\mathcal{T}$, which was computed in Section 4.4.2. The *constrained* vertices, denoted by $\mathcal{V}_c = \{v_1, \ldots, v_c\}$, correspond to the original vertices, lying along the curve network. These are the vertices that are at boundaries of cycles. The remaining vertices are called *free* and denoted by $\mathcal{V}_f = \{v_{c+1}, \ldots, v_{c+f}\}$. The set of all mesh vertices is the union $\mathcal{V} = \mathcal{V}_c \cup \mathcal{V}_f$, with the total number of vertices being $|\mathcal{V}| = c + f = n$. See also Section 2.8.

To compute the base geometry, we solve two biharmonic systems with hard boundary constraints along the discrete network. If we suppose that the indices of propagated mesh vertices $\mathbf{v}_i^*$ match the indices of the input curve vertices $\mathbf{x}_i$, then we want to solve

$$\Delta_S^2 \mathbf{v}_k^* = 0 \quad \text{s.t.} \quad \mathbf{v}_i^* = \mathbf{x}_i \quad \text{if} \quad v_i \in \mathcal{V}_c, \tag{4.12a}$$

$$\Delta_S^2 \mathbf{N}_k^* = 0 \quad \text{s.t.} \quad \mathbf{N}_i^* = \mathbf{N}_i \quad \text{if} \quad v_i \in \mathcal{V}_c. \tag{4.12b}$$

The weights for the discrete Laplacian Δ_S are obtained from the planar triangulation (Section 4.4.2) and assembled into the Laplacian matrix L. See Eq. (2.40) on p. 46. The above two biharmonic systems with hard constraints are then written in matrix form as

$$\begin{bmatrix} \mathrm{L}_f^2 & \\ \mathrm{I_c} & 0 \end{bmatrix} \begin{bmatrix} \mathrm{X}_c \\ \mathrm{X}_f \end{bmatrix} = \begin{bmatrix} 0 \\ \mathrm{X}_c \end{bmatrix}, \tag{4.13}$$

where X_f is the $f \times 3$ matrix with free positions/normals; X_c is the $c \times 3$ matrix with constrained positions/normals; L_f is the $f \times n$ Laplacian matrix of the free vertices. Both systems are resolved by eliminating the known constrained vertices X_c and solving for the unknown free vertices X_f.

After the solution has been computed, the propagated normals $\mathbf{N}_k^*$ are normalized. On rare occasions, the norm of $\mathbf{N}_k^*$ might vanish.[1] In this case, we compute the normal $\mathbf{N}_k^*$ by averaging the non-zero propagated normals in the neighborhood of $\mathbf{v}_k^*$. The neighborhood is chosen as the smallest n-neighborhood of $\mathbf{v}_k^*$ for which the average is not zero.

[1] Pierre Alliez brought this issue to my attention.

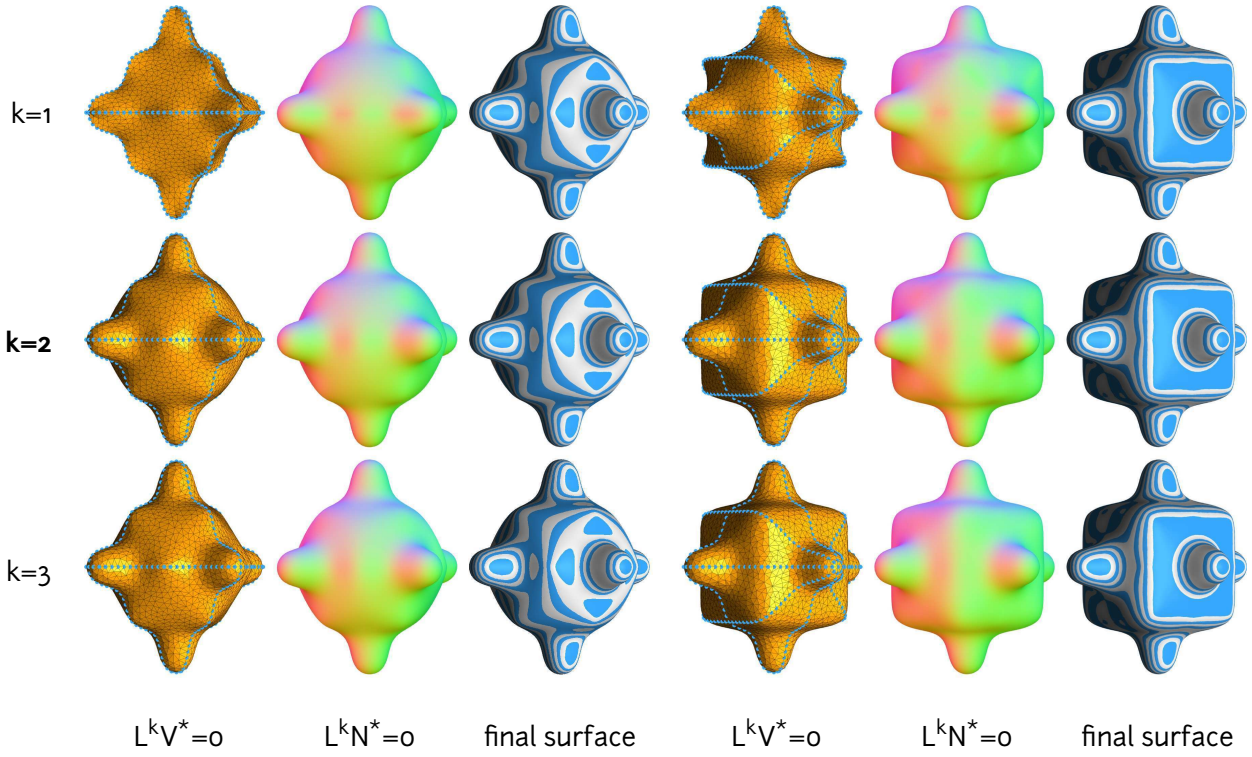


Figure 4.12: Comparison of three base geometries on two different bumpy cube networks. We use k=2 in our framework.

The choice of the base geometry is important for further steps of the framework, notably the estimation of the mean curvature that we describe in the next section. Fig. 4.12 shows that solving the biharmonic systems in Eqs. (4.12a) and (4.12b) to get $\mathbf{v}^*$ and $\mathbf{N}^*$ is a trade-off: in general, we obtain better results with the biharmonic base geometry than with the harmonic base geometry $LX = 0$; on the other hand, there is little difference between the final surfaces using biharmonic and triharmonic base geometries $L^3X = 0$. This is caused by the fact that only the vertices along the input curves are fixed when solving these systems.

In the next section, we use the computed initial geometry to estimate scalar mean curvature for the whole mesh.

4.5.2 Mean curvature estimation

In this section, we describe our approach for the estimation of the discrete mean curvature for the mesh $\mathcal{T}$ (see the penultimate step in Fig. 4.6). This estimation is the most important part of the computation of geometry. It provides the missing piece of information needed for minimization of the energy (4.11).

Our idea for estimating the mean curvature is to combine the initial positions $\mathbf{v}_k^*$ and propagated normals $\mathbf{N}_k^*$ (Section 4.5.1) into a compact curvature measure. In order to include the propagated normals into the estimation, we do not use the cotangent formula from Eq. (2.38) on p. 45, which is evaluated using positions only.

To get an alternative discretization of the mean curvature, we take a step back and look at the *integral* definition of the mean curvature normal $\mathbf{H}$.

To this end, let us consider a regular surface $\mathcal{S}$ and a point $\mathbf{x} \in \mathcal{S}$ on this surface. Next, let A be a neighborhood around $\mathbf{x}$, such that A is a topological disk (Fig. 4.13 left). The boundary of the neighborhood is a closed curve $\gamma = \partial \mathrm{A}$. Then it holds that

$$\iint_{\mathrm{A}} \mathbf{H}\, \mathrm{d}A = \iint_{\mathrm{A}} -2\mathrm{H}\mathbf{N}\, \mathrm{d}A = \oint_{\gamma} \mathbf{t} \times \mathbf{N}\, \mathrm{d}s = \oint_{\gamma} \mathbf{B}\, \mathrm{d}s. \tag{4.14}$$

Here, $\{\mathbf{t}, \mathbf{N}, \mathbf{B}\}$ is the Darboux frame of γ with respect to $\mathcal{S}$, with unit tangent $\mathbf{t}$, unit normal $\mathbf{N}$, and unit conormal $\mathbf{B}$. See Definition 2.11 on p. 27. Note that Eq. (4.14) can be proved using divergence theorem from Eqs. (2.30) and (2.31) [Sul08].

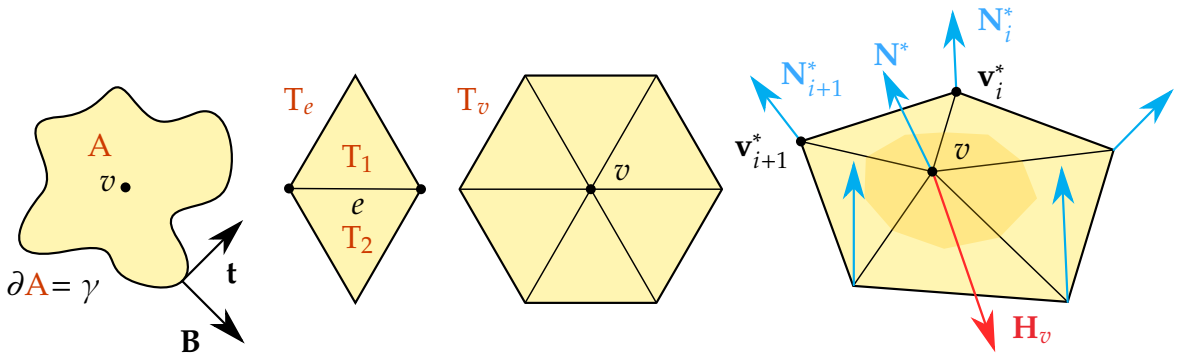


Figure 4.13: Integral definition of the mean curvature normal

We now describe how the Eq. (4.14) is discretized on a triangle mesh. For an interior edge $e = (v_i, v_j) \in \mathcal{E}$ in the triangle mesh $\mathcal{T}$, we denote by $\mathrm{T}_e = \mathrm{T}_1 \cup \mathrm{T}_2$ the union of the two triangles adjacent to e (Fig. 4.13 *middle left*). Fixing a region A_e around e, such that $e \subset \mathrm{A}_e \subset \mathrm{T}_e$, the *edge mean curvature* is defined as

$$\mathbf{H}_e = \iint_{\mathrm{A}_e} \mathbf{H}\, \mathrm{d}A = \oint_{\partial \mathrm{A}_e} \mathbf{t} \times \mathbf{N}\, \mathrm{d}s = \mathbf{e} \times \mathbf{N}_1 - \mathbf{e} \times \mathbf{N}_2 = \mathbf{e}^{\perp_1} - \mathbf{e}^{\perp_2} \tag{4.15}$$

where $\mathbf{e} = \mathbf{v}_j - \mathbf{v}_i$ is the edge vector and $\mathbf{N}_k$ are the unit outward normal of the triangle T_k. The symbol $^{\perp_k}$ denotes the rotation by 90 degrees counterclockwise in the plane of the triangle T_k.

For a mesh vertex v, the mean curvature $\mathbf{H}_v$ can be evaluated by summing the contributions of all adjacent edges. Denote by $\mathrm{T}_v$the set of all triangles adjacent

to the vertex v (Fig. 4.13 *middle right*):

$$\mathrm{T}_v = \bigcup_{v \in e} \mathrm{T}_e = \bigcup_{v \in \mathrm{T}} \mathrm{T}.$$

Then we have

$$2\mathbf{H}_v = \sum_{v \in e} \mathbf{H}_e = \iint_{\mathrm{T}_v} \mathbf{H}\, \mathrm{d}A = \oint_{\partial \mathrm{T}_v} \mathbf{t} \times \mathbf{N}\, \mathrm{d}s. \tag{4.16}$$

Using Eq. (4.15), $\mathbf{H}_v$ this is discretized as

$$\mathbf{H}_v = \tfrac{1}{2} \sum_{i=0}^{n-1} (\mathbf{v}_{i+1} - \mathbf{v}_i) \times \mathbf{N}_{\mathrm{T}}. \tag{4.17}$$

Here we suppose that the adjacent vertices $\{v_i\}_{i=0}^{n-1}$ are sorted radially around v and their indices are taken modulo n. The $\mathbf{N}_{\mathrm{T}}$ is the unit normal of the triangle $\mathrm{T} = (v, v_i, v_{i+1})$.

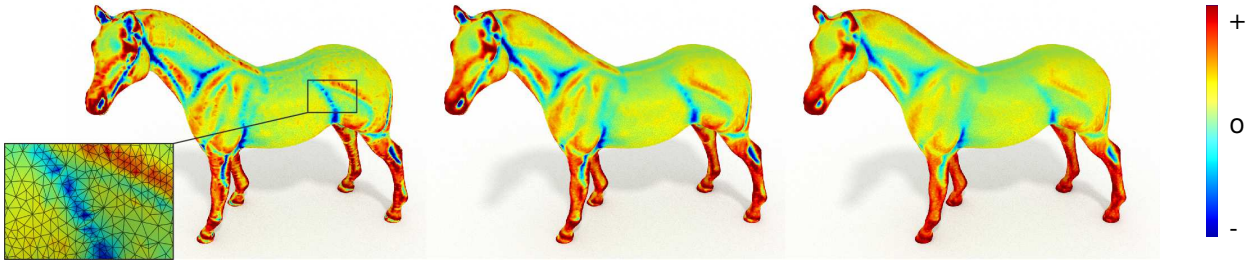
This integral is usually evaluated using the triangle normals defined by the mesh vertices $\mathbf{v}$, which leads to the cotangent formula (Eq. (2.38), p. 45). We evaluate this integral using the propagated normals $\mathbf{N}^*$ rather than the triangle normals (Fig. 4.13 right). For a vertex v, we compute the mean curvature vector by summing the contributions for all oriented opposite edges:

$$\mathbf{H}_v = \frac{1}{2\mathrm{A}_v} \sum_{i=0}^{n-1} \left(\mathbf{v}^*_{i+1} - \mathbf{v}^*_i\right) \times \frac{\mathbf{N}^* + \mathbf{N}^*_i + \mathbf{N}^*_{i+1}}{\left\| \mathbf{N}^* + \mathbf{N}^*_i + \mathbf{N}^*_{i+1} \right\|}. \tag{4.18}$$

Here, $\mathbf{N}^*$ denotes the propagated normal at v, n is the valence of v, A_v is the Voronoi area of v (Fig. 2.10), $\mathbf{v}^*_i$ are the positions of cyclically sorted neighbors of v (indices taken modulo n) and $\mathbf{N}^*_i$ are the corresponding propagated normals. Analogously to the cotan mean curvature, we scale the $\mathbf{H}_v$ by the inverse of the Voronoi area to get a scale-independent mesh formulation.

Formula (4.18) for computing the mean curvature is a key part to our method. Its originality lies in blending together the positional information (the initial vertices $\mathbf{v}^*$) with the additional normal information (the propagated normals $\mathbf{N}^*$) not directly inferred from the positions. In contrast, the usual discrete mean curvature formulations – such as the cotan formula from the Eq. (4.17) – rely solely on vertex positions.

We illustrate this originality in Fig. 4.14, where we show three discrete mean curvature measures: one based on the cotan formula (2.36), and two based on our formula (4.18) evaluated with the same geometry with different normal fields. It can further be observed in Fig. 4.14 that our mean curvature measure behaves at least as well as the standard measures even with a low quality mesh. Since we apply the



a) cotan formula **b)** (4.18) original **c)** (4.18) smoothed

Figure 4.14: Mean curvature of the irregular horse mesh computed using the cotan formula and the 3-averaging formula (4.18) with original normals and with smoothed normals.

formula (4.18) to good quality triangulations resulting from a planar Delaunay tessellation (Section 4.4.2), we do not investigate here the incorporation of propagated normals into more robust discrete mean curvature measures [Gri+06].

The next section describes how the estimated curvature is used to compute the final smooth geometry of the mesh.

4.5.3 Optimization

To summarize, our method so far consisted of the meshing step (Section 4.4), followed by the computation of an initial geometry (Section 4.5.1) and estimation of mean curvature (Section 4.5.2). The purpose of all of these steps is to obtain the necessary pieces of information in order to minimize the energy defined in Eq. (4.11).

Finally, we arrive to the point where the optimization can be discretized and resolved. On the triangle mesh $\mathcal{T}$, we discretize the energy (4.11) as

$$\mathrm{E}_{\mathrm{mean}}(\mathcal{V}) = \sum_{v \in \mathcal{V}} \left\| \Delta_{\mathcal{S}} \mathbf{v} - \mathbf{H}_v^* \right\|^2 . \tag{4.19}$$

The mean curvature normal $\mathbf{H}_v^*$ is defined using the propagated normal $\mathbf{N}^*$ and the scalar mean curvature estimated via the three-normal averaging formula (4.18):

$$\mathbf{H}_v^* = -\left\|\mathbf{H}_v\right\| \mathbf{N}^* .$$

In Section 4.5.1, the mesh vertices $\mathcal{V}$ were split into the constrained vertices $\mathcal{V}_c = \{v_1, \ldots, v_c\}$ and the free vertices $\mathcal{V}_f = \{v_{c+1}, \ldots, v_{c+f}\}$. To interpolate the input curves, the discretized energy from Eq. (4.19) is minimized subject to hard positional constraints along the curves:

$$\min \mathrm{E}_{\mathrm{mean}}(\mathcal{V}) \quad \text{s.t.} \quad \mathbf{v}_i = \mathbf{v}_i^* \text{ for all } v \in \mathcal{V}_c . \tag{4.20}$$

Denoting by H the matrix of the mean curvature normals $\mathbf{H}_v^*$, the energy $\mathrm{E}_{\mathrm{mean}}$ is written in matrix form as

$$\mathrm{E}_{\mathrm{mean}}(\mathcal{V}) = \|\mathrm{LV} - \mathrm{H}\|^2 \tag{4.21}$$

and minimized by solving the normal equations

$$\mathrm{L}^\intercal \mathrm{L}\,\mathrm{V} = \mathrm{L}^\intercal \mathrm{H}. \tag{4.22}$$

The positional constrains are enforced by eliminating the constrained vertices from the system. Splitting the matrix of the system $\mathrm{L}^\intercal \mathrm{L} = \begin{bmatrix}\mathrm{M}_c & \mathrm{M}_f\end{bmatrix}$ into blocks corresponding to (known) constrained vertices V_c and (unknown) free vertices V_f, the system (4.22) is equivalent to

$$\begin{bmatrix}\mathrm{M}_c & \mathrm{M}_f\end{bmatrix}\begin{bmatrix}\mathrm{V}_c \\ \mathrm{V}_f\end{bmatrix} = \mathrm{L}^\intercal \mathrm{H} \quad \Rightarrow \quad \mathrm{M}_f \mathrm{V}_f = \mathrm{L}^\intercal \mathrm{H} - \mathrm{M}_c \mathrm{V}_c, \tag{4.23}$$

cf. Eq. (2.54) on p. 49. The solution of (4.23) gives the final smooth geometry $\mathbf{v}$ of the mesh $\mathcal{T}$ (see the overview in Fig. 4.6, p. 112).

4.5.4 Hard constraints vs. soft constraints

If the input positions are noisy, a useful modification of the problem (4.20) is to incorporate the positions as soft constraints. To this end, the constrained vertices $\mathcal{V}_c$ are further partitioned into hard constraints $\mathcal{V}_h = \{v_1, \ldots, v_h\}$ and soft constraints $\mathcal{V}_s = \{v_{h+1}, \ldots, v_{h+s=c}\}$ with $h + s + f = n$ being the total number of vertices. The soft positions are added as an energy term:

$$\mathrm{E}_{\mathrm{soft}}(\mathcal{V}) = \mathrm{E}_{\mathrm{mean}}(\mathcal{V}) + w\mathrm{E}_{\mathrm{interp}}(\mathcal{V}) \quad \text{with} \quad \mathrm{E}_{\mathrm{interp}}(\mathcal{V}) = \sum_{v_s \in \mathcal{V}_s} \left\|\mathbf{v}_s - \mathbf{v}_s^*\right\|^2. \tag{4.24}$$

The matrix form of the modified functional $\mathrm{E}_{\mathrm{soft}}$ is

$$\mathrm{E}_{\mathrm{soft}}(\mathcal{V}) = \|\mathrm{LV} - \mathrm{H}\|^2 + w\left\|\mathrm{V}_s - \mathrm{V}_s^*\right\|^2, \tag{4.25}$$

and the corresponding normal equations are

$$\mathrm{A}^\intercal \mathrm{W}\mathrm{A}\mathrm{V} = \mathrm{A}^\intercal \mathrm{W}\,\mathrm{B}, \tag{4.26}$$

where

$$\mathrm{V} = \begin{bmatrix}\mathrm{V}_h \\ \mathrm{V}_s \\ \mathrm{V}_f\end{bmatrix}, \quad \mathrm{A} = \begin{bmatrix}\mathrm{L} \\ 0_{s\times h} \quad \mathrm{I}_s \quad 0_{s\times f}\end{bmatrix}, \quad \mathrm{B} = \begin{bmatrix}\mathrm{H} \\ \mathrm{V}_s^*\end{bmatrix}, \quad \mathrm{W} = \mathrm{diag}(\underbrace{1, \ldots, 1}_{n \text{ times}}, \underbrace{w, \ldots, w}_{s \text{ times}}).$$

The system (4.26) is solved by moving the hard constraints to the right-hand side using the procedure outlined at the end of the previous section. Fig. 4.15 illustrates the robustness of this approach; in this test, we have artificially perturbed the positions and normal directions along the input network. Our method with soft constraints produces stable output, while still preserving the shape fidelity.

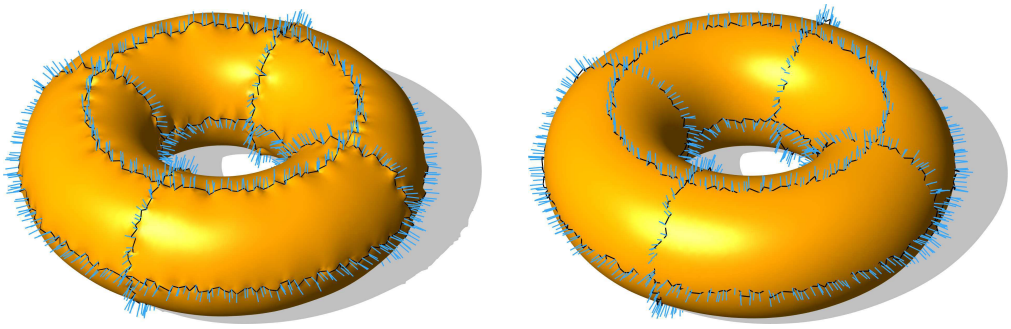


Figure 4.15: If the input data are noisy, soft constraints (*right*) produce better results than hard constraints (*left*) while still maintaining high overall shape fidelity. In this example, 5% of noise to both positions and normals.

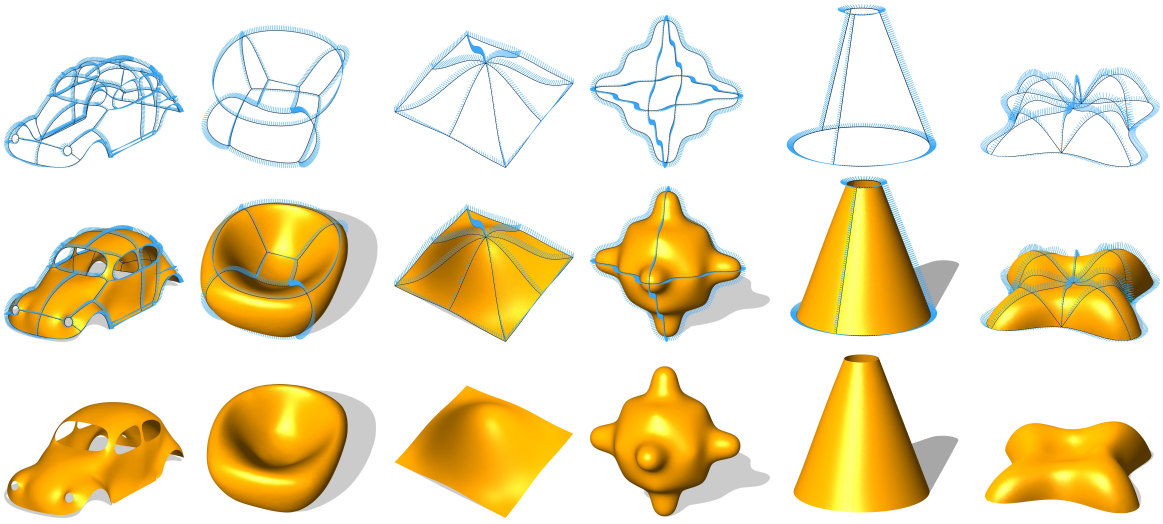


Figure 4.16: Smooth surfaces reconstructed using our method

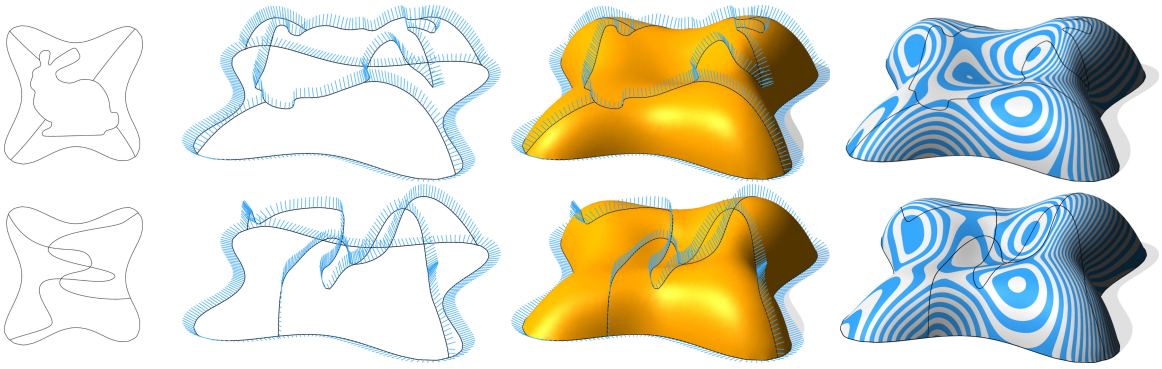


Figure 4.17: In terms of geometry, we place no constraints on the input. Our method can handle even these challenging networks with winding curves, all while preserving the global smoothness across patches.

4.6 Evaluation

In this section, we evaluate the method proposed in Sections 4.3 to 4.5. See also Fig. 4.6 on p. 112, which provides an overview of our surfacing method.

We mostly use synthetic data for evaluation. Fig. 4.19 also shows surfacing results for lilium networks from Fig. 3.14 in Chapter 3. Most surfacing results for real-world acquired data are presented in Chapter 5.

In order to obtain the synthetic networks (with surface normals) for testing, we have used the technique described in the beginning of Section 3.7. A curve network was traced on a known surface in Blender and the data (vertices and normals) were exported with help of a python script.

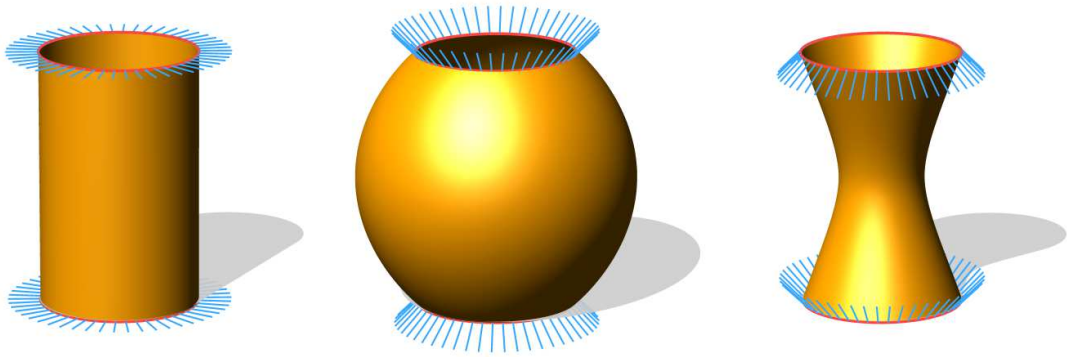


Figure 4.18: Influence of input normals. The red circles are given as input together with three types of normal constraints (blue). Propagation of the input normals over the surface guides the computation of three different shapes.

4.6.1 Normal control

In Fig. 4.18 we demonstrate the shape control provided by the input normals. The fixed vertex positions are sampled along two parallel circles from the same cylinder while prescribing three different sets of normal vectors along the circles.

With the original normals (Fig. 4.18 left), the cylindrical surface is nicely reconstructed. Using the two other sets of rotated normals (Fig. 4.18 middle, right) results in the barrel and bottleneck surfaces, as expected intuitively.

Our method works well even for challenging input data, such as the networks with large normal curvature variations and high valence curve intersections in Fig. 4.16, or networks with large curvature variation in the tangent plane in Fig. 4.17.

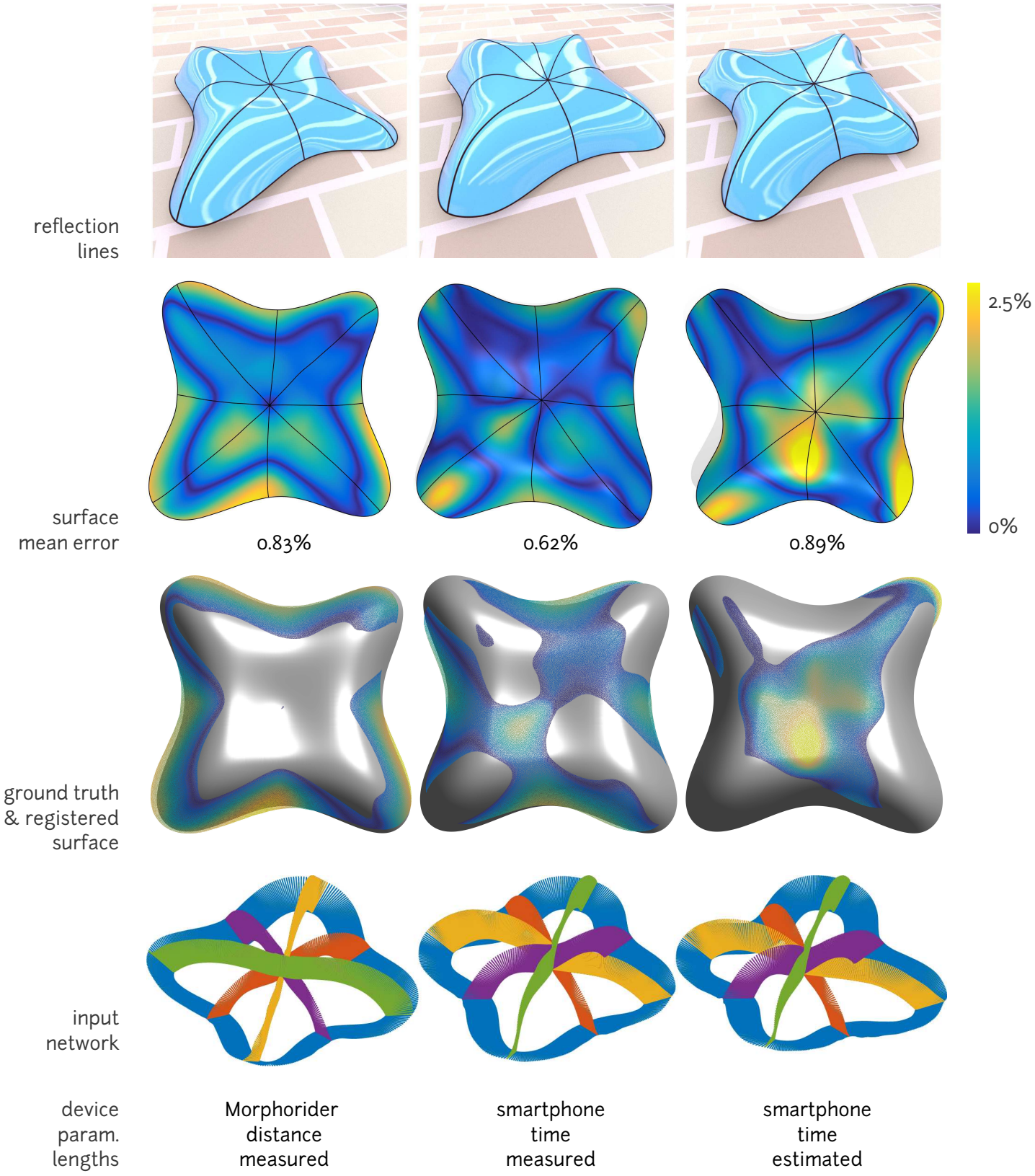


Figure 4.19: Surfacing error for lilium networks from Fig. 3.14. For filtering of orientations, we have used the weights $\lambda = 1, \mu = 100$. All lengths are relative to the diameter (length of AABB diagonal) of the ground truth surface.

4.6.2 Convergence & error measurements

We demonstrate in this section that for decreasing sampling distance, the surfaces computed using our method converge to a limit surface. Given an input network Γ and an initial sampling distance d, we have resampled Γ using the method from Appendix C by setting the sampling distance to $d/2^i$ for $i = 1, \ldots, 5$.

We then applied our surfacing method to the resampled networks Γ_i. This process results in a sequence of meshes $\mathcal{T}_i$. Since there is no analytic form of $\lim_{i\to\infty} \mathcal{T}_i$, we illustrate the convergence in Fig. 4.20 by plotting the Hausdorff distance between the consecutive meshes $\mathcal{T}_i, \mathcal{T}_{i+1}$. The three curves correspond to sphere (Fig. 4.22), the beetle and bumpy cube (Fig. 4.16) networks. The Hausdorff distance was computed using Meshlab [CRS98].

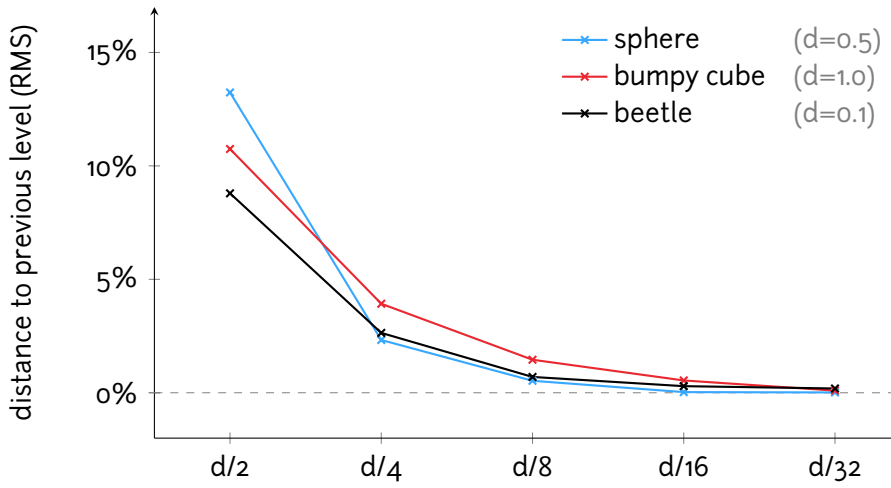
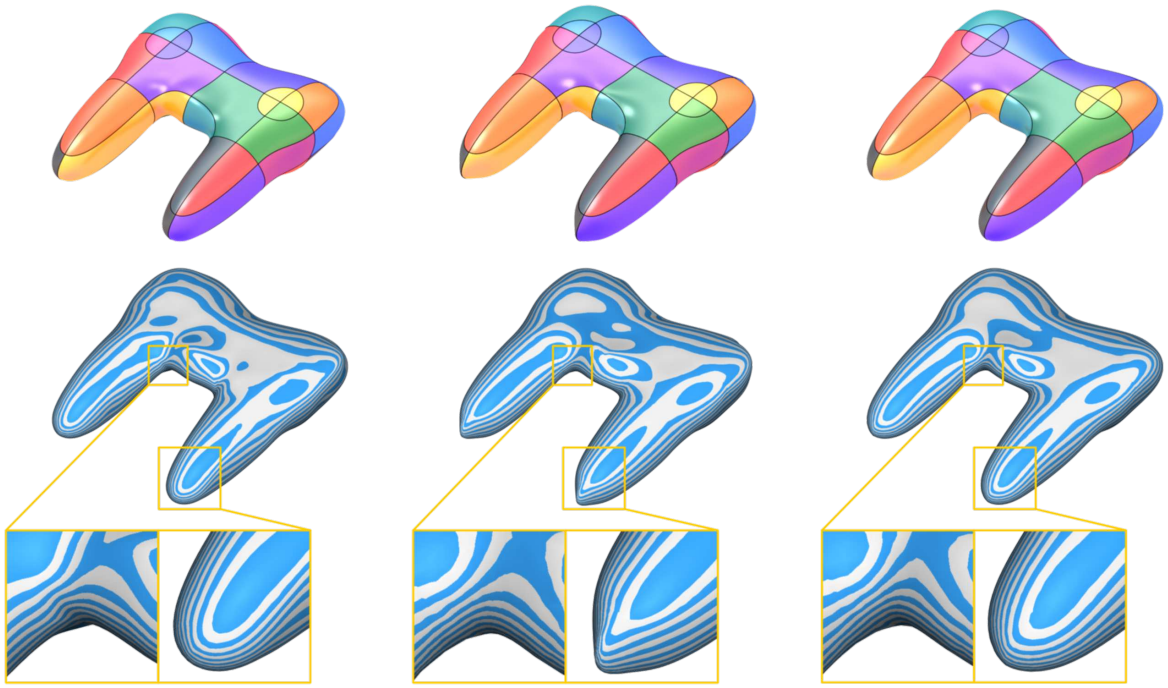


Figure 4.20: The meshes computed using our method converge towards a limit surface upon refinement of the sampling distance. For each sampling level, we plot Hausdorff distance to the previous level relative to the sampling distance d.

Fig. 4.19 shows results of our surfacing algorithm applied to the networks on lilium reconstructed for acquired data in Fig. 3.14. It compares the three acquisition setups described in Sections 3.7.3 and 3.7.4.

Reflection lines in the top row show that the surfaces are globally smooth. We also visualize the error of reconstruction with respect to a ground truth. The smallest mean error is obtained using smartphone orientations with measured distances, and all mean errors are less than 1%.

Chapter 5 includes a detailed analysis of surface reconstruction error with respect to ground truth.



[Pan+15] biharmonic + normals (4.27) our method

Figure 4.21: On sketched networks, the results of our algorithm are similar to the method of Pan et al. [Pan+15], which assumes that the input curves capture the flow field of the underlying surface. The isophotes on our surface vary more smoothly, suggesting higher order of continuity. All three meshes have the same normals along the curve network.

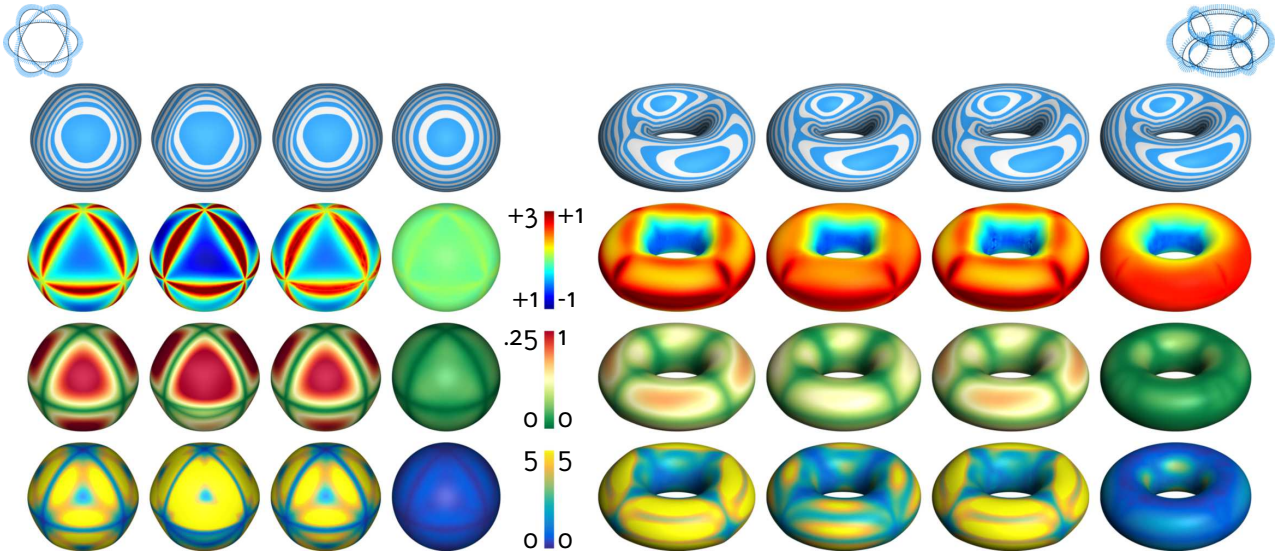


Figure 4.22: Various quality measures on the unit sphere and torus with radii 4 and 2. Top to bottom: isophotes, mean curvature, distance from ground truth, difference between propagated and computed normals (in degrees). Left to right: biharmonic $\Delta^2 = \mathbf{0}$; triharmonic $\Delta^3 = \mathbf{0}$; biharmonic with prescribed normals – solution of (4.27); our algorithm.

4.6.3 Comparison with Laplacian methods

The method of Botsch and Kobbelt [BK04], which we introduced in Section 4.1.1, uses k$^{\text{th}}$-order Laplacian with boundary conditions up to C^{k-1} to compute a smooth surface:

$$\Delta_{\mathcal{S}}^{k}\,\mathbf{v}\,(u,v) = \mathbf{0} \qquad (u,v) \in \Omega\backslash\partial\Omega, \tag{4.27a}$$

$$\Delta_{\mathcal{S}}^{j}\,\mathbf{v}\,(u,v) = \mathbf{b}_j(u,v) \qquad (u,v) \in \partial\Omega\,,\ j < k. \tag{4.27b}$$

Notice, however, that Botsch and Kobbelt [BK04] did not implement boundary constraints directly; instead, they fixed the positions of (k-1) rings of vertices to prescribe C^{k-1} boundary constraints.

This setting prevents dealing with arbitrary constrained curve networks without knowing the positions of (k-1) rings of vertices. It is therefore impossible to compare our method to theirs; we can however compare our method to the analogous formulation given by the system (4.27).

To this end, we have implemented (4.27) for $k = 2$. To avoid fixing the positions of 1-ring vertices along the constrained curves, we directly cast $\mathbf{b}_j$ as equality constraints of the linear system, and solve it in the least-squares sense.

In Fig. 4.22 we compare three error measures on the well-known geometries of sphere and torus. The three error measures are mean curvature, distance to ground truth and difference between propagated and computed normals. In addition to measuring the error for our surfaces and the surfaces obtained by solving (4.27), we also look at the standard biharmonic and triharmonic surfaces

$$\Delta^{k}\mathbf{v} = \mathbf{0}, \quad k = 2, 3,$$

with positional constraints and without any normal constraints.

Even though the networks on sphere and torus are only toy examples, we believe this comparison provides useful insight into how our method performs in comparison with the standard linear variational methods, with or without the normal constraints. The linear methods exhibit the well-known undesired defects due to the linearization of the energy functionals, and the shapes have high curvatures along the curve network and low curvature everywhere else.

Our solution has much smaller curvature variation. Many authors spend considerable effort in improving the shape of linear methods using e.g. reparametrizations or an iterative approach [SK01; BK04; Jac+10]. We can observe that our shapes succeed in mimicking the desired non-linear shape behaviors simply by combining two linear processing steps: normal propagation and constrained fitting, see the curvature plots in Fig. 4.22. We argue that the normal propagation step, which pre-computes a continuously varying normal field, is the basis for this desirable property.

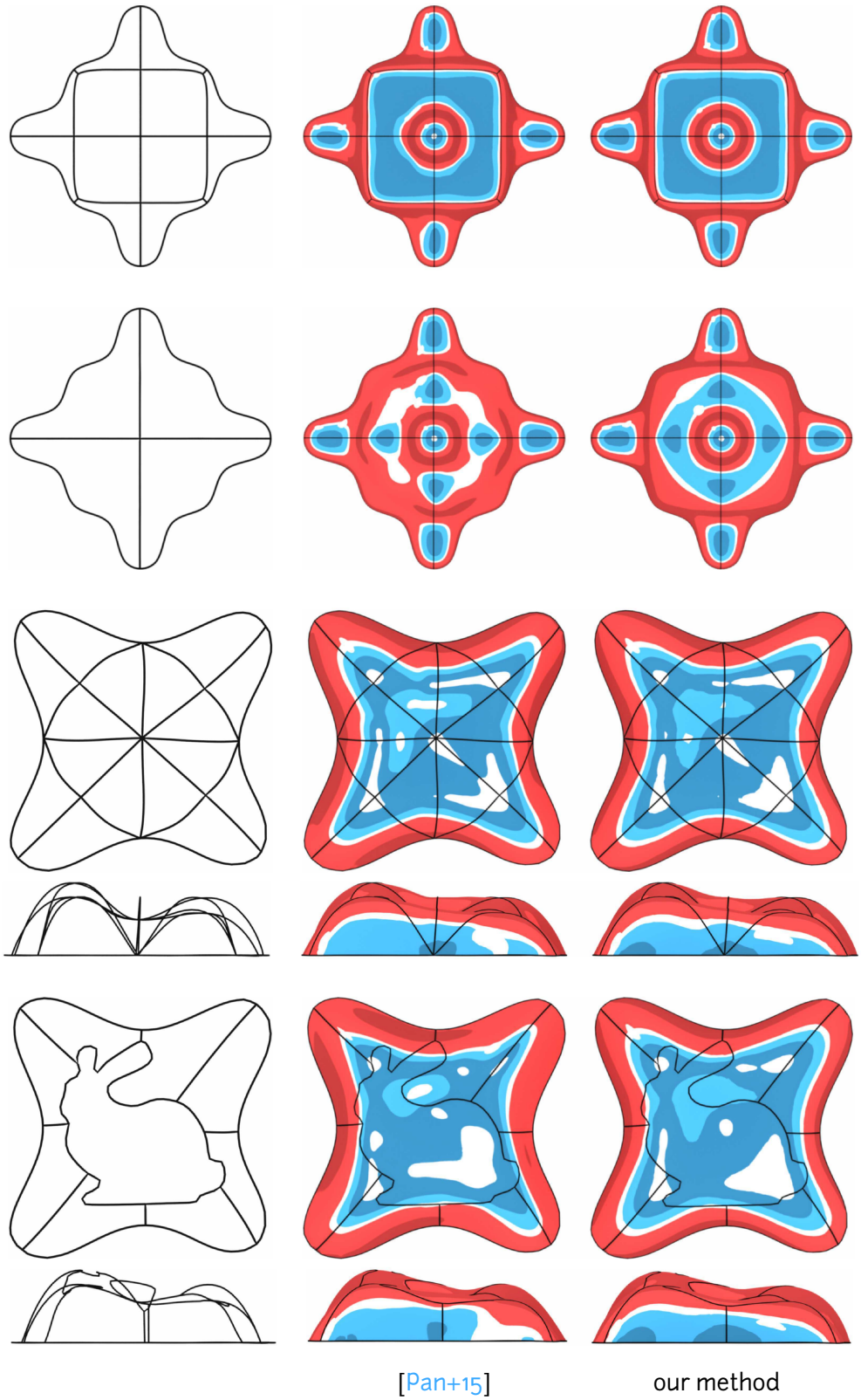


Figure 4.23: Comparison of our method with Pan et al. [Pan+15] on various curve networks. While both methods are capable of treating these highly curved inputs, our method reconstructs the underlying shapes more faithfully and is better at capturing the symmetry in the input networks. This visualization uses the normal mapping from Fig. 4.24.

4.6.4 Comparison with the flow-aligned surfacing

The method of Pan et al. [Pan+15] is considered the state of the art in surfacing of sketched curves. We find it interesting to include a comparison with this method, although the two algorithms do not share the same input since Pan et al. [Pan+15] do not assume the normal input. The comparison with their method on gamepad is shown in Fig. 4.21. The normals along the constrained curves, required by our method, were sampled from the final surface of Pan et al. [Pan+15]. From left to right, we show the surface of [Pan+15], the biharmonic surface with prescribed normals computed via the system (4.27), our surface.

Our method combines the algorithmic simplicity with high fidelity to the reconstructed shape, and at the same time maintains the fairness of the final surface. Interesting details are revealed by looking at the isophotes. On the surface of Pan et al. [Pan+15], the isophotes are of globally poor quality, with undesirable wiggles visible at closer inspection, see the close-up to the concave region. While the middle surface computed by solving (4.27) seems globally smoother than the left surface of [Pan+15], the linearization artifacts are evident (close-up, handle). The colored renderings of the three surfaces look similar at the first glance; notice however the improved quality of specular highlights on our gamepad compared to the left surface of [Pan+15].

In many cases, the results of the two algorithms are hard to tell apart if colored uniformly. To better understand and compare the qualitative properties, we shade the surfaces by mapping the normals to a red-and-blue texture with circular patterns, see Fig. 4.24 [Slo+01]. The shaded circles in the middle of the figure represent a hemisphere of unit normals oriented towards the camera (the mapping is view-dependent). Such normal-mapped texture is useful to examine continuity, regularity and symmetry of shapes.

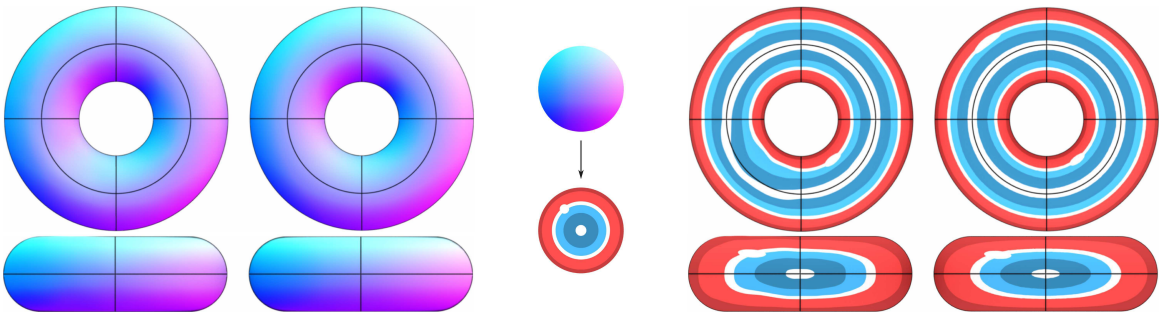


Figure 4.24: The normal mapping used to assess surface quality. In this case, the texture reveals that our torus (*right*) is more symmetric and regular than the one of [Pan+15] (*left*).

Pan et al. [Pan+15] assume the initial curve network is an output of a sketching system, then use this assumption to guide their optimization. Their algorithm yields reasonable surfaces even when applied to more general curve networks *not* coming from sketching tools, see for instance the bunny network in Fig. 4.23. While the resulting surfaces are smooth, the normal mapping from the Fig. 4.24 enables a more detailed analysis of the quality.

Focusing on bumpy cube, the algorithm of Pan et al. [Pan+15] clearly does not maintain the symmetry of the input network in the computed surface. Our algorithm keeps the symmetry: the final textured surfaces are fairly regular while the circles in the texture are nicely preserved. The same behavior can be observed with torus network on Fig. 4.24.

The evidence is less striking for the non-symmetric lilium networks; nevertheless, our surfaces are of globally better quality. Notice the white curve between red and blue regions on each surface. Visually, its behavior suggests our surfaces might be G^2 continuous, while the surfaces of Pan et al. [Pan+15] seem to be only G^1 continuous.

In the case of lilium and bumpy cube, it is interesting to observe how the computed shapes depend on the input curves. For two very different networks initially lying on the same surface (lilium), the results of our method are fairly close to each other; the results of Pan et al. [Pan+15] are more input-dependent.

4.7 Conclusion

This chapter introduced a Laplacian-based surface reconstruction method from curve and normal input. After propagating the input normals smoothly over the surface and computing the corresponding mean curvature vectors, the normal constraints are integrated into the energy functional. Efficiency and robustness are achieved by using a linearized objective functional, such that the global optimization amounts to solving a sparse linear system of equations.

Limitations. A weakness of our method lies in the fact that the cotangent weights for the Laplacian matrix L are inferred from the planar triangulation, computed for each patch individually as explained in Section 4.4.2. Such parametrization is not isometric to the actual surface patch; as a consequence, the weights are not optimal. Nevertheless our examples show that it does not impact the smoothness of our results across surface patches. The framework cannot automatically handle curve networks which are open or consist of more than one component. However, the optimization in Eq. (4.20) is not limited by the topology of the network, only by the availability of the initial mesh.

Future work. Our implementation runs at interactive time rates: the reconstruction takes about 0.1s for a mesh with 10k vertices and 1k constraints. However, when a new curve is added to an existing curve network, the whole surface needs to be recomputed from scratch. Our method could possibly be extended into an algorithm for *iterative* surfacing by adjusting the already computed surface when a new curve is added. Such extension could be valuable in the context of dynamic shape-from-sensors setup (Section 1.4). As stated in Section 4.4.2, the tessellation could be improved by using a more advanced 3D patching algorithm. Another possible modification might be to allow open curve networks on the input, which is potentially useful for iterative acquisition.

Acknowledgements. For testing, we have used synthetic curve networks with normals. These networks were traced on meshes provided by Cindy Grimm[2] (bowl), Pan et al. [Pan+15] (gamepad) and [libigl][3] (bumpy cube, lilium). The gamepad network is originally due to Xu et al. [Xu+14].

Hao Pan kindly provided the meshes used for comparison in Figs. 4.21, 4.23 and 4.24 (gamepad, torus, lilium, bumpy cube).

[2] http://web.engr.oregonstate.edu/~grimmc/meshes.php
[3] https://github.com/libigl/libigl/tree/master/tutorial/shared

5

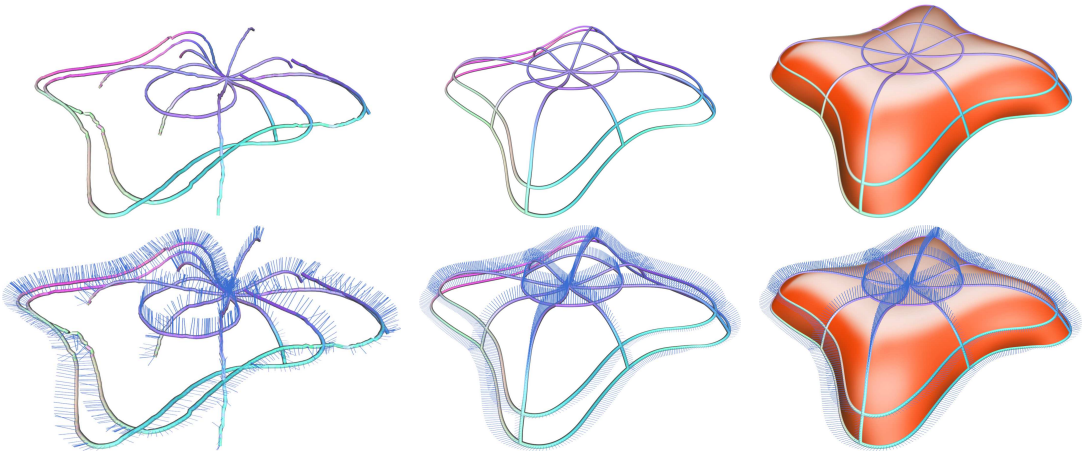
Applications : Reconstruction & Sketching

Algorithms from previous two chapters have many potential applications. Chapter 3 described a new algorithm for reconstruction of smooth and consistent curve networks from orientation data provided by inertial and magnetic sensors. Chapter 4 described a new algorithm for surfacing of curve networks with known surface normals.

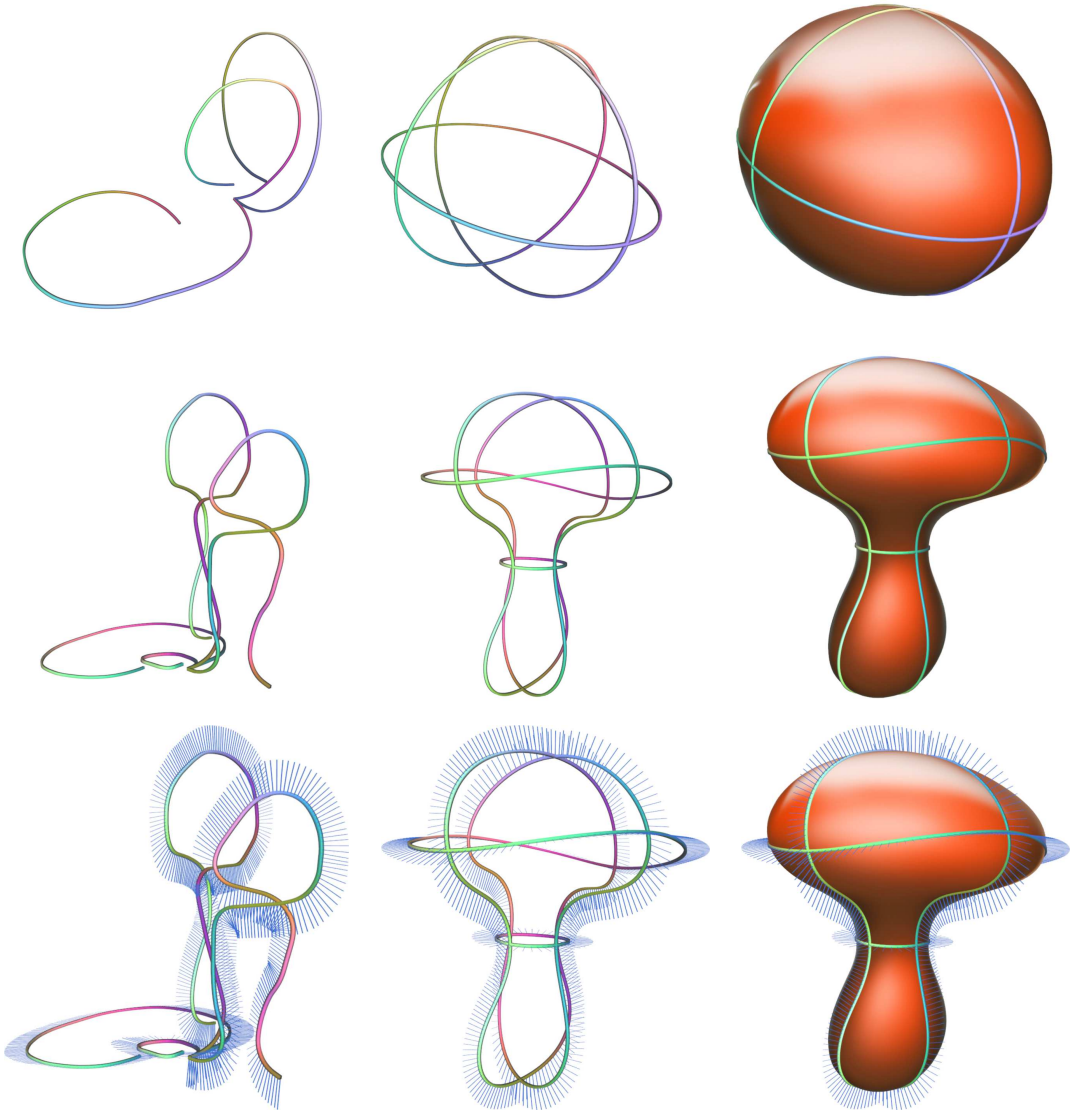
In this chapter, we combine and apply the two algorithms to real-world data in order to produce digital reconstructions of physical surfaces. We demonstrate results using data acquired with the two dynamic devices introduced in Section 1.4: Morphorider and a smartphone.

Motivated by the previous reconstruction methods developed for static sensor devices, our initial goal was to develop methods for reconstruction of shapes using data from dynamic sensor devices. In doing so, we have discovered that the same methods have a potential to be used in the context of 3D sketching.

We therefore distinguish two scanning modes in the context of our dynamic framework. *Reconstruction* (Fig. 5.1a) is the process of scanning an existing physical shape; see Section 1.1 for more details. Both Morphorider and smartphone are used to this



(a) Lilium acquired with the Morphorider



(b) Sphere and mushroom sketched with a smartphone

Figure 5.1: Examples of results obtained using our framework: **(a)** reconstruction, **(b)** sketching. Naive integration of raw, acquired data yields network with incorrect topology (*left*). Our reconstruction algorithms result in smooth curves with correct topology (*middle*) ready for surfacing (*right*).

end. *Sketching* (Fig. 5.1b) means creation of a new (3D) shape from scratch, using a smartphone.

This chapter has three main sections. Section 5.1 describes the fabrication of the lilium test surface. This surface was fabricated from a known digital model in order to be used as a ground truth. This allows us to evaluate the presented framework quantitatively by measuring the error of reconstruction.

Section 5.2 provides more details about the acquisition devices and describes our acquisition process. We also discuss our implementation of the different methods for acquisition, reconstruction and surfacing.

Finally, Section 5.3 presents results of the complete reconstruction pipeline, starting with raw orientations and resulting in a smooth surface. Computed surfaces with known ground truth are evaluated by measuring the error of reconstruction.

5.1 Experiments with fabricated surfaces

The target application of the algorithms presented in this thesis is reconstruction of real-world physical shapes. Fig. 5.1a shows an example of reconstruction of a physical object using our methods.

One way of testing the quality of the reconstructed models is visual inspection. Many surface interrogation techniques are available to assess the smoothness of a surface, such as curvature and porcupine plots (Fig. 3.12), isophotes (Fig. 4.21), reflection lines (Fig. 4.19), or normal mapping (Fig. 4.24).

To perform a more rigorous *quantitative* testing of the presented algorithms, our goal was to compare the reconstructed network and/or surface to a known ground truth. In practice, such models are seldom available, and there are essentially two options to obtain them. One way to obtain a ground truth model is to scan the physical object using a different technique with high precision. The other way is to fabricate the object from a digital model – this is what we chose to do.

We already had one fabricated model with known ground truth at our disposition – a polystyrene cone with base radius 0.5m and height 2m (Fig. 5.4 left). The same cone was used for testing in the thesis of Mathieu Huard [Hua13]. While the cone is certainly a suitable test object, we also wanted to test surfaces with more complex geometry in terms of curvature variation; cone, being a developable surface, has Gaussian curvature zero everywhere ($K \equiv 0$).

Today, 3D fabrication is accessible and low-cost; nevertheless, most consumer-available 3D printers have severe limitations when it comes to the size of the printed model. Such small-sized objects cannot be used for data acquisition with either of

our devices. In order to fabricate a surface of suitable size, we had to use a different fabrication method (injection molding) proposed by a specialized company.

Since only a single surface could have been fabricated, we have chosen the test object to be lilium (Fig. 5.4 right). The geometry of this shape is well-suited for testing our framework: its surface is smooth and contains elliptic points ($K > 0$), hyperbolic points ($K < 0$) and parabolic/planar points ($K = 0$).

The dimensions of the two fabricated models are given in Table 5.2.

	width w	depth d	height h	diameter $\sqrt{w^2 + d^2 + h^2}$
lilium	1.00	1.00	0.25	1.44
cone	1.00	1.00	2.00	2.45

Table 5.2: Dimensions of physical surfaces used as ground truth in our tests (Fig. 5.4). Lengths are given in meters and refer to the dimensions of the axis-aligned bounding box (AABB).

5.1.1 Fabrication of lilium

Lilium is available online as a triangle mesh[1] – this representation is however unsuitable for fabrication, which requires a B-spline surface on the input. Our solution was to infer the B-spline surface from the available triangle mesh: since there are no automatic tools for this type of conversion, we had to manually drape the mesh with a smooth surface.

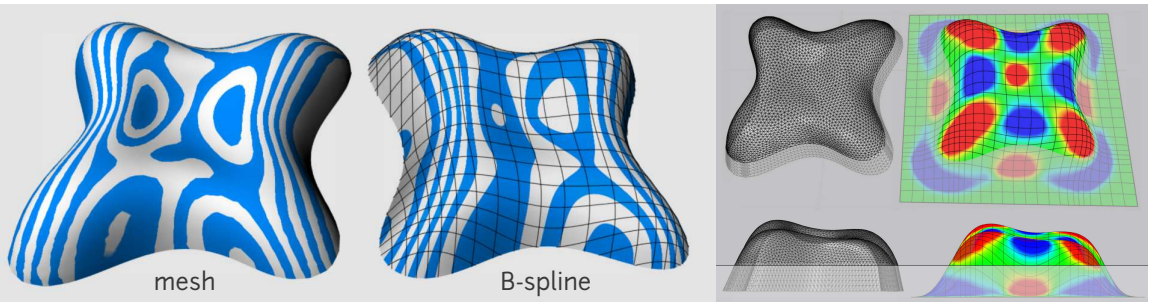


Figure 5.3: Lilium triangle mesh converted to a smooth B-spline surface, both shown with isophotes. On the right is the mesh wireframe and Gaussian curvature of the B-spline surface *(red – positive, green – zero, blue – negative)*.

[1] https://github.com/libigl/libigl/blob/master/tutorial/shared/lilium.obj

Figure 5.4: Physical surfaces used as ground truth in our tests: cone (*left*) and lilium (*right*).

We have created the B-spline surface in [Rhino] with help of algorithmic modeling provided by [Grasshopper]. The boundary curve of the mesh was first extruded. We then applied Rhino's drape operation to get a rough B-spline model, followed by iterated Laplacian smoothing. Fig. 5.3 shows the input triangle mesh and the B-spline surface we obtained, as well as its Gaussian curvature. The resulting smooth surface preserves lilium's characteristic shape and is a sufficient approximation for our needs.

5.2 Acquisition & implementation

In Section 1.4, we briefly introduced the devices that we use for acquisition in the dynamic setup. This section provides more details about the devices and about the acquisition process. We also describe our implementation of various tools for acquisition, reconstruction (Chapter 3) and surfacing (Chapter 4).

5.2.1 Morphorider

Morphorider is an acquisition device that motivated the development of algorithms presented in this thesis. It is our primary acquisition device. We already mentioned Morphorider in the context of estimation of orientations (Section 3.4.2) and evaluation of our methods (Sections 3.7 and 4.6). In this section, we provide more details on how this device operates.

Inertial measurement unit (IMU). To measure the orientation of the device, we use inertial and magnetic sensors able to provide their rotation with respect to a mea-

```
% Morphorider's MAC address is 00:12:F3:1C:75:56
% listen at port 99, has to be executed as root
system('sudo rfcomm bind 99 00:12:F3:1C:75:56');
mrider.port = '/dev/rfcomm99';
mrider.serial = serial( mrider.port,...
     'BaudRate',        115200,     ...
     'DataBits',        8,          ...
     'StopBits',        1,          ...
     'Parity',          'none',     ...
     'FlowControl',     'none',     ...
     'InputBufferSize',  22         );
fopen( mrider.serial );
% send 'Q' = char(81) : this triggers acquisition
fprintf( mrider.serial,'%c',81);
% read the measurement
[data,datasize] = fread( mrider.serial );
% process the measurement
% ...
```

Code snippet 5.5: Opening a serial connection in Matlab under Linux

```
# file: lilium5-mrider.rawnet
# number of curves in the network
5
# curve 1
  1  # 1=boundary, 0=interior
  1  # 1=closed,   0=open
  # number of nodes
  9
  # node indices
  1 51 86 121 159 194 226 252 282  #  local
  0 1 2 3 4 5 6 7 0                # global
  # number of datapoints
  282
  # data:
  # distance   tangent                  normal
  #            x       y       z        x       y       z
    +0.0000   +0.6442 +0.7626 -0.0570   +0.7277 -0.5884 +0.3524
    +0.0091   +0.6412 +0.7644 -0.0662   +0.7326 -0.5843 +0.3490
    +0.0178   +0.6537 +0.7540 -0.0637   +0.7236 -0.5983 +0.3440
     ...
    +2.4427   +0.7504 +0.6607 -0.0166   +0.6149 -0.6887 +0.3839
# curve 2
...
```

Code snippet 5.6: Rawnet file for storing network orientations and distances

sured field (Section 1.2). We use a 3A3M configuration, which combines a 3-axis accelerometer and a 3-axis magnetometer. An orthonormal frame that represents the 3D orientation is determined by combining the information from both sensors. The sensors are fixed in the device so that one of the axes of the orthonormal frame is aligned with the motion axis. The orientations are estimated from sensor measurements by solving the Wahba's problem using SVD (Section 3.4).

Sensor of displacement. Orientations provided by the IMU are by themselves not sufficient to reconstruct the spatial locations, we also need to know the displacement of the device along scanned curves. To this end, we use an odometer: the encoder disk sends a tick when 1/500 of a round is traveled. The displacement of the Morphorider has to be controlled to avoid sliding.

Data collection. The device contains a micro-controller, which collects the data. The micro-controller is managed by a software driver. Values from the IMU and the odometer are read sequentially via a serial bus, and send to a host computer via Bluetooth. Morphorider is equipped with a battery and the acquisition is wireless.

Acquisition process. We use a Matlab program to acquire data with the Morphorider (Fig. 5.7 left). To record the data, the device is connected to a host computer via a serial Bluetooth connection. In Matlab, such connection is established by executing the command `fopen` on an object created with the command `serial` (Code snippet 5.5). Acquisition is triggered by sending commands via `fprintf` and measured data are read using `fread`. A raw measurement consists of 22 bytes:

bytes	**1 – 4**	**5 – 6**	**7 – 10**	**11 – 16**	**17 – 22**
represent	control bytes	index	distance	accelerometer	magnetometer

Numbers are represented as 16-bit signed integers, first bit represents the sign. For the distance measurement, the two integers represent amounts of big and small ticks: one big tick corresponds to a wheel turn, one small tick corresponds to 1/500 of a wheel turn. Accelerometer and magnetometer measurements consist of three integers – each of these integers is mapped to the interval [-1,1]. The resulting raw unit vectors in $\mathbb{R}^3$ have to be calibrated to obtain the directions $\mathbf{e}_{\text{acc}}$ and $\mathbf{e}_{\text{mag}}$ [Bon+09]. The calibrated vectors are then used for estimation of orientations (cf. Section 3.4.2).

Prior to acquisition, a unique index between `0` and `n-1` is assigned to each of the `n` network nodes. Acquisition is controlled remotely using a wireless numpad (Fig. 1.15 left). When starting a new curve, we first indicate if the curve is at the boundary (the `-` key) or in the interior (the `+` key). Every time the device passes through a node, we record its index by pressing the corresponding key on the numpad. After the acquisition has been stopped, orientations, distances and network topology are exported into a `rawnet` file (Code snippet 5.6). This file is then directly available for automatic shape reconstruction using our framework without any post-processing.

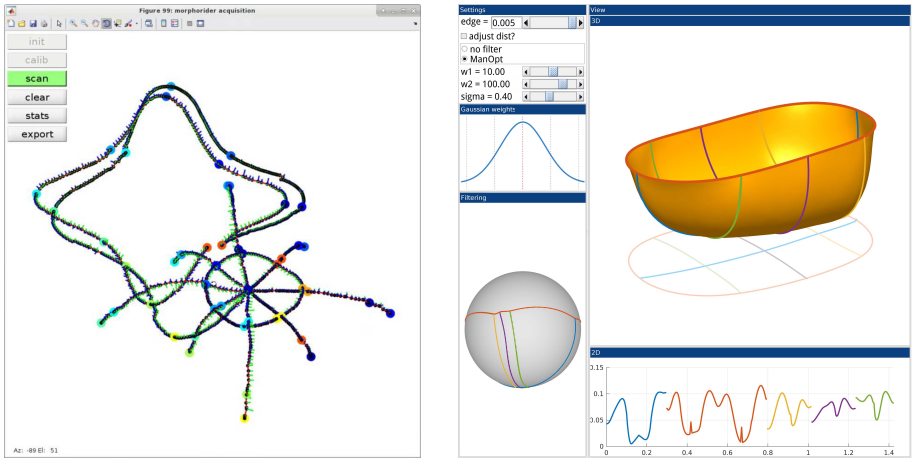


Figure 5.7: Matlab programs for acquisition (*Left*) and reconstruction (*Right*). Data are visualized in real time during acquisition, with positions computed using explicit Euler integration $\mathbf{x}_{i+1} = \mathbf{x}_i + d_i \mathbf{t}_i$.

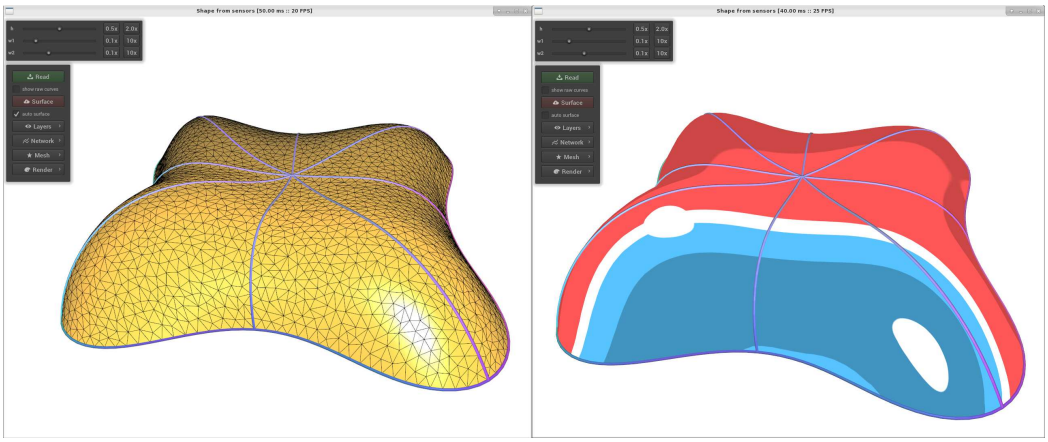


Figure 5.8: C++ implementation of our reconstruction framework

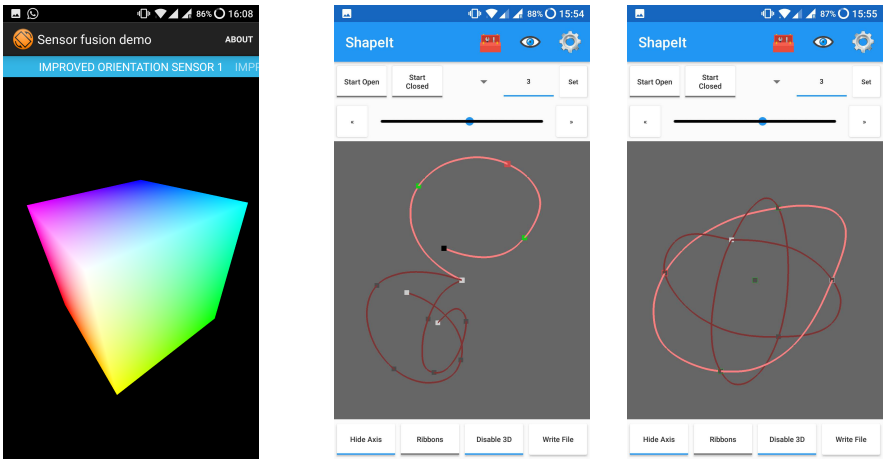


Figure 5.9: Android applications used for data acquisition with a smartphone. (*Left*) Sensor fusion. (*Right*) ShapeIt.

5.2.2 Reconstruction GUI

We implemented the reconstruction framework from Chapters 3 to 4 in Matlab (Fig. 5.7 right) and in C++ (Fig. 5.8).

For easier control and interactive feedback, the user manages the reconstruction from a GUI. Data are read from a rawnet file (Code snippet 5.6). User selects reconstruction parameters, such as the sampling density h, the convolution radius σ (Fig. 3.6), and the filtering weights λ and μ (Fig. 3.12).

The main part of the GUI is the rendering of the reconstructed shape. In addition, the Matlab program also shows the Gauss map and the curvature plot of the network. The C++ program enables visualization of the surface using the normal mapping technique described in Section 4.6, see Fig. 5.8 right.

5.2.3 Smartphone

Morphorider is certainly a device with interesting possibilities for applications waiting to be explored. For instance, imagine a mouse-like robot equipped with sensors that is autonomous or controlled by a human on distance. Such device could be used to acquire 3D information in hostile or unreachable environments.

While the possibilities are limitless, Morphorider is at the moment a one-of-a-kind prototype. When developing the algorithms exposed in previous two chapters, we were curious if the same methods could be used for reconstruction with smartphone-acquired data. This extension is not straightforward since smartphone cannot measure distances – at least not with the precision that we need.

To test the viability of this idea, we developed a minimal Android application based on the code of Alexander Pacha written for his master's thesis on sensor fusion [Pac13; MHV11]. This first prototype consisted of a single screen showing a cube rotating according to the orientation of the device. User controls the application by tapping the screen to start/stop the acquisition; a later version of the application also featured the possibility to indicate that the device is passing through a node via a vertical slide, but there was no visual feedback on the shape which was being scanned. The application can be seen in Fig. 5.9 left.

During acquisition, orientation data from the smartphone were recorded together with the timestamps and saved in a file on phone's local storage. Since the node indices were not specified during acquisition, the file needed to be manually post-processed by adding the indices where necessary. Afterwards, the data were ready for reconstruction. Examples of results obtained using this application are shown in Fig. 3.14 and Fig. 5.1b (mushroom).

5.2.4 ShapeIt

Encouraged by the initial results obtained with the smartphone-acquired data, we decided to further pursue the line of research related to shape acquisition with nothing but a smartphone. This is a challenging problem since there is currently no reliable method for estimating the displacement of a smartphone with precision to millimeters, or even centimeters – precise distance measurements are important for the network reconstruction algorithms from Chapter 3 to give the expected results.

The first prototype of the smartphone application for data acquisition (Section 5.2.3) was lightweight, but often tedious to use. Moreover, the output files required post-processing – the topology of the network had to be specified manually before the files could be read by our reconstruction programs.

Our initial focus was therefore on the development of a more user-friendly interface for acquiring network orientation data; this became the subject of a three-month internship of Lucas Lesage [Les17], which I co-supervised. The result is an Android application called ShapeIt (Fig. 5.9 right).

Three main goals were identified prior to the development of ShapeIt.

1. *Real-time visualization*. The application should provide a real-time feedback for the user by showing the scanned curves reconstructed using a simple forward integration without satisfying the topological constraints.
2. *Topology specification*. The application should provide a simple GUI for specification of the topological constraints (node indices), as well as for marking boundary/interior curves.
3. *Data export*. The application should provide a simple way for the user to export data in rawnet format compatible with our Matlab/C++ programs (Section 5.2.2).

The final application includes all three above features. We also included the Poisson network reconstruction implemented in C++ directly in the application (using Android Native Development Kit). This allows the user to reconstruct the network with correct topology directly in the smartphone. At the moment of writing this thesis, neither filtering nor surfacing are implemented directly in the application, and the Poisson reconstruction serves mainly for the purpose of visualization. In future, we plan to perform all the steps – acquisition, filtering, reconstruction and surfacing – directly on the smartphone.

Acquisition process. Data acquisition with ShapeIt is toggled by pushing a button. Surface is scanned by moving the smartphone along a fixed axis (see Fig. 5.10, the illustration in the middle). Similarly to Morphorider, each curve is acquired individually, and the nodes are marked during acquisition by pushing a button. User later

manually indicates the global indices of the marked nodes directly in the application. Acquired data are then exported to a `rawnet` file (cf. Code snippet 5.6), which can be processed by our reconstruction tools. See the example in Fig. 5.10.

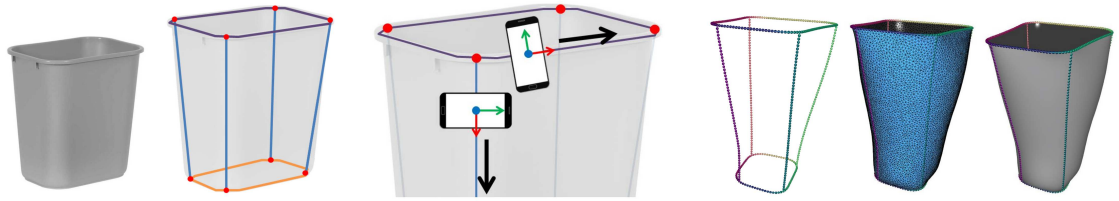


Figure 5.10: Example of a surface acquired with ShapeIt. Image from [Les17].

5.2.5 Sketching of virtual 3D objects

A major advantage of Morphorider over a smartphone is that it provides precise distance measurements and data that are parametrized by arc-length. Smartphone, on the other hand, has to rely on time parametrization and lengths either being input manually or estimated from total acquisition time. We have compared the three approaches in Section 3.7.4.

The apparent shortfalls of the smartphone setup can be turned to an advantage, enabling smartphone to do what Morphorider cannot do: sketching virtual shapes directly in 3D. Examples are shown in Fig. 5.11 and Fig. 5.12.

The application of our framework to 3D sketching was surprising and not intended. We believe this problem deserves a proper investigation on its own. Our current acquisition tools (such as ShapeIt) are prototypes, perhaps unintuitive for a first-time user. There is certainly great potential for future improvement, notably in terms of development of an intuitive user interface, but also in terms of algorithms customized for the particular task of 3D sketching.

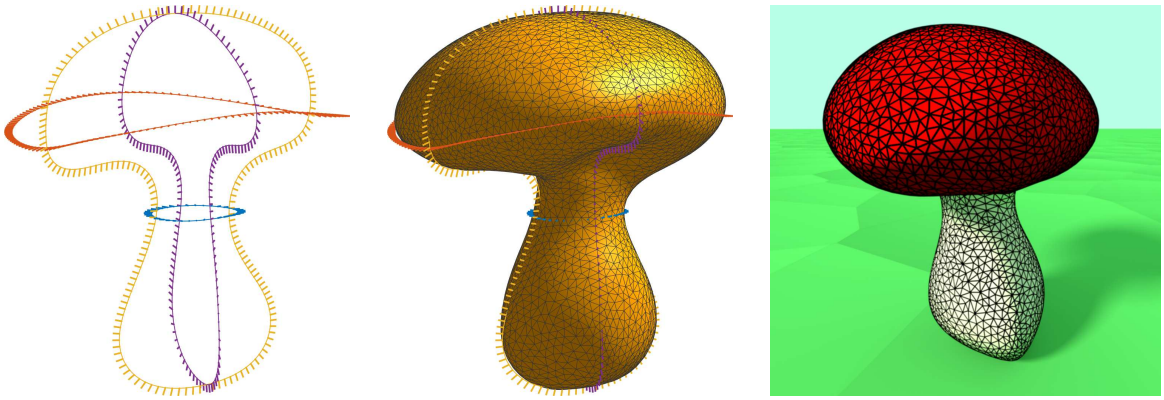


Figure 5.11: Mushroom network created from scratch entirely with a smartphone. Curve lengths were estimated from acquisition time and the network (*Left*) was reconstructed using algorithms from Chapter 3. Final surface was computed using the method from Chapter 4 with soft positional constraints *(middle)* and rendered in Blender (*Right*).

Figure 5.12: Inspired by the image on the right, we created a 3D model of sailboat by sketching the two networks on the left with a smartphone. The lengths were estimated from time of acquisition.

5.3 Reconstruction of physical surfaces

In the last part of this chapter, we go back to our original problem: acquisition and reconstruction of real-world shapes. We first present examples of acquisition and reconstruction of everyday objects. We then discuss the error of reconstruction with respect to ground truth for data acquired from cone and lilium.

5.3.1 Acquisition of everyday objects

We demonstrate the results obtained by scanning four different everyday objects: chair (Fig. 5.13), guitar (Fig. 5.14), baby bathtub (Fig. 5.15) and roof box (Fig. 5.16).

Fig. 5.13 demonstrates the importance of our filtering step. It shows four surfaces interpolating the chair networks from Fig. 3.13. The result in the first row was obtained by surfacing a curve network reconstructed from pre-filtered orientations without the filtering on SO(3). The other three shapes were obtained by surfacing networks reconstructed from filtered orientations using three different sets of weights.

The difference in quality of the final surfaces is evident: notice the specular highlights and silhouettes. The highlights on the unfiltered chair (top) indicate that the surface is not smooth and contains high-frequency noise. This noise is also visible by looking at the black curve network on the right.

In contrast, the filtered surfaces do not contain such artifacts, and the resulting shapes are globally smooth. Among the three filtered surfaces, the best results are arguably obtained with the weights $\lambda = 1$, $\mu = 1e4$, which provide the best trade-off between smoothing and fidelity.

Guitar (Fig. 5.14) and baby bathtub (Fig. 5.15) are additional examples of everyday objects acquired with the Morphorider. With our techniques, both shapes are reconstructed smoothly using only a few input curves instead of a dense point cloud.

During acquisition we faced the question of what curves to acquire in order to minimize the reconstruction error – different curve networks on the same surface usually produce different results. This is the case for the various networks taken from roof box in Fig. 5.16. While adding more curves generally improves the reconstruction, the overall shape clearly depends on how the curves are chosen. The optimal choice of curves to be scanned merits to be investigated further, but goes beyond the scope of this thesis.

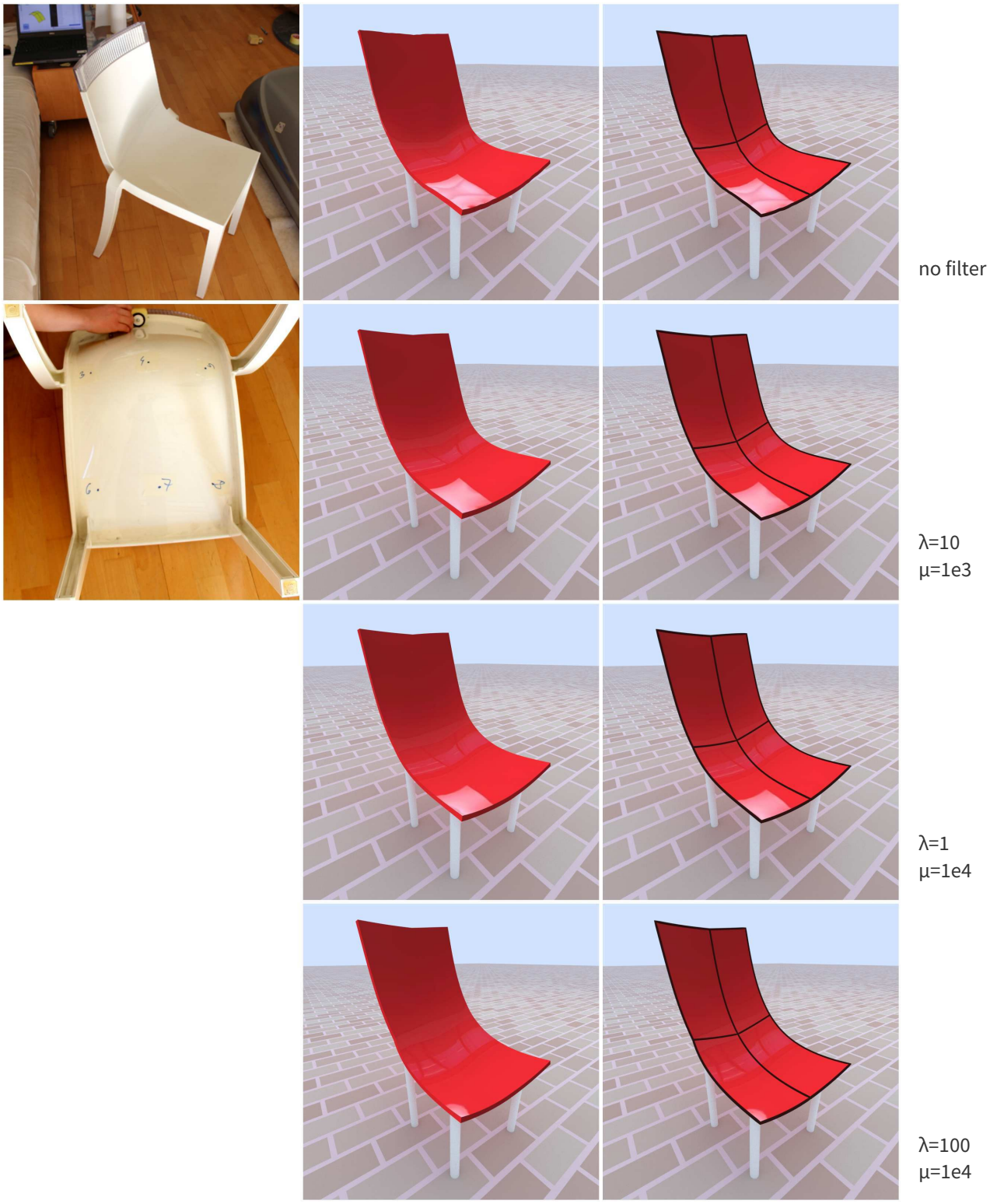


Figure 5.13: Surfaces interpolating the chair networks from Fig. 3.13. Photos on the left show the original chair during acquisition.

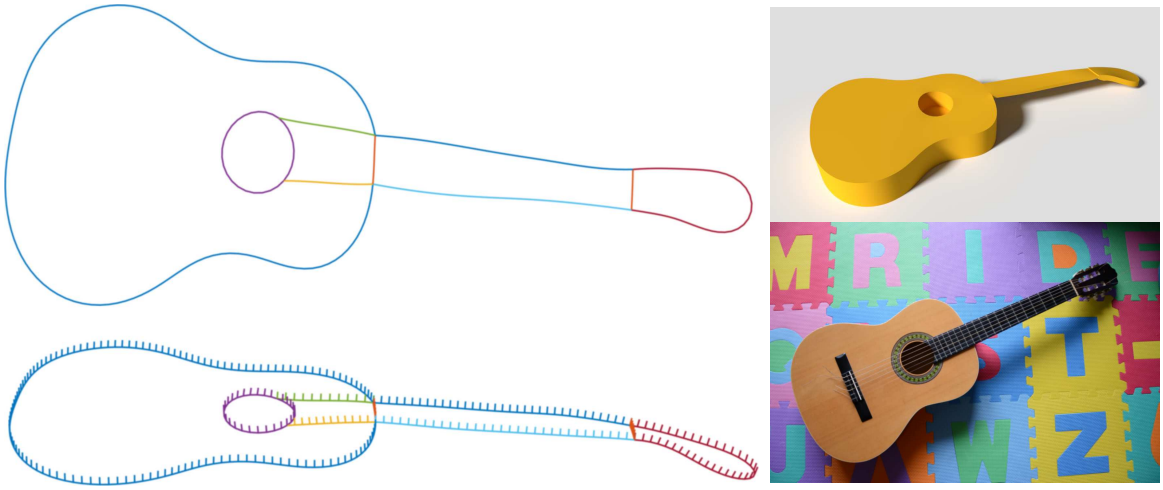


Figure 5.14: Guitar acquired with Morphorider. Digital guitar was created by extruding the surfaced network.

Figure 5.15: Baby bathtub acquired with Morphorider. Scanning this translucent object was not an issue.

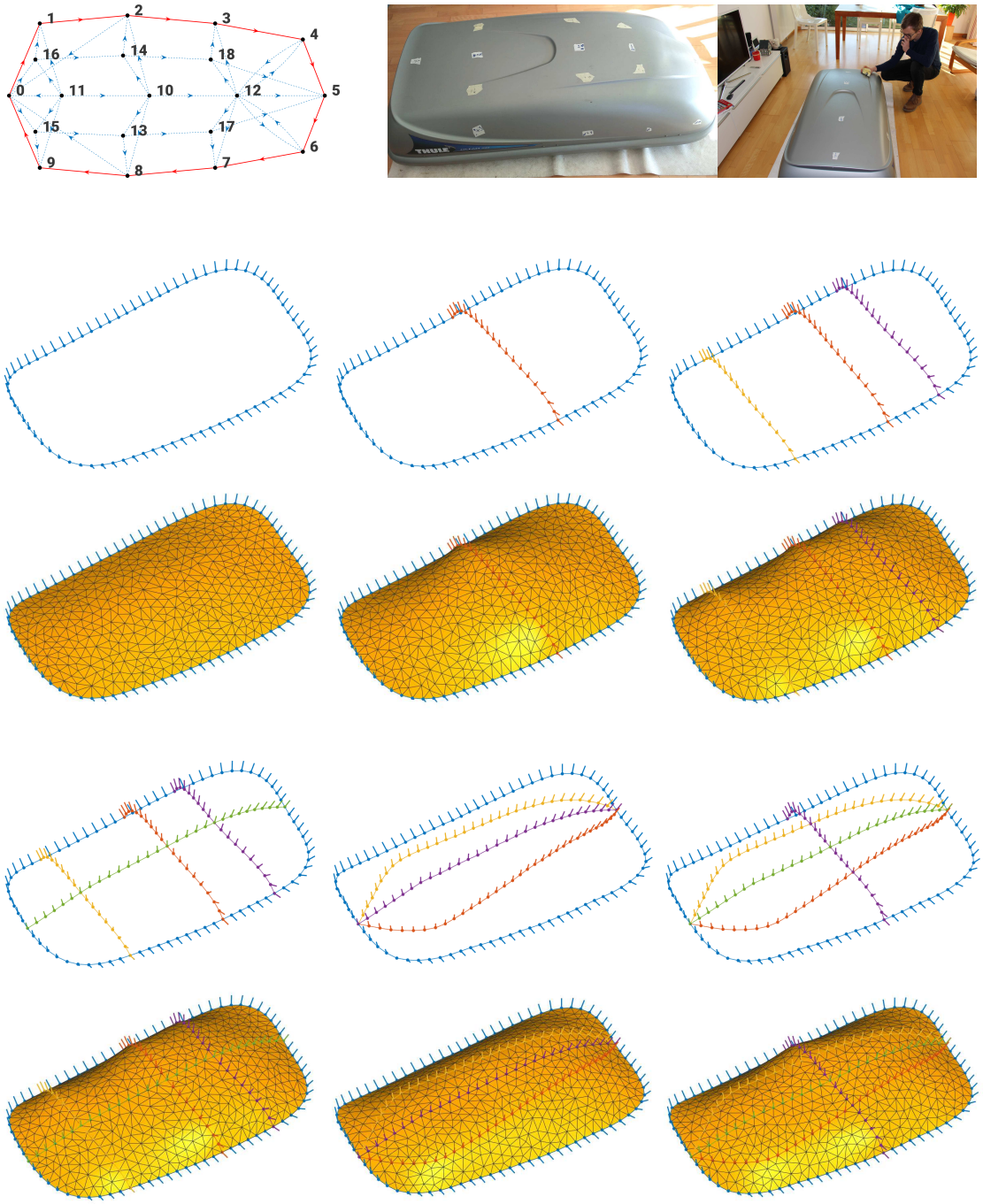


Figure 5.16: Roof box acquired using the Morphorider and reconstructed with different sets of curves

5.3.2 Measuring the error

In Figs. 3.18 to 3.22, we showed an analysis of reconstruction error for networks acquired using both Morphorider and smartphone. This error was computed for networks acquired from the fabricated surfaces – cone and lilium – with known ground truth models. See Section 3.7.3.

This section features analysis of reconstruction error for surfaces. Each of the reconstructed networks from Figs. 3.18 to 3.22, was surfaced using our method from Chapter 4. We then evaluated the error of reconstruction between the computed mesh $\mathcal{T}$ and the ground truth surface $\mathcal{S}$. Similarly to the network case, before computing the error, the two surfaces need to be registered together using ICP [BM92]. Then, for each vertex $\mathbf{v}$ in $\mathcal{T}$, the error is computed as the distance between $\mathbf{v}$ and its closest point on $\mathcal{S}$:

$$\text{error}(\mathbf{v}) = \min_{\mathbf{x}\in\mathcal{S}} \|\mathbf{v} - \mathbf{x}\| \,. \tag{5.1}$$

Results are shown in Figs. 5.17 to 5.21. Recall that for each dataset, we compare the surfaces obtained with four sets of filtering weights ($\lambda = 1, 100$ and $\mu = 1, 100$) and five different sampling distances

$$h \in \{6.4\%, 3.2\%, 1.6\%, 0.8\%, 0.4\%\} \,.$$

Plots on the right visualize the maximum, root mean square (rms), mean and minimum of the reconstruction error. As before, all lengths are relative to the diameter $d_{\mathcal{S}}$ of the corresponding ground truth surface $\mathcal{S}$.

For each surface, we visualize the point-wise error of reconstruction w.r.t. ground truth computed using Eq. (5.1). For lilium datasets, we also show the isophotes on the computed surfaces.

Remarkably, the mean error at the finest resolution is always smaller than 1%.

Cone (Fig. 5.17). Cone dataset was acquired with the Morphorider. In terms of error, the difference between reconstructed surfaces is most evident at coarse levels. At most levels, the smallest mean error is obtained with the weights $\lambda = 1$, $\mu = 100$, although the differences at the finest level are practically invisible. In terms of surface quality, the weights $\lambda = 100$, $\mu = 100$ provide the smoothest results, which is best seen by focusing at the boundary curves. The weights $\lambda = 100$, $\mu = 100$ seem to be the best choice in this case: at all levels, they combine visually smooth results with low reconstruction error.

Next, we look at the results with the two lilium datasets.

Lilium, 5 curves (Figs. 5.18 to 5.20). Recall that for lilium with 5 curves, we compare the three acquisition setups: Morphorider (with measured lengths), smartphone with estimated lengths, and smartphone with measured lengths. See Sections 3.7.3 and 3.7.4 for more details.

Let us first look at the error of reconstruction, which behaves similarly to the network error. For datasets with measured lengths, better results are obtained by setting small stretching weights ($\lambda = 1$). On the other hand, for the dataset with estimated lengths, better results are obtained by setting higher stretching weights ($\lambda = 100$). This behavior is possibly due to the fact that the estimated lengths tend to distort the shape; setting higher stretching weights smooths out this distortion.

Similarly to networks, the best reconstruction is obtained with smartphone data with measured lengths. The smallest mean error is 0.62% for the weights $\lambda = 1$, $\mu = 100$. Morphorider and smartphone with estimated lengths result in comparable error statistics. Overall, Morphorider provides slightly better results than smartphone with estimated lengths; moreover, comparing the colormaps of the two datasets, Morphorider reconstruction distributes the error more evenly.

In terms of surface quality, the results with measured lengths (with both devices) are superior to the results with estimated lengths. As expected intuitively, we obtain the smoothest and most regular isophotes for weights $\lambda = 100$, $\mu = 100$.

Lilium, 7 curves (Fig. 5.21). Finally, we discuss the reconstruction results using lilium dataset with 7 curves acquired with the Morphorider (Fig. 5.20). The mean error is slightly higher (around 1%) compared to the mean error of lilium with 5 curves acquired with Morphorider (around 0.85%). On the other hand, the isophotes on lilium with 7 curves seem to vary more smoothly that lilium with 5 curves. This means that adding new curves to the network does not necessarily improve the reconstruction in terms of error, but might improve the smoothness of the resulting surface.

5.3.3 Conclusion

This section presented the results obtained with our framework tested on a variety of real-world objects. Our experiments demonstrate that the framework produces surfaces of high quality, in terms of both visual smoothness and reconstruction error.

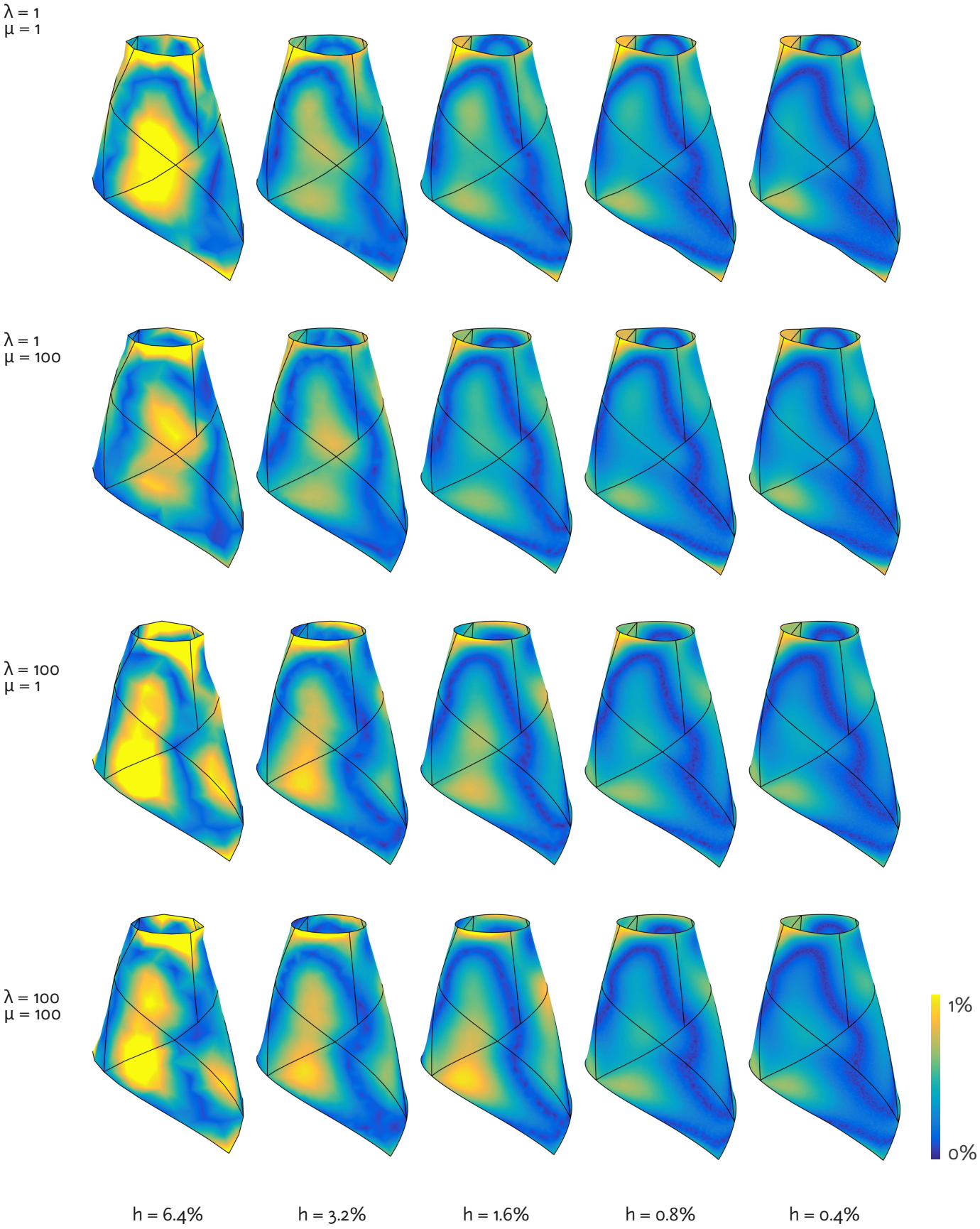
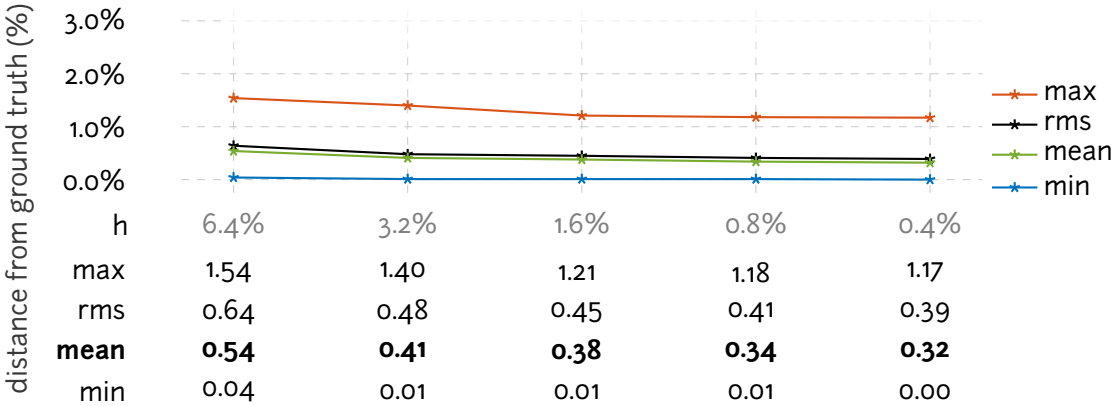


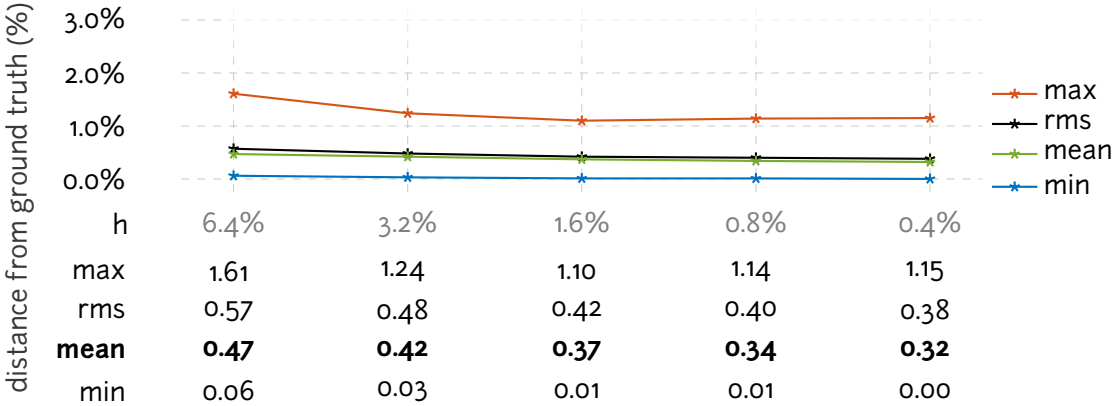
Figure 5.17: Surface error, cone, Morphorider

λ = 1
μ = 1



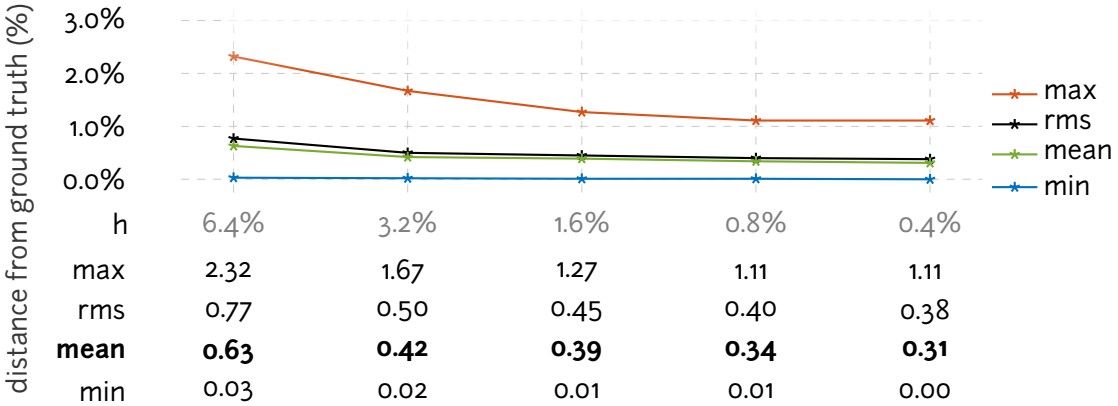
h	6.4%	3.2%	1.6%	0.8%	0.4%
max	1.54	1.40	1.21	1.18	1.17
rms	0.64	0.48	0.45	0.41	0.39
mean	**0.54**	**0.41**	**0.38**	**0.34**	**0.32**
min	0.04	0.01	0.01	0.01	0.00

λ = 1
μ = 100



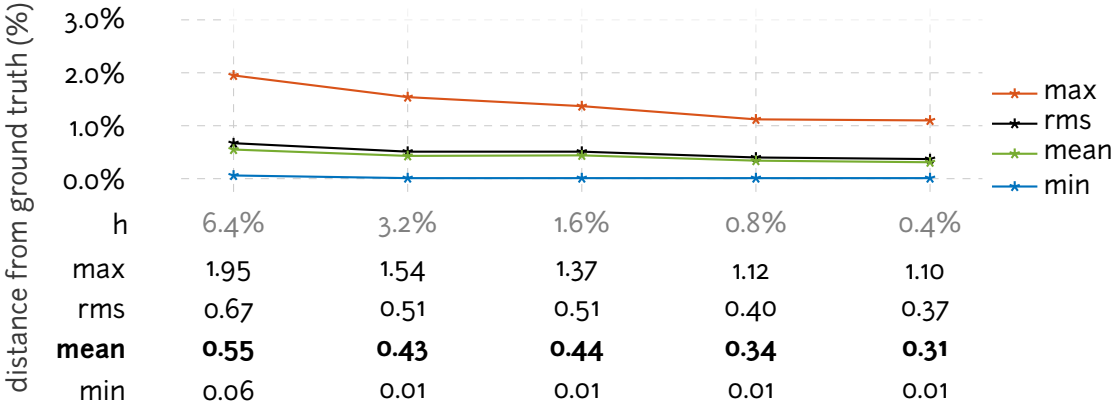
h	6.4%	3.2%	1.6%	0.8%	0.4%
max	1.61	1.24	1.10	1.14	1.15
rms	0.57	0.48	0.42	0.40	0.38
mean	**0.47**	**0.42**	**0.37**	**0.34**	**0.32**
min	0.06	0.03	0.01	0.01	0.00

λ = 100
μ = 1



h	6.4%	3.2%	1.6%	0.8%	0.4%
max	2.32	1.67	1.27	1.11	1.11
rms	0.77	0.50	0.45	0.40	0.38
mean	**0.63**	**0.42**	**0.39**	**0.34**	**0.31**
min	0.03	0.02	0.01	0.01	0.00

λ = 100
μ = 100



h	6.4%	3.2%	1.6%	0.8%	0.4%
max	1.95	1.54	1.37	1.12	1.10
rms	0.67	0.51	0.51	0.40	0.37
mean	**0.55**	**0.43**	**0.44**	**0.34**	**0.31**
min	0.06	0.01	0.01	0.01	0.01

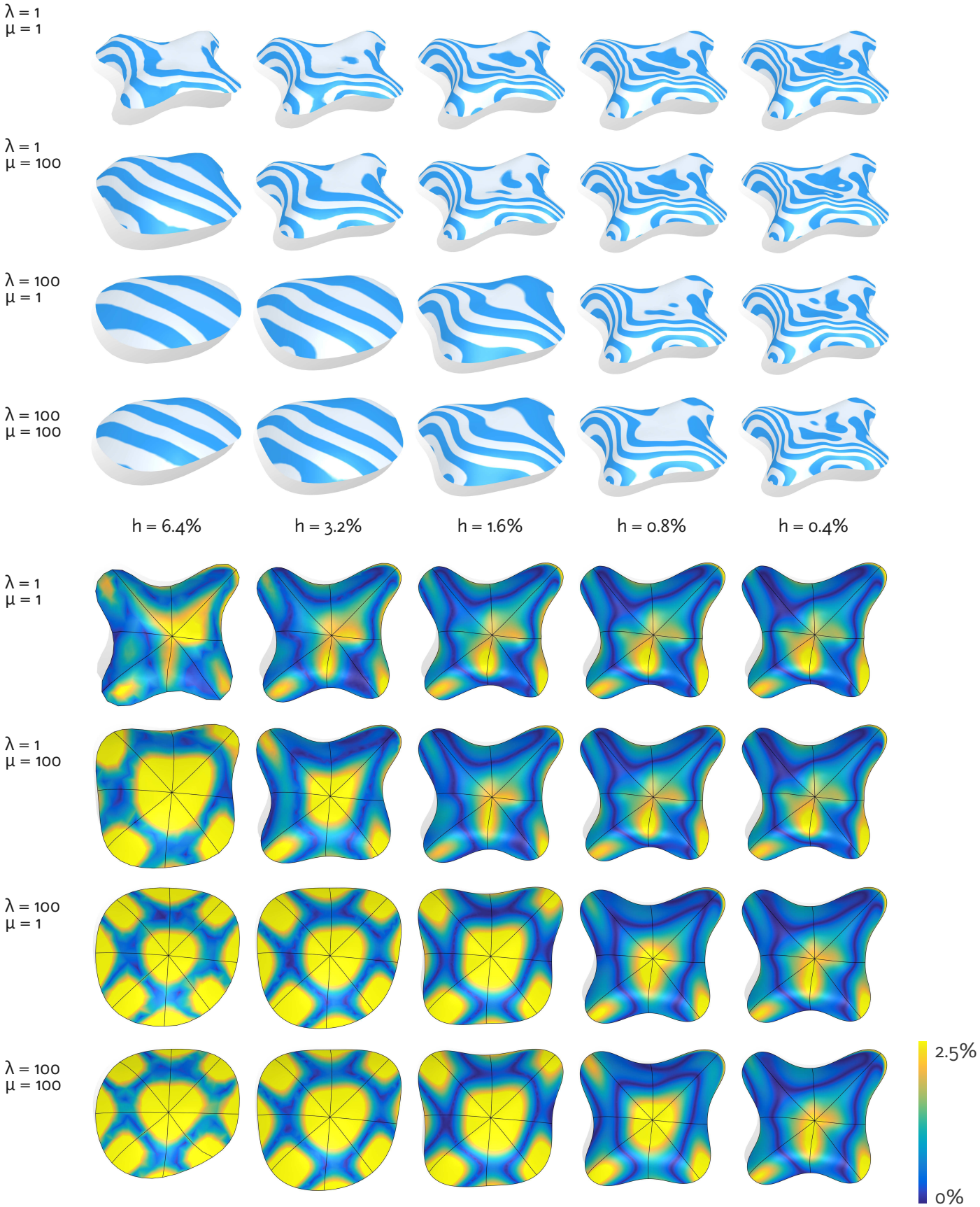
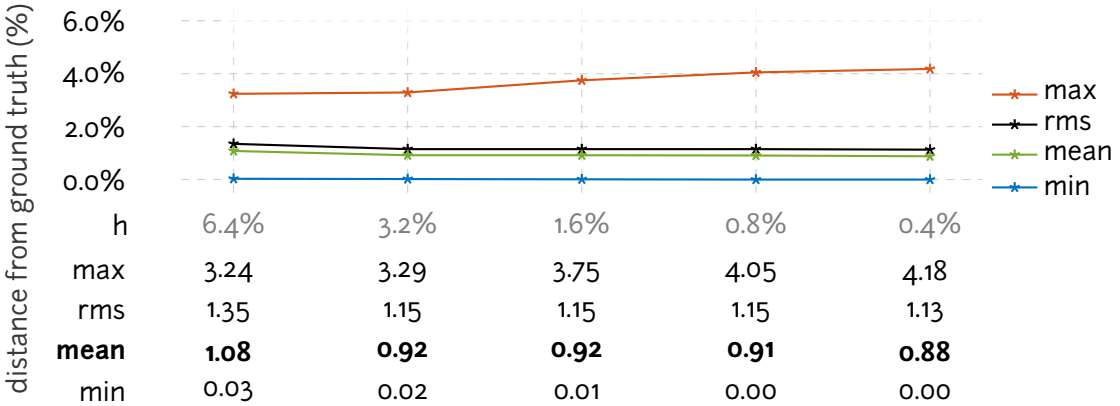


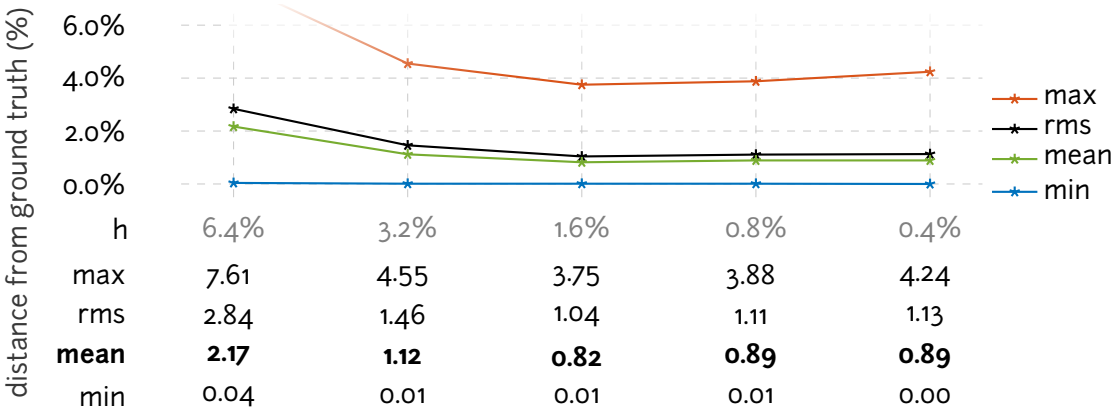
Figure 5.18: Surface error, lilium (5 curves), smartphone, estimated lengths

λ = 1
μ = 1



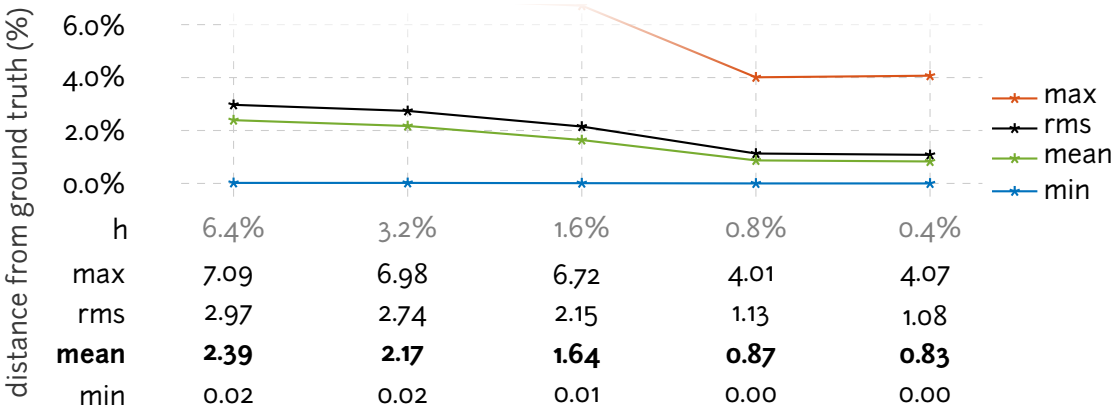
h	6.4%	3.2%	1.6%	0.8%	0.4%
max	3.24	3.29	3.75	4.05	4.18
rms	1.35	1.15	1.15	1.15	1.13
mean	**1.08**	**0.92**	**0.92**	**0.91**	**0.88**
min	0.03	0.02	0.01	0.00	0.00

λ = 1
μ = 100



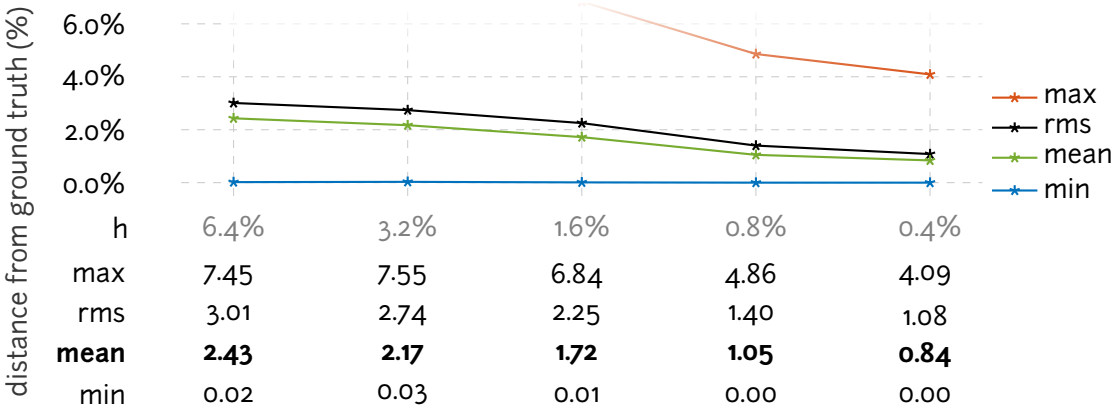
h	6.4%	3.2%	1.6%	0.8%	0.4%
max	7.61	4.55	3.75	3.88	4.24
rms	2.84	1.46	1.04	1.11	1.13
mean	**2.17**	**1.12**	**0.82**	**0.89**	**0.89**
min	0.04	0.01	0.01	0.01	0.00

λ = 100
μ = 1



h	6.4%	3.2%	1.6%	0.8%	0.4%
max	7.09	6.98	6.72	4.01	4.07
rms	2.97	2.74	2.15	1.13	1.08
mean	**2.39**	**2.17**	**1.64**	**0.87**	**0.83**
min	0.02	0.02	0.01	0.00	0.00

λ = 100
μ = 100



h	6.4%	3.2%	1.6%	0.8%	0.4%
max	7.45	7.55	6.84	4.86	4.09
rms	3.01	2.74	2.25	1.40	1.08
mean	**2.43**	**2.17**	**1.72**	**1.05**	**0.84**
min	0.02	0.03	0.01	0.00	0.00

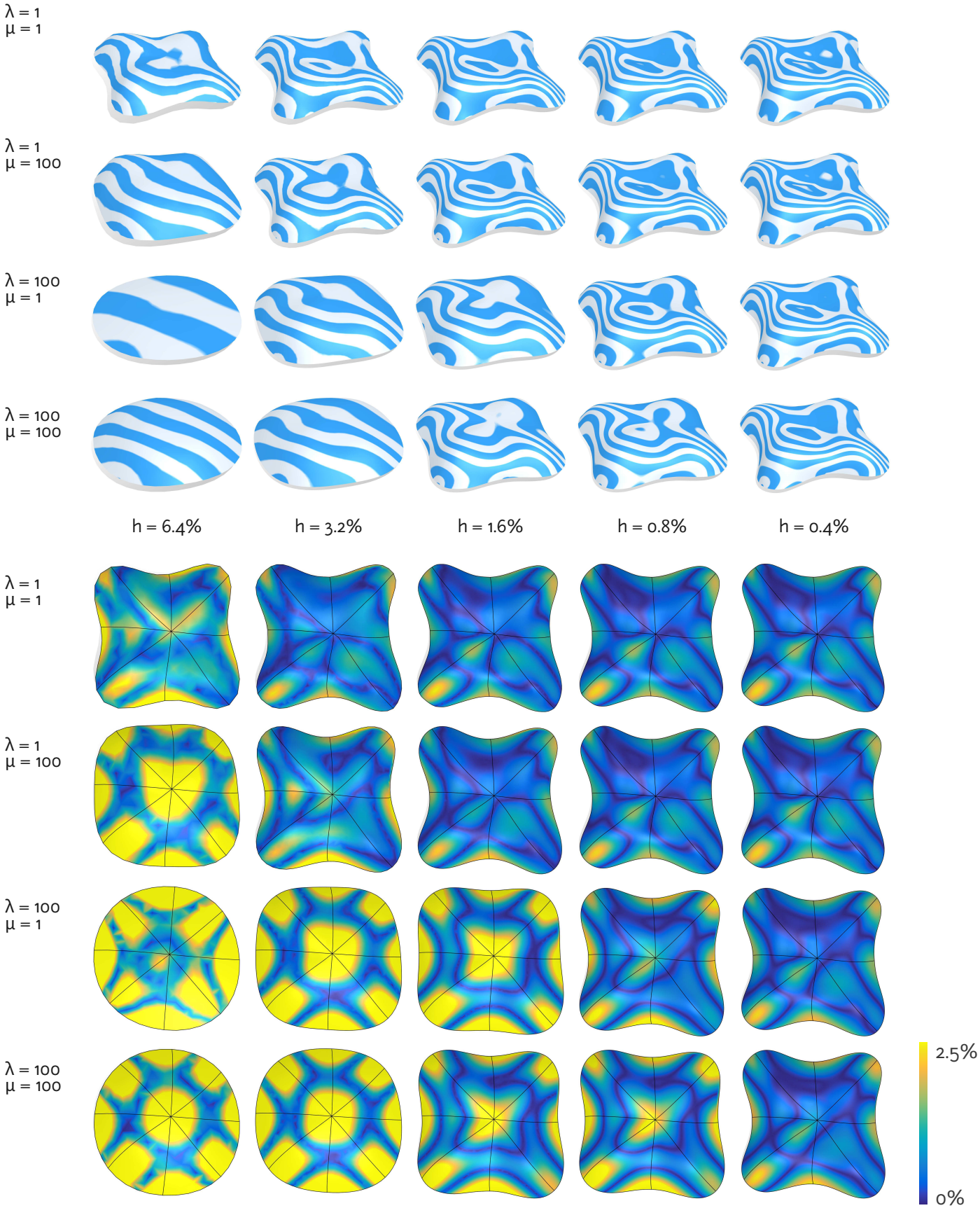
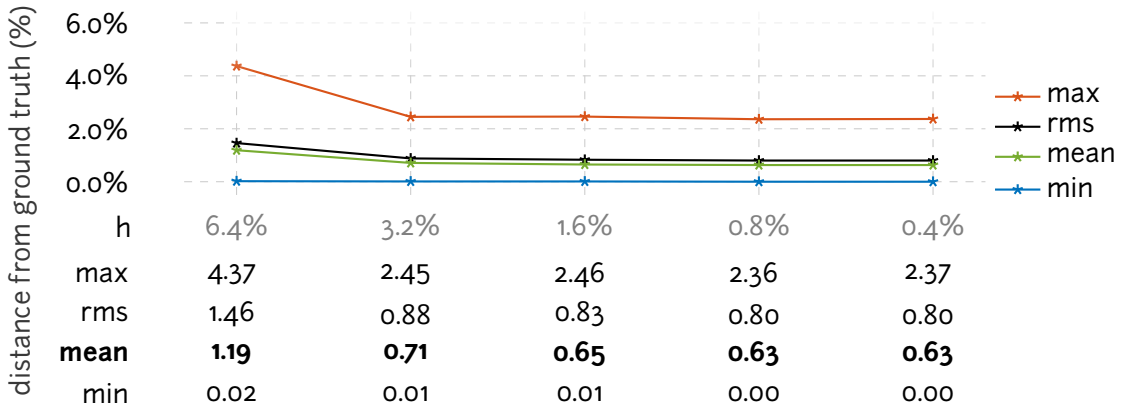


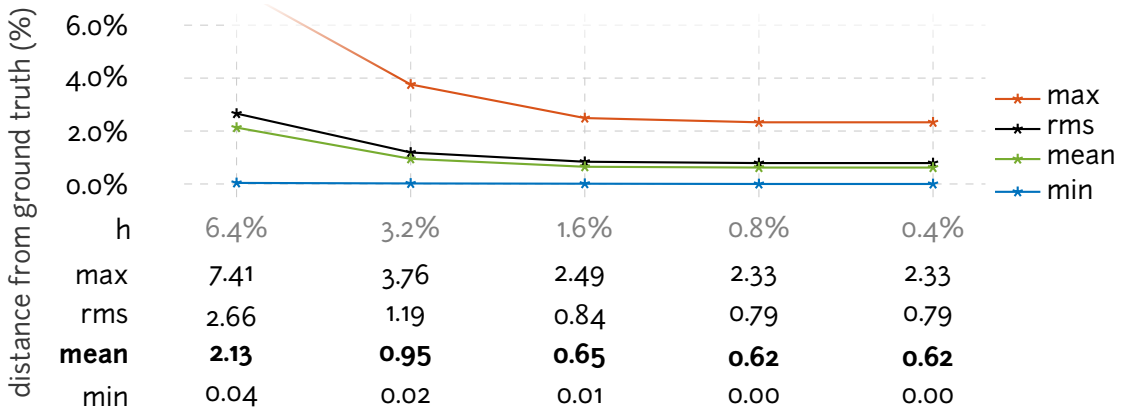
Figure 5.19: Surface error, lilium (5 curves), smartphone, measured lengths

λ = 1
μ = 1



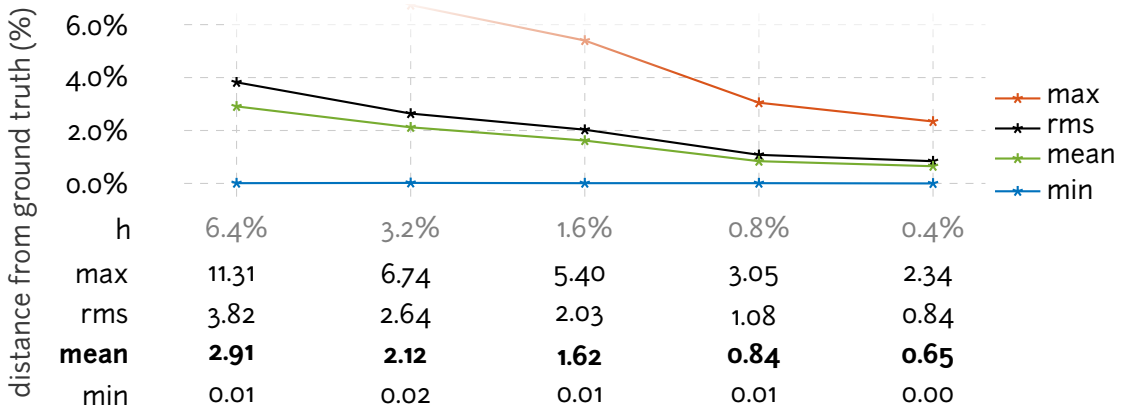
h	6.4%	3.2%	1.6%	0.8%	0.4%
max	4.37	2.45	2.46	2.36	2.37
rms	1.46	0.88	0.83	0.80	0.80
mean	**1.19**	**0.71**	**0.65**	**0.63**	**0.63**
min	0.02	0.01	0.01	0.00	0.00

λ = 1
μ = 100



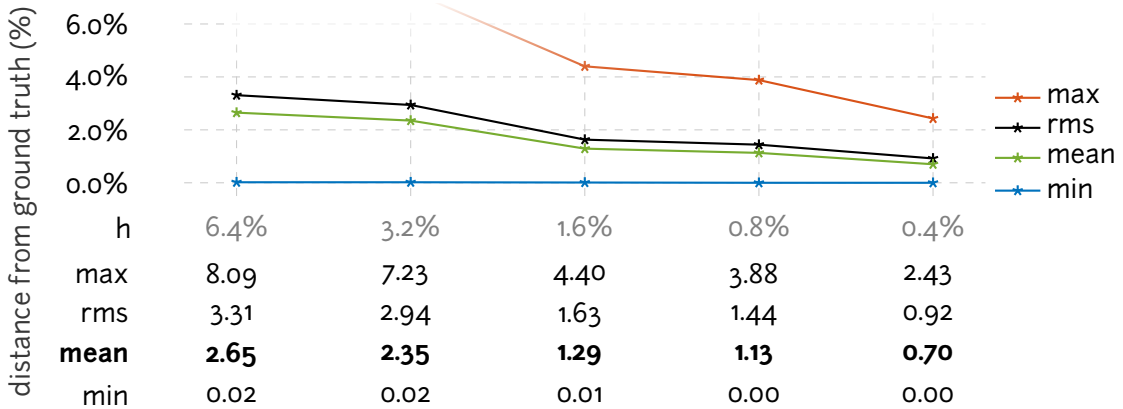
h	6.4%	3.2%	1.6%	0.8%	0.4%
max	7.41	3.76	2.49	2.33	2.33
rms	2.66	1.19	0.84	0.79	0.79
mean	**2.13**	**0.95**	**0.65**	**0.62**	**0.62**
min	0.04	0.02	0.01	0.00	0.00

λ = 100
μ = 1



h	6.4%	3.2%	1.6%	0.8%	0.4%
max	11.31	6.74	5.40	3.05	2.34
rms	3.82	2.64	2.03	1.08	0.84
mean	**2.91**	**2.12**	**1.62**	**0.84**	**0.65**
min	0.01	0.02	0.01	0.01	0.00

λ = 100
μ = 100



h	6.4%	3.2%	1.6%	0.8%	0.4%
max	8.09	7.23	4.40	3.88	2.43
rms	3.31	2.94	1.63	1.44	0.92
mean	**2.65**	**2.35**	**1.29**	**1.13**	**0.70**
min	0.02	0.02	0.01	0.00	0.00

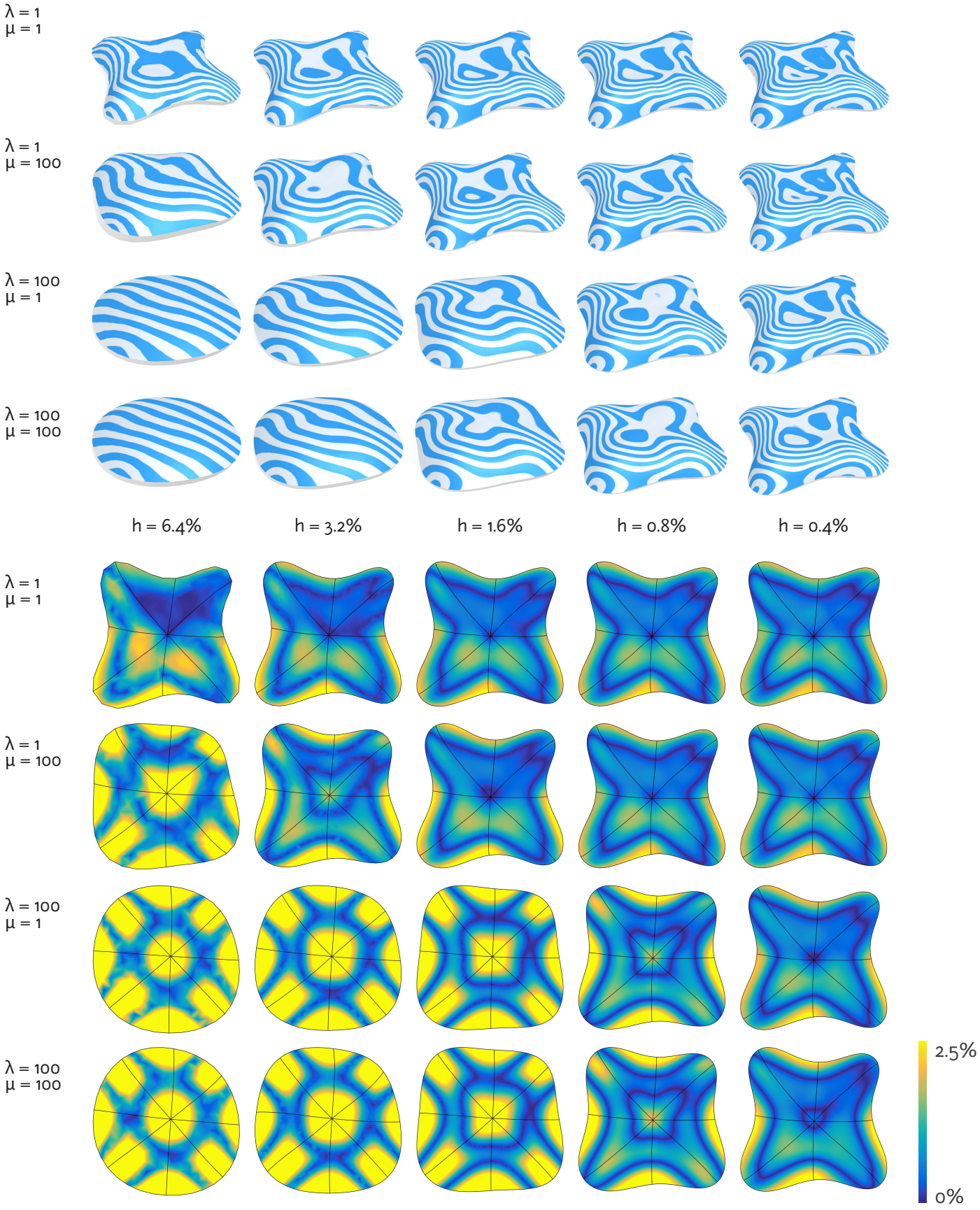
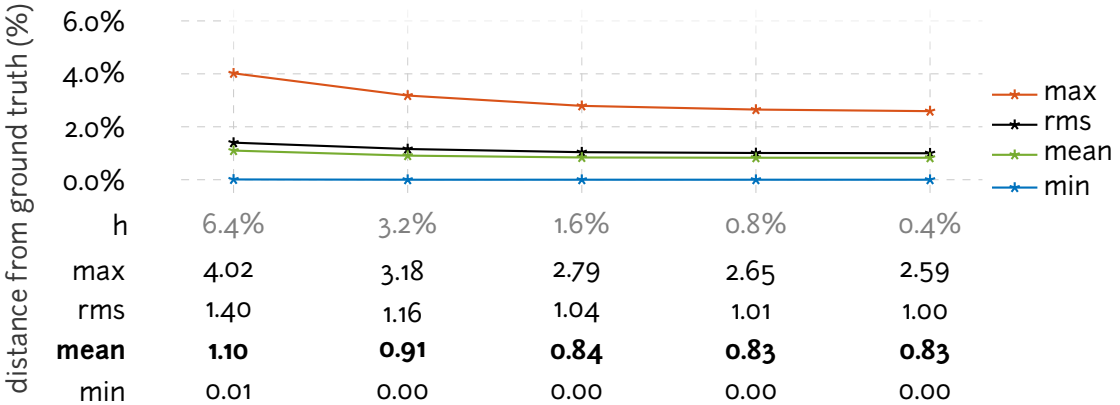


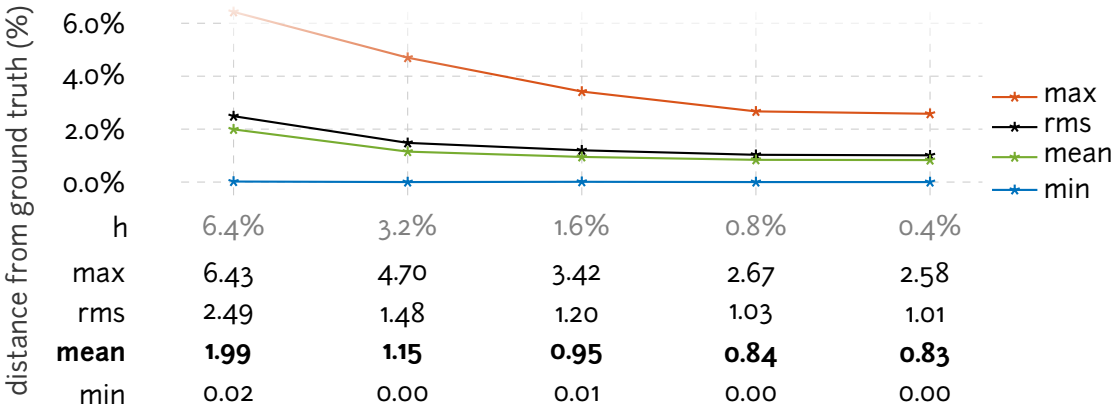
Figure 5.20: Surface error, lilium (5 curves), Morphorider

λ = 1
μ = 1



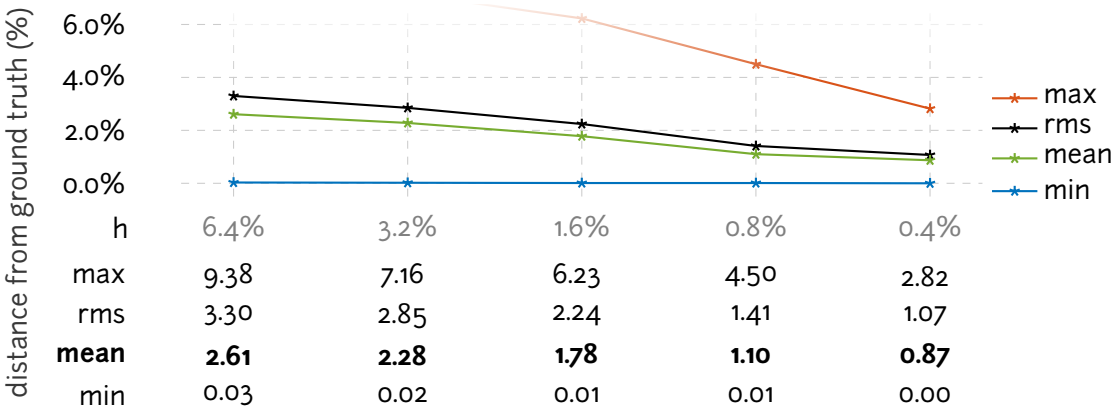
h	6.4%	3.2%	1.6%	0.8%	0.4%
max	4.02	3.18	2.79	2.65	2.59
rms	1.40	1.16	1.04	1.01	1.00
mean	**1.10**	**0.91**	**0.84**	**0.83**	**0.83**
min	0.01	0.00	0.00	0.00	0.00

λ = 1
μ = 100



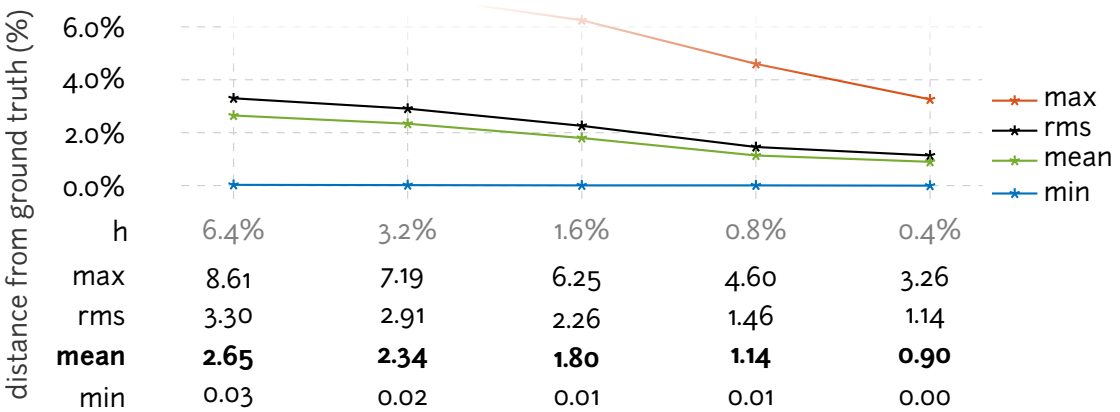
h	6.4%	3.2%	1.6%	0.8%	0.4%
max	6.43	4.70	3.42	2.67	2.58
rms	2.49	1.48	1.20	1.03	1.01
mean	**1.99**	**1.15**	**0.95**	**0.84**	**0.83**
min	0.02	0.00	0.01	0.00	0.00

λ = 100
μ = 1



h	6.4%	3.2%	1.6%	0.8%	0.4%
max	9.38	7.16	6.23	4.50	2.82
rms	3.30	2.85	2.24	1.41	1.07
mean	**2.61**	**2.28**	**1.78**	**1.10**	**0.87**
min	0.03	0.02	0.01	0.01	0.00

λ = 100
μ = 100



h	6.4%	3.2%	1.6%	0.8%	0.4%
max	8.61	7.19	6.25	4.60	3.26
rms	3.30	2.91	2.26	1.46	1.14
mean	**2.65**	**2.34**	**1.80**	**1.14**	**0.90**
min	0.03	0.02	0.01	0.01	0.00

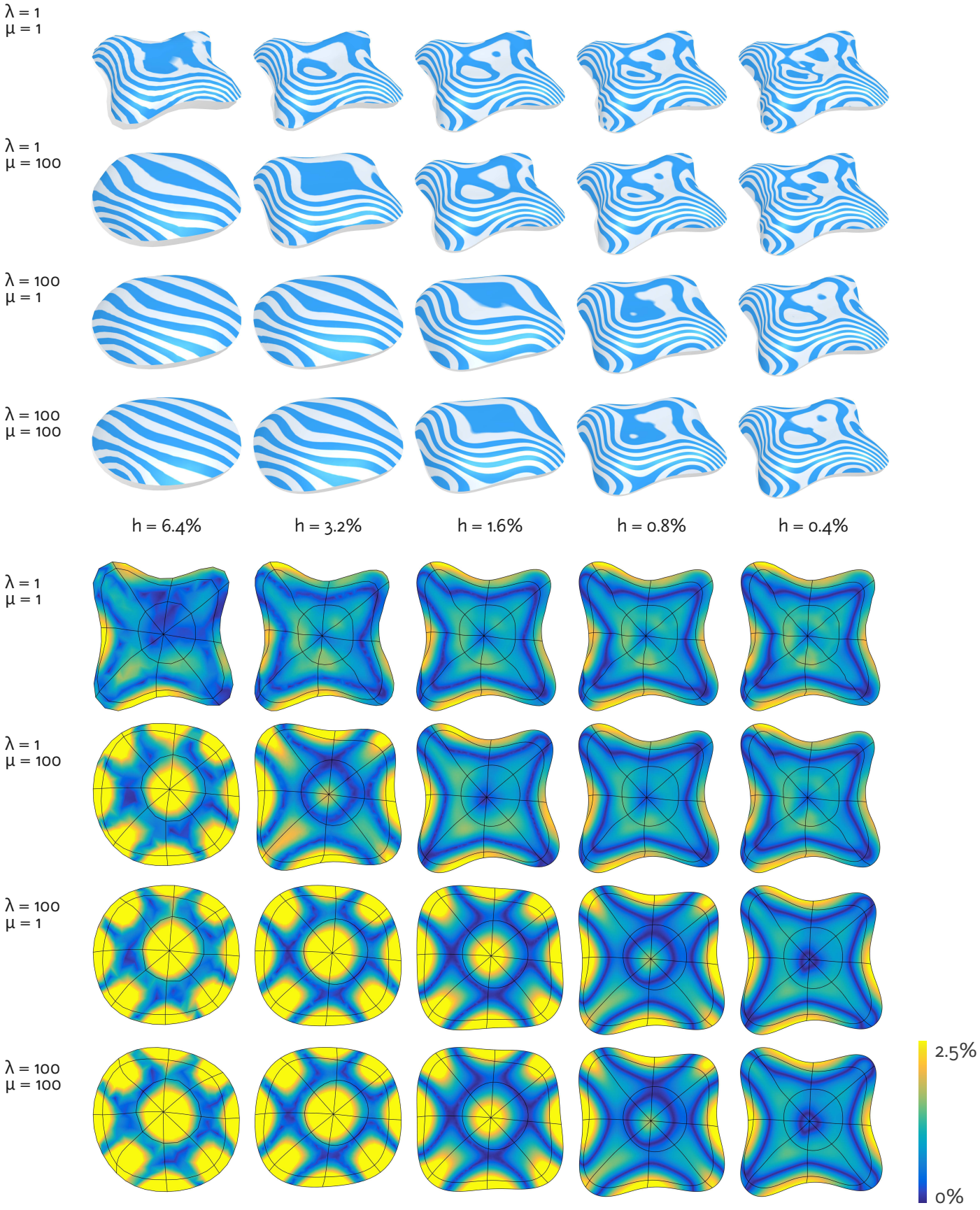
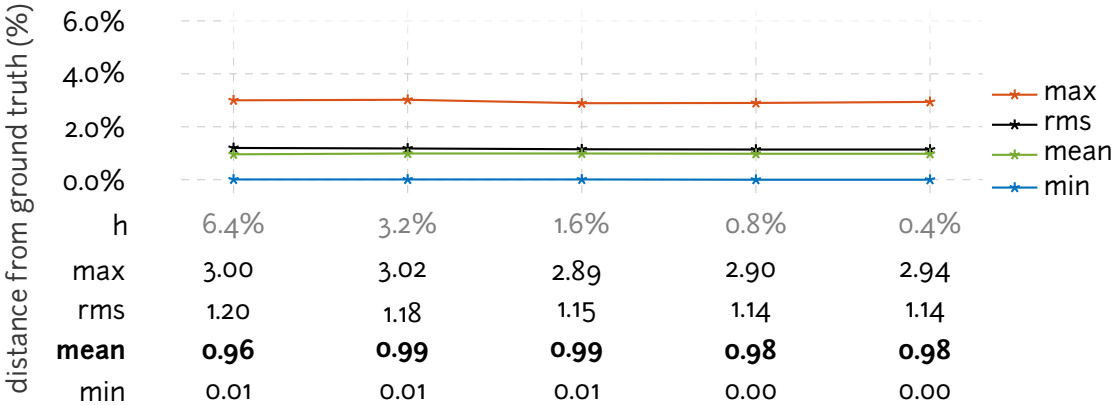


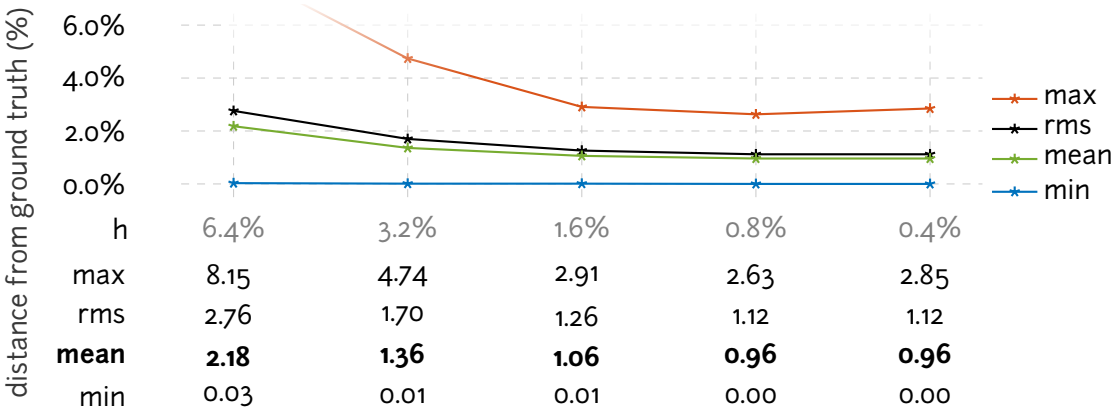
Figure 5.21: Surface error, lilium (7 curves), Morphorider

λ = 1
μ = 1



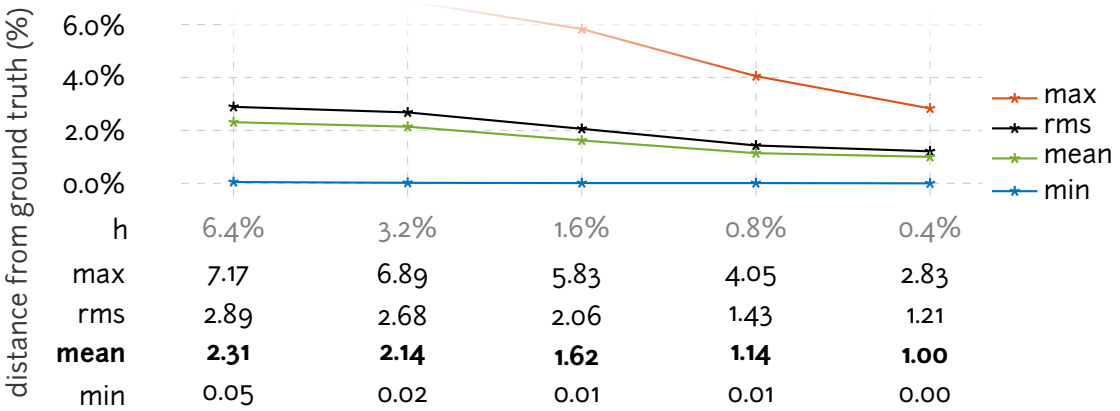
h	6.4%	3.2%	1.6%	0.8%	0.4%
max	3.00	3.02	2.89	2.90	2.94
rms	1.20	1.18	1.15	1.14	1.14
mean	**0.96**	**0.99**	**0.99**	**0.98**	**0.98**
min	0.01	0.01	0.01	0.00	0.00

λ = 1
μ = 100



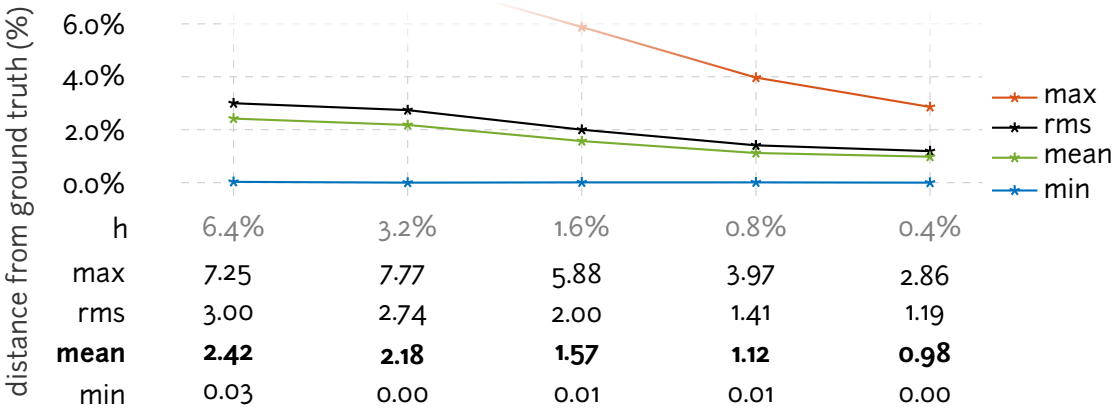
h	6.4%	3.2%	1.6%	0.8%	0.4%
max	8.15	4.74	2.91	2.63	2.85
rms	2.76	1.70	1.26	1.12	1.12
mean	**2.18**	**1.36**	**1.06**	**0.96**	**0.96**
min	0.03	0.01	0.01	0.00	0.00

λ = 100
μ = 1



h	6.4%	3.2%	1.6%	0.8%	0.4%
max	7.17	6.89	5.83	4.05	2.83
rms	2.89	2.68	2.06	1.43	1.21
mean	**2.31**	**2.14**	**1.62**	**1.14**	**1.00**
min	0.05	0.02	0.01	0.01	0.00

λ = 100
μ = 100



h	6.4%	3.2%	1.6%	0.8%	0.4%
max	7.25	7.77	5.88	3.97	2.86
rms	3.00	2.74	2.00	1.41	1.19
mean	**2.42**	**2.18**	**1.57**	**1.12**	**0.98**
min	0.03	0.00	0.01	0.01	0.00

6

Conclusion

We described new methods for 3D reconstruction with orientation sensors. The presented *shape-from-sensors* framework constitutes a complete solution for reconstruction of curves and surfaces using sensor devices. Below are the most important elements that differentiate our framework from related methods in sensor-based reconstruction.

Discrete formulation. The algorithms are formulated in terms of discrete curves (polylines) and discrete surfaces (triangle meshes). As a consequence, reconstruction is performed at interactive time rates.

Optimization-driven approach. Shapes are defined as solutions to optimization problems. These problems are formulated using results from discrete differential geometry.

User-specified topology. Curve network topology is required as user input and preserved during reconstruction by design. This approach avoids using ad-hoc stitching techniques to get the correct topology.

Guided by normals. Surface normals serve as important constraints in the optimization. During orientation filtering, we verify the normals are consistent at intersections; surfaces attempt to match the curvature estimated using propagated normals.

6.1 Summary of contributions

The general philosophy of this work was to adopt a mesh-related approach for solving the problem of shape reconstruction from sensor data. In contrast, most previous methods introduced in the context of sensors reconstruction use traditional spline-based methods. Our inspiration was mostly drawn from the emerging fields of geometry processing and discrete differential geometry. We have succeeded in adapting recent techniques from these fields to our specific setup – we introduced new reconstruction algorithms with rigorous mathematical background and extensive experimental results.

The effort was placed into developing algorithms that are as automatic as possible. While the acquisition has to be controlled manually, the reconstruction requires minimal user intervention. Essentially, the user only has to adjust a few parameters such as sampling density or filtering weights, and all computations are done automatically. Importantly, since our methods are interactive, the result is updated and rendered instantaneously when the parameters are changed.

We now sum up the two parts of the framework.

Our reconstruction method for curve networks has two important ingredients. First, our custom filtering directly on the manifold of orientations ensures the filtered data are smooth and consistent while remaining true to the raw acquired orientations. Second, the positions are obtained by resolving a Poisson system discretized by taking into account the topology of the network.

A key ingredient of our surfacing method is the equation matching vector mean curvature with the Laplace-Beltrami operator of the surface. This equation allows us to match the initial positions with mean curvature vectors estimated from input surface normals.

We applied the presented methods to reconstruction of surfaces from data acquired with two devices, Morphorider and smartphone. We have tested our approach on fabricated objects with known digital ground truth. Even for intricate shapes with high variations of curvature (lilium), the mean reconstruction error remains around 1%. Another possible application of our framework is 3D sketching using nothing but a smartphone, which we demonstrated with examples (mushroom, sailboat).

6.2 Future work

The present framework is the first step in a new direction, leaving space for future research. We summarize open questions together with ideas and perspectives for improvements – some have already been stated in previous chapters and are repeated here.

Curves. The versatility of the dynamic setup potentially allows us to acquire many different curve networks on the scanned surface (Fig. 5.16). It is unclear how the curves should be chosen in order to minimize network/surface reconstruction error. For instance, as we noted at the end of Section 2.1, the Darboux frame is rotation-minimizing along lines of curvature, i.e. curves with vanishing geodesic torsion. Should the acquired network trace lines of curvature on the surface? How many curves should we acquire? In which parts of the surface should the curves lie? These are the questions that remain open and deserve further investigation.

Experimental results indicate that surface normals along reconstructed networks are continuous but *not* differentiable (with derivatives defined with respect to the underlying surface). This means that the network is G^1, but not G^2 everywhere. Networks with smoothly varying normals might be obtained by a modification of the normal penalty in the energy used for orientation regression. The idea would be to take into account the normals not only at the vertex, but also at its neighborhood, with a suitable inverse-distance weighting scheme (possibly similar to the pre-filtering weighting scheme, see Fig. 3.6).

Surfaces. To obtain the initial tessellation, we parametrize each cycle by projection to a plane. The main reason why we use planar projection is that it introduces smaller conformal distortion compared to methods which map the cycle to a circle or a planar polygon (see the examples in Section 4.4.2). This approach has limitations since it cannot be used if the projection is not injective. The solution might be to use an existing method for triangulation of 3D curves [ZJC13]. Alternatively, the initial tessellation might be computed with help of a restricted Delaunay triangulation (DT). Restricted DT is a subcomplex of a three-dimensional DT which serves as a triangulation of a smooth surface embedded in $\mathbb{R}^3$ [CDS12, Chap. 13]. Restricted DT requires a sufficiently dense sample of points that could be sampled from a quadratic surface approximating the input curves and the input normals.

Devices. A next generation prototype of Morphorider currently in development might help in resolving some of the problems we experienced during acquisition. The current Morphorider is a proof of concept and suffers from construction drawbacks. Its relatively big size inhibits the acquisition; we often found it hard to manipulate, and some regions with high curvature (in absolute value) could not have been scanned (see Fig. 1.15). Another problem is that the distance-measuring wheel is not aligned with the sensor unit – in practice this means that there is an offset in the orientation

measurement. Moreover, the device relies on a magnetometer and cannot be used around ferromagnetic objects.

The biggest limitation of the smartphone setup is the inability to measure distances – these need to be estimated or obtained manually.

3D sketching with a smartphone remains a difficult task with current tools, such as ShapeIt. Nevertheless, we believe this method has a great potential, conditioned by further development of the algorithms. The ultimate goal would be a smartphone application that would allow the user to sketch 3D shapes in real time, without needing to specify the topology manually. First step in this direction might be an algorithm for automatic inference of nodes in the network, for instance using the continuity of network normals.

A

Riemannian connection & covariant derivative

This appendix includes additional results from Riemannian geometry not included in Section 2.4.

Definition A.1 (Connection, covariant derivative)**.** *For a Riemannian manifold $\mathcal{M}$, denote by $\tau(\mathcal{M})$ the set of all smooth vector fields over $\mathcal{M}$ and by $\mathrm{C}^\infty(\mathcal{M})$ the set of all smooth scalar fields over $\mathcal{M}$. A* connection *on $\mathcal{M}$ is a bilinear map*

$$\nabla : \tau(\mathcal{M}) \times \tau(\mathcal{M}) \to \tau(\mathcal{M}) \quad : \quad (\mathbf{X}, \mathbf{Y}) \mapsto \nabla_{\mathbf{X}}\mathbf{Y}$$

which satisfies the following three conditions:

(i) $\mathrm{C}^\infty(\mathcal{M})$*-linearity in the first variable :* $\nabla_{f\mathbf{X}+g\mathbf{Y}}\mathbf{Z} = f\,\nabla_{\mathbf{X}}\mathbf{Z} + g\nabla_{\mathbf{Y}}\mathbf{Z}$

(ii) $\mathbb{R}$*-linearity in the second variable :* $\nabla_{\mathbf{X}}(\alpha\mathbf{Y} + \beta\mathbf{Z}) = \alpha\nabla_{\mathbf{X}}\mathbf{Y} + \beta\nabla_{\mathbf{X}}\mathbf{Z}$

(iii) Product rule : $\nabla_{\mathbf{X}}(f\,\mathbf{Y}) = \mathrm{D}f\,[\mathbf{X}]\mathbf{Y} + f\,\nabla_{\mathbf{X}}\mathbf{Y}$

where $\mathbf{X}, \mathbf{Y}, \mathbf{Z} \in \tau(\mathcal{M})$ are vector fields on $\mathcal{M}$, $f, g \in \mathrm{C}^\infty(\mathcal{M})$ are scalar fields on $\mathcal{M}$, and $\alpha, \beta \in \mathbb{R}$. The notation $\mathrm{D}f\,[\mathbf{X}]$ is a shorthand for a scalar field on $\mathcal{M}$ defined as $\mathrm{D}f(\mathbf{x})\left[\mathbf{X}(\mathbf{x})\right]$ for all $\mathbf{x}$ on $\mathcal{M}$.

The vector field $\nabla_{\mathbf{X}}\mathbf{Y}$ *is called the* covariant derivative *of* $\mathbf{Y}$ *with respect to* $\mathbf{X}$ *for the connection* ∇.

Theorem A.2 (Fundamental theorem of Riemannian geometry)**.** *On any Riemannian manifold* $\mathcal{M}$*, there exists a unique symmetric torsion-free metric connection* ∇ *called the* Levi-Civita *or the* Riemannian connection.

Looking at the rather technical definition of covariant derivative, we should keep in mind its geometric interpretation: for each point $\mathbf{x}$, $(\nabla_{\mathbf{X}}\mathbf{Y})_{\mathbf{x}}$ is a vector which describes how the vector field $\mathbf{Y}$ changes in the direction $\mathbf{X}_{\mathbf{x}}$. It is therefore a generalization of the classical directional derivative from Euclidean spaces. For submanifolds immersed in $\mathbb{R}^n$, the following lemma provides a recipe on computing the covariant derivative as a projection of the (Euclidean) directional derivative. This is analogical to the computation of the Riemannian gradient from Lema 2.22.

Lemma A.3 (Covariant derivative as projection)**.** *Let* $\mathbf{X}, \mathbf{Y} \in \tau(\mathcal{M})$ *be two vector fields on the Riemannian submanifold* $\mathcal{M}$ *of* $\mathbb{R}^n$ *and* ∇ *is the Levi-Civita connection on* $\mathcal{M}$*. Then for all* $\mathbf{x} \in \mathcal{M}$*,*

$$(\nabla_{\mathbf{X}}\mathbf{Y})_{\mathbf{x}} = \mathrm{P}_{\mathbf{x}}\left(\mathrm{D}\mathbf{Y}(\mathbf{x})[\mathbf{X}_{\mathbf{x}}]\right)$$

where $\mathrm{D}\mathbf{Y}(\mathbf{x})[\mathbf{X}_{\mathbf{x}}]$ *denotes the classical directional derivative in the (Euclidean) space* $\mathbb{R}^n$ *and* $\mathrm{P}_{\mathbf{x}}$ *is the projection to the tangent space* $\mathrm{T}_{\mathbf{x}}\mathcal{M}$*.*

Definition A.4 (Acceleration along a curve)**.** *Let* $\gamma : \mathbb{R} \to \mathcal{M}$ *be a* C^2 *curve. The acceleration along* γ *is given by*

$$\frac{\mathrm{D}^2}{\mathrm{d}t^2}\gamma(t) = \frac{\mathrm{D}}{\mathrm{d}t}\dot{\gamma}(t) = \nabla_{\dot{\gamma}}\dot{\gamma}.$$

B

Notes on quaternion filtering

This appendix features additional ideas related to filtering of orientations in the space of quaternions. Even though the following computations turned out to be of little practical interest in our specific setup, we found it interesting to include the results. Hopefully, they will provide some insight into how the quaternion representation is related to the Darboux frame, and also why we chose to filter data using splines on SO(3).

In the following, we suppose the rotation matrix $\mathrm{A} = [\mathbf{t}\ \mathbf{N}\ \mathbf{B}]$ represents the Darboux frame $\mathcal{D} = \{\mathbf{t}, \mathbf{N}, \mathbf{B}\}$ of a curve on a surface. The surface normal $\mathbf{N}$ is obtained using projection on the second column of the matrix: $\mathbf{N} = \mathrm{A}|_{\mathbf{N}} = \mathrm{A}\mathbf{e}_2$. Recall that a unit quaternion $\mathbf{q} = (x, y, z, w) \in \mathbb{S}^3$ is related to the Darboux frame $\mathrm{A} \in \mathrm{SO}(3)$ by

$$\mathrm{A}(\mathbf{q}) = \begin{pmatrix} 2w^2 + 2x^2 - 1 & 2xy + 2wz & 2xz - 2wy \\ 2xy - 2wz & 2w^2 + 2y^2 - 1 & 2yz + 2wx \\ 2xz + 2wy & 2yz - 2wx & 2w^2 + 2z^2 - 1 \end{pmatrix}.$$

The coordinates of $\mathbf{N} = \mathrm{A}|_{\mathbf{N}} = (N_x, N_y, N_z)^\intercal$ can be expressed as function of $\mathbf{q}$ using quadratic forms defined in $\mathbf{q}$ (interpreted as a vector in $\mathbb{R}^4$):

$$N_x = \mathbf{q}^\intercal \mathrm{K}_x \mathbf{q}, \quad N_y = \mathbf{q}^\intercal \mathrm{K}_y \mathbf{q}, \quad N_z = \mathbf{q}^\intercal \mathrm{K}_z \mathbf{q},$$

with symmetric matrices

$$\mathrm{K}_x = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \quad \mathrm{K}_y = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \mathrm{K}_z = \begin{pmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{pmatrix}.$$

Filtering in parametric space. Unit quaternions, being a set of points on the 3-sphere, are parametrizable using *hyperspherical coordinates*. Hyperspherical coordinates generalize polar ($d = 1$) and spherical ($d = 2$) coordinates to d-spheres of any dimension $d > 2$. For a unit quaternion $\mathbf{q} \in \mathbb{S}^3$, hyperspherical coordinates (ϕ, θ, ψ) are related to the Cartesian coordinates (x, y, z, w) by

$$\left.\begin{aligned} w &= \cos\phi \\ x &= \sin\phi\cos\theta \\ y &= \sin\phi\sin\theta\cos\psi \\ z &= \sin\phi\sin\theta\sin\psi \end{aligned}\right\} \quad \begin{aligned} \phi &\in [0, \pi] \\ \theta &\in [0, \pi] \\ \psi &\in [0, 2\pi] \end{aligned} \quad .$$

Given a noisy sequence of quaternions, one way to approach the filtering would be to parametrize the quaternions via hyperspherical coordinates and filter in the parametric space $[0, \pi] \times [0, 2\pi] \times [0, 2\pi]$. In order for this approach to be valid, the periodicity of the three angle functions needs to be taken into account.

This simple filtering method has several drawbacks, for instance the fact that hyperspherical coordinates distort the metric of the space and are not injective. Moreover, an attempt to introduce normal constraints into this kind of filtering would result in a system with non-polynomial equations, which could only be resolved using numerical methods.

Average quaternion with constrained normal. In Section 3.5.2, the average rotation was defined by minimizing the quaternion energy

$$\max_{\mathbf{q} \in \mathbb{S}^3} \operatorname{trace}\left(\mathrm{A}\mathrm{B}^\intercal\right) = \max_{\mathbf{q} \in \mathbb{S}^3} \mathrm{E}_{\text{interp}}$$

where $\mathrm{B} = \sum_{i=1}^{n} \alpha_i \mathrm{A}_i$. A similar approach might be used to compute the average rotation while constraining the surface normal. Consider a modified energy, which includes an additional energy term controlled by the weight $w \in [0, 1]$:

$$\max_{\mathbf{q} \in \mathbb{S}^3} (1 - w)\mathrm{E}_{\text{interp}} + w\mathrm{E}_{\text{normal}},$$

where $\mathrm{E}_{\text{normal}} = \left\langle \mathbf{N}, \widehat{\mathbf{N}} \right\rangle$ if $\widehat{\mathbf{N}}$ is the constrained unit surface normal. The energy $\mathrm{E}_{\text{normal}}$ is expressed via the following quadratic form:

$$\mathrm{E}_{\text{normal}} = \mathbf{q}^\intercal \left(\widehat{N}_x \mathrm{K}_x + \widehat{N}_y \mathrm{K}_y + \widehat{N}_z \mathrm{K}_z \right) \mathbf{q}.$$

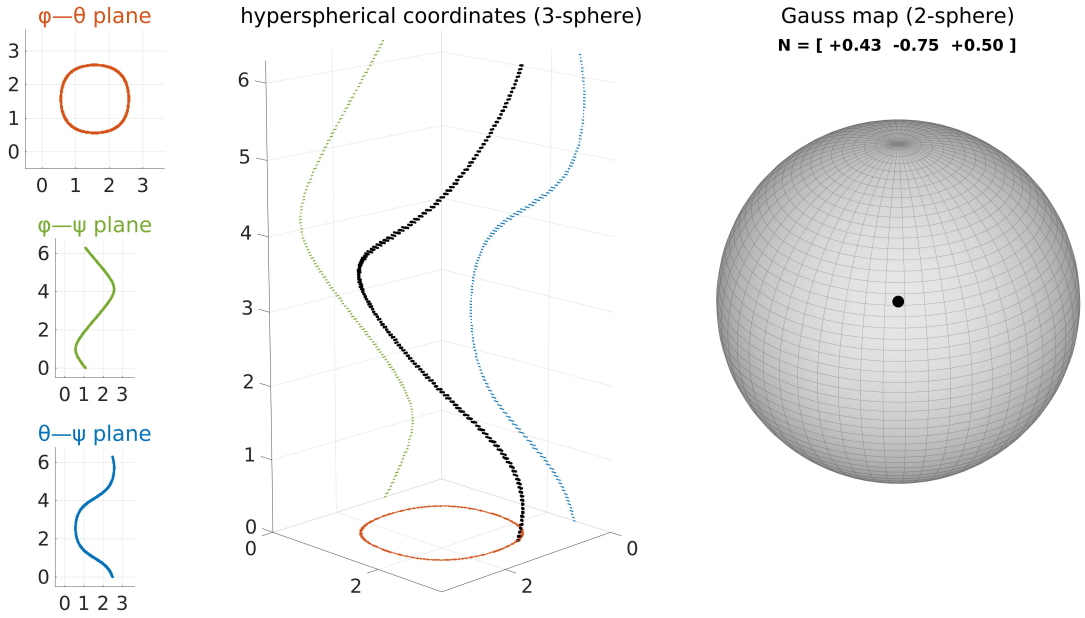


Figure B.1: Fixing a surface normal $\widehat{\mathbf{N}}$ (*right*), quaternions representing Darboux frames $[\mathbf{t}\ \widehat{\mathbf{N}}\ \mathbf{B}]$ correspond to a 3D curve (*left, black*) in the parametric space (ϕ, θ, ψ).

Since $\mathrm{E}_{\mathrm{interp}} = \mathbf{q}^{\mathsf{T}}\mathrm{K}\mathbf{q}$ with K defined in Proposition 3.4, it follows that

$$(1 - w)\mathrm{E}_{\mathrm{interp}} + w\mathrm{E}_{\mathrm{normal}} = \mathbf{q}^{\mathsf{T}}\widehat{\mathrm{K}}\mathbf{q}$$

where

$$\widehat{\mathrm{K}} = (1 - w)\,\mathrm{K} + w\left(\widehat{N}_x\mathrm{K}_x + \widehat{N}_y\mathrm{K}_y + \widehat{N}_z\mathrm{K}_z\right).$$

This mean that the modified energy is maximized if $\mathbf{q}$ is equal to the (unit) eigenvector of $\widehat{\mathrm{K}}$ corresponding to the largest eigenvalue.

This modification of the q method is almost identical to the algorithm presented in Section 3.5.2 without introducing any overhead. While using this modification of the averaging scheme could be used to constrain normals at nodes, it has several problems. First, the constrained normal $\widehat{\mathbf{n}}$ is not known *a priori*. A simple solution to this issue is to compute $\widehat{\mathbf{n}}$ as the average of all normals at the given node. Second, constrained averaging at isolated points creates sharp cusps in the orientations, which is not desirable. Our attempt to solve this issue was to constrain the normal within a fixed distance from a node, with inverse distance weights w. This approach turned out to be difficult to control.

In contrast, regression on SO(3) used for filtering orientations in Section 3.5 does not require the normals to be specified explicitly, and provides an elegant solution to the constrained filtering problem.

C

G^1 interpolation of discrete networks with normals

This appendix describes our previously unpublished method for interpolating a discrete network with a set of cubic splines. The result is a network of splines, which is tangent-plane continuous at nodes. This method is inspired by Nielson's *minimum norm networks* [Nie83].

Our surfacing method from Chapter 4 assumes a uniformly sampled dataset $\mathbf{x}_i$ and $\mathbf{N}_i$ on the input. The algorithm described in this appendix can be used to interpolate a non-uniformly sampled network with G^1 cubic splines. Sampling the computed cubic splines uniformly with respect to arc length provides suitable input for the method from Chapter 4. This additional procedure makes our surfacing method applicable to general curve networks with normals.

We define the G^1 interpolant by minimizing the integral of curvature for the whole network:

$$\mathrm{E}_{\mathrm{curv}}(\Gamma) = \sum_{\gamma \in \Gamma} \int_0^L \left\| \mathbf{x}'' \right\|^2 \mathrm{d}t. \tag{C.1}$$

It is known that this approach produces cubic splines [De 78; Nie83]. Boundary conditions – endpoint tangent vectors – need to be specified to fix all degrees of freedom.

It is precisely the computation of the endpoint tangent vectors that we address in this appendix. Since our input network is sampled from a surface, we work with the added constraint that endpoint tangent vectors of adjacent curves live in the same tangent plane. We use this constraint to guide our optimization, producing tangent-plane continuous curve networks (Fig. C.2).

We represent piecewise-cubic spline in the Hermite form. In the following section, we summarize the main elements of cubic Hermite interpolation.

C.1 Hermite interpolation and curvature energy

Hermite curve. Denote by $\gamma = \mathbf{h}[\mathbf{x}_0, \mathbf{x}_1, \mathbf{t}_0, \mathbf{t}_1]$ a cubic Hermite curve parametrized by $\mathbf{h} : [0,1] \to \mathbb{R}^3$. At endpoints, Hermite curve interpolates the points $\mathbf{x}_0, \mathbf{x}_1$ and its tangent matches the vectors $\mathbf{t}_0, \mathbf{t}_1$:

$$\mathbf{h}(0) = \mathbf{x}_0, \quad \mathbf{h}(1) = \mathbf{x}_1, \quad \mathbf{h}'(0) = \mathbf{t}_0, \quad \mathbf{h}'(1) = \mathbf{t}_1.$$

Explicitly, the parametrization is given by

$$\mathbf{h}(t) = \mathbf{x}_0\, h_{00}(t) + \mathbf{t}_0\, h_{10}(t) + \mathbf{x}_1\, h_{01}(t) + \mathbf{t}_1\, h_{11}(t), \quad t \in [0,1],$$

where the cubic Hermite polynomials are given by

$$h_{00}(t) = 2t^3 - 3t^2 + 1, \quad h_{10}(t) = t^3 - 2t^2 + t, \quad h_{01}(t) = -2t^3 + 3t^2, \quad h_{11}(t) = t^3 - t^2.$$

Denote by $\mathbf{a}$ and $\mathbf{b}$ the following vectors:

$$\mathbf{a} = 6\left(2(\mathbf{x}_0 - \mathbf{x}_1) + \mathbf{t}_0 + \mathbf{t}_1\right), \quad \mathbf{b} = -2\left(3(\mathbf{x}_0 - \mathbf{x}_1) + 2\mathbf{t}_0 + \mathbf{t}_1\right). \tag{C.2}$$

Then the scalar curvature of $\mathbf{h}$ is

$$\mathbf{h}''(t) = t\mathbf{a} + \mathbf{b}, \quad t \in [0,1].$$

The curvature energy of the curve $\mathbf{h}$ is defined as

$$\mathrm{E}_{\mathrm{curv}}(\mathbf{x}_0, \mathbf{x}_1, \mathbf{t}_0, \mathbf{t}_1) = \int_0^1 \left\|\mathbf{h}''\right\|^2 \mathrm{d}t = \int_0^1 \|t\mathbf{a} + \mathbf{b}\|^2 \,\mathrm{d}t = \tfrac{1}{3}\|\mathbf{a}\| + \langle \mathbf{a}, \mathbf{b}\rangle + \|\mathbf{b}\|. \tag{C.3}$$

The change of the curvature energy with respect to a vector $\mathbf{e}$ is

$$\frac{\partial \mathrm{E}_{\mathrm{curv}}}{\partial \mathbf{e}_i} = \left\langle \mathbf{a}, \tfrac{2}{3}\frac{\partial \mathbf{a}}{\partial \mathbf{e}_i} + \frac{\partial \mathbf{b}}{\partial \mathbf{e}_i} \right\rangle + \left\langle \mathbf{b}, \frac{\partial \mathbf{a}}{\partial \mathbf{e}_i} + 2\frac{\partial \mathbf{b}}{\partial \mathbf{e}_i} \right\rangle. \tag{C.4}$$

Hermite spline. Given a sequence of points $\mathbf{x}_0, \ldots, \mathbf{x}_{n+1}$ and endpoint tangent vectors $\mathbf{t}_0, \mathbf{t}_{n+1}$, the unique C^2 cubic spline γ that interpolates $\mathbf{x}_i$ and $\mathbf{t}_i$ has $n+1$ Hermite segments:

$$\mathbf{h}\left[\mathbf{x}_i, \mathbf{x}_{i+1}, \mathbf{t}_i, \mathbf{t}_{i+1}\right](t), \quad t \in [0,1], \quad i = 0, \ldots, n.$$

Fixing the end vectors $\mathbf{t}_0$ and $\mathbf{t}_n$, the inner tangents $\mathbf{t}_1, \ldots, \mathbf{t}_n$ are given by the $n \times n$ tridiagonal system

$$\underbrace{\begin{pmatrix} 4 & 1 & & & \\ 1 & 4 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 4 & 1 \\ & & & 1 & 4 \end{pmatrix}}_{\mathrm{A}} \underbrace{\begin{pmatrix} \mathbf{t}_1 \\ \mathbf{t}_2 \\ \vdots \\ \mathbf{t}_{n-1} \\ \mathbf{t}_n \end{pmatrix}}_{\mathrm{T}} = 3 \underbrace{\begin{pmatrix} \mathbf{x}_2 - \mathbf{x}_0 \\ \mathbf{x}_3 - \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n - \mathbf{x}_{n-2} \\ \mathbf{x}_{n+1} - \mathbf{x}_{n-1} \end{pmatrix}}_{\mathrm{X}} - \underbrace{\begin{pmatrix} \mathbf{t}_0 \\ 0 \\ \vdots \\ 0 \\ \mathbf{t}_{n+1} \end{pmatrix}}_{\mathrm{B}}. \tag{C.5}$$

Since the positions $\mathbf{x}_i$ are fixed, the derivative of this system with respect to some vector $\mathbf{e}$ is

$$\mathrm{A}\frac{\partial}{\partial \mathbf{e}}\mathrm{T} = -\frac{\partial}{\partial \mathbf{e}}\mathrm{B}. \tag{C.6}$$

C.2 Network of Hermite splines

Each curve in the input curve network Γ (Section 4.2) is represented by a Hermite spline interpolating the input positions $\mathbf{x}_i$. We will now compute the vectors $\mathbf{t}_i$ needed for Hermite interpolation so that the tangents at intersections lie in the same tangent plane.

We will use the following notation. Nodes in the curve network are denoted by v_i or v; number of nodes is $|\mathcal{V}| = \mathrm{V}$. Fixing a node v_i, we define the set of neighboring nodes as

$$\mathrm{link}\,(v_i) = \left\{v_j : \ v_i, v_j \text{ are the two endnodes of some segment } \gamma \in \Sigma\right\}. \tag{C.7}$$

The degree of v_i is the number of neighboring nodes,

$$d_i = \mathrm{degree}(v_i) = \left|\mathrm{link}\,(v_i)\right|.$$

For each node v_i, our algorithm needs an ordering of the neighboring vertices v_j around v_i. This ordering might for instance be provided by the user or determined by sorting the segments incident to v_i in the tangent plane defined by the surface normal $\mathbf{N}$ at v_i. In any case, we assume this order is known – it is a permutation with d_i elements:

$$\text{for } \ v_j \in \mathrm{link}\,(v_i) : \mu_{ij} = \mu_i(v_j) \in [1, \ldots, d_i] \ \text{ and } \ \mu_{ij_1} = \mu_{ij_2} \ \text{ iff } \ j_1 = j_2.$$

If γ is a segment with endnodes v_i and v_j, the input points and normals in this segment will be denoted by

$$\left\{\mathbf{x}_0^{ij},\ \mathbf{N}_0^{ij}\right\},\quad \left\{\mathbf{x}_1^{ij},\ \mathbf{N}_1^{ij}\right\},\quad \dots\quad \left\{\mathbf{x}_n^{ij},\ \mathbf{N}_n^{ij}\right\},\quad \left\{\mathbf{x}_{n+1}^{ij},\ \mathbf{N}_{n+1}^{ij}\right\},$$

where $n = n_{ij} = n(v_i, v_j)$ is the number of inner points in the spline connecting v_i and v_j. The unknown tangents are

$$\mathbf{t}_0^{ij},\quad \mathbf{t}_1^{ij},\quad \dots\quad \mathbf{t}_n^{ij},\quad \mathbf{t}_{n+1}^{ij}.$$

We often omit the upper indices ^{ij} when they are clear from the context.

The inner tangent vectors needed for Hermite interpolation are calculated by solving the System (C.5). To that end, the tangent vectors need to be fixed at nodes. Let $\mathbf{e}_i^u, \mathbf{e}_i^v \in \mathbb{R}^3$ be a linear basis of the tangent plane at node v_i. For each segment $\gamma = (v_i, v_j)$ starting at node v_i and ending at node v_j, we define the first tangent vector as

$$\mathbf{t}_0^{ij} = \underbrace{\cos\left(2\pi\tfrac{\mu_{ij}}{d_i}\right)}_{\alpha_{ij}}\mathbf{e}_i^u + \underbrace{\sin\left(2\pi\tfrac{\mu_{ij}}{d_i}\right)}_{\beta_{ij}}\mathbf{e}_i^v\,. \tag{C.8}$$

Recall that d_i is the degree of the node v_i and $\mu_{ij} \in [1, \dots, d_i]$ is the order of the node v_j with respect to v_i. Note that the last tangent vector of $\gamma = (v_i, v_j)$ is obtained by reversing the direction of the first tangent vector at v_j:

$$\mathbf{t}_{n+1}^{ij} = -\mathbf{t}_0^{ji} = -(\alpha_{ji}\mathbf{e}_j^u + \beta_{ji}\mathbf{e}_j^v).$$

Having fixed the endpoint tangent vectors $\mathbf{t}_0, \mathbf{t}_{n+1}$, we describe in the following section the computation of the basis vectors $\mathbf{e}_i^u, \mathbf{e}_i^v$.

C.3 Optimization

The basis vectors $\mathbf{e}_i^u, \mathbf{e}_i^v$ are the only missing piece of information in order to have a well-defined spline network, which is tangent plane continuous. For the moment, let us set aside the input normal vectors $\mathbf{N}_0^{ij}$. The normal vectors will be included in the computation in Section C.4.

To compute the basis $\mathbf{e}_i^u, \mathbf{e}_i^v$ for each node, we minimize the curvature energy for the whole spline network:

$$\mathrm{E}_{\mathrm{curv}} = \sum_{\gamma\in\Gamma} \mathrm{E}_\gamma \tag{C.9}$$

where E_γ is the energy of the spline γ defined as the sum of energies of the Hermite segments:

$$\mathrm{E}_\gamma = \sum_{k=0}^{n} \mathrm{E}_\gamma^k. \tag{C.10}$$

Recall that the energy of a Hermite segment from Eq. (C.3) is given by

$$\mathrm{E}_\gamma^k = \int_0^1 \left\| \mathbf{h}'' \left[\mathbf{x}_k, \mathbf{x}_{k+1}, \mathbf{t}_k, \mathbf{t}_{k+1} \right] \right\|^2 \mathrm{d}t = \tfrac{1}{3} \left\| \mathbf{a}_k \right\| + \left\langle \mathbf{a}_k, \mathbf{b}_k \right\rangle + \left\| \mathbf{b}_k \right\|. \tag{C.11}$$

The total spline network energy $\mathrm{E}_{\mathrm{curv}}$ is minimized by solving the linear system of 2V equations – two for each network node:

$$\frac{\partial \mathrm{E}_{\mathrm{curv}}}{\partial \mathbf{e}_i^u} = \mathbf{0}, \quad \frac{\partial \mathrm{E}_{\mathrm{curv}}}{\partial \mathbf{e}_i^v} = \mathbf{0}, \quad i = 1, \ldots, \mathrm{V} = |\mathcal{V}|.$$

The partial derivative of the total energy w.r.t. $\mathbf{e}_i = \mathbf{e}_i^u$ or $\mathbf{e}_i^v$ is the sum of partial derivatives of segments:

$$\frac{\partial \mathrm{E}_{\mathrm{curv}}}{\partial \mathbf{e}_i} = \frac{\partial \left(\sum_{\gamma \in \Gamma} \mathrm{E}_\gamma \right)}{\partial \mathbf{e}_i} = \sum_{\gamma \in \Gamma} \frac{\partial \mathrm{E}_\gamma}{\partial \mathbf{e}_i} = \sum_{\gamma \in \Gamma} \sum_{k=0}^{n} \frac{\partial \mathrm{E}_\gamma^k}{\partial \mathbf{e}_i}.$$

The derivative of a single segment from Eq. (C.4) is

$$\frac{\partial \mathrm{E}_\gamma^k}{\partial \mathbf{e}_i} = \left\langle \mathbf{a}_k, \frac{2}{3} \frac{\partial \mathbf{a}_k}{\partial \mathbf{e}_i} + \frac{\partial \mathbf{b}_k}{\partial \mathbf{e}_i} \right\rangle + \left\langle \mathbf{b}_k, \frac{\partial \mathbf{a}_k}{\partial \mathbf{e}_i} + 2 \frac{\partial \mathbf{b}_k}{\partial \mathbf{e}_i} \right\rangle = 2 \frac{\partial \mathbf{t}_{k+1}}{\partial \mathbf{e}_i} \mathbf{a}_k + 2 \left(\frac{\partial \mathbf{t}_{k+1}}{\partial \mathbf{e}_i} - \frac{\partial \mathbf{t}_k}{\partial \mathbf{e}_i} \right) \mathbf{b}_k \tag{C.12}$$

where the vectors $\mathbf{a}_k$ and $\mathbf{b}_k$ are defined in Eq. (C.2) as

$$\mathbf{a}_k = 12 \left(\mathbf{x}_k - \mathbf{x}_{k+1} \right) + 6\mathbf{t}_k + 6\mathbf{t}_{k+1}, \qquad \mathbf{b}_k = -6 \left(\mathbf{x}_k - \mathbf{x}_{k+1} \right) - 4\mathbf{t}_k - 2\mathbf{t}_{k+1}.$$

Plugging these definitions into Eq. (C.12) yields

$$\frac{1}{4} \frac{\partial \mathrm{E}_\gamma^k}{\partial \mathbf{e}_i} = 3 \left(\mathbf{x}_k - \mathbf{x}_{k+1} \right) \left(\frac{\partial \mathbf{t}_k}{\partial \mathbf{e}_i} + \frac{\partial \mathbf{t}_{k+1}}{\partial \mathbf{e}_i} \right) + \mathbf{t}_k \left(2 \frac{\partial \mathbf{t}_k}{\partial \mathbf{e}_i} + \frac{\partial \mathbf{t}_{k+1}}{\partial \mathbf{e}_i} \right) + \mathbf{t}_{k+1} \left(\frac{\partial \mathbf{t}_k}{\partial \mathbf{e}_i} + 2 \frac{\partial \mathbf{t}_{k+1}}{\partial \mathbf{e}_i} \right). \tag{C.13}$$

To compute the partial derivatives $\partial \mathbf{t}_k / \partial \mathbf{e}_i$, denote $\xi_k = \partial \mathbf{t}_k / \partial \mathbf{e}_i^u$ and $\chi_k = \partial \mathbf{t}_k / \partial \mathbf{e}_i^v$. From Eq. (C.8), we have

$$\xi_0 = \alpha_{ij}, \quad \chi_0 = \beta_{ij}, \quad \xi_{n+1} = 0, \quad \chi_{n+1} = 0. \tag{C.14}$$

Now, plugging $\mathbf{e}_i = \mathbf{e}_i^u$ into Eq. (C.6) yields

$$\begin{aligned}
4\,\xi_1 \;+\; \xi_2 &= -\alpha_{ij}, \\
\xi_1 \;+\; 4\,\xi_2 \;+\; \xi_3 &= 0, \\
&\ddots \; \ddots \; \ddots \qquad \vdots \\
\xi_{n-2} \;+\; 4\xi_{n-1} \;+\; \xi_n &= 0, \\
\xi_{n-1} \;+\; 4\xi_n &= 0.
\end{aligned}$$

An analogical system is obtained by replacing ξ_k with χ_k and α_{ij} with β_{ij}. Both systems can be directly solved for arbitrary n, and the solution is

$$\xi_k = \frac{c_{n+1-k}}{c_{n+1}}\alpha_{ij}, \qquad \chi_k = \frac{c_{n+1-k}}{c_{n+1}}\beta_{ij}, \qquad k = 1, \dots, n,$$

with c_i given by the recurrence relation

$$c_0 = 0\,, \quad c_1 = 1\,, \quad c_{k-1} + 4\,c_k + c_{k+1} = 0\,. \tag{C.15}$$

Using Eq. (C.14), we can actually write

$$\xi_k = \frac{c_{n+1-k}}{c_{n+1}}\alpha_{ij}, \qquad \chi_k = \frac{c_{n+1-k}}{c_{n+1}}\beta_{ij}, \qquad k = 0, \dots, n+1, \tag{C.16}$$

Plugging the computed partial derivatives into Eq. (C.13) yields

$$\begin{aligned}\frac{1}{4}\frac{\partial \mathrm{E}_\gamma^k}{\partial \mathbf{e}_i^u} &= 3(\mathbf{x}_k - \mathbf{x}_{k+1})(\xi_k + \xi_{k+1}) + \mathbf{t}_k(2\xi_k + \xi_{k+1}) + \mathbf{t}_{k+1}(\xi_k + 2\xi_{k+1}) \qquad \text{(C.17)}\\ &= \frac{\alpha_{ij}}{c_{n+1}}\left(3(\mathbf{x}_k - \mathbf{x}_{k+1})(c_{n-k+1} + c_{n-k}) + \mathbf{t}_k(2c_{n-k+1} + c_{n-k}) + \mathbf{t}_{k+1}(c_{n-k+1} + 2c_{n-k})\right)\end{aligned}$$

The expression for $\partial \mathrm{E}_\gamma^k / \partial \mathbf{e}_i^v$ is obtained by replacing α_{ij} with β_{ij}. Summing the contributions of all Hermite segments yields derivative of the energy of a spline γ:

$$\begin{aligned}\frac{\partial \mathrm{E}_\gamma}{\partial \mathbf{e}_i} = \sum_{k=0}^{n}\frac{\partial \mathrm{E}_\gamma^k}{\partial \mathbf{e}_i} = \frac{4y_{ij}}{c_{n+1}}\Big[3\sum_{k=0}^{n}(c_{n-k+1} + c_{n-k})(\mathbf{x}_k - \mathbf{x}_{k+1}) + \\ + \mathbf{t}_0(2c_{n+1} + c_n) + \mathbf{t}_1\underbrace{(c_{n+1} + 4c_n + c_{n-1})}_{=0} + \cdots + \mathbf{t}_n\underbrace{(c_2 + 4c_1 + c_0)}_{=0} + \mathbf{t}_{n+1}\underbrace{(c_1 + 2c_0)}_{=1}\Big]\end{aligned} \tag{C.18}$$

where either $\mathbf{e}_i = \mathbf{e}_i^u$, $y_{ij} = \alpha_{ij}$; or $\mathbf{e}_i = \mathbf{e}_i^v$, $y_{ij} = \beta_{ij}$. All inner tangent terms vanish since $(c_{k-1} + 4c_k + c_{k+1}) = 0$, and $(c_1 + 2c_0) = 1$; see Eq. (C.15). Denote the term independent from $\mathbf{e}_i$ by

$$C_\gamma = 3\sum_{k=0}^{n}\frac{c_{n-k+1} + c_{n-k}}{c_{n+1}}(\mathbf{x}_k - \mathbf{x}_{k+1}).$$

Using the definition of node tangent vectors from Eq. (C.8) gives

$$\begin{aligned}\frac{\partial \mathrm{E}_\gamma}{\partial \mathbf{e}_i} &= 4y_{ij}\Big[(2 + c_n/c_{n+1})\mathbf{t}_0 + 1/c_{n+1}\mathbf{t}_{n+1} + C_\gamma\Big] \qquad \text{(C.19)}\\ &= 4y_{ij}\Big[(2 + c_n/c_{n+1})(\alpha_{ij}\mathbf{e}_i^u + \beta_{ij}\mathbf{e}_i^v) - 1/c_{n+1}(\alpha_{ji}\mathbf{e}_j^u + \beta_{ji}\mathbf{e}_j^v) + C_\gamma\Big] \qquad \text{(C.20)}\end{aligned}$$

What about the derivative $\partial \mathrm{E}_{\mathrm{curv}}/\partial \mathbf{e}_i$ of the energy for the whole network? Importantly, only the curves adjacent to v_i affect the change of energy with respect to $\mathbf{e}_i$. Indeed, if $\gamma = (v_i, v_j)$ then

$$\frac{\partial \mathrm{E}_\gamma}{\partial \mathbf{e}_i} \neq \mathbf{0} \quad \text{iff} \quad v_j \in \mathrm{link}\,(v_i)\,.$$

Summing the energy (C.9) for all splines using the expression from Eq. (C.20) therefore yields a tensor-product linear system in $\mathbf{e}_i^u$, $\mathbf{e}_i^v$ of the form

$$\begin{bmatrix} \mathrm{A}_{11} & \dots & \mathrm{A}_{1\mathrm{V}} \\ \vdots & \ddots & \vdots \\ \mathrm{A}_{\mathrm{V}1} & \dots & \mathrm{A}_{\mathrm{VV}} \end{bmatrix} \begin{bmatrix} \mathrm{X}_1 \\ \vdots \\ \mathrm{X}_\mathrm{V} \end{bmatrix} = \begin{bmatrix} \mathrm{B}_1 \\ \vdots \\ \mathrm{B}_\mathrm{V} \end{bmatrix}. \tag{C.21}$$

Each A_{ij} is a 2×2 matrix, X_i is a 2×3 matrix, and B_i is a 2×3 matrix, defined by

$$\begin{aligned} \mathrm{A}_{ii} &= \sum\nolimits_{v_j \in \mathrm{link}(v_i)} -(2 + c_n/c_{n+1})(\alpha_{ij}\ \beta_{ij})^\intercal(\alpha_{ij}\ \beta_{ij}), \\ \mathrm{A}_{ij} &= \begin{cases} 1/c_{n+1}(\alpha_{ij}\ \beta_{ij})^\intercal(\alpha_{ji}\ \beta_{ji}) & : \quad v_j \in \mathrm{link}\,(v_i)\,, \\ 0_{2\times 2} & : \quad v_j \notin \mathrm{link}\,(v_i) \cup \{v_i\}\,, \end{cases} \\ \mathrm{B}_i &= \sum\nolimits_{v_j \in \mathrm{link}(v_i)} (\alpha_{ij}\ \beta_{ij})^\intercal \mathrm{C}_\gamma, \\ \mathrm{X}_i &= (\mathbf{e}_i^u\ \mathbf{e}_i^v)^\intercal. \end{aligned}$$

C.4 Constraining the normals

In the previous section, the basis vectors $\mathbf{e}_i^u, \mathbf{e}_i^v$ needed for Hermite interpolation were computed by unconstrained minimization of the curvature energy of the spline network. This approach ignores our knowledge of normals $\mathbf{N}_i$. In this section we modify the above optimization in order to respect the input normals at nodes. If $\mathbf{N}_i$ is the surface normal at node v_i, we want the basis vectors to lie in the plane orthogonal to $\mathbf{N}_i$, resulting in a set of linear constraints:

$$\min_{\mathbf{e}_i^u, \mathbf{e}_i^v} \mathrm{E}_{\mathrm{curv}} \qquad \text{s.t.} \qquad \mathbf{e}_i^u \cdot \mathbf{N}_i = 0, \quad \mathbf{e}_i^v \cdot \mathbf{N}_i = 0, \qquad i = 1, \dots, \mathrm{V}. \tag{C.22}$$

The two linear constraints per node are concisely written in matrix form as

$$\mathbf{N}_i^\intercal(\mathbf{e}_i^u\ \mathbf{e}_i^v) = \mathbf{N}_i^\intercal \mathrm{X}_i^\intercal = \mathbf{0}$$

and enforced using Lagrange multipliers.

Fig. C.2 shows results obtained using this method for input networks from Fig. C.1.

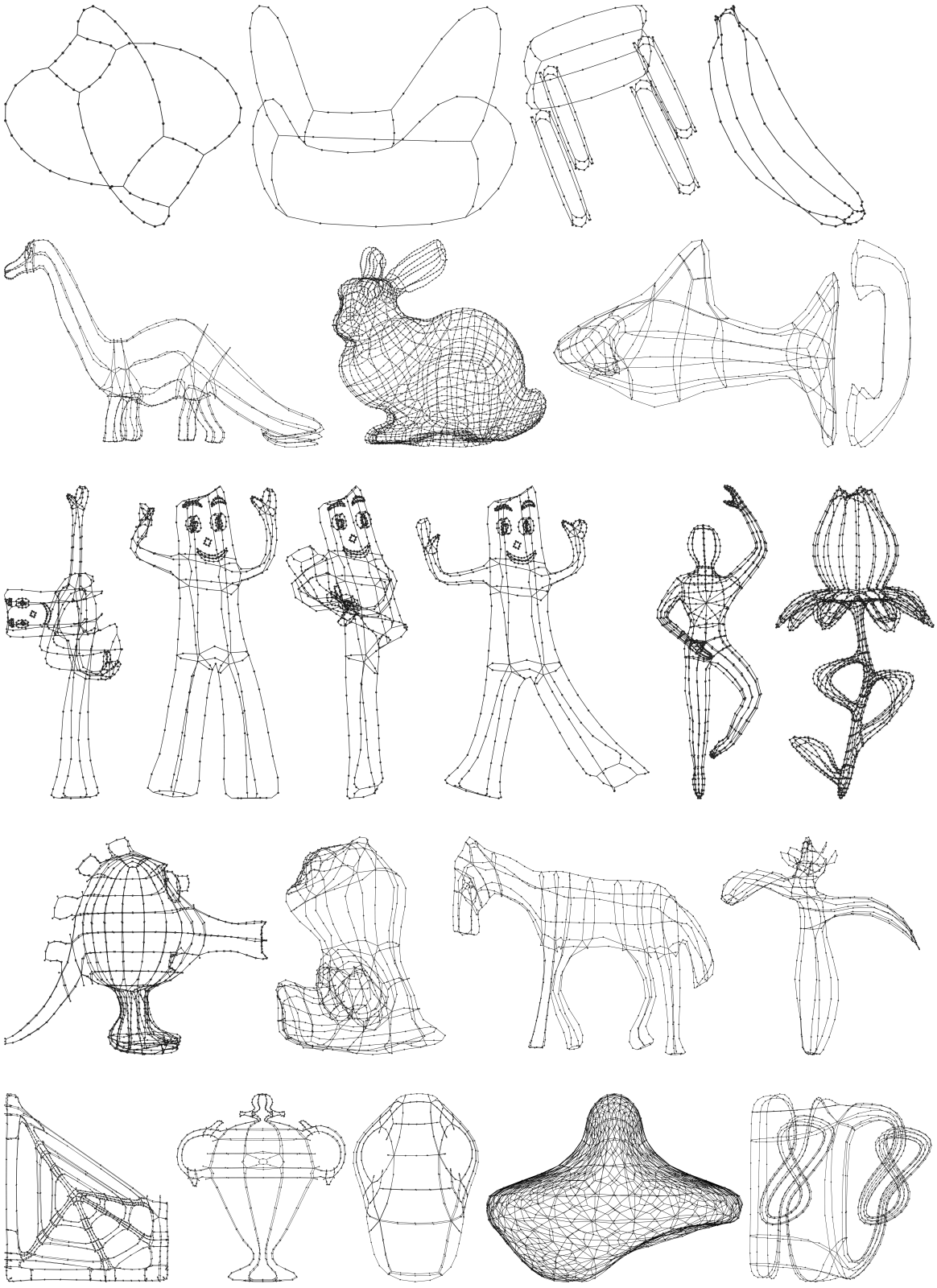


Figure C.1: Examples of input data for Hermite spline interpolation from Appendix C. All above networks of polylines were extracted from meshes provided by Cindy Grimm[1].

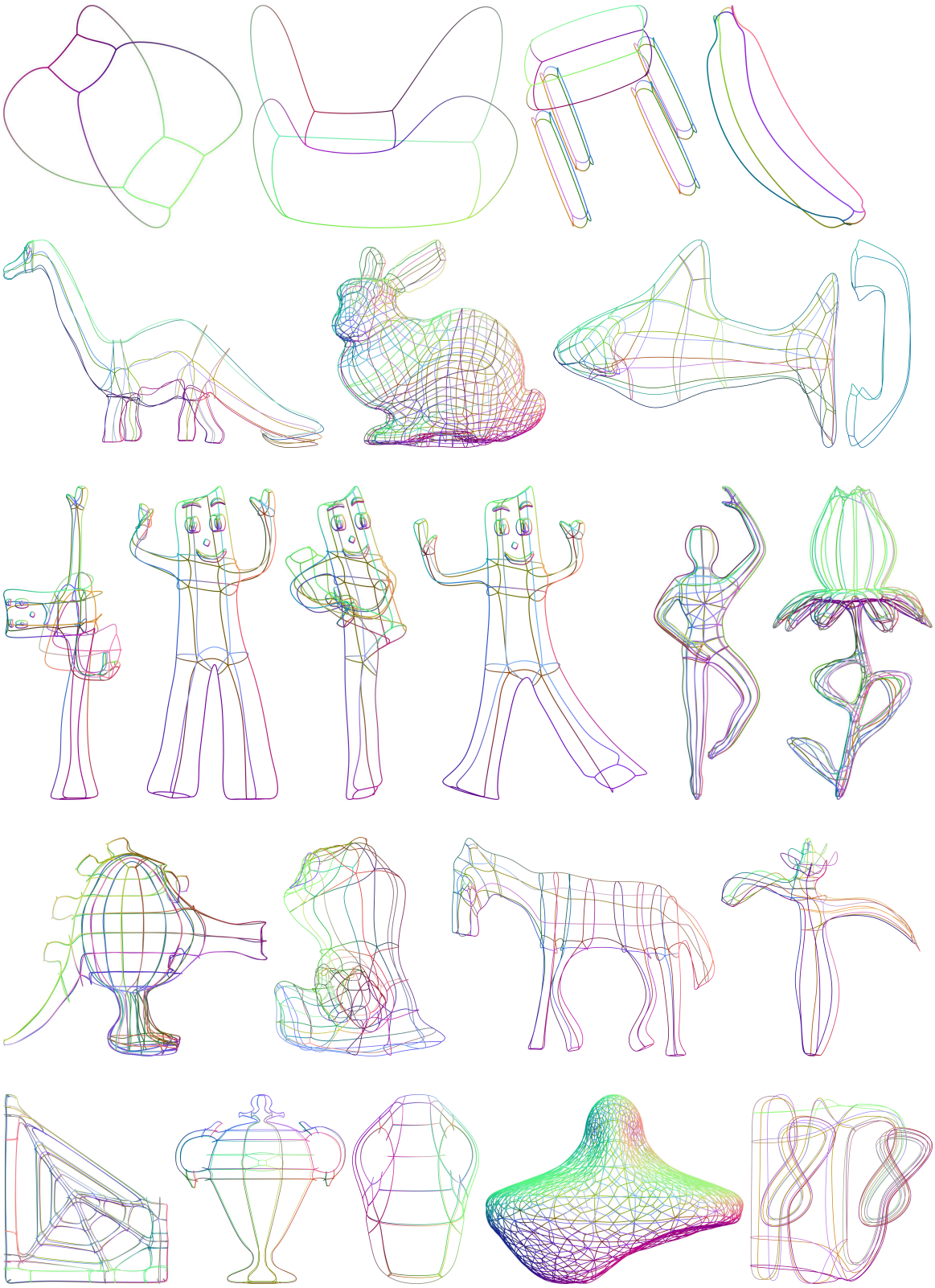


Figure C.2: Hermite spline networks with tangent plane continuity interpolating data from Fig. C.1 using the method from Appendix C.

Publications

Following is the list of articles I authored and conference talks I've given in the course of my doctoral research (October 2014 – October 2017). The articles include two conference papers and two journal papers. The journal article on *Shape from sensors* [Sta+17a] was published in the special issue of SMI 2017, where it was recognized with a Best Paper Award.

In the electronic version of this thesis, the colored titles are hyperlinked to their respective online version or publisher's page.

Articles

[Sta+16a] T. Stanko, S. Hahmann, G.-P. Bonneau, and N. Saguin-Sprynski. "Smooth Interpolation of Curve Networks with Surface Normals". In: *Proc. Eurographics – Short papers*. Lisbon, Portugal: Eurographics Association, May 2016, pp. 021–024 (p. 105).

[Sta+16b] T. Stanko, S. Hahmann, G.-P. Bonneau, and N. Saguin-Sprynski. "Surfacing Curve Networks with Normal Control". In: *Computers & Graphics* 60 (November 2016), pp. 1–8 (p. 105).

[Sta+17a] T. Stanko, S. Hahmann, G.-P. Bonneau, and N. Saguin-Sprynski. "Shape from Sensors: Curve Networks on Surfaces from 3D Orientations". In: *Computers & Graphics* 66 (August 2017). Proc. SMI, **Best Paper Award**, pp. 74–84 (p. 16, 53, 187).

[Sta+17b] T. Stanko, N. Saguin-Sprynski, L. Jouanet, S. Hahmann, and G.-P. Bonneau. "Morphorider: Acquisition and Reconstruction of 3D Curves with Mobile Sensors". In: *Proc. Conf. IEEE Sensors*. IEEE, October 2017, pp. 1170–1172 (p. 53).

Conference talks

[EG16] T. Stanko, S. Hahmann, G.-P. Bonneau, and N. Saguin-Sprynski. "Smooth Interpolation of Curve Networks with Surface Normals". Proc. Eurographics – Short papers. Lisbon, Portugal, May 10, 2016 (p. 105).

[GTMG16] T. Stanko, S. Hahmann, G.-P. Bonneau, and N. Saguin-Sprynski. "Smooth Interpolation of Curve Networks with Surface Normals". Proc. GTMG – Journées du Groupe de travail en Modélisation Géométrique. Dijon, France, March 23, 2016 (p. 105).

[jFIG17] T. Stanko, S. Hahmann, G.-P. Bonneau, and N. Saguin-Sprynski. "Surfacing Curve Networks with Normal Control". Proc. j•FIG – Journées françaises d'informatique graphique. Grenoble, France, December 2, 2016 (p. 105).

[SMI17] T. Stanko, S. Hahmann, G.-P. Bonneau, and N. Saguin-Sprynski. "Shape from Sensors: Curve Networks on Surfaces from 3D Orientations". SMI 2017 – Shape Modeling International. Berkeley, CA, June 22, 2017 (p. 53).

Bibliography

For reader's convenience, the bibliography is organized into multiple sections: books, theses, articles, online resources. In the electronic version of this thesis, the colored titles are hyperlinked to their respective electronic version, project page, or conference website.

Books

[AMS08] P.-A. Absil, R. Mahony, and R. Sepulchre. "Optimization Algorithms on Matrix Manifolds". Princeton University Press, 2008. ISBN: 978-0691132983 (p. 29).

[Ber03] M. Berger. "A Panoramic View of Riemannian Geometry". Springer Verlag, 2003. ISBN: 978-3642621215 (p. 29).

[Bot+10] M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, and B. Lévy. "Polygon Mesh Processing". CRC Press, 2010. ISBN: 978-1568814261 (p. 41, 48, 106, 113).

[CDS12] S.-W. Cheng, T. K. Dey, and J. Shewchuk. "Delaunay Mesh Generation". Chapman and Hall/CRC, 2012. ISBN: 978-1584887300 (p. 117, 169).

[De 78] C. De Boor. "A practical guide to splines". Vol. 27. Springer-Verlag New York, 1978. ISBN: 978-0-387-95366-3 (p. 178).

[doCa16] M. do Carmo. "Differential Geometry of Curves and Surfaces". 2nd ed. Dover Publications, December 2016. ISBN: 978-0486806990 (p. 13, 25, 26, 29).

[doCa92] M. do Carmo. "Riemannian Geometry". 4th ed. Birkhäuser, 1992. ISBN: 978-3764334901 (p. 25, 29).

[Far02] G. Farin. "Curves and Surfaces for CAGD: A Practical Guide". 5th ed. Morgan Kaufmann, 2002. ISBN: 978-1558607378 (p. 12, 53, 108, 110).

[Hat02] A. Hatcher. "Algebraic Topology". Cambridge University Press, 2002. ISBN: 978-0521795401 (p. 38).

[HC52] D. Hilbert and S. Cohn-Vossen. "Geometry and the Imagination". Chelsea Publishing House, 1952. ISBN: 0828400873 (p. 25).

[Mil65] J. Milnor. "Topology from the Differentiable Viewpoint". The University Press of Virginia, 1965. ISBN: 978-0813901817 (p. 29).

[Smi02] S. Smith. "Digital Signal Processing: A Practical Guide for Engineers and Scientists". Newnes, 2002. ISBN: 978-0750674447 (p. 65).

Theses

[Bou10] N. Boumal. "Discrete Curve Fitting on Manifolds". Master's thesis. Université catholique de Louvain, June 2010 (p. 29).

[Cur97] B. Curless. "New Methods for Surface Reconstruction from Range Images". PhD thesis. Stanford University, June 1997 (p. 2).

[Hua13] M. Huard. "Modélisation Géométrique et Reconstruction de Formes Equipées de Capteurs d'Orientation". PhD thesis. Université de Grenoble, September 2013 (p. 11, 13, 139).

[Jac13] A. Jacobson. "Algorithms and Interfaces for Real-time Deformation of 2D and 3D Shapes". PhD thesis. ETH Zürich, May 2013 (p. 41, 47, 106).

[Leš02] P. Leškovský. "Subdivision Surfaces". Master's thesis. Comenius University in Bratislava, May 2002 (p. 38).

[Pac13] A. Pacha. "Sensor Fusion for Robust Outdoor Augmented Reality Tracking on Mobile Devices". Master's thesis. 2013. ISBN: 978-3656964025 (p. 14, 145).

[Spr07] N. Sprynski. "Reconstruction de Courbes et de Surfaces à Partir de Données Tangentielles". PhD thesis. Université de Grenoble, July 2007 (p. 8, 10, 12, 13, 75).

Articles

[Abb+13] F. Abbasinejad, P. Joshi, C. Grimm, N. Amenta, and L. Simons. "Surface Patches for 3D Sketching". In: *Proc. Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM)*. 2013, pp. 53–60 (p. 108, 109).

[Abe+17] K. Aberman, O. Katzir, Q. Zhou, Z. Luo, A. Sharf, C. Greif, B. Chen, and D. Cohen-Or. "Dip Transform for 3D Shape Reconstruction". In: *ACM Trans. Graphics* 36.4 (2017). Proc. SIGGRAPH, 79:1–79:11 (p. 3, 5).

[ABG07] P.-A. Absil, C. Baker, and K. Gallivan. "Trust-Region Methods on Riemannian Manifolds". In: *Foundations of Computational Mathematics* 7.3 (2007), pp. 303–330 (p. 74).

[ABP16] C. Antonya, S. Butnariu, and C. Pozna. "Real-time Representation of the Human Spine with Absolute Orientation Sensors". In: *14th Proc. Int. Conf. on Control, Automation, Robotics and Vision (ICARCV)*. 2016, pp. 1–6 (p. 2, 9, 10).

[AJA11] F. Abbasinejad, P. Joshi, and N. Amenta. "Surface Patches from Unorganized Space Curves". In: *Computer Graphics Forum* 30.5 (2011). Proc. SGP, pp. 1379–1387 (p. 108, 109).

[AJC11] J. Andrews, P. Joshi, and N. Carr. "A Linear Variational System for Modelling from Curves". In: *Computer Graphics Forum* 30.6 (2011), pp. 1850–1861 (p. 53).

[Aro+17] R. Arora, R. H. Kazi, F. Anderson, T. Grossman, K. Singh, and G. Fitzmaurice. "Experimental Evaluation of Sketching on Surfaces in VR". In: *Proc. Conf. Human Factors in Computing Systems (CHI)*. ACM, 2017 (p. 53).

[BA11] N. Boumal and P.-A. Absil. "A Discrete Regression Method on Manifolds and its Application to Data on SO(n)". In: *Proc. World Congress of the Int. Federation of Automatic Control (IFAC)*. Milan, Italy, 2011, pp. 2284–2289 (p. 55, 71, 72).

[Bau+16] A. Bauer, A.-H. Dicko, F. Faure, O. Palombi, and J. Troccaz. "Anatomical Mirroring: Real-time User-specific Anatomy in Motion using a Commodity Depth Camera". In: *Proc. 9th Int. Conf. Motion in Games (MIG)*. ACM, 2016 (p. 2, 4).

[BBS08] S.-H. Bae, R. Balakrishnan, and K. Singh. "ILoveSketch: As-Natural-As-Possible Sketching System for Creating 3D Curve Models". In: *Proc. Symp. User Interface Software and Technology (UIST)*. ACM, 2008, pp. 151–160 (p. 53, 108).

[BC94] G. Brunnett and P. Crouch. "Elastic Curves on the Sphere". In: *Advances in Computational Mathematics* 2.1 (1994), pp. 23–40 (p. 71).

[Ber+10] E. Berberich, E. Fogel, D. Halperin, K. Mehlhorn, and R. Wein. "Arrangements on Parametric Surfaces I: General Framework and Infrastructure". In: *Mathematics in Computer Science* 4 (1 November 2010), pp. 45–66 (p. 38).

[Ber+14] M. Berger, A. Tagliasacchi, L. Seversky, P. Alliez, J. Levine, A. Sharf, and C. Silva. "State of the Art in Surface Reconstruction from Point Clouds". In: *Proc. Eurographics – STAR*. 2014 (p. 3).

[Bes+12] M. Bessmeltsev, C. Wang, A. Sheffer, and K. Singh. "Design-driven Quadrangulation of Closed 3D Curves". In: *ACM Trans. Graphics* 31.6 (2012). Proc. SIGGRAPH Asia, 178:1–178:11 (p. 108, 109).

[BFS10] L. Biard, R. Farouki, and N. Szafran. "Construction of Rational Surface Patches Bounded by Lines of Curvature". In: *Computer-Aided Geometric Design* 27.5 (2010), pp. 359–371 (p. 28, 109).

[BK04] M. Botsch and L. Kobbelt. "An Intuitive Framework for Real-time Freeform Modeling". In: *ACM Trans. Graphics* 23.3 (2004). Proc. SIGGRAPH, pp. 630–634 (p. 106–108, 131).

[BL17] D. Boltcheva and B. Lévy. "Surface Reconstruction by Computing Restricted Voronoi Cells in Parallel". In: *Computer Aided Design* 90 (2017), pp. 123–134 (p. 3).

[BLP10] I. Baran, J. Lehtinen, and J. Popović. "Sketching Clothoid Splines using Shortest Paths". In: *Computer Graphics Forum* 29.2 (2010), pp. 655–664 (p. 106).

[BM92] P. Besl and N. McKay. "A Method for Registration of 3-D Shapes". In: *Trans. Pattern Analysis and Machine Intelligence* 14 (2 1992), pp. 239–256 (p. 3, 84, 153).

[Bon+09] S. Bonnet, C. Bassompierre, C. Godin, S. Lesecq, and A. Barraud. "Calibration Methods for Inertial and Magnetic Sensors". In: *Sensors and Actuators A: Physical* 156.2 (2009), pp. 302–311 (p. 143).

[Bou+14] N. Boumal, B. Mishra, P.-A. Absil, and R. Sepulchre. "Manopt, a Matlab Toolbox for Optimization on Manifolds". In: *Journal of Machine Learning Research* 15 (2014), pp. 1455–1459 (p. 78).

[Bou13] N. Boumal. "Interpolation and Regression of Rotation Matrices". In: *Geometric Science of Information*. Vol. 8085. Lecture Notes in Computer Science. Springer Verlag, 2013, pp. 345–352 (p. 35, 36, 73).

[Car+15] M. Carmona, R. Perrier, L. Jouanet, N. Saguin-Sprynski, and O. Delcroix. "Morphopipe: Curvature Monitoring of Flexible Risers with MEMS Accelerometers". In: *Proc. IWSHM – 10th International Workshop on Structural Health Monitoring*. 2015 (p. 9).

[CL96] B. Curless and M. Levoy. "A Volumetric Method for Building Complex Models from Range Images". In: *Proc. SIGGRAPH 96. Annual Conference Series*. ACM, 1996, pp. 303–312 (p. 3).

[CMC07] J. Crassidis, L. Markley, and Y. Cheng. "Survey of Nonlinear Attitude Estimation Methods". In: *Journal of Guidance, Control, and Dynamics* 30.1 (2007), pp. 12–28 (p. 54, 63).

[CNW16] Y. Cao, L. Nan, and P. Wonka. "Curve Networks for Surface Reconstruction". 2016. arXiv: 1603.08753 [cs.GR] (p. 53).

[Coo67] S. Coons. "Surfaces for Computer-Aided Design of Space Forms". Tech. rep. Defense Technical Information Center (DTIC), 1967 (p. 12).

[Cor+11] F. Cordier, H. Seo, J. Park, and J. Y. Noh. "Sketching of Mirror-Symmetric Shapes". In: *IEEE Trans. Visualisation and Computer Graphics* 17.11 (November 2011), pp. 1650–1662 (p. 108).

[CPS13] K. Crane, U. Pinkall, and P. Schröder. "Robust Fairing via Conformal Curvature Flow". In: *ACM Trans. Graphics* 32 (4 2013). Proc. SIGGRAPH (p. 55, 75, 106).

[CR74] P. Ciarlet and P.-A. Raviart. "A Mixed Finite Element Method for the Biharmonic Equation". In: *Proc. Symp. Mathematical Aspects of Finite Elements in PDE*. 1974, pp. 125–145 (p. 108).

[Cra+13] K. Crane, F. de Goes, M. Desbrun, and P. Schröder. "Digital Geometry Processing with Discrete Exterior Calculus". In: *ACM SIGGRAPH 2013 Courses*. Anaheim, CA: ACM, 2013 (p. 41).

[CRS98] P. Cignoni, C. Rocchini, and R. Scopigno. "Metro: Measuring error on simplified surfaces". In: vol. 17. 2. Proc. Eurographics. Wiley Online Library, June 1998, pp. 167–174 (p. 129).

[CS00] B. Curless and S. Seitz. "3D Photography". In: *SIGGRAPH 2000 Courses*. New Orleans, LA, July 23, 2000 (p. 2).

[Dav68] P. Davenport. "A Vector Approach to the Algebra of Rotations with Applications". Tech. rep. National Aeronautics and Space Administration (NASA), 1968 (p. 60).

[DDG05] K. Das, P. Diaz-Gutierrez, and M. Gopi. "Sketching Free-Form Surfaces Using Network of Curves". In: *Proc. Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM)*. 2005, pp. 127–134 (p. 108).

[Des+99] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr. "Implicit Fairing of Irregular Meshes Using Diffusion and Curvature Flow". In: *Proc. SIGGRAPH 99. Annual Conference Series*. 1999, pp. 317–324 (p. 41, 106).

[Dou+16] M. Dou et al. "Fusion4D: Real-time Performance Capture of Challenging Scenes". In: *ACM Trans. Graphics* 35.4 (July 2016). Proc. SIGGRAPH, 114:1–114:13 (p. 2).

[Duc77] J. Duchon. "Splines Minimizing Rotation-Invariant Semi-Norms in Sobolev Spaces". In: *Constructive Theory of Functions of Several Variables. Proc. of a Conference Held at Oberwolfach April 25 – May 1, 1976*. Vol. 571. Lecture Notes in Mathematics. Springer Verlag, 1977, pp. 85–100 (p. 47).

[Far+87] G. Farin, G. Rein, N. Sapidis, and A. Worsey. "Fairing Cubic B-Spline Curves". In: *Computer-Aided Geometric Design* 4.1 (1987), pp. 91–103 (p. 106).

[Fie73] M. Fiedler. "Algebraic Connectivity of Graphs". In: *Czechoslovak Mathematical Journal* 23.2 (1973), pp. 298–305 (p. 46).

[Fon+17] A. Fondevilla, A. Bousseau, D. Rohmer, S. Hahmann, and M.-P. Cani. "Patterns from photograph: Reverse-engineering developable products". In: *Computers & Graphics* 66 (August 2017), pp. 4–13 (p. 108).

[GJ08] J. Giesen and M. John. "The Flow Complex: A Data Structure for Geometric Modeling". In: *Computational Geometry* (2008), pp. 178–190 (p. 109).

[Gri+06] E. Grinspun, Y. Gingold, J. Reisman, and D. Zorin. "Computing Discrete Shape Operators on General Meshes". In: *Computer Graphics Forum* 25.3 (2006). Proc. Eurographics, pp. 547–556 (p. 124).

[Guo+17] K. Guo, F. Xu, T. Yu, X. Liu, Q. Dai, and Y. Liu. "Real-time Geometry, Albedo and Motion Reconstruction Using a Single RGBD Camera". In: *ACM Trans. Graphics* (2017) (p. 2–4).

[GZ94] J. Gregory and J. Zhou. "Filling Polygonal Holes with Bicubic Patches". In: *Computer-Aided Geometric Design* 11.4 (1994), pp. 391–410 (p. 110).

[HCG16] A. Hermanis, R. Cacurs, and M. Greitans. "Acceleration and Magnetic Sensor Network for Shape Sensing". In: *IEEE Sensors Journal* 16.5 (March 2016), pp. 1271–1280 (p. 9, 10, 16, 75).

[HF02] D. Hansford and G. Farin. "Curve and Surface Constructions". In: *Handbook of Computer-Aided Geometric Design*. Ed. by G. Farin, J. Hoschek, and M.-S. Kim. Elsevier, 2002, pp. 165–192. ISBN: 978-0444511041 (p. 10).

[HP04] M. Hofer and H. Pottmann. "Energy-Minimizing Splines in Manifolds". In: *ACM Trans. Graphics* 23.3 (2004). Proc. SIGGRAPH, pp. 284–293 (p. 55, 71).

[HS08] T. Hoshi and H. Shinoda. "3D Shape Measuring Sheet Utilizing Gravitational and Geomagnetic Fields". In: *SICE Annual Conference*. IEEE, 2008, pp. 915–920 (p. 7–9).

[Hua+12] M. Huard, N. Sprynski, N. Szafran, and L. Biard. "Reconstruction de Surfaces Développables à Partir de Courbes Géodésiques". In: *Proc. GTMG – Journées du Groupe de travail en Modélisation Géométrique*. 2012 (p. 13).

[Hua+13] M. Huard, N. Sprynski, N. Szafran, and L. Biard. "Reconstruction of Quasi Developable Surfaces from Ribbon Curves". In: *Numerical Algorithms* 63.3 (2013), pp. 483–506 (p. 8, 9, 13, 14, 16).

[Hua+14] M. Huard, R. Farouki, N. Sprynski, and L. Biard. "C2 Interpolation of Spatial Data subject to Arc-Length Constraints using Pythagorean–Hodograph Quintic Splines". In: *Graphical Models* 76.1 (2014), pp. 30–42 (p. 9, 12).

[Hua+16] W. Huang, P.-A. Absil, K. Gallivan, and P. Hand. "ROPTLIB: an Object-Oriented C++ Library for Optimization on Riemannian Manifolds". Tech. rep. FSU16-14.v2. Florida State University, 2016 (p. 78).

[Ike+07] K. Ikeuchi, T. Oishi, J. Takamatsu, R. Sagawa, A. Nakazawa, R. Kurazume, K. Nishino, M. Kamakura, and Y. Okamoto. "The Great Buddha Project: Digitally Archiving, Restoring, and Analyzing Cultural Heritage Objects". In: *Int. Journal of Computer Vision* 75.1 (February 2007), pp. 189–208 (p. 2).

[IMT99] T. Igarashi, S. Matsuoka, and H. Tanaka. "Teddy: A Sketching Interface for 3D Freeform Design". In: *Proc. SIGGRAPH 99. Annual Conference Series*. 1999, pp. 409–416 (p. 53).

[Jac+10] A. Jacobson, E. Tosun, O. Sorkine, and D. Zorin. "Mixed Finite Elements for Variational Surface Modeling". In: *Computer Graphics Forum* 29.5 (2010). Proc. SGP, pp. 1565–1574 (p. 106–108, 131).

[JLW05] S. Jin, R. R. Lewis, and D. West. "A comparison of algorithms for vertex normal computation". In: *The Visual Computer* 21.1-2 (February 2005), pp. 71–82 (p. 113).

[JVV12] B. Jeuris, R. Vandebril, and B. Vandereycken. "A survey and comparison of contemporary algorithms for computing the matrix geometric mean". In: *Electronic Transactions on Numerical Analysis* 39 (2012), pp. 379–402 (p. 65).

[Kar77] H. Karcher. "Riemannian center of mass and mollifier smoothing". In: *Communications on pure and applied mathematics* 30.5 (1977), pp. 509–541 (p. 65).

[KBH06] M. Kazhdan, M. Bolitho, and H. Hoppe. "Poisson Surface Reconstruction". In: *Proc. Symp. Geometry Processing (SGP)*. 2006, pp. 61–70 (p. 3, 55, 75).

[KH13] M. Kazhdan and H. Hoppe. "Screened Poisson Surface Reconstruction". In: *ACM Trans. Graphics* 32.3 (2013). Proc. SIGGRAPH, 29:1–29:13 (p. 3).

[Kob97] L. Kobbelt. "Discrete Fairing". In: *Proc. 7th IMA Conf. on the Mathematics of Surfaces*. 1997, pp. 101–131 (p. 48, 106).

[KS07] L. Kara and K. Shimada. "Sketch-Based 3D-Shape Creation for Industrial Styling Design". In: *IEEE Computer Graphics and Applications* 27.1 (January 2007), pp. 60–71 (p. 108).

[Les17] L. Lesage. "ShapeIt: Développement d'une application de capture et de prétraitement de réseaux de courbes à partir de capteurs d'orientation sur smartphone". Tech. rep. RICM. Polytech Grenoble, August 2017 (p. 146, 147).

[Mar+07] L. Markley, Y. Cheng, J. Crassidis, and Y. Oshman. "Averaging Quaternions". In: *Journal of Guidance, Control, and Dynamics* 30.4 (2007), pp. 1193–1197 (p. 54, 65).

[Mey+03] M. Meyer, M. Desbrun, P. Schröder, and A. Barr. "Discrete Differential Geometry Operators for Triangulated 2-Manifolds". In: *Visualization and mathematics III*. Springer Verlag, 2003, pp. 35–57 (p. 41, 43, 106).

[MHV11] S. Madgwick, A. Harrison, and R. Vaidyanathan. "Estimation of IMU and MARG Orientation using a Gradient Descent Algorithm". In: *Proc. IEEE Int. Conf. Rehabilitation Robotics (ICORR)*. IEEE, June 2011, pp. 1–7 (p. 14, 145).

[Mil+16] B. Milosevic, F. Bertini, E. Farella, and S. Morigi. "A SmartPen for 3D Interaction And Sketch-based Surface Modeling". In: *The International Journal of Advanced Manufacturing Technology* 84.5 (2016), pp. 1625–1645 (p. 54).

[MM00] L. Markley and D. Mortari. "Quaternion Attitude Estimation using Vector Observations". In: *The Journal of the Astronautical Sciences* 48.2 (2000), pp. 359–380 (p. 54).

[MS91] H. Moreton and C. Séquin. "Surface Design with Minimum Energy Networks". In: *Proc. Symp. Solid Modeling and CAD/CAM Applications*. Austin, Texas, June 1991, pp. 291–301 (p. 105).

[MS92] H. Moreton and C. Séquin. "Functional Optimization for Fair Surface Design". In: *Computer Graphics* 26.2 (1992). Proc. SIGGRAPH, pp. 167–176 (p. 47, 105).

[Mur+16] O. Muratov, Y. Slynko, V. Chernov, M. Lyubimtseva, A. Shamsuarov, and V. Bucha. "3DCapture: 3D Reconstruction for a Smartphone". In: *IEEE Conf. on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, June 2016 (p. 3).

[Nea+07] A. Nealen, T. Igarashi, O. Sorkine, and M. Alexa. "FiberMesh: Designing Freeform Surfaces with 3D Curves". In: *ACM Trans. Graphics* 26.3 (July 2007). Proc. SIGGRAPH (p. 53, 108).

[New+11] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon. "KinectFusion: Real-time dense surface mapping and tracking". In: *Int. Symp. Mixed and Augmented Reality (ISMAR)*. IEEE, October 2011 (p. 2, 3).

[Nie04] G. Nielson. "ν-Quaternion Splines for the Smooth Interpolation of Orientations". In: *IEEE Trans. Visualisation and Computer Graphics* 10 (2 2004), pp. 224–229 (p. 11, 12, 55, 71).

[Nie83] G. Nielson. "A Method for Interpolating Scattered Data Based Upon a Minimum Norm Network". In: *Mathematics of Computation* 40.161 (1983), pp. 253–271 (p. 177, 178).

[Ort+16] S. Orts-Escolano et al. "Holoportation: Virtual 3D Teleportation in Real-time". In: *Proc. Symp. User Interface Software and Technology (UIST)*. ACM, 2016, pp. 741–754 (p. 2).

[Pan+15] H. Pan, Y. Liu, A. Sheffer, N. Vining, C.-J. Li, and W. Wang. "Flow-Aligned Surfacing of Curve Networks". In: *ACM Trans. Graphics* 34.4 (2015). Proc. SIGGRAPH, 127:1–127:10 (p. 104, 108–110, 119, 130, 132–135).

[PP93] U. Pinkall and K. Polthier. "Computing Discrete Minimal Surfaces and Their Conjugates". In: *Experimental Mathematics* 2.1 (1993), pp. 15–36 (p. 106).

[PR97] F. C. Park and B. Ravani. "Smooth Invariant Interpolation of Rotations". In: *ACM Trans. Graphics* 16.3 (July 1997), pp. 277–295 (p. 71).

[Rei67] C. Reinsch. "Smoothing by Spline Functions". In: *Numerische Mathematik* 10.3 (1967), pp. 177–183 (p. 55).

[Roc+01] C. Rocchini, P. Cignoni, C. Montani, P. Pingi, and R. Scopigno. "A Low Cost 3D Scanner Based on Structured Light". In: *Computer Graphics Forum* 20.3 (2001). Proc. Eurographics, pp. 299–308 (p. 2).

[Ros+07] K. Rose, A. Sheffer, J. Wither, M.-P. Cani, and B. Thibert. "Developable Surfaces from Arbitrary Sketched Boundaries". In: *Proc. Symp. Geometry Processing (SGP)*. 2007, pp. 163–172 (p. 108).

[Ros+14] G. Rosman, X.-C. Tai, R. Kimmel, and A. M. Bruckstein. "Augmented-Lagrangian Regularization of Matrix-Valued Maps". In: *Methods and Applications of Analysis* 21.1 (2014), pp. 121–138 (p. 54).

[Rus16] S. Rusinkiewicz. "Active 3D scanning methods". COS 429 – Computer Vision course at Princeton University. 2016 (p. 2).

[Sag+14] N. Saguin-Sprynski, L. Jouanet, B. Lacolle, and L. Biard. "Surfaces Reconstruction via Inertial Sensors for Monitoring". In: *Proc. EWSHM – 7th European Workshop on Structural Health Monitoring*. 2014 (p. 8, 9, 12, 16).

[Sag+16] N. Saguin-Sprynski, M. Carmona, L. Jouanet, and O. Delcroix. "New Generation of Flexible Risers Equipped with Motion Capture – Morphopipe System". In: *Proc. EWSHM – 8th European Workshop on Structural Health Monitoring*. 2016 (p. 2, 5, 9).

[Sch+09] R. Schmidt, A. Khan, K. Singh, and G. Kurtenbach. "Analytic Drawing of 3D Scaffolds". In: *ACM Trans. Graphics* 28.5 (December 2009). Proc. SIGGRAPH Asia, 149:1–149:10 (p. 108).

[Sch+17] N. Schertler, M. Tarini, W. Jakob, M. Kazhdan, S. Gumhold, and D. Panozzo. "Field-Aligned Online Surface Reconstruction". In: *ACM Trans. Graphics* 36.4 (July 2017). Proc. SIGGRAPH (p. 3).

[She96] J. Shewchuk. "Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator". In: *Proc. Applied Computational Geometry Towards Geometric Engineering*. Lecture Notes in Computer Science. Springer Verlag, 1996, pp. 203–222 (p. 119).

[Sho85] K. Shoemake. "Animating Rotation with Quaternion Curves". In: *Computer Graphics* 19.3 (1985). Proc. SIGGRAPH, pp. 245–254 (p. 11, 37, 54, 71).

[SK01] R. Schneider and L. Kobbelt. "Geometric Fairing of Irregular Meshes for Free-Form Surface Design". In: *Computer-Aided Geometric Design* 18.4 (2001), pp. 359–379 (p. 106, 119, 131).

[SLB11] N. Sprynski, B. Lacolle, and L. Biard. "Motion Capture of an Animated Surface via Sensors' Ribbons". In: *Int. Conf. Pervasive and Embedded Computing and Communication Systems (PECCS)*. 2011, pp. 421–426 (p. 9, 14).

[Slo+01] P.-P. Sloan, W. Martin, A. Gooch, and B. Gooch. "The Lit Sphere: A Model for Capturing NPR Shading from Art". In: *Proc. Graphics Interface*. Toronto, Canada: Canadian Information Processing Society, 2001, pp. 431–150 (p. 133).

[Spr+07a] N. Sprynski, D. David, B. Lacolle, and L. Biard. "Curve Reconstruction via a Ribbon of Sensors". In: *Proc. Int. Conf. Electronics Circuits Systems (ICECS)*. 2007, pp. 407–410 (p. 9, 12).

[Spr+07b] N. Sprynski, B. Lacolle, L. Biard, and D. David. "Curve and Surface Reconstruction via Tangential Information". In: *Curve and Surface Design*. Int. Conf. Curves and Surfaces, June 29–July 5, 2006, Avignon. Nashboro Press, Brentwood, 2007, pp. 254–263 (p. 9).

[Spr+08] N. Sprynski, N. Szafran, B. Lacolle, and L. Biard. "Surface Reconstruction via Geodesic Interpolation". In: *Computer Aided Design* 40.4 (April 2008), pp. 480–492 (p. 8, 9).

[SS14] B. Sadri and K. Singh. "Flow-Complex-Based Shape Reconstruction from 3D Curves". In: *ACM Trans. Graphics* 33.2 (2014), 20:1–20:15 (p. 108, 109).

[Sul08] J. Sullivan. "Curvatures of Smooth and Discrete Surfaces". In: *Discrete Differential Geometry*. Ed. by A. Bobenko, J. Sullivan, P. Schröder, and G. Ziegler. Vol. 38. Oberwolfach Seminars. Birkhäuser, 2008, pp. 175–188. ISBN: 978-3764386207 (p. 122).

[SWZ04] S. Schaefer, J. Warren, and D. Zorin. "Lofting Curve Networks Using Subdivision Surfaces". In: *Proc. Symp. Geometry Processing (SGP)*. 2004, pp. 103–114 (p. 108–110).

[Tan+13] P. Tanskanen, K. Kolev, L. Meier, F. Camposeco, O. Saurer, and M. Pollefeys. "Live Metric 3D Reconstruction on Mobile Phones". In: *IEEE Int. Conf. on Computer Vision*. IEEE, December 2013 (p. 3, 4).

[Tau95] G. Taubin. "A Signal Processing Approach to Fair Surface Design". In: *Proc. SIGGRAPH 95. Annual Conference Series*. 1995, pp. 351–358 (p. 46, 106).

[Tka+17] A. Tkach, A. Tagliasacchi, E. Remelli, M. Pauly, and A. Fitzgibbon. "Online Generative Model Personalization for Hand Tracking". In: *ACM Trans. Graphics* 36 (6 November 2017). Proc. SIGGRAPH Asia (p. 2, 3).

[TL94] G. Turk and M. Levoy. "Zippered Polygon Meshes from Range Images". In: *Proc. SIGGRAPH 94. Annual Conference Series*. ACM, 1994 (p. 3).

[VSK16] T. Várady, Salvi, and G. Karikó. "A Multi-sided Bézier Patch with a Simple Control Structure". In: *Computer Graphics Forum* 35.2 (2016). Proc. Eurographics, pp. 307–317 (p. 110).

[Wah65] G. Wahba. "A Least Squares Estimate of Satellite Attitude". In: *SIAM Review* 7.3 (July 1965), pp. 409–409 (p. 59).

[Wan+08] W. Wang, B. Jüttler, D. Zheng, and Y. Liu. "Computation of Rotation Minimizing Frames". In: *ACM Trans. Graphics* 27.1 (2008), p. 2 (p. 28, 109).

[Wim16] M. Wimmer. "Computational Aspects of Digital Fabrication – Module 3 (Scanning & Acquisition Pipeline)". 2016 (p. 2).

[Xu+14] B. Xu, W. Chang, A. Sheffer, A. Bousseau, J. McCrae, and K. Singh. "True2Form: 3D Curve Networks from 2D Sketches via Selective Regularization". In: *ACM Trans. Graphics* 33.4 (2014). Proc. SIGGRAPH, 131:1–131:13 (p. 53, 108, 109, 135).

[Zhu+13] Y. Zhuang, M. Zou, N. Carr, and T. Ju. "A General and Efficient Method for Finding Cycles in 3D Curve Networks". In: *ACM Trans. Graphics* 32.6 (2013). Proc. SIGGRAPH Asia, 180:1–180:10 (p. 108, 109, 115, 119).

[ZJC13] M. Zou, T. Ju, and N. Carr. "An Algorithm for Triangulating Multiple 3D Polygons". In: *Computer Graphics Forum* 32.5 (2013). Proc. SGP, pp. 157–166 (p. 109, 119, 169).

Online

[Creaform] Creaform. 2017. https://www.creaform3d.com (p. 4).

[ESA17] European Space Agency. *Gravity Field and Ocean Circulation Explorer (GOCE)*. https://earth.esa.int/web/guest/missions/esa-operational-eo-missions/goce (p. 6).

[Goo] Google. *Tilt Brush*. https://tiltbrush.com (p. 53, 54).

[Grasshopper] *Grasshopper - algorithmic modeling for Rhino*. http://www.grasshopper3d.com/ (p. 141).

[Ins] InstruMMents. *The 01 dimensioning instrument*. https://instrummments.com (p. 54).

[Lev+03] M. Levoy et al. *The Digital Michelangelo Project*. Last update: March 27, 2015. 2003. http://graphics.stanford.edu/projects/mich/ (p. 2).

[libigl] A. Jacobson, D. Panozzo, et al. *libigl: A Simple C++ Geometry Processing Library*. 2017. https://libigl.github.io (p. 135).

[Mau10] S. Maus. *Geomagnetism: Historical Main Field Change and Declination*. August 20, 2010. http://geomag.org/info/declination.html (p. 6).

[Morphosense] Morphosense. http://morphosense.com (p. 1, 2).

[Nai06] M. Nair. *Wandering of the Geomagnetic Poles*. July 31, 2006. https://ngdc.noaa.gov/geomag/GeomagneticPoles.shtml (p. 6).

[Rhino] *Rhinoceros*. https://www.rhino3d.com/ (p. 141).

[Sparkfun] Sparkfun. *9 Degrees of Freedom - Razor IMU*. https://www.sparkfun.com/products/retired/10736 (p. 7).

[Stanford3D] M. Levoy et al. *The Stanford 3D Scanning Repository*. 2014. http://graphics.stanford.edu/data/3Dscanrep/ (p. 2).

[Tau08] G. Taubin. *The IBM Pieta Project: A Historical Perspective*. 2008. http://mesh.brown.edu/3dpgp-2008/notes/IBM-PietaProject.pdf (p. 2).

https://tiborstanko.sk